

29. konferenca
Sodobne informacijske
tehnologije in storitve



ots

2026

Zbornik
prispevkov

Uredniki
Luka Pavlič
Tina Beranič
Marjan Heričko

Oblračne
rešitve
Mobilne aplikacije
Spletne aplikacije
Agilni razvoj
DevSecOps
Mikrostoritve
Umetna inteligenca
Podatki
Kakovost
Agenti



Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

OTS 2026

Sodobne informacijske tehnologije in storitve

Zbornik 29. konference

Uredniki

Luka Pavlič

Tina Beranič

Marjan Heričko

September 2026

Naslov <i>Title</i>	OTS 2026 Sodobne informacijske tehnologije in storitve <i>OTS 2026 Advanced Information Technologies and Services</i>
Podnaslov <i>Subtitle</i>	Zbornik 29. konference <i>Conference Proceedings of the 29th Conference</i>
Uredniki <i>Editors</i>	Luka Pavlič (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko) Tina Beranič (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko) Marjan Heričko (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)
Tehnična urednika <i>Technical editors</i>	Luka Pavlič (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko) Jan Perša (Univerza v Mariboru, Univerzitetna založba)
Oblikovanje ovitka <i>Cover designer</i>	Špela Čučko (Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)
Grafika na ovitku <i>Cover graphics</i>	OTS 2026, Čučko, 2026
Grafične priloge <i>Graphics material</i>	Vsi viri so lastni, če ni navedeno drugače. Avtorji in Pavlič, Beranič, Heričko (uredniki), 2026
Konferenca <i>Conference</i>	OTS 2026 Sodobne informacijske tehnologije in storitve
Kraj in datum <i>Location and date</i>	Maribor, Slovenija, 2. in 3. september 2026
Programski odbor <i>Program committee</i>	Luka Pavlič, vodja (Univerza v Mariboru, Slovenija), Marjan Heričko (Univerza v Mariboru, Slovenija), Tina Beranič (Univerza v Mariboru, Slovenija), Boštjan Grašič (Slovenija), Dean Korošec (Slovenija), Mateja Verlič Brunčič (Slovenija), Boštjan Kežmah (Slovenija), Boštjan Šumak (Univerza v Mariboru, Slovenija), Muhamed Turkanović (Univerza v Mariboru, Slovenija), Bojan Štok (Slovenija), Milan Gabor(Slovenija).
Organizacijski odbor <i>Organizing committee</i>	Tina Beranič (vodja), Luka Pavlič, Špela Čučko, Luka Četina, Lucija Brezočnik, Saša Brdnik, Mitja Gradišnik, Tjaša Heričko, Katja Kous, Alen Rajšp, Vasilka Saklamaeva, Lidija Vincekovič (vsi Univerza v Mariboru, Slovenija).
Založnik <i>Published by</i>	Univerza v Mariboru Univerzitetna založba Slomškov trg 15, 2000 Maribor, Slovenija https://press.um.si , zalozba@um.si
Izdajatelj <i>Issued by</i>	Univerza v Mariboru Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko Koroška cesta 46, 2000 Maribor, Slovenija https://feri.um.si , feri@um.si
Izdaja <i>Edition</i>	Prva elektronska izdaja
Vrsta publikacije <i>Publication type</i>	E-knjiga
Izdano <i>Published at</i>	Maribor, Slovenija, september 2026
Dostopno na <i>Available at</i>	https://press.um.si/index.php/ump/catalog/book/1156

Katalogni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni
knjižnici
COBISS.SI-ID 288695555
ISBN 978-961-299-188-3 (PDF)

Zbornik je bil uredniško jezikovno pregledan.



© Univerza v Mariboru, Univerzitetna založba
/ University of Maribor, University of Maribor Press

Besedilo/ Text © avtorji prispevkov in Pavlič, Beranič, Heričko, 2026

To delo je objavljeno pod licenco Creative Commons Priznanje avtorstva-Nekomercialno-Brez predelav 4.0 Mednarodna. / *This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International License.*

Uporabnikom je dovoljeno reproduciranje brez predelave avtorskega dela, distribuiranje, dajanje v najem in priobčitev javnosti samega izvirnega avtorskega dela, in sicer pod pogojem, da navedejo avtorja in da ne gre za komercialno uporabo.

Vsa gradiva tretjih oseb v tej knjigi so objavljena pod licenco Creative Commons, razen če to ni navedeno drugače. Če želite ponovno uporabiti gradivo tretjih oseb, ki ni zajeto v licenci Creative Commons, boste morali pridobiti dovoljenje neposredno od imetnika avtorskih pravic.

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

ISBN 978-961-299-188-3 (pdf)
978-961-299-189-0 (USB ključ)
978-961-299-190-6 (mehka vezava)

DOI <https://doi.org/10.18690/um.feri.7.2026>

Cena
Price Brezplačni izvod

Odgovorna oseba založnika
For publisher Prof. dr. Dean Korošak, rektor Univerze v Mariboru

Citiranje
Attribution Pavlič, L., Beranič, T., Heričko, M. (ur.). (2026). *OTS 2026 Sodobne informacijske tehnologije in storitve: Zbornik 29. konference*. Univerza v Mariboru, Univerzitetna založba. doi: 10.18690/um.feri.7.2026

<https://www.ots.si>

Prispevki predstavljajo stališča avtorjev, ki niso nujno usklajena s stališči organizatorja, programskega odbora in urednikov zbornika. Zato ne sprejemajo nobene formalne odgovornosti zaradi morebitnih avtorjevih pravopisnih ali strokovnih napak, netočnosti in neustrezne rabe virov.

Spoštovane, spoštovani,

konferenca OTS predstavlja platformo za izmenjavo izkušenj in spoznanj iz uspešnih IT projektov, identifikacijo prihodnjih trendov ter argumentirano izmenjavo mnenj o dogajanju v industriji. Stalnica na konferenci so bogate in raznovrstne vsebine, med katerimi so sodobne IT arhitekture, IT integracije, organizacijski vidiki razvojno-operativnih aktivnosti, podatkovne in inteligentne rešitve na podlagi umetne inteligence, kibernetska varnost, jeziki in ogrodja za učinkovit razvoj sodobnih uporabniških vmesnikov, zagotavljanje kakovosti informacijskih rešitev in storitev ter mnoge druge. Z njimi spodbujamo napredek na vseh področjih poslovanja organizacij kot tudi delovanja in življenja posameznikov, a hkrati opozarjamo na pasti, ki jih lahko poraja nekritična uporaba informacijskih rešitev.

Zbornik konference OTS 2026 vsebuje zanimive prispevke, ki iz različnih zornih kotov osvetlujejo tehnične in organizacijske vidike razvoja, nadgradnje, prilagajanja in upravljanja informacijskih rešitev in storitev. Umetna inteligenca, zlasti generativna umetna inteligenca, pogovorni roboti in inteligentni agenti, vse pomembneje vplivajo na celoten življenjski cikel sodobnih IS/T rešitev ter spreminjajo tudi vloge razvijalcev, arhitektov in drugih IT strokovnjakov. Zato prispevki zajemajo mnogo konkretnih izkušenj iz industrije. Še posebej izpostavljene vsebine prispevkov se nanašajo na kibernetsko varnost, zagotavljanje kakovosti, sodobne pristope k hranjenju in obdelavi podatkov, regulativne okvire in standardizacijo ter odgovorno in varno uporabo novih tehnologij. Veseli nas, da konferenca OTS še naprej prispeva k boljši povezanosti IT strokovnjakov, informatikov, arhitektov, razvijalcev naprednih IT rešitev in storitev ter akademske sfere in gospodarstva.

Ponosni smo na izjemne uspehe sodelujočih podjetij, ki z znanjem, pogumom in sposobnostmi vpeljujejo odlične tehnološko-organizacijske rešitve z visoko dodano vrednostjo za uporabnike v regiji in širše.

Luka Pavlič

vodja programskega
odbora OTS 2026

Tina Beranič

vodja organizacijskega
odbora OTS 2026

Marjan Heričko

predsednik konference
OTS 2026

KAZALO

Ko močna avtentikacija ni dovolj: forenzični pogled na bančno goljufijo z oddaljenim dostopom Boštjan Kežmah, Miha Strehar	1
Izkušnje z uporabo lokalnih modelov LLM v sistemu COBISS Rene Svenšek, Andrej Krajnc, Bojan Štok	13
x402: arhitektura in praktična uporaba strojnih plačil z uporabo HTTP 402 Martin Ferenec, Adnan Bratanović, Teo Lah, Muhamed Turkanović	25
Sistem SKRINJA: odpiranje podatkov za javnost in napovedno analitiko z uporabo umetne inteligence Paula Jožefa Kolenko, Alenka Krebs	45
Agentni sistem RAG za podporo zaposlenim v večjih podjetjih Rok Hladin	55
Ko uporabnik ni več samo človek: od priporočil do avtonomnih agentov v poslovnih aplikacijah Boštjan Grašič, Gal Badrov, Hanan Mešič, Saša Grašič	65
Od dveh generalov do dveh storitev Matjaž Prtenjak	77
Zrelost cevovodov neprekinjene dostave informacijskih rešitev Luka Četina, Luka Pavlič	87
Izboljšanje pretoka dela v kompleksnem projektu Tadej Volgemut	97
AI v kibernetski varnosti: nov napadalni vektor in pametnejša obramba Milan Gabor	105
Sistemske napadi na programske dobavne verige v dobi umetne inteligence Gregor Jošt, Viktor Taneski	115
Kdo in zakaj potrebuje Program nagrajevanja za varnostne hrošče? Denis Tomašević	141
Evropske digitalne denarnice EUDIW in EBW: od regulative do prakse Martin Domajnko, Vid Keršič, Muhamed Turkanović	145
Semantična podatkovna plast: manjkajoči temelj analitike nove generacije Jure Jeraj, Stevanče Nikoloski	161
Podatkovne semantične smernice kot temelj upravljanja podatkov in semantične interoperabilnosti v javni upravi Martina Furlan, Miha Jesenko, Karmen Kern Pipan	173
CODORQ: ogrodje za orkestracijo razvoja programske opreme z različnimi agentskimi orodji Tomaž Kolmanič	183

Migracija MAUI aplikacij z uporabo agentov UI Alen Granda _____	201
Ko umetna inteligenca prevzame razvoj internih orodij Gregor Kovač, Tomaž Kozlar, Žan Žvokelj _____	211
Nov, na prstnem odtisu temelječ biometrični varnostni sistem Komlen Lalović, Dalibor Trajkovic, Nebojsa Bacanin Dzakula _____	229
Samodejna priprava prometnih poročil za obveščanje javnosti preko radijskih postaj Ruben Ferreira, Rok Kužel, Tímotej Knez, Slavko Žitnik _____	235
Od podatkov do prvakov: integracija podatkov za podpora odločanja v športu Jernej Flisar _____	251

Ko močna avtentikacija ni dovolj: forenzični pogled na bančno goljufijo z oddaljenim dostopom

Boštjan Kežmah, Miha Strehar

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
bostjan.kezmah@um.si, miha.strehar@um.si

Prispevek obravnava porast bančnih prevar, pri katerih napadalci namesto iskanja tehničnih ranljivosti uporabljajo napreden socialni inženiring (telefonske klice) in legitimna orodja za oddaljeni dostop (npr. AnyDesk). Čeprav močna avtentikacija (SCA) znižuje splošno tveganje, so trenutni pristopi v elektronskem bančništvu ranljivi, saj vse varnostne faktorje združujejo na eni kompromitirani napravi. Vrzel lahko premostijo napredne tehnologije, kot je standard v pripravi Secure Payment Confirmation (SPC). Banke trenutno odgovornost pogosto prelagajo na žrtve zaradi »hude malomarnosti«. Novejša sodna praksa in izkušnje alternativnega reševanja sporov (IRPS) potrjujejo, da podleganje manipulaciji ne sme samodejno pomeniti hude malomarnosti. Prihajajoča evropska zakonodaja (PSR in PSD3) bo to izrecno prepovedala in ob uspešnem lažnem predstavljanju uvedla jasnejšo odgovornost bank ter pogoje povračila sredstev. Poudarek kibernetске varnosti se tako premika od preverjanja identitete (»kdo si«) k potrjevanju dejanskega razumevanja in namena transakcije.

Ključne besede:

bančne prevare
socialni inženiring
močna avtentikacija
huda malomarnost
plačilni predpisi

1 Uvod

Digitalizacija finančnih storitev in implementacija revidirane direktive o plačilnih storitvah (PSD2) sta v evropskem prostoru bistveno spremenili varnostne zahteve elektronskega bančništva. Z obvezno uvedbo močne avtentikacije strank (angl. »Strong Customer Authentication« - SCA) naj bi se zmanjšalo število klasičnih prevar, ki so temeljile na preprosti kraji identitete ali prestrezanju statičnih gesel. Vendar kibernetški prostor ostaja dinamičen ekosistem, v katerem se napadalci hitro prilagajajo novim regulativnim zahtevam. Namesto neposrednih tehničnih vdorov v bančne sisteme so napadalci težišče svojih napadov preusmerili na najšibkejši člen kibernetške varnosti - človeka.

Skupno poročilo Evropskega bančnega organa (EBA) in Evropske centralne banke (ECB) o plačilnih prevarah izpostavlja zaskrbljujoč trend. Čeprav močna avtentikacija znižuje splošno stopnjo tveganja, se je preostalo tveganje izrazito zgostilo znotraj transakcij, ki so uspešno avtentificirane z mehanizmi SCA. Danes več kot polovico vrednosti vseh kreditnih prenosov v Evropskem gospodarskem prostoru (EGP), ki temeljijo na goljufiji, predstavljajo prevare vrste »odobreno potisno plačilo« (angl. »Authorised Push Payment« - APP), pri katerih je plačnik s pomočjo socialnega inženiringa zmanipuliran do te mere, da sam z uporabo vseh zahtevanih varnostnih elementov odobri sporno transakcijo. [1]

Izhodišče te raziskave je specifičen, forenzično obravnavan in vse pogostejši tip incidenta, pri katerem storilec z uporabo naprednega socialnega inženiringa, najpogosteje v obliki telefonskega klica (angl. »vishing«) in popolnoma legitimnih aplikacij za oddaljeni dostop (angl. »Remote Access Tools« - RAT), kot sta AnyDesk in TeamViewer, prevzame nadzor nad uporabnikovo napravo. Z namenom iskanja sistemskih rešitev za izoliranje najšibkejšega člena kibernetške varnosti se ta prispevek osredotoča na identifikacijo in analizo tehnološke vrzeli na področju arhitekture avtentikacijskih mehanizmov. Primerja najsodobnejše (angl. »state-of-the-art« - SOTA) pristope k potrjevanju transakcij, kot so kriptografski ključi (angl. »passkeys«) in specifikacija za varno potrjevanje plačil, ki je še v pripravi (angl. »Secure Payment Confirmation« - SPC), z dejansko razširjenimi, a pogosto nezadostnimi praksami, ki jih trenutno uporablja bančna industrija v EU.

Vzporedno s tehnološko analizo prispevek obravnava pravni in regulatorni vidik tovrstnih dogodkov. Jedro pravnih sporov ob zlorabah elektronskih bank namreč predstavljata opredelitev in dokazovanje »hude malomarnosti« (»gross negligence«) na strani uporabnika. Analiza vključuje slovenski zakonodajni okvir, usmeritve in opozorila nacionalnih organov ter organizacij (SI-CERT, Urad Vlade RS za informacijsko varnost - URSIV, Združenje bank Slovenije - ZBS, Varni na internetu), prakse alternativnega reševanja sporov pri ZBS ter sodno prakso in prihajajoča evropska direktiva (PSD3) in uredba (PSR), ki bosta temeljito prenovili opredelitev izhodišč za določanje stopnje zadostne skrbnosti in odgovornosti ponudnikov plačilnih storitev.

2 Forenzična anatomija bančne goljufije z oddaljenim dostopom

Za razumevanje omejitev trenutnih avtentikacijskih mehanizmov bomo razčlenili potek sodobnega napada, ki združuje elemente psihološke manipulacije in zlorabe tehnologije. Forenzične preiskave incidentov, obravnavanih v digitalnih laboratorijih in prijavljenih nacionalnemu odzivnemu centru SI-CERT, razkrivajo visoko strukturiran način dela napadalcev. Ti napadi ne temeljijo na iskanju ranljivosti vrste »ničelnega dne« (angl. »zero-day«) v bančni aplikaciji, temveč na t. i. taktiki »živeti od zemlje« (angl. »Living off the Land« - LotL), kjer napadalci zlorabljajo obstoječa, legitimna sistemska orodja in procese.

2.1. Faze napada in socialni inženiring

Napad se običajno odvija v več skrbno načrtovanih fazah, ki so zasnovane za postopno grajenje zaupanja in ustvarjanje občutka nujnosti:

1. Pri pripravi in vzpostavitvi prvega stika napadalcu najprej pridobijo osnovne kontaktne podatke žrtev, pogosto iz javno dostopnih virov ali predhodnih vdorov v baze podatkov. Žrtev prejme navidezno legitimno sporočilo SMS ali elektronsko pošto, ki obvešča o izrednem dogodku, na primer »varnostni posodobitvi zaradi združevanja bank«, »blokadu računa zaradi sumljive aktivnosti« ali »odprtem kreditnem zahtevku«. Sporočilo vsebuje povezavo do prepričljive lažne spletne strani, kjer žrtev vnese svoje osnovne prijavnne podatke in telefonsko številko.
2. Sledita lažno predstavljanje in glasovna manipulacija v obliki telefonskega klica. Klicatelji komunicirajo v tekočem slovenskem jeziku, izkazujejo visoko raven profesionalnosti in se pogosto predstavijo kot uslužbenci tehnične pomoči banke, operaterji SI-CERT-a ali celo kot predstavniki policije in Banke Slovenije. Z uporabo tehnologije potvarjanja klicne številke se lahko na zaslonu žrtve celo prikaže dejanska telefonska številka banke, kar močno poveča verodostojnost klica.
3. Pod pretvezo, da mora žrtev zaradi varnosti nujno nadgraditi sistemsko zaščito ali preveriti stanje na računu, jo klicatelj usmerja, naj iz uradne trgovine z aplikacijami (Google Play, Apple App Store) ali prek spletne strani prenese aplikacijo za oddaljeni dostop (AnyDesk, TeamViewer, Chrome Remote Desktop). Ker gre za legitimna komercialna orodja, namenjena tehnični podpori, varnostni mehanizmi operacijskega sistema in nameščeni protivirusni programi teh aplikacij ne prepoznajo kot zlonamerno programsko opremo (angl. »malware«).

2.2. Forenzični artefakti in izkoriščanje seje

Ko žrtev namesti aplikacijo in napadalcu posreduje kodo za vzpostavitev seje, napadalec pridobi popoln ali delni nadzor nad napravo. Z vidika digitalne forenzike je to ključni trenutek. Forenzične analize pomnilnika, registrov (v okolju Windows) ali datotečnega sistema Android (prenesene datoteke, medpomnilnik, dnevniške datoteke aplikacije AnyDesk) sicer jasno pokažejo, da je bila oddaljena seja vzpostavljena, da so bile prenesene datoteke ali da so se izvajali kliki. Vendar pa ti artefakti zgolj potrjujejo prisotnost oddaljene seje, pa ne omogočajo preprečevanja zlorabe, saj potekajo znotraj avtoriziranega kanala.

Napadalec na tej točki izkoristi specifične funkcionalnosti orodij za oddaljeni dostop: [2]

1. Da bi preprečil, da bi žrtev videla, kaj se dogaja na zaslonu, napadalec aktivira funkcijo zatemnitve zaslona, ki jo omogočajo orodja, kot je AnyDesk za »zasebnost« med oddaljenim delom.
2. Medtem ko je zaslon črn, napadalec na napravi žrtve odpre mobilno banko ali pa žrtev prosi, naj mu »za preverjanje« prebere kodo SMS, ki jo je pravkar prejela. Če mobilna banka uporablja potisna (angl. »push«) sporočila za avtorizacijo (npr. potrditev z vnosom varnostne PIN kode znotraj aplikacije), lahko napadalec, če je pridobil ustrezna pooblastila (npr. z uporabo storitev za dostopnost v sistemu Android), potrdi transakcijo kar sam ali pa preprosto prosi žrtev, naj na slepo ali pod lažno pretvezo potrdi postopek.
3. Sredstva se nato prenesejo na račune t. i. denarnih mul, pogosto v tujino, ali v kriptomenjalnice, kjer se sled za denarjem hitro izgubi.

3 Omejitve obstoječih sistemov avtentikacije

Iz opisane anatomije napada sledi arhitekturna napaka v načinu, kako bančne institucije v EU in Sloveniji implementirajo zahteve direktive PSD2 v delu, ki se nanaša na močno avtentikacijo strank. [3][4] Predpisi od ponudnikov plačilnih storitev zahtevajo preverjanje na podlagi dveh od treh neodvisnih elementov (nekaj, kar vem, nekaj, kar imam, nekaj, kar sem). Kljub temu se v praksi soočamo z resnimi omejitvami.

3.1. Močna avtentikacija pri napadenem komitentu

Evropski bančni organ (EBA) opozarja, da čeprav se je število prevar z ukradenimi poverilnicami (angl. »unauthorised fraud«) zmanjšalo, je stopnja uspešnosti prevar pri transakcijah, ki so bile podvržene močni avtentikaciji (SCA), presenetljivo visoka. [5] Metodologija SCA, kot se trenutno izvaja v praksi (npr. SMS OTP ali preprost pritisk na gumb »Potrdi« v mobilnem žetonu ob uporabi prstnega odtisa), ne upošteva kognitivnega stanja uporabnika. Če je bil uporabnik žrtev socialnega inženiringa in je prepričan, da s potrditvijo transakcije dejansko »odpravlja blokado računa«, bo brez oklevanja zagotovil zahtevana faktorja (telefon in prstni odtis).

V takšnem primeru SCA ne deluje več kot varnostna pregrada pred prevaro, temveč postane izjemno zanesljivo forenzično in kriptografsko orodje za potrditev napačne odločitve. Napadi z dajanjem navodil po telefonu in z uporabo orodij za oddaljeni dostop (RAT) uspejo prav zato, ker obidejo tehnične kontrole in izkoristijo uporabnikovo napačno razumevanje konteksta.

3.2. Tveganje zgostitve vseh faktorjev na eni napravi

Največja praktična in arhitekturna slabost sodobnega elektronskega bančništva je zgostitev (koncentracija) vseh avtentikacijskih faktorjev in odločevalskih procesov na eni sami končni napravi (npr. pametnem telefonu). Koncept večfaktorske avtentikacije zgodovinsko temelji na predpostavki ločenih in neodvisnih kanalov (angl. »Out-of-Band« - OOB). Če napadalec kompromitira računalnik, nima dostopa do pametne kartice v žepu uporabnika ali do njegovega ločenega telefona.

V sodobnem mobilnem bančništvu pa uporabnik na isti napravi:

1. Posluša navodila napadalca (telefonski klic).
2. Upravlja vmesnik spletne ali mobilne banke.
3. Prejema drugi varnostni faktor (SMS žeton ali potisno potrditev v ločeni aplikaciji).
4. Dovoljuje izvajanje aplikacije za oddaljeni dostop, ki zajema zaslon.

S tem se odpira vprašanje dejanske neodvisnosti faktorjev, ki ga zahteva 9. člen Regulativnih tehničnih standardov (RTS) za SCA (Uredba EU 2018/389). Z oddaljenim dostopom se vzpostavi enotno kompromitirano okolje. Prevarant z dostopom do zaslona (ali na primer z uporabo storitev za dostopnost za branje vsebine na sistemu Android) lahko v realnem času prestereže kodo OTP iz SMS sporočila in jo vnese v plačilni nalog, ne da bi uporabnik, katerega zaslon je zatemnjen, to opazil. Zanašanje industrije izključno na posest naprave in biometrično odklepanje zaslona postane neučinkovito, če naprava ni pod popolnim nadzorom uporabnika.

4 Napredne metode potrjevanja

Da bi presegli omejitve trenutnih praks in uspešno zajezili goljufije z oddaljenim dostopom, tehnološki konzorciji (W3C, FIDO Alliance) in napredni deležniki v plačilni industriji razvijajo in uvajajo nove asimetrične kriptografske metode avtentikacije. Te metode rešujejo problem napadov z ribarjenjem in manipulacije na strukturni ravni.

4.1. FIDO2, WebAuthn in Passkeys

Sodobna zaščita identitete (SOTA) opušča statična gesla in SMS potrditve ter prehaja na arhitekturo FIDO2 (angl. »Fast IDentity Online«) in standard WebAuthn, bolj znan pod komercialnim imenom »passkeys« (kriptografski ključi). [6]

Namesto da bi si uporabnik in banka delila »skrivnost« (geslo ali kodo), passkeys temeljijo na asimetrični kriptografiji javnega in zasebnega ključa. Zasebni ključ se generira in trajno hrani izključno v varni strojni enklavi naprave (angl. »Secure Enclave« ali »Trusted Platform Module« - TPM) in v različici passkeys, ki je vezana na napravo, naprave nikoli ne zapusti.

Avtentikacija se izvede lokalno: banka (angl. »relying party«) pošlje kriptografski izziv (angl. »challenge«), naprava pa preveri identiteto uporabnika (z biometrijo ali kodo PIN) in izziv podpiše z zasebnim ključem. Ker je kriptografski podpis neločljivo vezan na specifično spletno domeno (angl. »origin-bound«), je ta metoda že po zasnovi odporna proti klasičnim napadom z ribarjenjem. Tudi če uporabnik podatke vpisuje na lažni strani (npr. lažnabanka.si), se kriptografski podpis, ustvarjen za to domeno, ne bo ujema z javnim ključem na strežniku prave banke.

Omejitve tehnologije passkeys v zvezi z direktivo PSD2 so povezane s tem, da čeprav so passkeys učinkoviti za prijavo v sistem, v svoji osnovni obliki protokola WebAuthn (metoda webauthn.get) ne zadoščajo za izpolnjevanje specifičnih zahtev PSD2 glede potrjevanja plačil. PSD2 (v 5. členu RTS) namreč zahteva dinamično povezovanje (angl. »Dynamic Linking«). To pomeni, da mora biti avtentikacijska koda neločljivo specifična za natančen znesek transakcije in identiteto prejemnika. Kakršna koli poznejša sprememba zneska ali prejemnika mora povzročiti neveljavnost avtentikacijske kode. Poleg tega mora biti zagotovljeno zavedanje plačnika (angl. »Payer Awareness«) - uporabniku morata biti pred potrditvijo nedvoumno prikazana znesek in prejemnik. Navaden izziv WebAuthn je zgolj naključno generiran niz podatkov brez semantičnih informacij o transakciji, zato sama prijava v sistem s tehnologijo passkey ne ustreza določilom o dinamičnem povezovanju.

4.2. Standard Secure Payment Confirmation (SPC) kot rešitev tehnološke vrzeli

Za premostitev te vrzeli je konzorcij W3C (angl. »World Wide Web Consortium«) v sodelovanju z zavezništvom FIDO in plačilno industrijo razvil specializirani standard spletnega vmesnika API za varno potrjevanje plačil (angl. »Secure Payment Confirmation« - SPC). [7] Ta gradi neposredno na infrastrukturi WebAuthn in ji dodaja ključne plačilne mehanizme, ki omogočajo popolno skladnost z zahtevami PSD2 in prihajajoče direktive PSD3.

Mehanizem SPC vzpostavlja model »kar vidiš, to podpišeš« (angl. »What-you-see-is-what-you-sign«) z naslednjimi tehnološkimi značilnostmi:

1. Vgrajeni prikaz v brskalniku (angl. »Browser-native UI«). Kadar banka zahteva potrditev plačila z uporabo SPC, API v uporabnikovem brskalniku (ali operacijskem sistemu) sproži prikaz posebnega, standardiziranega pogovornega okna (dialoga). To okno nadzoruje operacijski sistem in ne spletna stran, zato ga zlonamerna skripta ali prevarant z osnovnim zajemom zaslona bistveno težje prikrije ali ponaredi. V tem oknu so neločljivo prikazani točen znesek, valuta in ime prejemnika (angl. »merchant/payee«).
2. Najpomembnejša inovacija SPC je kriptografska povezava transakcijskih podatkov z neposredno vključitvijo prikazanih konkretnih plačilnih podatkov v podatkovno strukturo, ki je podpisana. Tako sta znesek in podatki o prejemniku vključena v podatke, ki jih SPC prikaže uporabniku.
3. Ker so transakcijski podatki kriptografsko zapečateni in vgrajeni v sam avtentikacijski zahtevek neposredno z uporabo strojne enklave, napadalec ne more naknadno spremeniti podatkov o nakazilu, ne da bi s tem razveljavil veljavnost ustvarjene potrditvene kode.

Kot izhaja iz primerjave v tabeli 1, bi implementacija arhitekture SPC namesto enostavnih enkratnih gesel SMS ali generičnih mobilnih žetonov predstavljala pomemben napredek na področju varnosti. Kljub temu tehnološka vrzel ostaja velika, saj banke zaradi stroškov integracije, skrbi glede uporabniške izkušnje in počasnega sprejemanja standardov v vseh brskalnikih še vedno vztrajajo pri zastarelih in ranljivih metodah. S tem tveganje oddaljenega prevzema naprave nesorazmerno prenašajo na komitente.

Tabela 1: Primerjava varnosti metod potrjevanja plačil

Merilo varnosti (PSD2/PSD3 zahteve)	SMS OTP (praksa v postopnem opuščanju)	Potisna avtentikacija	W3C Secure Confirmation (SOTA)	Payment
Prikaz zavedanja plačnika (angl. »Payer Awareness«)	Znotraj besedila SMS sporočila (pogosto ignorirano).	Ločena aplikacija (pogosto le obvestilo »Potrdite transakcijo«).	Vgrajeni dialog operacijskega sistema, specifičen za plačilo, nadzorovan zunaj vpliva spletne strani.	
Odpornost na ribarjenje (prestrežanje poverilnic)	Nizka. Uporabnik prebere in vnese kodo na lažni strani.	Srednja. Zanaša se na to, da uporabnik preveri podrobnosti.	Odporna. Kriptografija je neločljivo povezana s pravo domeno banke (angl. »origin-bound«).	
Dinamično povezovanje (angl. »Dynamic Linking«)	Koda je povezana z zneskom na zalednem sistemu banke.	Koda je poslana v aplikacijo, znesek je prikazan v uporabniškem vmesniku.	Kriptografsko vgrajeno. Znesek in prejemnik sta del objekta CollectedClientData, podprtega s strojnim ključem.	
Odpornost proti zlorabi z AnyDesk/RAT	Ranljiv. Napadalec prebere SMS na zatemnjenem zaslonu ali ga razkrije žrtev sama.	Nizka odpornost. Napadalec lahko sam klikne gumb »potrdi« ali usmerja žrtev.	Visoka odpornost. Zahteva eksplicitno lokalno potrditev točnega zneska na ravni operacijskega sistema, ki ga RAT težje manipulira brez prekinitve podpisa.	

4.3. Preverjanje prejemnika (Verification of Payee - VoP) kot komplementarni zaščitni sloj

Kadar tehnološke ovire same po sebi ne morejo preprečiti, da bi zavedeni uporabnik izvedel avtentikacijo transakcije, je nujno uvesti kontrolne mehanizme, ki prekinajo kognitivno zanko napadalca in ustvarijo t. i. pozitivno trenje (angl. »friction«) pred končnim odlivom sredstev. Forenzična analiza primerov zlorab z oddaljenim dostopom in telefonskim klicem (RAT/vishing) kaže, da ena ključnih točk prevare nastopi, ko storilec žrtvi narekuje vnos tuje številke IBAN (račun denarne mule) in ji zagotovi, da gre za »varen račun«, »rezervni račun pri Banki Slovenije« ali »FURS«. Žrtev pod vplivom stresa ne preverja lastništva računa, saj do nedavnega to v sistemu SEPA ni bilo niti tehnološko mogoče v realnem času.

Kot ključen varnostni ukrep in rešitev tega problema je Evropska unija v okviru Uredbe o takojšnjih plačilih in prihajajoče Uredbe o plačilnih storitvah (PSR) uzakonila storitev Preverjanje prejemnika (angl. »Verification of Payee – VoP«). [8] VoP vzpostavlja obveznost bank, da plačniku še pred potrditvijo in izvršitvijo kreditnega prenosa omogočijo preverjanje ujemanja med vnesenim imenom prejemnika in dejanskim imenom imetnika ciljnega računa IBAN. Ta mehanizem je postal obvezen in brezplačen za kreditne prenose v eurih v evroobmočju 9. oktobra 2025, nova uredba PSR pa ga širi na vse valute znotraj EU.

V scenariju, kjer napadalec z uporabo orodja za oddaljeni dostop (npr. AnyDesk) prevzame nadzor in vnese račun IBAN denarne mule, medtem ko žrtvi zatrjuje, da gre za »pokojninski sklad«, sistem VoP pred zahtevo po biometrični potrditvi vrne opozorilo o neujemanju ali prikaz dejanskega imena lastnika računa. Takšno eksplicitno in v realnem času generirano opozorilo v številnih primerih zadostuje, da posvari žrtev v stanju psihološke manipulacije in zmanjša uspešnost socialnega inženiringa na primer pri direktorskih prevarah in prevar vmešavanja v komunikacijo (angl. »Business Email Compromise« - BEC), vendar je brez učinka, kadar lahko napadalec z oddaljenim dostopom popolnoma samostojno in brez sodelovanja žrtve ustvari in potrdi plačilni nalog s podatki poljubnega prejemnika.

5 Pravni okvir in doktrina hude malomarnosti v slovenskem in evropskem pravu

Presečišče kompleksnih informacijskih tehnologij in prava pride do polnega izraza v trenutku po izvršeni zlorabi, ko komitent (potrošnik) od svoje banke zahteva vračilo odtujenih sredstev. Pravni spori, povezani z zlorabami elektronskega bančništva z uporabo socialnega inženiringa, se primarno osredotočajo na porazdelitev odgovornosti in razlago pravnih standardov skrbnosti.

5.1. ZPlaSSIED in odgovornost ponudnika

V slovenskem pravnem redu je področje izvajanja plačilnih storitev in odgovornosti zanje urejeno v Zakonu o plačilnih storitvah, storitvah izdajanja elektronskega denarja in plačilnih sistemih (ZPlaSSIED), s katerim je bila direktiva PSD2 prenesena v nacionalno zakonodajo. [9]

Zakon izhaja iz temeljnega načela zaščite potrošnika. V primeru neodobrene plačilne transakcije je v skladu s 136. členom ZPlaSSIED ponudnik plačilnih storitev (banka) dolžan plačniku nemudoma oziroma najpozneje do konca naslednjega delovnega dne, povrniti znesek te transakcije. V digitalnem forenzičnem laboratoriju opazimo, da se banke pogosto sklicujejo na izjemo, predpisano v 137. členu ZPlaSSIED. Ta člen določa odstopanje od obveznosti vračila in prenos celotnega bremena izgube na uporabnika, če je do izvršitve neodobrene plačilne transakcije prišlo zaradi uporabnikovega naklepa ali zato, ker je uporabnik iz hude malomarnosti (angl. »gross negligence«) kršil svoje obveznosti glede varovanja osebnih varnostnih elementov, določene v 132. členu istega zakona.

5.2. Pravni standard hude malomarnosti in splošni pogoji bank

Pojem »huda malomarnost« v ZPlaSSIED ni izrecno opredeljen, temveč gre za t. i. nedoločen pravni pojem ali pravni standard, katerega vsebino morajo v vsakem posamičnem primeru napolniti sodišča na podlagi konkretnih okoliščin.

V sodni praksi velja, da je huda malomarnost podana takrat, ko oseba ravna skrajno nepazljivo in znatno odstopa od skrbnosti, ki se v posameznem primeru lahko pričakuje od povprečno skrbnega človeka, kar nakazuje na zavestno zanemarjanje običajnih varnostnih ukrepov.

Da bi zmanjšale svojo izpostavljenost tveganjem, banke ta nedoločen pravni pojem pogosto enostransko opredeljujejo v svojih splošnih pogojih poslovanja (SPP). V teh pogojih so kot huda malomarnost rutinsko naštetja dejanja, kot so:

- Posredovanje uporabniškega imena, gesla ali potrditvene kode SMS kateri koli tretji osebi ali njihov vnos na nepreverjeno spletno stran.
- Neuporaba posodobljenih licenčnih protivirusnih programov na napravi.
- Omogočanje oddaljenega dostopa do naprave (z uporabo programov RAT, kot je AnyDesk) na zahtevo neznanih oseb.

Zveza potrošnikov Slovenije (ZPS) in Evropska potrošniška organizacija (BEUC) ostro opozarjata na nevzdržnost prakse, pri kateri si banke prisvajajo pravico, da same določajo meje hude malomarnosti. Po prepričanju teh organizacij banke s samodejnim pripisovanjem hude malomarnosti ob uspešnem ribarjenju izkrivljajo bistvo direktive PSD2 in prelagajo celotno odgovornost (in s tem tudi tveganje zaradi lastnih tehnoloških vrzeli v avtentikaciji) izključno na potrošnika. BEUC poudarja, da gola dejstva (npr. da so bili vneseni vsi varnostni elementi) ne bi smela veljati za neizpodbiten dokaz hude malomarnosti in da mora dokazno breme nedvoumno ostati na strani banke. [14]

5.3. Slovenska in evropska sodna praksa

V slovenski sodni praksi obstajajo pomembne odločitve višjih sodišč in vrhovnega sodišča (npr. VSL I Cpg 160/2025 z dne 17. 9. 2025, VSL Sodba I Cpg 82/2019, VSL Sodba II Cp 1212/2022, VSL Sklep I Cpg 76/2024), ki presojajo vprašanje malomarnosti.

Sodišča so v nekaterih primerih pritrdila bankam in presodila, da popolna odsotnost kritične presoje, na primer posredovanje varnostnih podatkov na očitno lažni spletni strani in zamolčanje te okoliščine banki ob prijavi incidenta, izpolnjuje standard hude malomarnosti. Kljub temu pa je presoja v primerih dobro načrtovanega in izvedenega telefonskega klica ter zlorabe z AnyDeskom bolj zapletena.

Ob upoštevanju psihološkega stresa, ki ga povzroči avtoritativni klic »policista« ali »bančnika«, ki žrtvi grozi z izgubo življenjskih prihrankov, je mnogim potrošnikom zelo težko očitati zavestno zanemarjanje skrbnosti. Takšno ravnanje pogosto ustreza navadni (lahki) malomarnosti, pri kateri pa finančno breme nosi banka. V širšem evropskem prostoru se vzpostavlja strožja doktrina odgovornosti ponudnikov plačilnih storitev. Sodišče Evropske unije (CJEU) v zadevah, kot je C-70/25 (Tukowiecka), ki je še v teku, obravnava vprašanje takojšnjega vračila sredstev. [10] Trenutno pravno razumevanje (ki so ga med drugim potrdila španska, francoska in italijanska vrhovna sodišča) poudarja, da zgolj elektronski zapis v informacijskem sistemu banke, ki potrjuje, da je bila transakcija uspešno avtentificirana z močnimi varnostnimi elementi, še ne zadostuje kot dokaz, da je transakcijo avtoriziral uporabnik, niti ne zadostuje za dokazovanje hude malomarnosti. Sodišča (npr. v Nemčiji) v številnih primerih razsojajo, da to, da se žrtev ujame v past socialnega inženiringa (kljub vnosu podatkov) ne predstavlja hude malomarnosti, zaradi česar morajo banke prevzeti finančno odgovornost za škodo. Prav tako primeri iz Avstrije (npr. OLG Linz) opozarjajo na odgovornost banke. Tudi če je bil potrošnik malomaren, sodišča presojajo soodgovornost banke, kadar njen sistem za spremljanje prevar (angl. »Fraud Transaction Monitoring« - FTM) ni pravočasno prepoznal in blokiral očitno nenavadnih serijskih transakcij takoj po namestitvi nove aplikacije (deljena odgovornost 50:50). [11]

5.4. Izkušnje iz alternativnega reševanja sporov pri Združenju bank Slovenije (IRPS)

Kadar potrošnik in banka ne dosežeta sporazuma v internem pritožbenem postopku glede odgovornosti za spletno zlorabo, imajo komitenti možnost sprožiti postopek pri izvajalcu izvensodnega reševanja potrošniških sporov (IRPS), ki deluje v okviru Združenja bank Slovenije (ZBS).

Poročila o delovanju IRPS za leti 2023 in 2024 kažejo, da so spori, ki izvirajo iz spletnih in mobilnih prevar (ribarjenje, investicijske prevare z zlorabo oddaljenega dostopa), med najpogostejšimi zadevami. Leta 2023 je IRPS obravnaval in zaključil 130 zadev. Leta 2024 je IRPS posebej zabeležil porast novih tipov zlorab, pri katerih so uporabniki s prenosom zlonamerne kode ali aplikacij omogočili goljufom oddaljen dostop do naprav in s tem do mobilne banke [12][13]. ZBS na svojem portalu »Varni na internetu« in v svojih smernicah redno opozarja na te grožnje in komitentom predpisuje obsežen seznam deloma tudi zahtevnih varnostnih priporočil.

Čeprav je postopek za potrošnika brezplačen in naj bi omogočal hitro rešitev spora, je njegova ključna pomanjkljivost v dejstvu, da so izdana mnenja posrednika v sporu za banke nezavezujoča.

Tudi v primerih, ko posrednik IRPS po pregledu dejstev ugotovi, da potrošnik, ki je bil žrtev naprednega ribarjenja, ni ravnal s hudo malomarnostjo in izda mnenje v njegovo korist, se številne banke odločijo, da tega mnenja preprosto ne upoštevajo in odklonijo povračilo sredstev. V javnosti je odmeval primer potrošnika, ki je v nočnem

času kupoval prek spleta (Amazon) in je bil tarča zlorabe. Iz njegovega računa je bilo v Estonijo prenakazanih skoraj 8.000 EUR. Potrošnik je imel ustrezno zaščito računalnika. Čeprav mu je poravnalni svet ZBS pritrdil in podal mnenje v njegovo korist, je banka povračilo vztrajno zavračala, kar je potrošnika prisililo k vložitvi drage in tvegane tožbe na sodišču. Takšna praksa nakazuje, da alternativno reševanje sporov brez zavezujočih odločitev v trenutnem pravnem vakuumu razlaga hude malomarnosti ne zagotavlja ustreznega varstva potrošnikov pred naraščajočimi kibernetскими tveganji. Kljub vsemu pa podatek, da so banke v letu 2024 pri IRPS sklenile sporazume ali poravnave v približno 27 odstotkih obravnavanih zadev (37 primerov), kaže na to, da se tudi znotraj bančne industrije postopoma krepi zavedanje o nesmiselnosti prelaganja absolutne odgovornosti na manipulirane žrtve, zlasti ob potencialnih pomanjkljivostih lastnih opozorilnih sistemov.

5.5. Prihajajoča regulativna paradigma: Uredba o plačilnih storitvah (PSR) in PSD3

Zaradi nezadostnosti trenutnega okvira PSD2 v boju proti prevaram in zlorabam socialnega inženiringa je Evropska komisija pripravila predloga tretje direktive o plačilnih storitvah (PSD3) in uredbe o plačilnih storitvah (PSR). Ker bo PSR sprejeta v obliki uredbe, se bo neposredno in enotno uporabljala v vseh državah članicah, s čimer bo odpravljeno trenutno razdrobljeno nacionalno razumevanje varnosti in odgovornosti. [15][16]

Nova zakonodaja prinaša prelomne premike:

1. PSR uvaja jasnejšo odgovornost bank za primere, ko goljuf, ki se izdaja za uslužbenca banke, zavede potrošnika. Tudi če je uporabnik transakcijo tehnično avtoriziral (pod vplivom telefonskega klica in oddaljenega dostopa), mora banka v primeru pravočasne prijave incidenta ter podane prijave policiji uporabniku povrniti celoten znesek izgube. S tem se odgovornost za sistemsko preprečevanje prevar izrecno prenaša nazaj na ponudnike infrastrukture.
2. Banke bodo še vedno imele možnost zavrniti povračilo v primerih potrošnikove »hude malomarnosti«, vendar bo prag dokazovanja znatno višji. Samo dejstvo, da je potrošnik pod vplivom naprednega socialnega inženiringa nasedel prevari, zakonsko ne bo več obravnavano kot huda malomarnost. Banke bodo morale izvesti temeljito preiskavo in predložiti neizpodbitne dokaze o grobem kršenju pravil.
3. PSR prepozna dejstvo, da se socialni inženiring in deljenje zlonamernih povezav najpogosteje ne zgodita znotraj same bančne aplikacije, temveč na telekomunikacijskih omrežjih, družbenih omrežjih (Facebook, TikTok) in iskalnikih. Nova ureditev omogoča bankam, da pod določenimi pogoji regresirajo (zahtevajo povračilo) od velikih spletnih in telekomunikacijskih družb, če so te z neukrepanjem po prijavi ali z neustreznim nadzorom (npr. omogočanjem lažnega prikazovanja klicnih števil) dopustila izvedbo prevare.
4. Banke bodo morale obvezno uvesti in nadgraditi mehanizme za spremljanje tveganj, predvsem pa bodo upravičene in zakonsko obvezane blokirati in zadržati zelo tvegana nakazila v realnem času, za kar bodo lahko prosto izmenjevale indikatorje zlorab z drugimi ponudniki (angl. »shared fraud data«).

6 Sklep

Sodobne prevare z uporabo aplikacij za oddaljeni dostop in naprednega socialnega inženiringa izpodbijajo klasične predpostavke kibernetiske varnosti v elektronskem bančništvu. Dokler bodo finančne institucije zahteve po dveh avtentikacijskih faktorjih združevale na eni sami končni napravi z uporabo generičnih potisnih sporočil in tehnologij SMS, bo oddaljena zloraba te naprave uspešno zaobšla celoten varovalni obroč. Identificirana tehnološka vrzel nakazuje, da so v pripravi izboljšave v obliki standarda W3C Secure Payment Confirmation (SPC), ki temelji na biometričnem in kriptografskem potrjevanju na ravni strojne enklave, pri čemer na zaslonu operacijskega sistema nespremenljivo in nezamenljivo prikaže dejanski znesek in prejemnika nakazila. Izgovori za vztrajanje pri zastareli avtentikaciji ob pojavu takšnih naprednih standardov niso več strokovno vzdržni. Rutinsko prelaganje celotne finančne odgovornosti na uporabnike pod izgovorom hude malomarnosti samo zato, ker so

pod hudim psihološkim pritiskom »lažnega uslužbenca« posredovali podatke ali namestili legitimno programsko opremo, lahko predstavlja izkrivljanje stanja. Forenzična analiza in izkušnje IRPS ter tuja sodišča kažejo, da zgolj zavajanje potrošnika v napredni prevari ne sme dosegati praga hude malomarnosti, dokazno breme pa mora ostati na banki. Nova pravila Uredbe o plačilnih storitvah (PSR) bodo to usmeritev uzakonila z uvedbo obveznega povračila sredstev pri poosebljanju bank. Izključno tehnično preverjanje identitete ni dovolj. Varnostni sistemi morajo preverjati namen in kontekst transakcije. Uvedba storitve Verification of Payee (VoP) bo z obveznim opozarjanjem na neujemanje med imenom in številko IBAN prejemnika predstavljala pomemben dodatni element kognitivnega trenja, ki ga napadalci ob uporabi oddaljenega dostopa ne bodo imeli možnosti zlahka prikriti s prepričljivo zgodbo. Varovanje sredstev komitentov se premika od preverjanja »ali ste to res vi« k preverjanju »ali zares veste in razumete, komu in zakaj nakazujete sredstva«. Šele praksa pa bo pokazala, ali se bodo uporabniki elektronskega bančništva pustili pretentati tako, da bodo na podlagi navodil napadalca sami potrjevali sumljive transakcije. Za slabo presojo pa trenutno nimamo tehnične rešitve.

Literatura

- [1] European Banking Authority, European Central Bank: 2025 Report on Payment Fraud, EBA/REP/2025/40, december 2025, <https://www.eba.europa.eu/sites/default/files/2025-12/1709846a-84d9-47cf-86a0-b155efb34d66/EBA%20and%20ECB%20Report%20on%20Payment%20Fraud.pdf>, obiskano 3. 7. 2026.
- [2] SI-CERT: Napredni phishing napadi na spletne banke podjetij z elementom telefonskega klica, 29. 10. 2025, <https://www.cert.si/napredni-phishing-napadi-na-spletne-banke-podjetij-z-elementom-telefonskega-klica/>, obiskano 3. 7. 2026.
- [3] Direktiva (EU) 2015/2366 z dne 25. novembra 2015 o plačilnih storitvah na notranjem trgu, spremembah direktiv 2002/65/ES, 2009/110/ES in 2013/36/EU ter Uredbe (EU) št. 1093/2010 in razveljavitvi Direktive 2007/64/ES, Uradni list Evropske unije, L 337, 23. 12. 2015, str. 35–127, <https://eur-lex.europa.eu/eli/dir/2015/2366/oj>, obiskano 5. 7. 2026.
- [4] Delegirana uredba Komisije (EU) 2018/389 z dne 27. novembra 2017 o dopolnitvi Direktive (EU) 2015/2366 glede regulativnih tehničnih standardov za močno avtentikacijo strank ter skupne in varne odprte standarde komunikacije, Uradni list Evropske unije, L 69, 13. 3. 2018, str. 23–43, https://eur-lex.europa.eu/eli/reg_del/2018/389/oj, obiskano 5. 7. 2026.
- [5] <https://www.zbs-giz.si/oblike-prevar-njihovi-znaki-in-opisi-ter-priporocljiva-ravnanja-bank>, Združenje bank Slovenije: Oblike prevar, njihovi znaki in opisi, obiskano 23. 7. 2026.
- [6] W3C: Web Authentication: An API for Accessing Public Key Credentials – Level 2, W3C Recommendation, 8. 4. 2021, <https://www.w3.org/TR/webauthn-2/>, obiskano 7. 7. 2026.
- [7] W3C: Secure Payment Confirmation, Candidate Recommendation Draft, 2. 7. 2026, <https://www.w3.org/TR/secure-payment-confirmation/>, obiskano 3. 8. 2026.
- [8] Uredba (EU) 2024/886 z dne 13. marca 2024 o spremembi uredb (EU) št. 260/2012 in (EU) 2021/1230 ter direktiv 98/26/ES in (EU) 2015/2366 glede takojšnjih kreditnih prenosov v eurih, Uradni list Evropske unije, L 2024/886, 19. 3. 2024, <https://eur-lex.europa.eu/eli/reg/2024/886/oj>, obiskano 3. 8. 2026.
- [9] Zakon o plačilnih storitvah, storitvah izdajanja elektronskega denarja in plačilnih sistemih – ZPlaSSIED, Uradni list Republike Slovenije, št. 7/18, 9/18 – popr. in 102/20, <https://pisrs.si/pregledPredpisa?id=ZAKO7574>, obiskano 9. 7. 2026.
- [10] RANTOS Athanasios: Sklepni predlogi generalnega pravobranilca v zadevi C-70/25, Tukowiecka, z dne 5. 3. 2026, ECLI:EU:C:2026:153, Sodišče Evropske unije, <https://infocuria.curia.europa.eu/tabs/jurisprudence?publishedId=C-70%2F25>, obiskano 3. 7. 2026.
- [11] OBERLANDESGERICHT LINZ: Odločba 1 R 45/25f z dne 22. 5. 2025, Rechtsinformationssystem des Bundes – RIS, <https://www.ris.bka.gv.at/>, obiskano 3. 7. 2026.
- [12] Združenje bank Slovenije: Poročilo o delovanju izvajalca izvensodnega reševanja potrošniških sporov pri Združenju bank Slovenije za leto 2023, 2024, <https://www.zbs-giz.si/irps/>, obiskano 5. 7. 2026.
- [13] Združenje bank Slovenije: Poročilo o delovanju izvajalca izvensodnega reševanja potrošniških sporov pri Združenju bank Slovenije za leto 2024, 2025, <https://www.zbs-giz.si/irps/>, obiskano 5. 7. 2026.

- [14] The European Consumer Organisation: Payment Services: BEUC's Recommendations on the Payment Services Regulation and the Payment Services Directive 3, BEUC-X-2023-123, 29. 9. 2023, https://www.beuc.eu/sites/default/files/publications/BEUC-X-2023-123_Payments_Services_Recommendations%20on%20the%20PSR-PSD3.pdf, obiskano 9. 7. 2026.
- [15] Predlog direktive Evropskega parlamenta in Sveta o plačilnih storitvah in storitvah elektronskega denarja na notranjem trgu, spremembi Direktive 98/26/ES ter razveljavitvi direktiv (EU) 2015/2366 in 2009/110/ES, COM(2023) 366 final, 2023/0209(COD), 28. 6. 2023, <https://eur-lex.europa.eu/legal-content/SL/TXT/?uri=CELEX:52023PC0366>, obiskano 10. 7. 2026.
- [16] Proposal for a Directive of the European Parliament and of the Council on payment services in the internal market, amending Directive 98/26/EC and repealing Directives (EU) 2015/2366 and 2009/110/EC – Confirmation of the final compromise text with a view to agreement, dokument ST 8222/26, 17. 4. 2026, <https://data.consilium.europa.eu/doc/document/ST-8222-2026-INIT/en/pdf>, obiskano 16. 7. 2026.

Izkušnje z uporabo lokalnih modelov LLM v sistemu COBISS

Rene Svenšek, Andrej Krajnc, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija
rene.svenssek@izum.si, andrej.krajnc@izum.si, bojan.stok@izum.si

V preteklih letih smo v sistemu COBISS na podlagi velikih jezikovnih modelov in pristopa RAG razvili pametnega asistenta. Začetna implementacija je temeljila na uporabi zunanjih storitev (OpenAI), nato pa smo z uvedbo lokalnih modelov in knjižnice LangChain4j poenostavili razvoj in izboljšali razširljivost rešitev. Pozitivne izkušnje so potrdile našo strateško usmeritev k uporabi lokalno nameščenih odprtokodnih modelov, kar omogoča večji nadzor nad podatki, večjo zasebnost in nižje stroške. Umetno inteligenco smo razširili tudi na druga področja sistema COBISS. V aplikaciji COBISS Lib smo nadgradili obstoječe asistente in dodatno razvili več novih rešitev. Mikrostoritev CAT-AI omogoča avtomatizirano ekstrakcijo bibliografskih metapodatkov iz dokumentov in slik. Z umetno inteligenco podprta pomoč uporabnikom (angl. *help desk*) omogoča semantično iskanje po interni bazi znanja. Implementirali smo iskanje v naravnem jeziku, ki uporabniške poizvedbe pretvori v strukturirane iskalne zahteve za iskanje po bibliografskih oziroma raziskovalnih virih. Rešitve temeljijo na javanskem ekosistemu (Quarkus, LangChain4j) in arhitekturi mikrostoritev. V prispevku so predstavljene praktične izkušnje pri uvajanju lokalnih modelov in integraciji umetne inteligence v sistem COBISS.

Ključne besede:

COBISS

Quarkus

LangChain4j

lokalni modeli

veliki jezikovni modeli

1 Uvod

Umetna inteligenca v zadnjih letih vse izraziteje prodira v različna področja družbe in postaja pomemben gradnik sodobnih informacijskih sistemov. Posebej velik vpliv imajo veliki jezikovni modeli, ki omogočajo razumevanje in tvorjenje naravnega jezika, odgovarjanje na vprašanja, semantično iskanje informacij in avtomatizacijo številnih opravil, ki so bila doslej v veliki meri odvisna od človeškega dela. Zaradi teh zmožnosti se umetna inteligenca vse pogosteje uporablja tudi za izboljšanje uporabniške izkušnje in optimizacijo poslovnih procesov.

Tudi knjižnično okolje ponuja številne primere uporabe, pri katerih lahko umetna inteligenca pomembno prispeva k učinkovitejšemu delu in lažjemu dostopu do informacij. V sistemu COBISS se je njen razvoj začel z uporabo zunanjih storitev velikih jezikovnih modelov, ki so omogočile hitro preverjanje uporabnosti tovrstnih tehnologij v praksi. Nadaljnji razvoj odprtokodnih modelov in lokalne infrastrukture je omogočil prehod na arhitekturo, ki zagotavlja večji nadzor nad podatki, boljšo razširljivost in ponovno uporabo skupnih komponent pri razvoju različnih AI-storitev.

2 Od zunanjih storitev do lokalnih jezikovnih modelov

Razvoj rešitev umetne inteligence v sistemu COBISS je potekal postopno, skladno z razvojem velikih jezikovnih modelov in spremljajočih tehnologij. Prve implementacije so temeljile na uporabi zunanjih storitev, saj so omogočale hiter razvoj in preverjanje uporabnosti umetne inteligence v knjižničnem okolju [1]. Z nadaljnjim razvojem odprtokodnih modelov in lokalne infrastrukture se je pokazala možnost prehoda na arhitekturo, ki omogoča večji nadzor nad podatki, boljšo razširljivost in učinkovitejšo integracijo umetne inteligence v različne storitve sistema COBISS [2].

2.1. Prva generacija AI-rešitev

Prve rešitve umetne inteligence v sistemu COBISS so temeljile na velikih jezikovnih modelih, dostopnih prek storitve OpenAI. Takšen pristop je omogočil hitro preverjanje uporabnosti velikih jezikovnih modelov pri podpori knjižničnim storitvam in razvoj prve generacije pametnega asistenta v aplikaciji COBISS Lib. Asistent je uporabnikom pomagal pri odgovarjanju na vprašanja o uporabi sistema in pri hitrejšem iskanju informacij v obstoječi dokumentaciji.

Ker veliki jezikovni modeli sami po sebi nimajo dostopa do internih podatkov sistema COBISS, je bila rešitev zasnovana na pristopu Retrieval-Augmented Generation (RAG). Uporabniška poizvedba se najprej pretvori v vektorsko predstavitev, nato pa se v vektorski bazi poiščejo vsebinsko najbolj podobni dokumenti. Pridobljeni izseki se vključijo v poziv (angl. *prompt*) jezikovnemu modelu, ki pripravi odgovor na podlagi posredovanega konteksta. Takšen pristop zmanjšuje pojav halucinacij in omogoča uporabo ažurnega domenskega znanja brez dodatnega učenja modela.

Začetna implementacija je pokazala, da je kombinacija velikih jezikovnih modelov in pristopa RAG primerna za različne primere uporabe v sistemu COBISS. Pri uporabi zunanjih storitev pa so se hkrati pokazale tudi omejitve, povezane z odvisnostjo od zunanjega ponudnika, stroški uporabe, zakasnitvami pri komunikaciji z oddaljenimi storitvami ter vprašanji varovanja podatkov in zasebnosti. Čeprav se v oblak niso pošiljali občutljivi podatki, je bila možnost njihove obdelave zunaj lastne infrastrukture pomemben dejavnik pri načrtovanju nadaljnjega razvoja.

2.2. Evolucija k lokalnim modelom

Z razvojem odprtokodnih velikih jezikovnih modelov in zmogljivejše strojne opreme je postala izvedljiva tudi uporaba lokalno nameščenih modelov. Zato smo postopno prešli na arhitekturo, ki omogoča uporabo lokalnih modelov kot privzete izbire. S tem smo pridobili popoln nadzor nad izvajanjem modelov, zmanjšali stroške uporabe in poenostavili vključevanje umetne inteligence v druge storitve sistema COBISS. Prehod ni pomenil zgolj

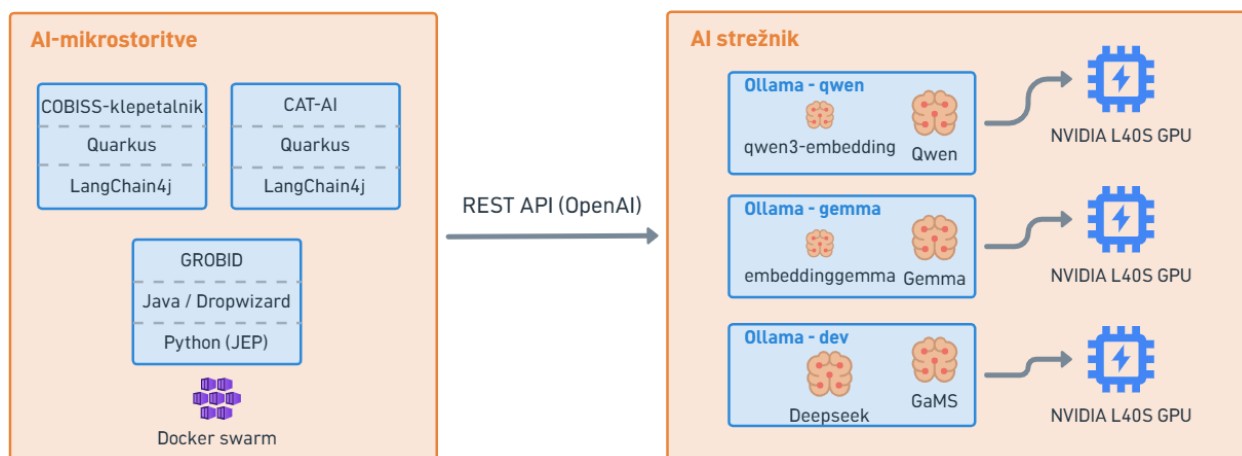
zamenjave ponudnika jezikovnega modela, temveč tudi priložnost za poenotenje razvoja AI-rešitev, saj so skupni gradniki, kot so RAG, vektorske baze in enotni programski vmesniki, omogočili ponovno uporabo komponent pri razvoju različnih aplikacij. Posledično je bilo mogoče umetno inteligenco razširiti tudi na nova področja, kot so katalogizacija, pomoč uporabnikom in iskanje v naravnem jeziku.

Kot kažejo izkušnje pri razvoju sistema COBISS, so sodobni odprtokodni modeli za številne domensko omejene naloge že dovolj zmogljivi, da predstavljajo alternativo komercialnim storitvam. Zunanji modeli praviloma še vedno zagotavljajo višjo kakovost odgovorov in hitrejši dostop do najnovejših modelov, lokalni modeli pa omogočajo večji nadzor nad podatki, bolj predvidljive stroške in lažjo integracijo v interno okolje. Prav zato je bila odločitev za lokalno izvajanje modelov predvsem odgovor na potrebe po večji neodvisnosti, boljši obvladljivosti in tesnejši integraciji umetne inteligence v obstoječi ekosistem COBISS.

Prehod na lokalne modele je zahteval tudi prilagoditev programske arhitekture. V nadaljevanju je predstavljena arhitektura AI-rešitev, ki omogoča uporabo različnih jezikovnih modelov in ponovno uporabo skupnih komponent pri razvoju storitev v sistemu COBISS.

3 Arhitektura AI-rešitev v sistemu COBISS

Po prehodu na lokalne velike jezikovne modele je bilo treba zagotoviti arhitekturo, ki omogoča njihovo uporabo v različnih delih sistema COBISS. Namesto razvoja ločenih, izoliranih rešitev za posamezne primere uporabe je bil zasnovan poenoten razvojni pristop. Ta temelji na mikrororitveni arhitekturi, enotnem razvojnem ogrodju in skupni infrastrukturi za izvajanje jezikovnih modelov. Takšna zasnova omogoča enostavnejši razvoj novih AI-rešitev, njihovo neodvisno nameščanje in ponovno uporabo preverjenih razvojnih vzorcev.



Slika 1: Infrastruktura AI-rešitev v sistemu COBISS

3.1. Infrastruktura za izvajanje modelov

Lokalni veliki jezikovni modeli se izvajajo na namenskem strežniku, opremljenem s tremi grafičnimi procesorji NVIDIA L40S, od katerih ima vsak 48 GB grafičnega pomnilnika (skupaj 144 GB VRAM-a). Takšna konfiguracija omogoča hkratno izvajanje več različnih modelov in zagotavlja zadostno zmogljivost za podporo zahtevnejšim storitvam umetne inteligence.

Za strežbo in upravljanje modelov se uporablja platforma Ollama; ta deluje kot vmesna plast (angl. *inference engine*) in zagotavlja enoten vmesnik REST API, ki je popolnoma združljiv s standardom OpenAI (angl. *OpenAI-compatible API*). Ta združljivost je ključna, saj omogoča preprosto integracijo z obstoječimi orodji, ogrodji (npr. LangChain, LlamaIndex) in knjižnicami brez potrebe po prilagajanju izvorne kode aplikacij — zamenja se le ciljni naslov (angl. *base URL*) strežnika.

Ollama v tej arhitekturi skrbi za:

- Dinamično upravljanje pomnilnika. Samodejno nalaga in prazni modele iz grafičnega pomnilnika glede na trenutne zahteve AI-storitev.
- Porazdelitev bremena (angl. load balancing). Optimizira izrabo vseh treh razpoložljivih grafičnih procesorjev, kar omogoča vzporedno obdelavo več zahtev hkrati.
- Fleksibilnost storitev. Na strežniku je lahko hkrati nameščenih več različnih modelov (npr. manjši modeli za hitro generiranje besedila in večji za kompleksnejše analize), posamezna AI-storitev pa glede na svoje specifične zahteve dinamično izbere najprimernejšega.

Takšno centralizirano izvajanje modelov bistveno poenostavlja njihovo upravljanje, posodabljanje, dodajanje novih različic in natančno spremljanje delovanja celotnega sistema.

3.2. Mikrostoritvena arhitektura AI-rešitev

Vse AI-rešitve v sistemu COBISS so implementirane kot samostojne mikrostoritve. Takšen modularni pristop omogoča neodvisen razvoj, testiranje in nameščanje posameznih komponent, hkrati pa bistveno poenostavlja njihovo integracijo v obstoječi ekosistem COBISS.

Za razvoj mikrostoritev se uporablja napredno ogrodje Quarkus, ki je optimizirano za okolji Kubernetes in Java v oblaku [3]. Izbira tega ogrodja prinaša več ključnih prednosti. To so:

- Učinkovitost in odzivnost. Quarkus omogoča izdelavo lahkih aplikacij s hitrim časom zagona (angl. fast startup time) in nizko porabo pomnilnika, kar je ključno pri izvajanju v kontejnerskih okoljih.
- Možnost nativnega prevajanja. Podpora za GraalVM omogoča prevajanje aplikacij neposredno v nativno strojno kodo, s čimer se še dodatno zmanjša pomnilniški odtis (angl. memory footprint) posamezne mikrostoritve.
- Ohlapna povezanost (angl. loose coupling). Posamezne AI-storitve z drugimi komponentami sistema komunicirajo prek standardiziranih REST-vmesnikov. Zaradi tega so storitve med seboj neodvisne, kar zmanjšuje medsebojne odvisnosti in preprečuje, da bi morebitna napaka v eni storitvi vplivala na delovanje preostalega sistema.

Takšna arhitekturna zasnova zagotavlja visoko stopnjo razširljivosti (angl. *scalability*), saj je mogoče najbolj obremenjene AI-storitve poljubno in neodvisno skalirati glede na trenutne potrebe uporabnikov sistema COBISS.

3.3. Integracija velikih jezikovnih modelov

Za integracijo velikih jezikovnih modelov v razvojno okolje Java se uporablja knjižnica LangChain4j, ki predstavlja enoten abstrakcijski sloj za komunikacijo z modeli [4]. Posamezna AI-mikrostoritev prek ogrodja LangChain4j definira parametre za izbrani model – naj gre za klasični jezikovni model (LLM), model vlaganja (angl. *embedding model*) ali multimodalni veliki jezikovni model (angl. Multimodal-Large-Language Model – MLLM) za analizo multimodalnih podatkov. Vsi zahtevki se nato prek REST-vmesnika posredujejo centraliziranemu strežniku Ollama.

Takšen pristop standardizira razvoj AI-rešitev, saj razvijalcem omogoča uporabo enakega programskega modela ne glede na tip in različico izbranega modela v ozadju. Zamenjava ali posodobitev uporabljenega modela praviloma zahteva le spremembo konfiguracijskih nastavitev aplikacije, medtem ko jedro poslovne logike posamezne storitve ostane nespremenjeno.

Poleg osnovne komunikacije z modeli lahko posamezne storitve glede na specifične potrebe vključujejo tudi kompleksnejše arhitekturne gradnike. Mednje spadajo sistemi za iskanje s podporo znanja: RAG, ustvarjanje besedilnih vektorjev in integracija z vektorskimi bazami podatkov. Povezava med ogrodjema Quarkus in LangChain4j tako tvori tehnično osnovo za uvajanje funkcionalnosti umetne inteligence v sistem COBISS.

Opisana arhitektura predstavlja skupno osnovo za razvoj vseh AI-rešitev v sistemu COBISS. Poenoten razvojni pristop, centralizirano izvajanje jezikovnih modelov in mikrostoritvena zasnova omogočajo hitre razvojne iteracije in enostavno vključevanje novih funkcionalnosti.

4 Implementirane rešitve in primeri uporabe

Na podlagi predstavljene arhitekture je bilo v sistemu COBISS razvitih več rešitev, ki uporabljajo velike jezikovne modele za podporo različnim poslovnim procesom. Čeprav so te rešitve namenjene različnim profilom uporabnikov in rešujejo posebne težave, vse temeljijo na enotnem, poenotenem razvojnem pristopu, opisanem v prejšnjem poglavju. V nadaljevanju so predstavljene ključne implementirane rešitve, njihove specifične arhitekturne značilnosti in konkretni primeri uporabe v praksi.

4.1. COBISS-klepetalnik za knjižničarje in uporabnike knjižnic

Prva rešitev, razvita na predstavljeni arhitekturi, je COBISS-klepetalnik. Gre za na umetni inteligenci temelječo storitev, ki nadomešča klasične klepetalne kanale, v katerih odgovore podajajo operaterji.

Sprva je bil uveden v aplikaciji COBISS Lib kot pametni asistent za knjižničarje, v katerem podpira vsakodnevno delo z zagotavljanjem hitrih odgovorov na vprašanja o funkcionalnostih sistema in uporabi programske opreme.

Na spletnem portalu COBISS Plus bo klepetalnik kmalu na voljo tudi slovenskim uporabnikom knjižnic. V tem primeru je namenjen predvsem pridobivanju informacij o storitvah sistema COBISS in uporabi kataloga ter odgovarjanju na splošna vprašanja uporabnikov.

Obe rešitvi temeljita na skupni zaledni mikrostoritvi. Ključna razlika med njima je v uporabniškem vmesniku in v naboru dokumentov, ki sestavljajo bazo znanja, na podlagi katere se ustvarjajo odgovori. Medtem ko klepetalnik za knjižničarje temelji predvsem na priročnikih in strokovnih navodilih, klepetalnik za uporabnike knjižnic izhaja predvsem iz zbirk najpogostejše zastavljenih vprašanj.

Arhitektura je zasnovana z vidika ponovne uporabe in razširljivosti. Namesto razvoja ločenih klepetalnikov za posamezne aplikacije je implementiran enoten zaledni sistem, ki omogoča:

- indeksiranje različnih zbirk dokumentacije,
- izbiro ustreznega vira znanja glede na kontekst uporabe,
- generiranje odgovorov, prilagojenih posamezni skupini uporabnikov.

Takšen pristop omogoča uvajanje klepetalnika v druge dele sistema COBISS ob minimalnih dodatnih razvojnih stroških. Dodajanje novih klepetalnikov v ekosistem COBISS ne zahteva posegov v zaledno programsko kodo, temveč zgolj pripravo in indeksiranje ustrezne dokumentacije.

Vse zaledne izboljšave – bodisi fino nastavljanje iskalnih algoritmov, optimizacija priprave konteksta ali nadgradnje komunikacijskega sloja z jezikovnim modelom – so tako samodejno na voljo vsem aplikacijam, ki uporabljajo skupno zaledno storitev.

4.1.1. Postopek indeksiranja in priprava baze znanja za COBISS-klepetalnik

Implementacija RAG temelji na uporabi lokalnega namenskega modela Qwen3-Embedding, ki se uporablja tako pri pripravi baze znanja kot tudi pri obdelavi uporabniških poizvedb.

Postopek gradnje baze znanja poteka v naslednjih korakih:

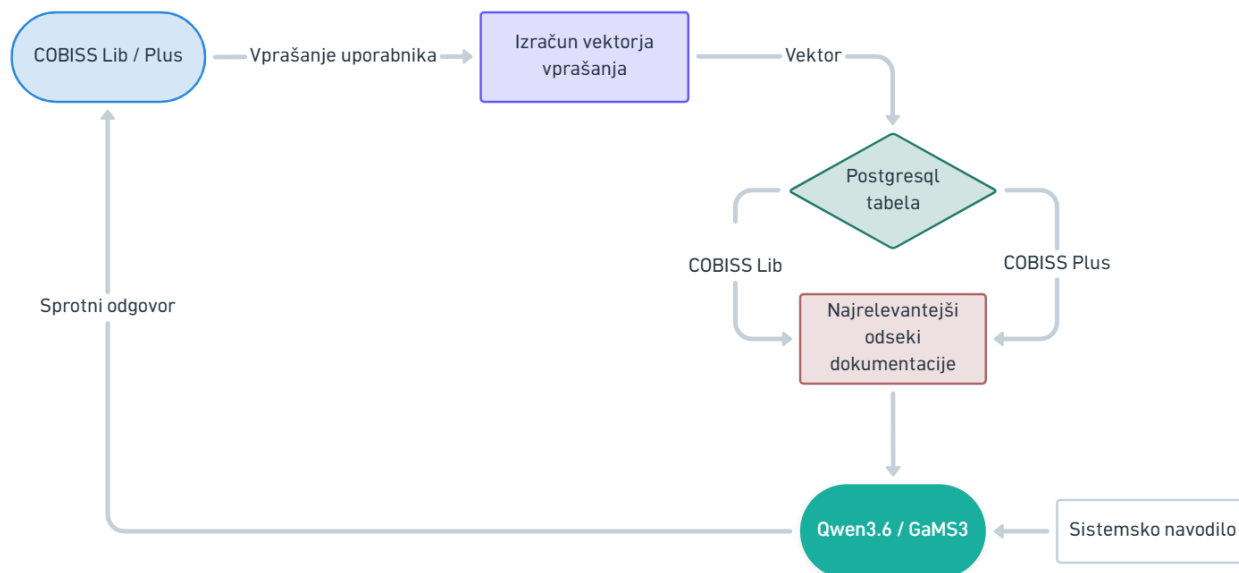
1. Razrez dokumentov (angl. *chunking*). Izvorna dokumentacija posamezne storitve se razdeli na manjše, logično zaokrožene vsebinske odseke, s čimer se zmanjša šum pri kasnejšem iskanju.
2. Vektorizacija z modelom Qwen3-Embedding. Za vsak besedilni odsek se na podlagi modela vlaganja (angl. *embedding model*) izračuna njegova vektorska predstavitev, ki matematično opiše semantični pomen besedila.
3. Vnos v namenske tabele. Vektorji se skupaj z izvirnim besedilom in pripadajočimi metapodatki shranijo v ločeno podatkovno tabelo v PostgreSQL, ki je dodeljena posamezni AI-storitvi.

4.1.2. Obdelava poizvedb v COBISS-klepetalniku

Ko zaledna mikrostoritev prejme uporabniško vprašanje, se celoten postopek obdelave izvede dinamično glede na izvor poizvedbe, in sicer kot sledi:

1. Pretvorba v vektor. Uporabniško vprašanje se najprej pretvori v vektorsko predstavitev z enakim modelom Qwen3-Embedding.
2. Semantično iskanje in usmerjanje. Sistem glede na izvirno aplikacijo (COBISS Lib ali portal COBISS Plus) izbere pripadajočo tabelo v bazi podatkov in iz nje izlušči semantično najpomembnejše odseke dokumentacije.
3. Kontekstualizacija poziva. Pridobljeni odseki se v obliki konteksta priložijo uporabniškemu vprašanju in sistemskim navodilom (angl. *system prompt*), pri čemer COBISS-klepetalnik za knjižničarje uporablja lokalni model Qwen3.6 (boljša večjezična podpora), COBISS-klepetalnik za slovenske uporabnike knjižnic pa uporablja lokalni model GaMS3 (boljša podpora za slovenski jezik).

Celoten potek obdelave poizvedbe je vizualno prikazan na sliki 2.



Slika 2: Potek obdelave poizvedbe v COBISS-klepetalniku

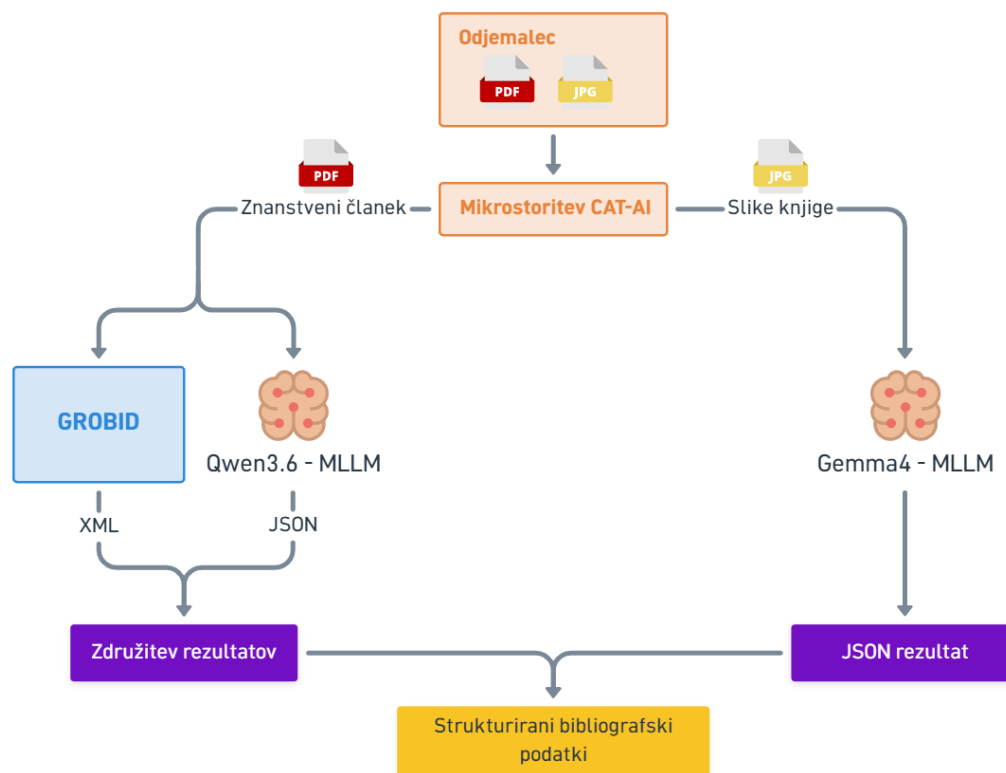
4.2. CAT-AI

CAT-AI je namenska mikrororitvev, razvita za avtomatizirano ekstrakcijo bibliografskih metapodatkov iz različnih vrst gradiva. Njena osrednja naloga je priprava strukturiranih metapodatkov, ki predstavljajo temelj za nadaljnjo gradnjo bibliografskih zapisov, s čimer se zmanjša obseg ročnega vnosa podatkov pri katalogizaciji.

Mikrororitvev je zasnovana modularno, zato lahko glede na vrsto vhodnega dokumenta dinamično izbere najprimernejši cevovod obdelave. Trenutno podpira dva ključna scenarija:

- ekstrakcija metapodatkov iz znanstvenih člankov in
- ekstrakcija metapodatkov iz naslovne in zadnje strani knjige.

Čeprav oba postopka sledita istemu cilju, tj. pripravi kakovostnih bibliografskih metapodatkov, uporabljata različne tehnološke pristope, prilagojene značilnostim posamezne vrste vhodnega gradiva.



Slika 3: Potek obdelave znotraj mikrororitve CAT-AI

Potek obdelave znotraj mikrororitve CAT-AI se začne tako, da po prejemu vhodnega dokumenta sistem najprej določi njegov tip in nato izbere ustrezno procesno vejo. Pri znanstvenih člankih se izvede vzporedna obdelava z orodjem GROBID za ekstrakcijo bibliografskih podatkov in jezikovnim modelom Qwen3.6, ki dodatno analizira vsebino dokumenta. Pri obdelavi naslovne in zadnje strani knjige pa ekstrakcijo metapodatkov neposredno iz slikovnih vhodov izvede vizualni jezikovni model Gemma4.

Ne glede na izbrano procesno pot je rezultat vedno poenoten nabor strukturiranih bibliografskih metapodatkov v standardizirani obliki. Tako pripravljeni podatki so neposredno uporabni v nadaljnjih postopkih katalogizacije in se lahko brez dodatnih prilagoditev vključijo v osrednji katalogizacijski proces sistema COBISS.

4.2.1. Ekstrakcija metapodatkov iz znanstvenih člankov

Pri znanstvenih člankih gre praviloma za dokumente PDF, iz katerih lahko pridobimo besedilno vsebino. Pri obdelavi znanstvenih člankov v besedilni obliki mikrorstitev vzporedno sproži dva neodvisna postopka obdelave. To sta:

- Strukturna analiza (GROBID). Ta postopek temelji na uporabi zunanje storitve GROBID, ki je specializirana za strojno učenje in analizo strukture znanstvenih člankov ter iz njih lušči strukturirane bibliografske informacije.
- Semantična analiza (Qwen3.6). Ta postopek uporablja lokalni veliki jezikovni model Qwen3.6. Modelu se kot kontekst posredujejo uvodne strani dokumenta PDF, saj je na njih praviloma večina ključnih podatkov, kot so naslov članka, avtorji, povzetek in podatki o publikaciji.

Oba postopka se izvajata sočasno, njuni delni rezultati pa se po zaključeni obdelavi v zaledju deterministično združijo v enoten, skladen nabor metapodatkov. Takšen hibridni pristop uspešno združuje prednosti obeh metod: GROBID zagotavlja stabilno ekstrakcijo strogo strukturiranih podatkov iz znanstvenih publikacij, medtem ko lokalni jezikovni model dopolni oziroma preveri informacije, ki jih GROBID zaradi posebnih postavitev strani morda ni prepoznal ali jih ni mogel enoznačno določiti. Kombinacija obeh metod se je v praksi izkazala za znatno zanesljivejšo od samostojne uporabe katerega koli izmed navedenih orodij.

4.2.2. Ekstrakcija metapodatkov iz naslovnice knjig

Naslovna in zadnja stran knjige sta slikovno gradivo, pri katerem nimamo na voljo besedilne vsebine, temveč le grafiko. Pri razvoju komponente za ekstrakcijo metapodatkov iz slikovnega gradiva sta bila preizkušena dva različna pristopa, ki nazorno prikazujeta prednost uvedbe sodobnih vizualnih modelov.

Prvotni pristop je temeljil na lastni storitvi OCR (optična prepoznavna znakov), ki je iz slik najprej izluščila golo besedilo, nato pa je aplikacija s programsko logiko poskušala prepoznati posamezne bibliografske elemente (naslov, avtor, založba). Ta pristop ni dosegal zadovoljive stopnje natančnosti, saj je bila uspešnost močno odvisna od kakovosti fotografije, kompleksnosti postavitve besedila in grafične zasnove naslovnice.

Zaradi omenjenih omejitev je bil uveden sodobnejši pristop, ki v celoti temelji na uporabi multimodalnega jezikovnega modela (angl. Multimodal Large Language Model – MLLM) Gemma4. Mikrorstitev CAT-AI lahko v okviru ene zahteve sočasno obdela več slik iste publikacije (npr. fotografije naslovne strani, hrbtna strani in kolofona). Model Gemma4 nato na podlagi celotnega nabora vizualnih vhodov pripravi strukturiran odgovor v obliki zapisa JSON z vsemi izluščenimi bibliografskimi metapodatki. Ta multimodalni pristop se je izkazal za bistveno uspešnejšega, saj model pri interpretaciji ne bere le golega besedila, temveč upošteva celotno vizualno hierarhijo, velikost in razmerja med posameznimi grafičnimi elementi na slikah, kar mu omogoča natančno razlikovanje med naslovom knjige, podnaslovom in imenom avtorja.

4.3. Iskanje v naravnem jeziku

Tretja implementirana rešitev omogoča iskanje po bibliografskih zapisih z uporabo naravnega jezika. Namenjena je poenostavitvi uporabniške izkušnje, predvsem za tiste uporabnike, ki ne poznajo zapletene sintakse bibliografskih poizvedb ali razpoložljivih iskalnih polj v sistemu COBISS. Namesto ročnega sestavljanja strukturiranih iskalnih izrazov lahko uporabnik svojo zahtevo opiše v prostem, naravnem jeziku, sistem na zaledni ravni pa jo samodejno pretvori v pravilno oblikovano bibliografsko poizvedbo, ki jo nato izvede obstoječi iskalni mehanizem COBISS.

Če želimo poiskati knjige Ivana Cankarja, izdane leta 2000, moramo zdaj v aplikaciji COBISS Plus uporabiti izbirno ali ukazno iskanje.

Slika 4: Izbirno iskanje v aplikaciji COBISS Plus

Slika 5: Ukazno iskanje v aplikaciji COBISS Plus

Poleg tega smo uvedli iskanje v naravnem jeziku, na primer: »Poišči knjige Ivana Cankarja, izdane leta 2000.«

Slika 6: Iskanje v naravnem jeziku

Rešitev temelji na uporabi lokalnega jezikovnega modela, katerega primarna naloga ni ustvarjanje neposrednega besedilnega odgovora uporabniku, temveč napredna semantična analiza uporabnikove zahteve in njena pretvorba v strukturiran iskalni niz (angl. *Text-to-Query*). Med tem procesom model iz uporabnikovega besedila prepozna posamezne bibliografske entitete in attribute ter jih preslika v ustrezna iskalna polja. To so:

- Identifikacija entitet. Prepoznavanje avtorja, naslova, založnika, jezika ali ključnih besed.
- Logično povezovanje. Če uporabnik znotraj ene zahteve poda več pogojev, model natančno ohrani njihove logične odvisnosti in jih poveže z ustreznimi Boolovimi operatorji (AND, OR, NOT).
- Normalizacija vrednosti. Model prepoznane vrednosti standardizira — osebna imena pretvori v osnovno obliko (imenovalnik), prepozna in pravilno zapiše časovne intervale (npr. »izdano leta 2000« pretvori v ustrezen operator enakosti) ter izraze ustrezno razvrsti.

Pomemben del implementacije predstavlja strogo usmerjanje jezikovnega modela prek sistemskih navodil (angl. *system prompt*). Z natančno definiranimi pravili, kontekstualnimi omejitvami in natančnim opisom dovoljenih operacij je model usmerjen tako, da iz uporabniške zahteve izlušči izključno podatke, ki jih je uporabnik dejansko navedel. Modelu je prek sistema poziva izrecno prepovedano kakršno koli sklepanje ali dopolnjevanje manjkajočih informacij, rezultat njegove obdelave pa je izključno čista, izvorna iskalna poizvedba (angl. *native query*) brez dodatnega pojasnjevalnega besedila ali klepetalnih razlag.

Tako pripravljena in strukturirana poizvedba se nato posreduje obstoječemu, uveljavljenemu iskalnemu jedru sistema COBISS. Ta pristop prinaša ključno arhitekturno prednost: umetna inteligenca se uporablja izključno kot vmesnik za razumevanje uporabnikove namere (angl. *intent recognition*), medtem ko iskanje, filtriranje in vračanje zadetkov še vedno izvaja preverjen in zanesljiv iskalni sistem. S tem je zagotovljena 100-odstotna združljivost z

obstoječimi indeksi in poslovno logiko COBISS-a, popolna odsotnost tveganja za halucinacije modela glede obstoja gradiva in hkrati izrazito poenostavljena uporabniška izkušnja.



Slika 7: Potek obdelave poizvedbe v naravnem jeziku

Proces transformacije nazorno ponazarja naslednji scenarij:

1. Uporabnikov vnos: »Poišči knjige Ivana Cankarja, izdane leta 2000.«
2. Obdelava z lokalnim LLM-jem. Model analizira vnos na podlagi pravil v sistemskega pozivu, prepozna avtorja ter časovni pogoj in generira čisto bibliografsko poizvedbo: (cankar ivan)/AX AND PY=2020
3. Izvedba. Nastala poizvedba se v ozadju neposredno izvede nad bibliografsko zbirko, sistem pa uporabniku vrne natančne zapise, ki ustrezajo vsem podanim kriterijem.

5 Pridobljene izkušnje in praktične ugotovitve

Predstavljene implementacije jasno potrjujejo, da izbrana skupna arhitektura omogoča razširljiv razvoj raznolikih AI-storitev za podporo procesom v sistemu COBISS. Kljub specifičnim zahtevam posameznih primerov uporabe se je pri razvoju izkazalo, da si vse rešitve delijo sorodne tehnične izzive. Ti so povezani predvsem z integracijo lokalnih jezikovnih modelov, učinkovitim upravljanjem strojnih virov, inženiringom pozivov (angl. *prompt engineering*) ter zagotavljanjem zanesljivosti in determinističnosti rezultatov v produkcijskem okolju.

5.1. Izbira in kakovost lokalnih modelov

Ena ključnih ugotovitev pri razvoju rešitev je, da izbira ustreznega jezikovnega modela bistveno vpliva na kakovost končnih rezultatov. Med razvojem smo preizkusili več odprtokodnih modelov (družine Qwen, Llama in Mistral, modela Gemma in GaMS ...). Ker so se modeli razlikovali po zmogljivosti in primernosti za različne naloge, smo izbiro vedno prilagodili konkretni funkcionalnosti.

Večji modeli z več parametri na splošno zagotavljajo kakovostnejše odgovore in boljše razumevanje kompleksnih nalog, čeprav velikost sama po sebi ne zadošča za doseganje zadovoljivih rezultatov. Enako pomembna sta se izkazala ustrezno zasnovano sistemsko navodilo in kakovost vhodnega konteksta. Pri tem je bilo treba posebno pozornost nameniti omejitvi kontekstnega okna, saj preseganje največjega dovoljenega števila vhodnih žetonov (angl. *tokens*) onemogoči pravilno obdelavo zahteve.

Pri izbiri modelov so pomembno vlogo igrale njihove specifične lastnosti:

- Večjezičnost. Za večjezične funkcionalnosti, kot je AI-klepetalnik v aplikaciji COBISS Lib, se je družina modelov Qwen izkazala za primernejšo od drugih, saj je zagotavljala kakovostnejše odgovore v različnih jezikih.
- Način razmišljanja (angl. *thinking mode*). Pri zahtevnejših nalogah lahko ta način bistveno izboljša kakovost rezultatov, čeprav podaljša čas generiranja odgovorov.
- Optimizacija virov. Največji razpoložljivi model ni vedno najboljša izbira. Če manjši model dosega primerljivo kakovost, pogosto predstavlja primernejšo rešitev za produkcijsko okolje, in sicer zaradi nižje porabe pomnilnika, hitrejšega izvajanja in večje prepustnosti sistema.

Univerzalnega jezikovnega modela, ki bi bil optimalen za vse primere uporabe, ni. Učinkovita uporaba lokalnih modelov zahteva izbiro glede na zahteve posamezne naloge, pri čemer je poleg zmogljivosti modela treba upoštevati tudi kakovost navodil (angl. *prompt*), pripravo konteksta in razpoložljive strojne vire. Uspešna uvedba lokalnih modelov je rezultat usklajenega delovanja modela, ustrezno zasnovanih navodil in premišljene systemske arhitekture.

5.2. Tehnični izzivi pri razvoju in uvajanju rešitev

Poleg izbire ustreznih modelov so pomemben del razvoja predstavljali tehnični izzivi, povezani z njihovim učinkovitim izvajanjem, stabilnostjo infrastrukture in zagotavljanjem strukturiranih izhodov.

Prvi večji izziv je bil povezan z upravljanjem več modelov na isti strojni infrastrukturi. Privzeto okolje Ollama modele dinamično nalaga v pomnilnik in jih po določenem času neaktivnosti odstrani. Če je bilo zaradi omejene količine grafičnega pomnilnika (VRAM) hkrati uporabljenih več različnih modelov, so se pojavljale situacije, ko je sistem že naloženi model odstranil in ga zamenjal z drugim, ki ga je zahtevala nova poizvedba. To je povzročalo nepredvidljivo obnašanje in občutno daljše odzivne čase pri sočasni uporabi različnih funkcionalnosti.

Za odpravo teh težav smo infrastrukturo prilagodili tako, da posamezne storitve Ollama izvajajo vnaprej določene modele na točno določenih grafičnih procesorjih (GPU), kot je prikazano na sliki 1. S to segmentacijo smo dosegli predvidljivo razporeditev virov, preprečili nepotrebno ponovno nalaganje modelov in izboljšali splošno odzivnost.

Dodatno težavo je predstavljalo generiranje strukturiranih odgovorov v obliki JSON. Kljub ustrezno oblikovanim sistemskim navodilom modeli niso vedno vrnilo veljavne sintakse, zato smo implementirali mehanizem za dodatno validacijo odgovorov. Če sistem zazna neveljaven odgovor, samodejno izvede ponovno poizvedbo z dodatnimi navodili, ki model usmerijo k popravku strukture. Takšen večstopenjski pristop se je izkazal za bistveno zanesljivejšega od zanašanja na enkratno generiranje.

Pri razvoju smo se soočali tudi s hitrimi spremembami v AI-ekosistemu. Posodobitve okolja Ollama in ustreznih knjižnic so občasno spremenile privzeto delovanje sistema ali prepisale obstoječe konfiguracije. Izrazit primer je bila sprememba obnašanja ogrodja Vulkan, zaradi katere se modeli niso več nalagali na vnaprej določeno kartico GPU, temveč so se razporedili čez vse razpoložljive naprave. To je zahtevalo takojšnje prilagajanje konfiguracijskih datotek in uvedbo protokola za ponovno preverjanje celotne infrastrukture po vsaki večji posodobitvi knjižnic.

6 Zaključek

Izkušnje pri razvoju in uvajanju rešitev na podlagi lokalnih velikih jezikovnih modelov v sistemu COBISS potrjujejo, da so odprtokodni modeli dovolj zmogljivi za uporabo v produkcijskih okoljih. Pri tem so se kot ključni dejavniki uspeha izkazali premišljena izbira modelov glede na posamezen primer uporabe, kakovostno načrtovanje sistemskih navodil ter zanesljiva in razširljiva infrastruktura.

Razvoj predstavljenih rešitev je prinesel tudi številne tehnične izzive, povezane z upravljanjem strojnih virov, konfiguracijo izvajalnega okolja in zagotavljanjem zanesljivih izhodov. Njihovo reševanje je omogočilo oblikovanje enotne arhitekture, ki predstavlja trdno osnovo za nadaljnji razvoj in širšo uporabo umetne inteligence v sistemu COBISS.

Nadaljnji razvoj bo usmerjen v posodobitev infrastrukture z uvedbo ogradij vLLM in LiteLLM ter uporabo nove strežniške opreme z zmogljivejšimi grafičnimi procesorji. vLLM je ogrodje, optimizirano za učinkovito izvajanje velikih jezikovnih modelov, ki omogoča hitrejšo generiranje odgovorov in boljšo izrabo pomnilnika grafičnih procesorjev [5]. LiteLLM pa predstavlja enoten vmesnik za povezovanje in upravljanje različnih jezikovnih modelov ter ponudnikov LLM-storitev, kar poenostavlja integracijo, preklapljanje med modeli in njihovo centralizirano upravljanje [6].

S tem pričakujemo učinkovitejše izvajanje modelov, boljšo izrabo strojnih virov in podporo večjim jezikovnim modelom, kar bo omogočilo razvoj novih in zahtevnejših funkcionalnosti. Tako bomo lahko razvijali naprednejše rešitve, ki bodo še učinkoviteje podpirale delo knjižničarjev in uporabnikov sistema COBISS.

Literatura

- [1] AMBROŽIČ, Vojko, KRAJNC, Andrej, ŠTOK, Bojan. Primer razvoja pametnega asistenta. V: PAVLIČ, Luka (ur.), BERANIČ, Tina (ur.), HERIČKO, Marjan (ur.). OTS 2024 : sodobne informacijske tehnologije in storitve : zbornik 27. konference : [Maribor, 4. in 5. september 2024]. 1. izd. Maribor: Univerza v Mariboru, Univerzitetna založba, 2024. Str. 15-24, ilustr. ISBN 978-961-286-893-2. <https://press.um.si/index.php/ump/catalog/book/903/chapter/68>, DOI: 10.18690/um.feri.4.2024.2
- [2] KRAJNC, Andrej, AMBROŽIČ, Vojko, ŠTOK, Bojan. Uporaba javanske knjižnice za velike jezikovne modele. V: PAVLIČ, Luka (ur.), BERANIČ, Tina (ur.), HERIČKO, Marjan (ur.). OTS 2025 : sodobne informacijske tehnologije in storitve : zbornik 28. konference : [Maribor, 3. in 4. september 2025]. 1. izd. Maribor: Univerza v Mariboru, Univerzitetna založba, 2025. Str. 175-186, ilustr. ISBN 978-961-299-036-7. <https://press.um.si/index.php/ump/en/catalog/book/1000/chapter/788>, DOI: 10.18690/um.feri.7.2025
- [3] Quarkus, <https://quarkus.io/>, obiskano 29. 7. 2026
- [4] LangChain4j, <https://github.com/langchain4j/langchain4j/>, obiskano 29. 7. 2026
- [5] vLLM, <https://vllm.ai/>, obiskano 29. 7. 2026
- [6] LiteLLM, <https://www.litellm.ai/>, obiskano 29. 7. 2026

x402: arhitektura in praktična uporaba strojnih plačil z uporabo HTTP 402

Martin Ferenec, Adnan Bratanović, Teo Lah, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
martin.ferenec@um.si, adnan.bratanovic@student.um.si, teo.lah@student.um.si,
muhamed.turkanovic@um.si

Članek predstavi protokol x402, odprt standard za strojno izvajanje plačil na spletu, ki po treh desetletjih neuporabe statusno kodo HTTP 402 prvič uveljavlja kot dejansko delujoč plačilni mehanizem. Opisano je trenutno stanje zaračunavanja dostopa do podatkov ter ovire, kot so odvisnost od posrednikov, visoki stroški mikrotransakcij in odsotnost vgrajenega plačilnega mehanizma, ki so pripeljale do nastanka protokola. Podrobneje je predstavljena arhitektura x402 (odjemalec, strežnik vira, denarnica, plačilne sheme, plačilni posrednik), delovanje in primerjava različic v1 in v2. Sledi primerjava z alternativnim protokolom Machine Payments Protocol (MPP) s priporočilom, kdaj izbrati katerega. Prispevek se konča s predstavitvijo uporabe x402 v referenčnih implementacijah agentov umetne inteligence in v podatkovnih prostorih, na primeru povezovalnika DSX Engine v podatkovnem prostoru DADS (DIH AGRIFOOD Data Space).

Ključne besede:

HTTP 402

spletna plačila

strojna plačila

plačila za AI agente

pametne pogodbe

1 Uvod

Digitalno gospodarstvo v vse večji meri temelji na avtomatizirani izmenjavi podatkov med sistemi. Pri spletnih storitvah se je kot prevladujoč pristop uveljavil arhitekturni slog REST, formaliziran leta 2000 [1]. Kljub tehnični zrelosti tovrstnih spletnih vmesnikov pa področje neposredne strojne monetizacije podatkov dolgo ni imelo odprtega in standardiziranega plačilnega mehanizma. Standard HTTP je sicer že od svojih začetkov predvideval statusno kodo 402 (Payment Required), namenjeno digitalnim plačilom, vendar trenutno veljavna specifikacija to kodo označuje kot rezervirano za prihodnjo uporabo, zaradi česar v praksi ni bila nikoli splošno uveljavljena [2]. Posledično ponudniki podatkov večinoma uporabljajo lastne modele naročnin ali posredniške plačilne sisteme, ki povečujejo stroške, uvajajo odvisnost od centraliziranih ponudnikov ter zmanjšujejo interoperabilnost med sistemi [3] [4].

To vprašanje je v zadnjih letih dobilo novo težo z uveljavljanjem avtonomnih agentov umetne inteligence, ki podatkovne vire in programske vmesnike kličejo brez sprotnega človekovega posredovanja. Postopki, zasnovani okoli človeka kot plačnika, torej registracija računa, sklenitev naročnine in ročna potrditev plačila, v takem okolju postanejo ozko grlo, saj jih agent ne more izvesti samostojno. Prispevek zato obravnava vprašanje, kako plačilo vgraditi neposredno v protokol HTTP tako, da je izvedljivo strojno in brez predhodno vzpostavljenega poslovnega razmerja med odjemalcem in ponudnikom.

Da je tak mehanizem sploh smisel, morajo biti izpolnjeni tudi ekonomski pogoji. Analize mikroplačil dosledno kažejo, da ta niso spodletela zaradi tehnoloških omejitev, temveč zaradi stroška na transakcijo in kognitivnega bremena, ki ju človeški plačnik ni bil pripravljen sprejeti [5] [6]. Oba dejavnika se v strojnem okolju bistveno spremenita. Agent nima miselnih transakcijskih stroškov, poravnava s stabilnimi kriptožetoni v nizkocenovnih omrežjih pa strošek posamezne transakcije zniža na raven, pri kateri je zaračunavanje posameznega klica programskega vmesnika ekonomsko vzdržno [7].

Prispevek prinaša štiri sklope. Prvi je sistematičen pregled ovir, ki jih ima obstoječa plačilna infrastruktura pri zaračunavanju strojnega dostopa do podatkov. Drugi je celovit opis arhitekture in poteka protokola x402, vključno s primerjavo različic v1 in v2 ter pregledom obstoječih implementacij plačilnih posrednikov. Tretji je primerjalna analiza z alternativnim protokolom Machine Payments Protocol (MPP) s priporočili, kdaj je smiselna izbira posameznega protokola. Četrti je predstavitev dveh referenčnih implementacij, ki x402 uporabita v praksi.

Prispevek je organiziran tako, da naslednje poglavje opiše prevladujoče načine izmenjave in zaračunavanja dostopa do podatkov ter ovire, ki so pripeljale do nastanka protokola. Sledi podroben opis arhitekture in delovanja x402, primerjava različic, pregled primerov uporabe in implementacij plačilnih posrednikov ter primerjava z MPP. Zadnji vsebinski del predstavi uporabo x402 pri agentih umetne inteligence in v podatkovnih prostorih na primeru povezovalnika DSX Engine v podatkovnem prostoru DADS, sklep pa povzame ugotovitve in odprta vprašanja.

2 Trenutno stanje

2.1. Prevladujoč način izmenjave podatkov

Izmenjava podatkov med sistemi dandanes v veliki meri poteka prek spletnih vmesnikov, zasnovanih po arhitekturnem slogu REST, ki ga je leta 2000 v svoji doktorski disertaciji formaliziral Roy T. Fielding in ki od takrat vodi zasnovo arhitekture sodobnega spleta. Ponudnik podatkov izpostavi množico končnih točk, odjemalec pa nanje pošilja zahteve HTTP in v odgovoru prejme podatke [1]. Dostop do teh vmesnikov je pri komercialni rabi skoraj vedno vezan na avtentikacijo ter avtorizacijo, saj morajo ponudniki vsak zahtevek pripisati določenemu potrošniku, še posebej kadar želijo za svoje storitve zaračunavati [8].

2.2. Zaračunavanje dostopa

Ponudniki podatkov in vmesnikov (API-jev) zaradi odsotnosti standardiziranega plačilnega mehanizma, zgrajenega na podlagi protokola HTTP, uporabljajo vrsto lastnih poslovnih modelov monetizacije. Empirična analiza 268 dejanskih vmesnikov API in 54 podrobneje modeliranih cenovnih shem je pokazala, da so te sheme sestavljene iz naročniških paketov z vnaprej določenimi omejitvami in stroškom, ki je lahko fiksni (npr. mesečni znesek) ali dinamičen (npr. odvisen od porabe), pri čemer se dodaten strošek za preseganje kvote pojavi v približno 18,2 % analiziranih cenovnih shem [3]. V širšem smislu API-je vse pogostejše obravnavajo kot samostojne poslovne izdelke, katerih odpiranje navzven ustvarja nove poslovne priložnosti in organizacijske strukture (t. i. platformni posli) [8]. Nastali so tudi trgi podatkov (angl. data marketplaces), ki delujejo kot dvo- ali večstranske platforme za prodajo in nakup podatkov med uporabniki, ki so razpršeni.

Ne glede na izbrano cenovno shemo je skupna značilnost vseh naštetih pristopov, da je **plačilo strukturno ločeno od samega zahtevka** za podatke, pri čemer se mora uporabnik najprej registrirati, izbrati naročniški paket in šele nato prejeti poverilnico, s katero lahko dostopa do podatkov [3]. Del ponudnikov namesto lastne infrastrukture uporablja centralizirane trge podatkov oz. API-je, ki v zameno za poenostavljeno odkrivanje, naročanje in obračunavanje podatkov prevzamejo del neposrednega sporazumevanja s končnim uporabnikom. Raziskave kažejo, da tak model, čeprav zmanjšuje vstopne stroške za manjše ponudnike, hkrati osredotoča izmenjavo podatkov in denarja pri operaterju platforme ter uvaja dodatne tehnične in organizacijske ovire pri zagotavljanju varnosti, suverenosti, preglednosti, učinkovitosti in skladnosti z zakonodajo [4].

2.3. Izvedba plačila

Plačilo se v opisanih modelih izvede kot ločen, praviloma enkratni ali periodični poslovni dogodek, ne kot sestavni del posameznega zahtevka za podatke. Uporabnik plača vnaprej ali pa se mu obračuna naknadno, na podlagi agregirane porabe [3]. Pri čezmejnih plačilih, ki so za mednarodne podatkovne prostore in trge pogosta, se v transakcijo vključi dodatna veriga finančnih posrednikov. Po analizah Banke za mednarodne poravnave (BIS) in pobude G20 za izboljšanje čezmejnih plačil povprečni stroški čezmejnih plačil še vedno znatno presegajo ciljno vrednost 1 %, ki si jo je skupina G20 zastavila za leto 2027, pri čemer so stroški v posameznih regijah, na primer v podsaharski Afriki, občutno višji od globalnega povprečja, napredek pri hitrosti, dostopnosti in preglednosti teh plačil pa je počasnejši od načrtovanega [9].

2.4. Ovire

Analiza obstoječih načinov izmenjave in zaračunavanja dostopa do podatkov razkriva pet ponavljajočih se ovir. Prve tri so ekonomske in vedenjske narave, preostali dve pa izvirata iz razdrobljenosti plačilne infrastrukture in tehnične zahtevnosti na strani ponudnika.

Strukturna odvisnost od posrednikov. Centralizirani trgi podatkov in API-jev v menjava vnašajo dodatno plast, ki nadzoruje razmerje s stranko in zase zadrži del vrednosti transakcije, zaradi česar so ponudniki odvisni od infrastrukture, nad katero nimajo neposrednega nadzora [4].

Visoki stroški mikrotransakcij. Tradicionalna plačilna infrastruktura ima stroške na transakcijo, ki so pri zneskih pod nekaj centov ali dolarjev nesorazmerno visoki glede na sam znesek. Andrew Odlyzko je že leta 2003 na konferenci Financial Cryptography utemeljeval, da ovire za uporabo mikroplačil niso predvsem tehnološke, temveč ekonomske, sociološke in psihološke narave. Potrošniki dosledno raje izberejo pavšalno ceno kot veliko število drobnih plačil, kar sistemsko spodkopava poslovni model mikroplačil [5].

Kognitivni napor. Evropska centralna banka v svoji analizi mikroplačil opozarja, da so t. i. miselni transakcijski stroški, kognitivni napor pri vsaki posamezni odločitvi o plačilu, relevantni predvsem takrat, ko je vsaj ena od strani v transakciji človek, medtem ko lahko postanejo nepomembni, ko transakcije med seboj izvajajo avtomatizirane naprave [6]. To neposredno kaže na omejitev vseh danes prevladujočih plačilnih načinov, ki so bili zasnovani okoli človeka kot plačnika (izpolnjevanje obrazcev, vnos kartice, potrditev nakupa), ne pa okoli strojnega plačevanja z uporabo na primer avtonomnih programskih agentov.

Geografska omejenost in nizka interoperabilnost. Razpršenost plačilnih shem, različne regulativne zahteve in visoki stroški pri čezmejnem poslovanju povzročajo, da ostaja dostop do plačljivih podatkovnih virov omejen, sistemi med različnimi ponudniki podatkov pa med seboj niso interoperabilni [9].

Tehnično in organizacijsko breme za ponudnike. Vzpostavitev merjenja porabe, uveljavljanja omejitev in zanesljivega obračunavanja je za ponudnika zahtevna tako s tehničnega kot z organizacijskega vidika. Raziskave s področja trgovanja s podatki ugotavljajo, da prav ti izzivi in ne le pomanjkanje povpraševanja pojasnjujejo, zakaj le maloštevilna podjetja uspejo ustvariti oprijemljivo vrednost neposredno iz svojih podatkov [4].

Naštete ovire, torej strukturna odvisnost od posrednikov, nesorazmerni stroški mikrotransakcij, kognitivni napor človeškega plačnika, geografska razdrobljenost plačilnih shem in tehnično-organizacijsko breme za ponudnike, so skupaj z odsotnostjo v protokol vgrajenega plačilnega mehanizma ozadje, na katerem je nastal protokol x402, ki kodo HTTP 402 prvič uveljavlja kot dejansko delujoč, standardiziran način zahtevanja in izvedbe plačila neposredno v obstoječem zahtevku HTTP.

3 Protokol x402

x402 je odprt plačilni protokol, ki omogoča neposredno zaračunavanje dostopa do spletnih virov na podlagi protokola HTTP. Temelji na statusni kodi 402 Payment Required, ki je bila v standardu HTTP predvidena za primere, ko je za dostop do vira potrebno plačilo, vendar v običajnih spletnih arhitekturah doslej ni bila splošno uporabljena. x402 tej statusni kodi dodaja standardiziran postopek izmenjave podatkov o plačilu, preverjanja avtorizacije in izvedbe poravnave [7].

Glavni namen protokola je omogočiti strojna plačila brez tradicionalnega postopka registracije uporabniškega računa, vzpostavljanja naročnine ali pridobivanja ključa API. Plačljivi spletni vir lahko zato uporablja človek, običajna programska aplikacija ali avtonomni programski agent. Plačilo je vključeno neposredno v cikel zahtevka in odgovora HTTP, zaradi česar je x402 primeren predvsem za plačljive vmesnike API, digitalne vsebine, mikrostoritve in storitve, ki jih uporabljajo agenti umetne inteligence [7] [10].

Protokol ne določa enega samega plačilnega omrežja ali ponudnika infrastrukture. Plačilne zahteve so ločene od konkretnega načina poravnave, zato je mogoče podpirati različne verige blokov, kriptovalute, plačilne sheme in ponudnike storitev za preverjanje ter poravnavo plačila. Referenčna implementacija trenutno največ pozornosti namenja plačilom s stabilnimi kriptozetoni, kot je USDC, vendar je protokol zasnovan razširljivo in neodvisno od posamezne kriptovalute in verige blokov, na kateri je kriptovaluta [7].

3.1. Delovanje

Komunikacija x402 praviloma poteka v dveh zaporednih zahtevkih HTTP. Pri prvem zahtevku odjemalec še nima podatkov o plačilu. Strežnik zato dostop zavrne s HTTP kodo 402 Payment Required in posreduje zahteve za izvedbo plačila [7] [10].

V različici x402 v2 so podatki o zahtevanem plačilu vključeni v glavo PAYMENT-REQUIRED. Ti podatki vsebujejo [11]:

- Uporabljeno različico protokola x402
- Podatkovni vir (angl. resource):
 - spletni naslov URL do tega podatkovnega vira
 - Opis vira (neobvezno)
 - format MIME dejanskih podatkov, ki bodo poslani po uspešnem plačilu (neobvezno)
- Seznam posameznih plačilnih informacij za izvedbo plačila, ki jih ponuja ponudnik:
 - Plačilno shemo
 - Omrežje - verigo blokov v CAIP-2 formatu
 - Znesek
 - Naslov pametne pogodbe na zgoraj opredeljeni verigi blokov (kriptožeton, kot je na primer USDC)
 - Naslov kripto denarnice, ki prejme plačilo (kripto denarnica ponudnika podatkov)
 - Največja dovoljena časovna omejitev v sekundah za izvedbo plačila po prejemnikovi avtorizaciji plačila
 - Dodatne informacije, potrebne za uporabo določenih kriptožetonov (neobvezno)
- Opis napake (neobvezno)

Odjemalec na podlagi prejetega seznama plačilnih informacij izbere eno podprto kombinacijo omrežja in plačilne sheme. V svoji denarnici nato pripravi in kriptografsko podpiše plačilni objekt. Prvotni zahtevek ponovi, podpisano avtorizacijo pa posreduje v glavi PAYMENT-SIGNATURE.

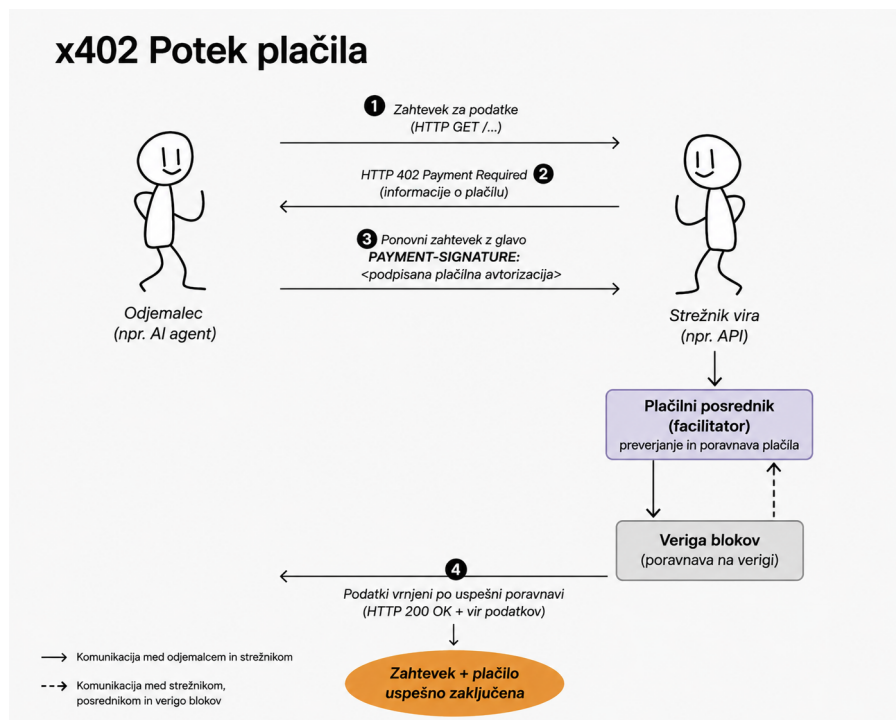
Strežnik ali izbrani plačilni posrednik preveri veljavnost podpisa, skladnost z zahtevanim zneskom, naslov kripto denarnice, časovno omejitev in podprte verige blokov. Če je preverjanje uspešno, se plačilo poravnava v skladu z uporabljenim plačilno shemo. Strežnik nato vrne rezultat plačila v glavi PAYMENT-RESPONSE in v primeru uspešnega plačila zahtevani vir [7] [10].

Celoten postopek je shematsko prikazan na sliki 1 [7] [11].

Takšen način delovanja odpravlja potrebo po preusmeritvi na ločeno plačilno stran. Plačilna logika ostane v okviru istega programskega vmesnika, zaradi česar jo je mogoče vključiti v knjižnice, strežniško vmesno programsko opremo in avtonomne agente.

Za preprečevanje večkratnega zaračunavanja pri ponovitvah zahtevkov je na voljo razširitev za identifikator plačila (angl. Payment-Identifier). Odjemalec zahtevku doda enolični identifikator, na podlagi katerega lahko strežnik prepozna že obdelano plačilo. To je pomembno pri prekinitvah omrežne povezave, samodejnih ponovitvah in arhitekturah z več uravnoteženimi strežniki [7].

Pri paketni poravnavi se avtorizacija in končni prenos sredstev časovno ločita. Odjemalec za posamezne zahtevke podpisuje zunajveržna dokazila, strežnik pa več takšnih avtorizacij pozneje poravnava v eni transakciji. S tem se zmanjšajo stroški in zakasnitev posameznega klica, kar je posebej pomembno pri storitvah z velikim številom nizko vrednostnih transakcij [7].



Slika 1: Potek plačila po protokolu x402 (prirejeno po [12])

3.2. Arhitektura

Arhitekturo x402 sestavlja pet osnovnih elementov: odjemalec, strežnik vira, denarnica, plačilna shema in plačilni posrednik za preverjanje ter poravnavo plačil (angl. facilitator) [7].

Odjemalec je aplikacija, spletni brskalnik, programska knjižnica ali avtonomni agent, ki želi dostopati do plačljivega vira. Odjemalec mora znati obdelati odgovor s statusom 402, pripraviti ustrezen plačilni objekt, ga podpisati in prvotni zahtevek ponovno poslati skupaj s plačilno avtorizacijo.

Strežnik vira zagotavlja plačljivi vmesnik API, vsebino ali drugo digitalno storitev. Za zaščiteni vir določi ceno, podprta omrežja, sprejeta sredstva, naslov prejemnika in uporabljeno plačilno shemo. Ob prejemu zahtevka brez ustreznega plačila vrne podatke, na podlagi katerih lahko odjemalec pripravi plačilno avtorizacijo.

Denarnica ima dvojno vlogo. Predstavlja plačilni mehanizem, s katerim odjemalec avtorizira prenos sredstev, hkrati pa lahko deluje kot kriptografska identiteta uporabnika ali programskega agenta. Zasebni ključ praviloma ostane pri odjemalcu, podpisovanje pa se izvede lokalno. Strežnik zato ne potrebuje dostopa do zasebnega ključa ali skrbništva nad sredstvi uporabnika [7].

Plačilna shema določa semantiko obračuna in poravnave. Ponudnik določi plačilno shemo glede na naravo podatkov oziroma storitve, ki jo izpostavlja v vmesniku API. Na voljo so tri plačilne sheme [7] [11]:

- exact, pri kateri odjemalec avtorizira plačilo natančnega zneska, ki ga določi ponudnik;
- upto, pri kateri odjemalec avtorizira najvišji dovoljeni znesek, strežnik pa poravnava samo dejansko uporabo;
- batch-settlement, pri kateri se posamezne avtorizacije zbirajo in pozneje poravnajo v skupni transakciji.

Shema exact je primerna za vire s fiksno ceno, na primer za dostop do posameznega dokumenta ali enkratni klic vmesnika API. Shema upto je namenjena storitvam, katerih cena je odvisna od dejanske porabe, kot so število obdelanih žetonov jezikovnega modela, čas izvajanja, prenesena količina podatkov ali poraba računalniških virov.

Paketna poravnava je namenjena visokofrekvenčnim interakcijam, pri katerih bi bila izvedba ločene verižne transakcije za vsak zahtevek prepočasna ali ekonomsko neučinkovita [7].

Samostojen plačilni posrednik je neobvezna, vendar priporočena infrastrukturna komponenta. Strežniku omogoča, da preverjanje plačil in izvedbo poravnave prenese na zunanjo storitev. Strežniku zato ni treba neposredno upravljati povezave z verigami blokov, preverjati vseh podpisov ali izvajati transakcij. Protokol je zasnovan brez dovoljenj, zato lahko plačilnega posrednika upravlja sam ponudnik podatkov, ponudnik javne infrastrukture, podjetje ali upravljavec zasebnega sistema [7].

Plačilni posrednik po preverjanju in poravnavi plačila rezultat vrne strežniku, ki odjemalcu posreduje zahtevani vir. Ker je komunikacija izvedena prek običajnih zahtevkov HTTP in ne zahteva trajne uporabniške seje, je osnovni postopek brez stanja (angl. stateless). Takšna zasnova olajša horizontalno skaliranje strežnikov, uporabo posredniških strežnikov in vključitev x402 v obstoječe spletne arhitekture.

Protokol dopolnjujejo izbirne razširitve. Med pomembnejšimi so prijava z denarnico Sign-In-With-X, identifikator plačila za zagotavljanje idempotentnosti, podpisane ponudbe in potrdila ter mehanizmi za sponzoriranje stroškov transakcij. Sign-In-With-X temelji na standardu CAIP-122 in omogoča, da uporabnik dokaže svojo identiteto s podpisom sporočila z isto denarnico, s katero je predhodno že plačal, kar omogoča seje za večkratno uporabo brez ponavljanja celotnega plačilnega postopka. Identifikator plačila strežniku omogoča, da posamezno plačilno zahtevo enolično poveže z izdanim odgovorom in tako prepozna podvojene poskuse (npr. zaradi ponovnih zahtev ob omrežnih napakah), ne da bi prišlo do dvojnega zaračunavanja ali zamenjave zahtevane storitve s cenejšo. Podpisane ponudbe in potrdila osnovno dokazilo o plačilu razširjajo v kriptografsko preverljiv zapis, ki poleg izvedbe plačila dokazuje tudi pogoje nakupa ter to, kdo in za kakšen namen ga je odobril, kar je pomembno za revizijsko sledljivost, zlasti pri plačilih, ki jih v imenu uporabnika izvajajo avtonomni agenti. Mehanizmi za sponzoriranje stroškov transakcij (npr. prek standarda EIP-2612 ali sponzoriranja odobritve ERC-20) pa omogočajo, da namesto kupca strošek odobritve plačilnega žetona krije plačilni posrednik, zato kupec za celoten postopek plačila ne potrebuje lastnega goriva (angl. gas) [7] [11]. Razširitve se vključijo v življenjski cikel preverjanja in poravnave, ne da bi bilo treba spremeniti osnovni potek komunikacije.

3.3. Primerjava zmožnosti različice 2 z različico 1

Različica x402 v2 ni zgolj sprememba poimenovanja ali strukture paketov. Uvaja standardizirane identifikatorje, modularno programsko arhitekturo, formalizirane razširitve in boljšo podporo zahtevnejšim ekonomskim modelom [7] [10]. Ključne razlike med različicama povzema tabela 1.

Prehod na identifikatorje CAIP-2 odpravlja dvoumnosti pri poimenovanju omrežij ter izboljšuje združljivost med različnimi verižnimi ekosistemi. Omrežje Base se na primer ne označuje več z lastnim besedilnim imenom, temveč s standardiziranim identifikatorjem, ki vključuje prostor imen in referenco omrežja [7] [10].

Druga pomembna sprememba je modularizacija knjižnic. Funkcionalnosti za jedro protokola, posamezna omrežja, odjemalce HTTP in strežniška ogrodja so razdeljene v ločene pakete. Razvijalec lahko zato vključi samo komponente, ki jih njegova aplikacija dejansko potrebuje.

Različica v2 formalizira tudi sistem razširitev. Funkcije, kot so avtentikacija z denarnico, idempotentnost, podpisana potrdila in odkrivanje storitev, se lahko vključijo kot sestavljivi dodatki. Protokol se tako iz mehanizma za enkratno plačilo na zahtevek razvija v splošnejšo ekonomsko plast za programske storitve [7].

Migracija vključuje zamenjavo glav HTTP, prehod na identifikatorje CAIP-2, posodobitev paketov in spremembo konfiguracije strežniških poti. Referenčna programska oprema omogoča tudi začasno podporo obema različicama, kar olajša postopno nadgradnjo obstoječih sistemov [10].

Tabela 1: Primerjava x402 v1 in x402 v2

<i>Vidik</i>	<i>x402 v1</i>	<i>x402 v2</i>
<i>Glava s plačilnim podpisom</i>	X-PAYMENT	PAYMENT-SIGNATURE
<i>Rezultat poravnave plačila</i>	X-PAYMENT-RESPONSE	PAYMENT-RESPONSE
<i>Posredovanje plačilnih zahtev</i>	V telesu odgovora	HTTP glava PAYMENT-REQUIRED
<i>Identifikacija omrežja</i>	Lastna imena, npr. base-sepolia	Standard CAIP-2, npr. eip155:84532
<i>Programska arhitektura</i>	Manj modularni paketi	Modularni paketi @x402/*
<i>Konfiguracija strežnika</i>	Enostavnejša konfiguracija posamezne poti	Seznam accepts z več shemami in omrežji
<i>Razširitve</i>	Omejeno formalizirane	Standardiziran sistem razširitev
<i>Odkrivanje storitev</i>	Ni del standardiziranega toka	Podpora razširitvi za odkrivanje in sistemu Bazaar
<i>Identiteta denarnice</i>	Predvsem plačilni mehanizem	Podlaga za avtentikacijo in ponovni dostop
<i>Ekonomski modeli</i>	Predvsem enkratna plačila	Fiksna, porabniška, predplačniška in večstopenjska plačila
<i>Združljivost</i>	Osnovna različica	Referenčni SDK-ji lahko podpirajo tudi v1

3.4. Primeri uporabe

Protokol x402 se v praksi uporablja na številnih področjih. Primeri uporabe so:

- Dostop agentov umetne inteligence do API-jev, kjer agent plača za posamezen klic (npr. vremenski podatki, finančne kotacije, lokacijske storitve);
- Monetizacija vsebin in podatkov, na primer plačljiv dostop do posameznih člankov, raziskovalnih objav ali podatkovnih tokov v realnem času, kjer stranka plača le za dejansko uporabljeno vsebino namesto za celoten paket;
- Trgi podatkov (angl. data marketplaces), kjer ponudniki podatkov in API-jev omogočajo avtomatizirano trgovanje z nabori podatkov ali tokovi podatkov;
- Storitve, obračunane po porabi, kjer se aplikacije in računalniške storitve (npr. inferenca jezikovnih modelov, obdelava žetonov, računski viri) zaračunavajo sproti, glede na dejansko porabljene vire;
- Igre in virtualna sredstva, kjer se x402 uporablja za nakupe v igrah in prenose virtualnih sredstev;
- Plačila med agenti (agent-to-agent), kjer avtonomni agenti med seboj najemajo storitve ali si izmenjujejo vire in plačila izvedejo neposredno, brez posredovanja človeka;
- Naprave interneta stvari (IoT) in druga strojna plačila, kjer nizkocenovne naprave samostojno plačujejo za dostop do storitev ali podatkov.

Podobno kot pri drugih odprtih protokolih se je tudi okoli x402 razvila širša razvijalska skupnost, ki jo spodbujajo številna tekmovanja, ki jih organizirajo podjetja, kot so Coinbase, Solana in Cronos. Tovrstna tekmovanja so pospešila nastanek velikega števila konkretnih aplikacij, ki uporabljajo protokol x402.

Med prijavljenimi projekti tovrstnih tekmovanj velja izpostaviti naslednje primere:

- Avtonomne agentske ekonomije, kjer agenti drug drugega najamejo in plačujejo za opravljene storitve z mikroplačili v kriptožetonih;
- Orodja za samodejno investiranje, ki uporabljajo x402 za izvajanje strategij dolarskega povprečenja stroškov (angl. dollar-cost averaging) za avtonomne portfelje;
- Infrastrukturo za varna agentska plačila, ki agentom omogoča premikanje sredstev z omejenimi, programsko določenimi pooblastili;
- Rešitve za samostojno plačevanje naprav IoT, kjer nizkocenovne naprave samostojno izvajajo plačila na verigi blokov;
- Avtomatizacijo naročil pri spletni trgovini, kjer agenti umetne inteligence samostojno izvedejo nakup v imenu uporabnika [10];
- Sisteme za vzpostavljanje bonitete in ocenjevanja agentov, ki omogočajo vzpostavitev zaupanja med avtonomnimi storitvami na podlagi zgodovine plačil;
- Poenostavljeno integracijo x402 prek konfiguracijskih datotek, namenjeno hitrejšemu vključevanju plačil v obstoječe aplikacije.

Raznolikost prijavljenih projektov kaže, da se protokol x402 ne uporablja zgolj kot teoretičen koncept, temveč že v številnih produkcijsko usmerjenih rešitvah, hkrati pa nakazuje smer nadaljnjega razvoja ekosistema, ki se osredotoča predvsem na avtonomno delovanje agentov umetne inteligence, mikroplačila in medstrojna plačila.

3.5. Implementacije plačilnih posrednikov

Ker je plačilni posrednik zamenljiva infrastrukturna komponenta, je nastalo več neodvisnih implementacij, ki se med seboj razlikujejo po podprtih omrežjih in kriptožetonih, zanesljivosti, uporabnosti in ceni.

Primeri uveljavljenih implementacij [7]:

- CDP Facilitator (Coinbase): brezplačna poravnava USDC na omrežju Base Mainnet z vgrajenim preverjanjem skladnosti transakcij (angl. Know Your Transaction, KYT) ter preverjanjem sankcijskih seznamov urada OFAC (angl. Office of Foreign Assets Control) pri vsaki transakciji. Zahteva avtentikacijo s ključem CDP API in uporablja omejitve hitrosti (angl. rate limits) glede na raven računa.
- PayAI Facilitator: pokriva več verig blokov (Base, Ethereum, Polygon, Avalanche, Sei, IoTeX, Peaq, Solana) in podpira vse žetone EIP-3009 ter SPL. Avtentikacija ni potrebna; zasnovan je za podporo širokemu naboru verig blokov.
- Mogami Facilitator: podpira Base in Base Sepolia, avtentikacija ni potrebna. Posebej je optimiziran za odjemalske in strežniške implementacije v programskem jeziku Java prek lastnega Java SDK-ja.
- Polygon Facilitator: produkcijsko zrel plačilni posrednik, namenjen izključno omrežju Polygon. Podpira preverjanje in poravnavo USDC prek standarda EIP-3009 na omrežju Polygon Mainnet in testnem omrežju Polygon Amoy.

3.6. Alternative in primerjava s protokolom Machine Payments Protocol (MPP)

Ena od alternativ protokolu x402 je MPP, odprt protokol za plačila med programskimi sistemi, ki sta ga zasnovala *Tempo* in *Stripe*. MPP je namenjen zaračunavanju zahtevkov API, klicev programskih orodij in dostopa do digitalnih vsebin. Podobno kot x402 omogoča izvedbo plačila neposredno v okviru komunikacije HTTP, ne da bi bila potrebna preusmeritev uporabnika na ločeno plačilno stran [13].

Tako x402 kot MPP uporabljata kodo HTTP 402 Payment Required, vendar se razlikujeta v načinu vključitve plačilnih podatkov v protokol HTTP. x402 uporablja namenske glave PAYMENT-REQUIRED, PAYMENT-SIGNATURE in PAYMENT-RESPONSE. MPP pa plačilo obravnava kot posebno shemo avtentikacije HTTP, imenovano Payment. Strežnik plačilni izziv posreduje v standardni glavi WWW-Authenticate, odjemalec dokazilo o plačilu pošlje v glavi Authorization, strežnik pa lahko rezultat poravnave vrne v glavi Payment-Receipt [7] [10] [14].

Plačilni izziv MPP vsebuje enolični identifikator, področje veljavnosti, plačilno metodo, namen plačila in podatke, potrebne za njegovo izvedbo. Vključuje lahko tudi čas poteka, opis, zgoščeno vrednost telesa zahtevka in podatke za povezovanje plačila s strežniškim stanjem. Enolični identifikator izziva in možnost vezave izziva na vsebino zahtevka sta namenjena preprečevanju sprememb zahtevka, ponovnega predvajanja plačilnih dokazil in uporabe iste avtorizacije za drug vir [14].

MPP je neodvisen od posamezne valute ali plačilnega omrežja. Osnovna specifikacija opredeljuje splošno ogrodje, posamezne plačilne metode pa so opisane v ločenih specifikacijah. Uradni seznam vključuje metode za plačilne kartice (fiat denar), omrežja EVM, Hedera, Lightning, Solano, Stellar, Tempo in USDC ter integracije Stripe in Near Intents [14].

Protokol razlikuje med različnimi nameni plačila. Namen *charge* je enkratno plačilo določenega zneska. Namen *session* omogoča postopno obračunavanje storitev, pri katerih se poraba spreminja med izvajanjem. Pri seji lahko odjemalec odpira plačilni kanal in z zaporednimi podpisanimi dokazili povečuje skupni odobreni znesek. Takšen model je primeren za pretočno izvajanje jezikovnih modelov, obračunavanje posameznih žetonov, dolgotrajne računske postopke in druge merjene storitve [14].

Specifikacije MPP vključujejo tudi mehanizem za odkrivanje plačljivih storitev in preslikavo plačilne sheme na komunikacijo JSON-RPC ter Model Context Protocol (MCP). MPP zato poleg običajnih virov HTTP neposredno obravnava tudi plačljiva orodja in storitve, ki jih uporabljajo agenti umetne inteligence [14].

Glavne razlike med protokoloma so prikazane v tabeli 2 [7] [10] [14]. Prednost x402 je razmeroma preprosta integracija, jasna vloga plačilnega posrednika in obstoječ nabor referenčnih knjižnic za priljubljena strežniška ogrodja. Plačilni posrednik lahko ponudnika storitve razbremeni neposrednega preverjanja transakcij in izvedbe poravnave plačil na verigah blokov.

MPP se tesneje opira na obstoječo semantiko avtentikacije HTTP. Uporaba glav WWW-Authenticate in Authorization omogoča vključitev plačil v mehanizem, ki ga spletni strežniki, posredniki in knjižnice HTTP že poznajo. Njegova dodatna prednost je formalna ločitev med abstraktnim protokolom, namenom plačila in konkretno plačilno metodo.

MPP z namenom session neposredno podpira postopna plačila za pretočne in merjene storitve [14]. Protokol x402 podobne primere uporabe obravnava s shemama upto in batch-settlement, pri katerih se plačilo prilagodi dejanski porabi oziroma se več avtorizacij pozneje združi v skupno poravnavo [7].

Pri oceni zrelosti je treba upoštevati, da so bile osnovne specifikacije MPP julija 2026 objavljene kot internetni osnutki. Internetni osnutek predstavlja dokument v razvoju, ki še nima statusa dokončnega standarda RFC. Specifikacije se lahko zato pred dokončno standardizacijo spremenijo, nadomestijo ali umaknejo [14].

Tabela 2: Primerjava x402 in MPP

<i>Vidik</i>	<i>x402</i>	<i>MPP</i>
<i>Osnovni transport</i>	HTTP in koda 402	HTTP in koda 402
<i>Posredovanje plačilnega izživa</i>	PAYMENT-REQUIRED	WWW-Authenticate: Payment
<i>Posredovanje dokazila</i>	PAYMENT-SIGNATURE	Authorization: Payment
<i>Potrдіlo o plačilu</i>	PAYMENT-RESPONSE	Payment-Receipt
<i>Protokolni model</i>	Namenske glave in podatkovni objekti x402	Standardna shema avtentikacije HTTP
<i>Abstrakcija plačila</i>	Omrežje in plačilna shema	Plačilna metoda in namen plačila
<i>Osnovni plačilni modeli</i>	exact, upto in paketna poravnava	charge in session
<i>Preverjanje in poravnava</i>	Izvede ju strežnik ali plačilni posrednik	Določena sta v specifikaciji posamezne plačilne metode
<i>Vloga plačilnega posrednika</i>	Plačilni posrednik je izrecno definiran element arhitekture	Protokol ne določa enotne vloge plačilnega posrednika
<i>Odkrivanje storitev</i>	Bazaar in razširitve za odkrivanje	Ločena specifikacija za odkrivanje storitev
<i>Podpora MCP</i>	Vodič in integracija strežnika MCP	Posebna specifikacija transporta JSON-RPC in MCP
<i>Stanje standardizacije</i>	Različica x402 v2 in referenčni SDK-ji	Specifikacije so objavljene kot osnutki IETF

Izbira med protokoloma za razvijalce, ki načrtujejo integracijo plačilnega mehanizma na svojih API-jih, je smiselno utemeljiti glede na naravo storitve in profil uporabnikov (agentov), ki jo bodo klicali. x402 je priporočljiva izbira za tiste, ki ponujajo ozko usmerjene (npr. funkcije z enim specializiranim namenom), visokofrekvenčne storitve z nizko vrednostjo posamezne transakcije, kot so dostop do podatkovnih virov, klici modelov AI, cenovni tokovi ali računski viri, kjer je ciljna publika že seznanjena s kripto plačili oziroma tehnično zmožna upravljati denarnice in podpisovati transakcije. Prednost je hitra in enostavna integracija brez potrebe po vzpostavitvi računov ali plačilne infrastrukture, dodaten strošek na transakcijo pa je zanemarljiv (le gorivo na verigah blokov). MPP je smiselna izbira za razvijalce, ki gradijo storitve za t. i. poslovne stranke, ponujajo fizično blago ali storitve, ki zahtevajo skladnost s predpisi, nadzor tveganj ali možnost plačila s poslovnimi karticami in kjer agent deluje v imenu uporabnika ali organizacije, ki nima kripto infrastrukture. Model session je primeren za storitve z neprekinjenim, visoko pretočnim značajem, kjer bi plačevanje za vsako posamezno zahtevo posebej (kot pri x402) povzročalo nepotrebne obremenitve. Razvijalcem, ki jim viri in čas to dopuščajo, velja priporočiti podporo obema protokoloma hkrati, saj lahko strežnik v istem odgovoru 402 oglašuje oba plačilna mehanizma in prepusti odjemalcu (agentu), da izbere tistega, ki ga podpira, kar razvijalcu zagotavlja najširšo možno dostopnost njegovih storitev.

4 Uporaba x402 v praksi

4.1. Agenti umetne inteligence

Agenti umetne inteligence lahko za izvedbo uporabnikovih nalog uporabljajo zunanje podatkovne vire, programske vmesnike in računalniške storitve. Pri tem klasični modeli dostopa, ki zahtevajo predhodno registracijo uporabniškega računa, sklenitev naročnine ali ročno izvedbo plačila, omejujejo njihovo avtonomno delovanje.

Protokol x402 jim omogoča, da informacije o ceni in pogojih plačila prejmejo neposredno kot del komunikacije HTTP, pripravijo ustrezno plačilno avtorizacijo ter po uspešni poravnavi nadaljujejo z uporabo zahtevanega vira. Agent tako ne potrebuje ločene interakcije uporabnika s plačilno stranjo, temveč lahko celoten postopek izvede programsko v okviru izvajanja naloge.

Primer takšne uporabe je agent, ki mora za pripravo odgovora zbrati spletne podatke, do katerih nima brezplačnega dostopa (npr. zbiranje objav z družbenih omrežij, pridobivanje podatkov o poslovnih subjektih ali odčitavanje cenovnih tokov), uporabnik pa za nalogo določi proračun.

Kot referenčno okolje uporabljamo platformo Apify, ki od junija 2026 omogoča, da agenti poženejo in plačajo njene programske komponente s stabilnim kriptozetonom USDC na omrežju Base prek protokola x402, in sicer brez uporabniškega računa, vzpostavljenega obračunavanja ali ključa API [15].

Agent v katalogu ponudnika Apify poišče orodje, ki ga potrebuje za izvedbo uporabnikove naloge, in zahtevek za njegov klic prek protokola MCP preda plačilni komponenti, ki je agentu izpostavljena kot strežnik MCP z orodjem, ki sprejme opis zahtevka HTTP (naslov, metoda, telo) in vrne odgovor ponudnika. Komponenta hrani namensko kriptodenarnico s sredstvi, dodeljenimi za pridobivanje plačljivih virov, in plačilno politiko – strojno berljiv zapis omejitev, ki določa dovoljene plačilne sheme, omrežja in kriptozetone, najvišji dovoljeni znesek posameznega plačila ter razpoložljivi proračun naloge. Takšna razmejitev ima dve posledici. Zasebni ključ, s katerim se podpisujejo plačila, ostane zunaj procesa agenta, hkrati pa jezikovni model o izvedbi plačila ne odloča, saj lahko plačilo kvečjemu predlaga s klicem orodja, avtorizirati pa ga ne more. Vrednotenje politike je namreč determinističen postopek nad podpisanimi plačilnimi pogoji in ne nad besedilom, ki ga model prejme od ponudnika ali iz vsebine pridobljenih podatkov. Primer takšne politike prikazuje izpis 1.

```
{
  "task": {
    "id": "task-8f21c4",
    "budgetUsd": "2.50",
    "spentUsd": "0.87"
  },
  "maxAmountPerRequestUsd": "0.25",
  "allowedNetworks": [
    "eip155:8453"
  ],
  "allowedAssets": [
    {
      "network": "eip155:8453",
      "address": "0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913",
      "symbol": "USDC"
    }
  ],
  "allowedSchemes": [
    "exact",
    "upto"
  ],
  "maxTimeoutSeconds": 600
}
```

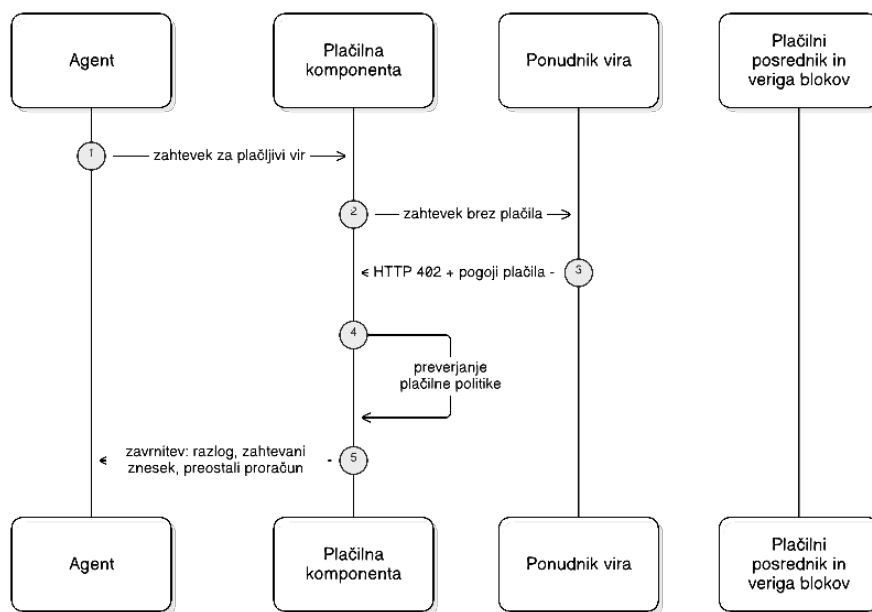
Izpis 1: Primer plačilne politike agenta

Potek. Postopek je enak osnovnemu poteku plačila po protokolu x402, prikazanemu na sliki 1, le da vlogo odjemalca prevzame plačilna komponenta, ki deluje v imenu agenta, vlogo strežnika vira pa ponudnik orodja.

Poteka v šestih korakih:

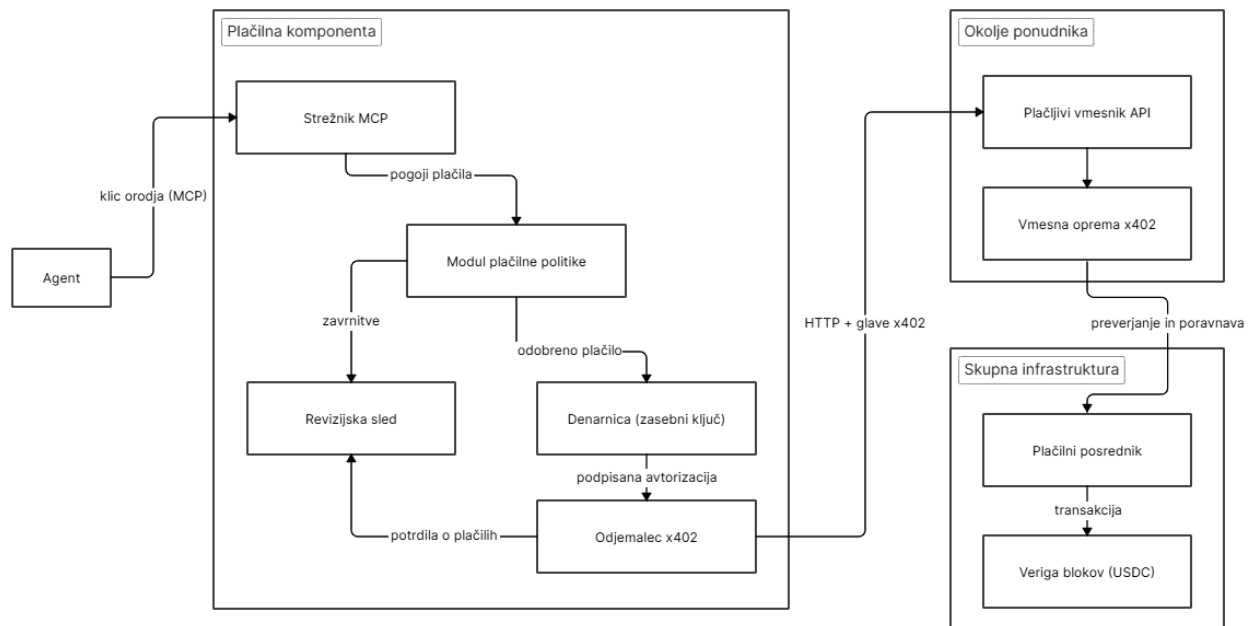
1. agent preda klic plačljive končne točke plačilni komponenti, ta pa zahtevek posreduje ponudniku;
2. ponudnik dostop zavrne z odgovorom HTTP 402 Payment Required in v glavi PAYMENT-REQUIRED posreduje seznam sprejetih plačilnih pogojev;
3. plačilna komponenta vsako ponujeno kombinacijo plačilne sheme, omrežja, kriptožetona in zneska ovrednoti glede na plačilno politiko in razpoložljivi proračun naloge;
4. če je katera od kombinacij odobrena, komponenta v denarnici lokalno pripravi in podpiše plačilno avtorizacijo ter zahtevek ponovi z glavo PAYMENT-SIGNATURE, ki ji doda enolični identifikator plačila;
5. ponudnik prek plačilnega posrednika preveri podpis in skladnost z zahtevanimi pogoji, plačilo poravna na verigi blokov ter vrne zahtevani vir skupaj s potrdilom o poravnavi v glavi PAYMENT-RESPONSE;
6. plačilna komponenta potrdilo zabeleži v revizijsko sled, zmanjša razpoložljivi proračun naloge in odgovor vrne agentu, ki ga uporabi pri oblikovanju odgovora uporabniku.

Zavrnitev plačila. Če plačilna politika nobene od ponujenih kombinacij ne odobri, ker zahtevani znesek presega dovoljeno mejo, ker preostali proračun naloge ne zadošča ali ker ponudnik ne sprejema nobenega dovoljenega omrežja oziroma kriptožetona, plačilna avtorizacija ni podpisana. Komponenta v tem primeru agentu vrne strukturirano napako z razlogom zavrnitve, zahtevanim zneskom in preostalim proračunom, zavrnitev pa zabeleži v isto revizijsko sled kot izvedena plačila. Odločitev o nadaljevanju je nato na strani agenta, ki lahko poskusi z drugim, cenejšim virom, od uporabnika zahteva izrecno odobritev nakupa ali nalogo prekine in razlog vključi v odgovor. Ta postopek je prikazan na sliki 2.



Slika 2: Diagram plačila agenta z zavrnitvijo po plačilni politiki

Slika 3 prikazuje razporeditev komponent rešitve. Iz tega sta razvidni dve lastnosti zasnove. Prvič, zasebni ključ ostane znotraj plačilne komponente in ni dosegljiv procesu, v katerem se izvaja jezikovni model, zato agent plačila ne more podpisati mimo nje. Drugič, plačilna komponenta je edina pot, po kateri zahtevki za plačljive vire zapustijo okolje agenta, zato je plačilna politika uveljavljiva omejitev in ne le priporočilo. Zaradi obojega plačilna komponenta ni zgolj programska knjižnica znotraj agenta, temveč ločena storitev z lastnim varnostnim obsegom, ki jo je mogoče uporabiti tudi za več agentov hkrati pri čemer se revizijska sled zbira na enem mestu.



Slika 3: Arhitektura rešitve za plačevanje dostopa agenta do plačljivih virov po protokolu x402

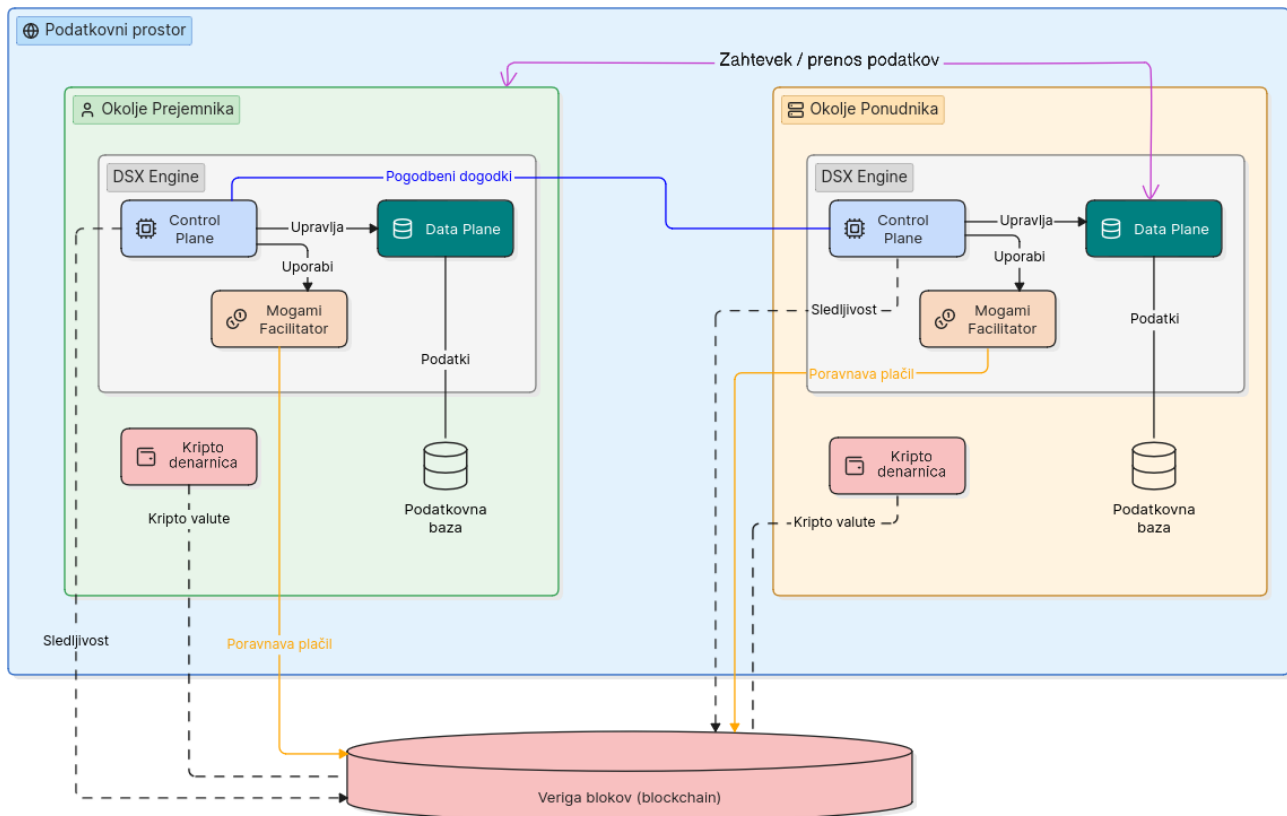
4.2. Podatkovni prostori

Zahteve, ki jih morajo izpolnjevati podatkovni prostori in ki temeljijo na evropski strategiji za podatke ter standardu združenja International Data Spaces Association (IDSA) [16], vključujejo suverenost udeležencev nad lastnimi podatki, preglednost postopkov in sledljivost izvedenih transakcij. Te zahteve se dobro ujemajo z lastnostmi protokola x402. Ker je x402 odprt standard, ki ni vezan na posrednike ali centralizirano infrastrukturo, ponudnikom podatkov omogoča, da sami določajo pogoje monetizacije svojih podatkov, kar je skladno z načelom suverenosti. Ker plačilne zahteve in odgovori potekajo prek standardiziranih glav HTTP (na primer PAYMENT-REQUIRED in PAYMENT-RESPONSE), je postopek plačila razumljiv tako za ponudnika kot za prejemnika. Ker se vsaka izvedena transakcija zabeleži v verigi blokov in jo je mogoče navesti v potrdilu v okviru glave PAYMENT-RESPONSE, x402 podpira tudi sledljivost transakcij v podatkovnem prostoru. Njegova neodvisnost od posamezne panoge, tehnologije ali posrednika ter enostavna integracija v obstoječo infrastrukturo HTTP omogočata, da se lahko x402 vključi v različne implementacije podatkovnih prostorov.

Decentraliziran povezovalnik podatkovnih prostorov DSX Engine [16] podpira monetizacijo podatkov v podatkovnih prostorih z uporabo protokola x402 [17], s čimer omogoča dinamično odkrivanje plačilnih informacij za monetizirane podatke in izvedbo pripadajočih plačil. DSX Engine z decentralizacijskimi mehanizmi in novimi funkcionalnostmi v obliki razširitev nadgrajuje referenčno implementacijo povezovalnika Eclipse Dataspace Components (EDC) podatkovnih prostorov, ki temelji na referenčni arhitekturi in standardih IDSA ter specifikacijah Gaia-X krovnega okvirja zaupanja. Na referenčni implementaciji EDC s svojimi razširitvami temeljijo tudi drugi podatkovni prostori, kot je Catena-X na področju avtomobilske industrije [16].

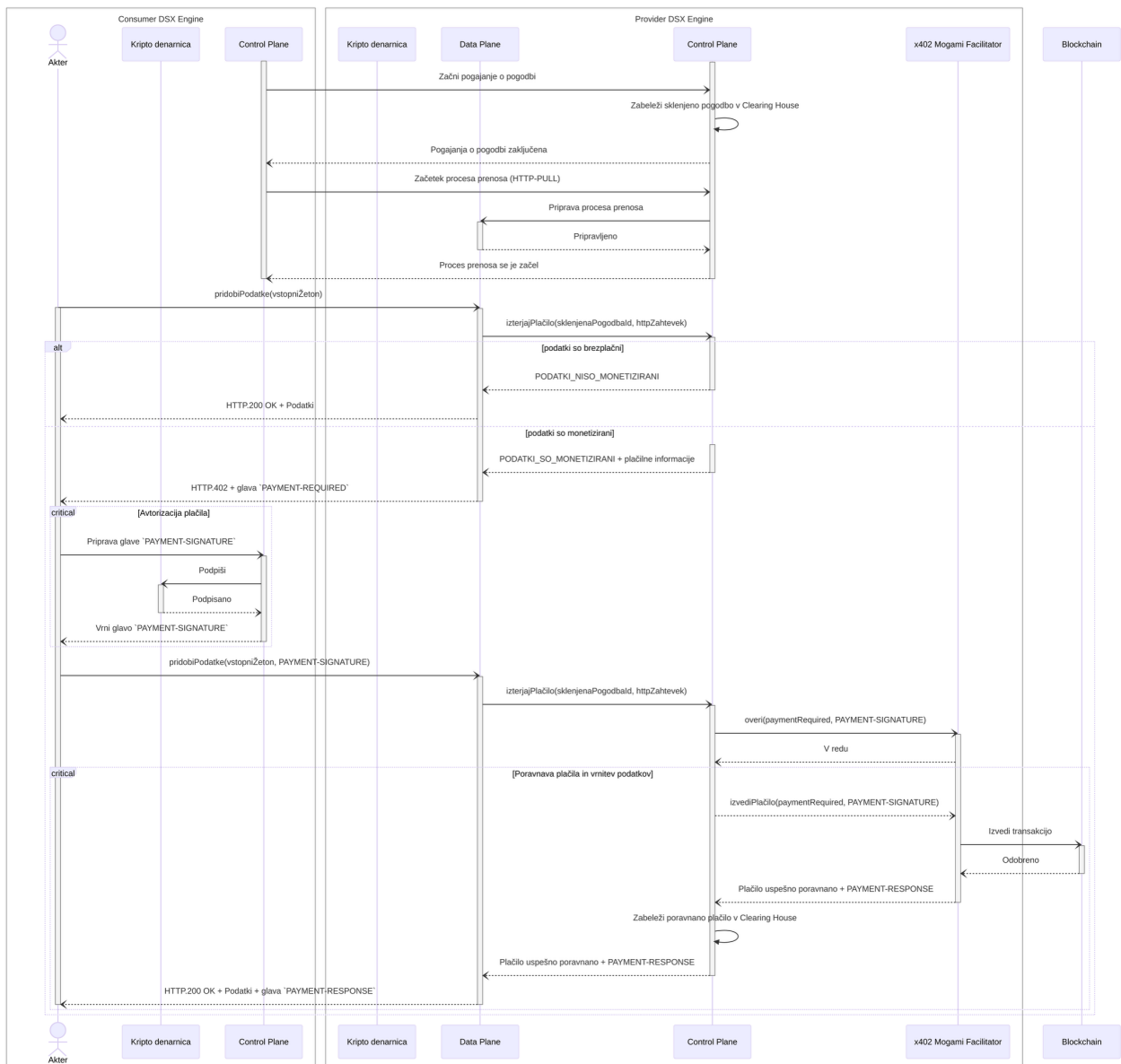
DSX Engine povezovalniki se uporabljajo v podatkovnem prostoru DADS (DIH AGRIFOOD Data Space), ki deluje v realnem okolju na področju kmetijstva. Koncept podatkovnih prostorov in zasnova povezovalnika DSX Engine sta bila podrobneje predstavljena že na lanski konferenci OTS 2025, zato ju v tem prispevku ne obravnavamo ponovno; podrobnosti so na voljo v [16], [17] in [18]. Ključna komponenta, ki omogoča izvedbo plačila x402 v DSX Engine, je Mogami Facilitator, ki je ena od implementacij plačilnih posrednikov x402. DSX Engine vsebuje svoj Mogami Facilitator, kot je razvidno na sliki 4, vendar je možna tudi uporaba zunanjega posrednika Mogami Facilitator. Funkcionalnost pridobivanja monetiziranih podatkov v podatkovnem prostoru omogoča razširitev monetizacije (angl. Monetization Extension), ki je del povezovalnika DSX Engine.

Komponenta, ki prejemu ob zahtevku po monetiziranih podatkih vrne odgovor po protokolu x402, je ponudnikova komponenta Data Plane.



Slika 4: Komponentni diagram DSX Engine za pridobivanje monetiziranih podatkov

Član podatkovnega prostora, v katerem se uporabljajo DSX Engine povezovalniki, pridobi želene (monetizirane) podatke od ponudnika podatkov tako, da z njim najprej sklene pogodbo (glej sliko 5). Po sklenitvi pogodbe med prejemnikom in ponudnikom podatkov v podatkovnem prostoru prejemnik sproži postopek prenosa tipa HTTP-PULL in do podatkov dostopa tako, da zahtevo pošlje na javni vmesnik HTTP API ponudnikove komponente Data Plane [18]. Data Plane je komponenta na strani ponudnika, ki skrbi za dejanski prenos podatkov in komunikacijo s prejemnikom. Ponudnikov Data Plane prejeto zahtevo najprej overi, pri čemer med drugim preveri, ali so zahtevani podatki monetizirani. To preverjanje delegira Control Plane komponenti, ki v ozadju upravlja pravila, politike in odločitve o tem, ali in pod kakšnimi pogoji se podatki lahko prenesejo, ločeno od komponente, ki prenos dejansko izvede. Control Plane tako na podlagi sklenjene pogodbe dinamično izračuna ceno trenutnega dostopa do podatkov in določi, ali je prenos dovoljen, hkrati pa po potrebi generira objekt s plačilnim zahtevkom ali izvede plačilo s pomočjo lokalnega plačilnega posrednika x402 Mogami, ki vrne potrdilo transakcije, ki bo vključeno v končni odgovor komponente Data Plane prejemniku. Kadar je za dostop do podatkov treba plačati, Data Plane s pomočjo Control Plane komponente vrne odgovor HTTP 402 Payment Required, ki vsebuje informacije o zahtevanem plačilu v skladu s protokolom x402. Če so podatki sicer monetizirani, a je bilo plačilo zanje že izvedeno v preteklosti in za trenutno zahtevo ni več potrebno, Data Plane vrne zahtevane podatke skupaj s potrdilom o predhodno izvedeni transakciji, vključenim v HTTP glavo PAYMENT-RESPONSE. Če podatki niso monetizirani, pa Data Plane preprosto vrne zahtevane podatke, uporaba x402 pa v tem primeru ni potrebna.



Slika 5: Sekvenčni diagram DSX Engine pridobivanja podatkov z uporabo x402

Kot je določeno v protokolu x402, se ob pridobivanju monetiziranih podatkov pojavijo tri vrste glav HTTP, ki so PAYMENT-REQUIRED, PAYMENT-SIGNATURE in PAYMENT-RESPONSE. Ponudnikov DSX Engine Data Plane ob zahtevku po monetiziranih podatkih brez dodane glave HTTP PAYMENT-SIGNATURE kot odgovor vrne HTTP 402 z glavo PAYMENT-REQUIRED, ki jo prikazuje izpis 2 (po pretvorbi iz formata BASE64).

```
{
  "x402Version": 2,
  "error": "PAYMENT-SIGNATURE header is required",
  "resource": {
    "url": "http://127.0.0.1:29291/api/public",
    "description": "Data asset with id 'asset-1'",
    "mimeType": "application/octet-stream"
  },
  "accepts": [
    {
      "scheme": "exact",
      "network": "eip155:84532",
      "amount": "24335",
      "asset": "0x036CbD53842c5426634e7929541eC2318f3dCF7e",
      "payTo": "0x04ec5b689e74a259b6fb6874391156926c4143b7",
      "maxTimeoutSeconds": 600,
      "extra": {
        "name": "USDC", "version": "2"
      }
    }
  ]
}
```

Izpis 2: Odgovor HTTP 402 z glavo PAYMENT-REQUIRED

```
{
  "x402Version": 2,
  "resource": {
    "url": "placeholder-url",
    "description": "Data asset with id 'test-asset-5'",
    "mimeType": "application/octet-stream"
  },
  "accepted": {
    "scheme": "exact",
    "network": "eip155:84532",
    "amount": "24335",
    "asset": "0x036CbD53842c5426634e7929541eC2318f3dCF7e",
    "payTo": "0x04ec5b689e74a259b6fb6874391156926c4143b7",
    "maxTimeoutSeconds": 600,
    "extra": {
      "name": "USDC", "version": "2"
    }
  },
  "payload": {
    "signature":
"0x5bb7e1910b7229779fe0d77217631d64d33edbf74545319ce2a52db717debd1a1c6281772a62c310c25e117ec23e5e
f5e240b8e3012e6ae1b9665225985d3a5b1c",
    "authorization": {
      "from": "0x25bb103e321bbf0b6ce3c08cd12f92a3292c09a5",
      "to": "0x04ec5b689e74a259b6fb6874391156926c4143b7",
      "value": "24335",
      "validAfter": "1785327538",
      "validBefore": "1785328138",
      "nonce": "0xfd362eae021141a56bde660036644d1b26c6d76610ed99c280351f9a2bb87c2b"
    }
  },
  "extensions": {}
}
```

Izpis 3: Glava PAYMENT-SIGNATURE, ki jo prejemnik pošlje ponudniku

Izpis 3 prikazuje primer glave HTTP PAYMENT-SIGNATURE, ki jo prejemnik pošlje ponudniku (po pretvorbi iz formata BASE64):

Tehnične značilnosti implementacije podpore monetizaciji podatkov v povezovalniku DSX Engine povzema tabela 3. Pri implementaciji podpore monetizaciji podatkov v DSX Engine smo najprej želeli problem zahtevanja in izvedbe plačila rešiti z uporabo lastnih pametnih pogodb, ki bi znotraj podatkovnega prostora urejale preverjanje in izvedbo plačil. Izkazalo se je, da implementacija takšnih pametnih pogodb zahteva veliko časa in znanja ter prinaša tveganja zaradi nepravilnosti, ki bi se odkrile pozneje med uporabo, saj pametnih pogodb ni mogoče spreminjati po objavi. Uvesti bi bilo treba tudi mehanizem posodabljanja pametnih pogodb, za preverjanje in izvajanje monetizacije ter plačil, kar je kompleksno. Protokol x402 nam je omogočil enake funkcionalnosti na standardiziran in odprt način, kjer je še vedno zagotovljena sledljivost, preglednost in suverenost. Prav tako so komponente, kot so plačilni posredniki x402 že razvite in na voljo za uporabo, zaradi česar smo se lahko popolnoma posvetili integraciji x402 v DSX Engine namesto lastni celoviti implementaciji mehanizma plačil z lastnimi pametnimi pogodbami. Skladnost s standardom x402 nam je prav tako omogočila morebitni razvoj pridobivanja podatkov z DSX Engine v podatkovnih prostorih z uporabo agentov umetne inteligence, pri katerem bi agenti našli želene podatke v podatkovnem prostoru in jih samodejno pridobili oziroma kupili.

Tabela 3: Tehnične značilnosti implementacije podpore monetizaciji podatkov v DSX Engine

<i>Podprta različica x402</i>	x402 v2
<i>Podprta omrežja oz. verige blokov</i>	Vse verige blokov, ki jih podpira uporabljeni plačilni posrednik x402. Mogami Facilitator podpira verigi blokov Base in Base Sepolia. Član podatkovnega prostora mora imeti kriptodenarnico v verigah blokov, v katerih želi poslovati.
<i>Podprti kriptožetoni</i>	Vsi ERC-20 žetoni in potencialno tudi drugi žetoni, kot so žetoni v verigi blokov Solana, če jih podpira tudi uporabljeni plačilni posrednik x402. Mogami Facilitator podpira vse žetone ERC-20 (na podprtih verigah blokov).
<i>Podprti plačilni posredniki x402</i>	Trenutno je podprt le Mogami Facilitator, vendar je mogoče implementirati razširitev za podporo tudi drugim plačilnim posrednikom [17].
<i>Način namestitve</i>	Podpora monetizaciji podatkov s protokolom x402 je del decentraliziranega povezovalnika DSX Engine podatkovnih prostorov. Namestitve je enaka osnovni namestitvi DSX Engine povezovalnika [16], le da mora član podatkovnega prostora imeti še kriptodenarnico v verigi blokov, v kateri želi prejemati oz. pošiljati kriptožetone, ki jih mora tudi pridobiti [17] [18].
<i>Programski jezik in ogrodje</i>	DSX Engine temelji na EDC. Implementiran je v programskem jeziku Java [18].

5 Zaključek

Protokol x402 ponuja odgovor na ovire, ki so bile identificirane pri obstoječih pristopih k zaračunavanju dostopa do podatkov in storitev: strukturno odvisnost od centraliziranih posrednikov, nesorazmerno visoke stroške tradicionalne plačilne infrastrukture pri mikrotransakcijah in plačilne poti, zasnovane okoli človeka kot plačnika. Glavna prednost protokola je vključitev plačilne logike neposredno v cikel zahtevka in odgovora HTTP, kar odpravlja potrebo po registraciji, naročnini ali ključu API in omogoča dostop tako ljudem kot avtonomnim agentom umetne inteligence. Enako pomembna je odprtost protokola, saj ni vezan na enega ponudnika, omrežje ali kriptovaluto, kar udeležencem omogoča prosto izbiro infrastrukture, plačilne sheme in plačilnega posrednika.

Različica v2 to prednost dodatno krepi s standardiziranimi identifikatorji CAIP-2, modularno arhitekturo in formaliziranim sistemom razširitev [7] [10] [11].

Kljub temu protokol ostaja v razmeroma zgodnji fazi zrelosti. Zanašanje na tehnologijo verige blokov pomeni, da so uporabniška izkušnja, regulativna obravnava in stabilnost vrednosti plačilnih sredstev odvisne od širšega kripto ekosistema, kar za del ponudnikov in uporabnikov predstavlja vstopno oviro. Vloga plačilnega posrednika uvaja novo točko trenja, saj mora ponudnik podatkov izbrani komponenti zaupati preverjanje in poravnavo plačil ali pa jo izvajati sam [7].

Primerjava s protokolom MPP pokaže dva sorodna, a različna pristopa k istemu problemu. Izbira med x402 in MPP je predvsem odvisna od infrastrukture in ciljne skupine uporabnikov. Za razvijalce, ki vstopajo v prostor strojnih (agentskih) plačil, je pragmatičen pristop naslednji: če gre za preprosto, visokofrekvenčno storitev z odjemalci, ki že poznajo kriptosredstva, je smiselno začeti z x402 zaradi hitre integracije, zrelega nabora SDK-jev in nizkih stroškov; če ciljate na podjetniške uporabnike, fizično blago ali potrebujete skladnost s predpisi in fiat plačila, izberite MPP. Kadar viri to dopuščajo, pa je dolgoročno najbolj smiselna podpora obema protokoloma hkrati, saj tako storitev ostane dostopna najširšemu krogu odjemalcev ne glede na to, kateri standard bo v prihodnje prevladal.

Literatura

- [1] R. T. Fielding, "Architectural styles and the design of network-based software architectures", Ph.D. dissertation, Dept. Inf. and Comput. Sci., Univ. of California, Irvine, CA, USA, 2000. Dostopno: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- [2] R. T. Fielding, M. Nottingham in J. Reschke, "RFC 9110: HTTP Semantics", junij 2022. Dostopano: 29. junij 2026. [Na spletu]. Dostopno: <https://www.rfc-editor.org/rfc/rfc9110.html>
- [3] R. Fresno-Aranda, P. Fernandez, A. Gamez-Diaz, A. Duran, in A. Ruiz-Cortes, "Pricing4APIs: A rigorous model for RESTful API pricings", Computer Standards & Interfaces, let. 91, str. 103878, jan. 2025, doi: 10.1016/j.csi.2024.103878.
- [4] Q. Ramadan, Z. Boukhers, M. AlShaikh, C. Lange, in J. Jürjens, "Data Trading and Monetization: Challenges and Open Research Directions", 17. januar 2024, arXiv: arXiv:2401.09199. doi: 10.48550/arXiv.2401.09199.
- [5] A. Odlyzko, "The Case Against Micropayments", v Financial Cryptography, let. 2742, R. N. Wright, Ur., v Lecture Notes in Computer Science, vol. 2742, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, str. 77–83. doi: 10.1007/978-3-540-45126-6_6.
- [6] European Central Bank. A big future for small payments?: Micropayments and their impact on the payment ecosystem. LU: Publications Office, 2023. doi: 10.2866/07262.
- [7] x402 Foundation, "x402 documentation", x402. Dostopano: 6. julij 2026. [Na spletu]. Dostopno: <https://docs.x402.org>
- [8] M. Zachariadis in P. Ozcan, "The API Economy and Digital Transformation in Financial Services: The Case of Open Banking", SSRN Journal, 2016, doi: 10.2139/ssrn.2975199.
- [9] T. Lammer, D. Rees, T. Rice, in T. Shirakami, "Enhancing cross-border payments: state of play and way forward", Bank for International Settlements, BIS Bulletins 119, dec. 2025. Dostopno: <https://www.bis.org/publ/bisbull119.pdf>
- [10] Coinbase, "x402 documentation", *Coinbase*. Dostopano: 6. julij 2026. [Na spletu]. Dostopno: <https://docs.cdp.coinbase.com/x402>
- [11] x402 Foundation, "x402", GitHub repozitorij. Dostopano: 6. julij 2026. [Na spletu]. Dostopno: <https://github.com/x402-foundation/x402>
- [12] Tiger Research, "x402: Coinbase and the Beginning of the AI Agent Era", CoinGecko. Dostopano: 3. avgust 2026. [Na spletu]. Dostopno: <https://www.coingecko.com/learn/x402-autonomous-ai-agent-payment-coinbase>
- [13] "Machine Payments Protocol", MPP. Dostopano: 8. julij 2026. [Na spletu]. Dostopno: <https://mpp.dev/overview>
- [14] "Machine Payments Protocol Specifications," PaymentAuth.org. Dostopano: 8. julij 2026. [Na spletu]. Dostopno: <https://paymentauth.org>

- [15] Apify, "Agentic payments with x402 protocol", Apify Documentation. Dostopano: 3. avgust 2026. [Na spletu]. Dostopno: <https://docs.apify.com/platform/integrations/x402>
- [16] M. Ferenec, T. Lah, in M. Turkanović, "DSX Engine: decentraliziran povezovalnik podatkovnih prostorov", v OTS 2025: sodobne informacijske tehnologije in storitve: zbornik 28. konference, L. Pavlič, T. Beranič, in M. Heričko, Ur., 1. izd. Maribor: Univerza v Mariboru, Univerzitetna založba, 2025, str. 109–130. doi: 10.18690/um.feri.7.2025.11.
- [17] M. Ferenec, T. Lah, M. Bunderla, D. Copot, in M. Turkanović, "Podpora ekonomiji podatkov v podatkovnih prostorih: implementacija monetizacije z DSX Engine", predstavljeno na 33. konferenci Dnevi slovenske informatike 2026: Digitalna družba na razpotju: ko se srečata regulacija in inovacija, Portorož, Slovenija, 15. april 2026.
- [18] M. Ferenec in M. Turkanović, "DSX Engine CE", GitHub repozitorij, 2025. [Na spletu]. Dostopno: <https://github.com/blockchain-lab-um/DSX-Engine-CE>

Sistem SKRINJA: odpiranje podatkov za javnost in napovedna analitika z uporabo umetne inteligence

Paula Jožefa Kolenko, Alenka Krebs

Ministrstvo za notranje zadeve in javno upravo Republike Slovenije, Ljubljana, Slovenija
paula.kolenko@gov.si, alenka.krebs@gov.si

V prispevku predstavljamo nadgradnjo poslovno-inteligenčnega sistema SKRINJA, horizontalne platforme za podporo odločanju v organih državne uprave. Sistem v ciljnem stanju omogoča zajem, integracijo in analizo podatkov iz različnih informacijskih virov ter zagotavlja ažurne informacije o ključnih kazalnikih poslovanja za podporo operativnemu in strateškemu odločanju. V zadnjem razvojnem obdobju je bil nadgrajen z dvema novima moduloma: Odprta Skrinja in Napovedna analitika. Modul Odprta Skrinja omogoča avtomatizirano objavo odprtih podatkov in njihovo interaktivno predstavitev z orodji poslovne inteligence. Kot prvi primer uporabe je bila izvedena objava podatkov o oddanih javnih naročilih na portalu e-JN, kjer so uporabnikom na voljo v obliki interaktivnih nadzornih plošč Power BI. Modul Napovedna analitika uporablja metode strojnega učenja za napovedovanje prihodnjih trendov. Njegova uporabnost je bila preverjena na podatkih Skupne kmetijske politike Ministrstva za kmetijstvo. Omogoča napovedovanje trendov opuščanja kmetijske dejavnosti ter pripravo analitičnih podlag za pravočasno načrtovanje ukrepov. Predstavljeni rešitvi sta bili razviti kot generični storitvi platforme SKRINJA, zato ju je mogoče uporabiti tudi na drugih področjih državne uprave.

Ključne besede:

horizontalna platforma
podatkovno skladišče
poslovna inteligenca
javna objava podatkov
napovedna analitika

1 Uvod

Prispevek predstavlja razvoj sistema SKRINJA kot horizontalne podatkovno-inteligenčne platforme, ki poleg klasične poslovne analitike vključuje tudi modul za objavo odprtih podatkov (Odprta Skrinja) in modul napovedne analitike z uporabo metod strojnega učenja ter prikazuje njuno uporabo na konkretnih primerih v državni upravi.

2 Razvoj sistema SKRINJA

2.1. Horizontalna platforma za organe državne uprave

Sistem SKRINJA je horizontalna poslovno-inteligenčna platforma, namenjena organom državne uprave. Njegov namen je omogočiti podatkovno podprto odločanje in oblikovanje politik na vseh ravneh upravljanja z zagotavljanjem enotnega analitičnega okolja za različna vsebinska področja. Platforma temelji na skupni metodologiji razvoja podatkovnih skladišč, standardiziranih analitičnih modelih ter sodobnih orodjih poslovne inteligence, kar omogoča ponovno uporabo rešitev in učinkovitejše uvajanje novih podatkovnih virov. Sistem upravlja Ministrstvo za notranje zadeve in javno upravo kot skupno horizontalno storitev državne uprave.

Platforma je bila vzpostavljena leta 2020. Vanjo so bili vključeni podatkovni viri s področja plač v javnem sektorju, javnih naročil in centralne kadrovske evidence. V nadaljnjem razvoju se platforma postopoma širi z uvedbo novih podatkovnih virov, hkrati pa se nadgrajuje z dodatnimi analitičnimi funkcionalnostmi, kot so objava odprtih podatkov na spletnih portalih in napovedna analitika z uporabo strojnega učenja. Takšna modularna zasnova omogoča dolgoročen razvoj sistema brez večjih posegov v obstoječo arhitekturo.

Osnovna organizacijska enota platforme je podatkovni vir, ki predstavlja logično zaključeno področje poslovanja posameznega organa državne uprave. Posamezen podatkovni vir lahko vključuje podatke iz več operativnih informacijskih sistemov, ki so združeni v skupno področno podatkovno skladišče na podlagi ustreznih pravnih podlag in poslovnih pravil. Takšna zasnova omogoča neodvisen razvoj posameznih področij, hkrati pa zagotavlja uporabo enotne metodologije vodenja postopka uvedbe, skupnih dimenzij oziroma šifrantov in analitičnih metod na ravni celotne platforme.

Do danes je bilo v sistem vključenih več podatkovnih virov, med drugim s področij plač v javnem sektorju, javnih naročil, kadrovske evidence, upravnih postopkov, inšpekcijskih postopkov in izvajanja Skupne kmetijske politike. Raznolikost podatkovnih virov potrjuje generično zasnovo platforme, ki omogoča uporabo enotnega tehnološkega in metodološkega pristopa ne glede na vsebinsko področje.

2.2. Arhitektura sistema SKRINJA

Horizontalna poslovno-inteligenčna platforma SKRINJA temelji na modularni večplastni arhitekturi, ki omogoča ločeno upravljanje operativnih podatkov, analitičnih podatkovnih skladišč, analitičnih storitev in uporabniškega predstavitvenega sloja. Takšna zasnova omogoča vključevanje novih podatkovnih virov in funkcionalnosti brez posegov v obstoječo informacijske sisteme posameznih organov državne uprave.

V kontekstu sistema SKRINJA izraz *horizontalna* pomeni, da platforma ni namenjena enemu vsebinskemu področju ali posameznemu organu državne uprave, temveč zagotavlja skupno metodologijo razvoja, tehnološko infrastrukturo in analitične storitve, ki jih je mogoče uporabljati na različnih področjih državne uprave.

Podatki iz operativnih informacijskih sistemov se prek standardiziranih ETL postopkov prenašajo v področna podatkovna skladišča, vzpostavljena po enotni metodologiji razvoja sistema SKRINJA. Vsako področno podatkovno skladišče predstavlja samostojno analitično okolje za posamezno vsebinsko področje (npr. javna naročila, centralna kadrovska evidenca, plače v javnem sektorju ali skupna kmetijska politika), pri čemer vsa uporabljajo skupne razvojne standarde, podatkovne modele in tehnološko platformo. Takšen pristop omogoča neodvisen razvoj posameznih področij ob hkratni ponovni uporabi skupnih komponent poslovne inteligence.

Nad posameznimi področnimi podatkovnimi skladišči je vzpostavljen skupni analitični sloj SQL Server Analysis Services (SSAS), ki predstavlja model za pripravo analiz in poslovnih kazalnikov. Predstavitveni sloj temelji na orodju Power BI, prek katerega uporabniki dostopajo do interaktivnih poročil in nadzornih plošč, prilagojenih posameznim področjem in uporabniškim skupinam.

Arhitektura platforme SKRINJA omogoča postopno širitev z novimi analitičnimi storitvami, ne da bi bilo treba spreminjati obstoječe informacijske sisteme ali razvijati ločene rešitve za posamezne organe državne uprave. S tem platforma predstavlja skupno podatkovno-analitično infrastrukturo, ki podpira učinkovitejše upravljanje podatkov, podatkovno podprto odločanje v državni upravi in večjo preglednost.

Tabela 1: Razvoj funkcionalnosti platforme SKRINJA

Razvojna faza	Ključna funkcionalnost	Dodana vrednost
Osnovna platforma	Poslovna inteligenca nad področnimi podatkovnimi skladišči	Enotno analitično okolje in podpora odločanju
Nadgradnja	Modul Odprta Skrinja	Preglednost in objava odprtih podatkov
Nadgradnja	Modul Napovedna analitika	Napovedovanje prihodnjih trendov z uporabo strojnega učenja

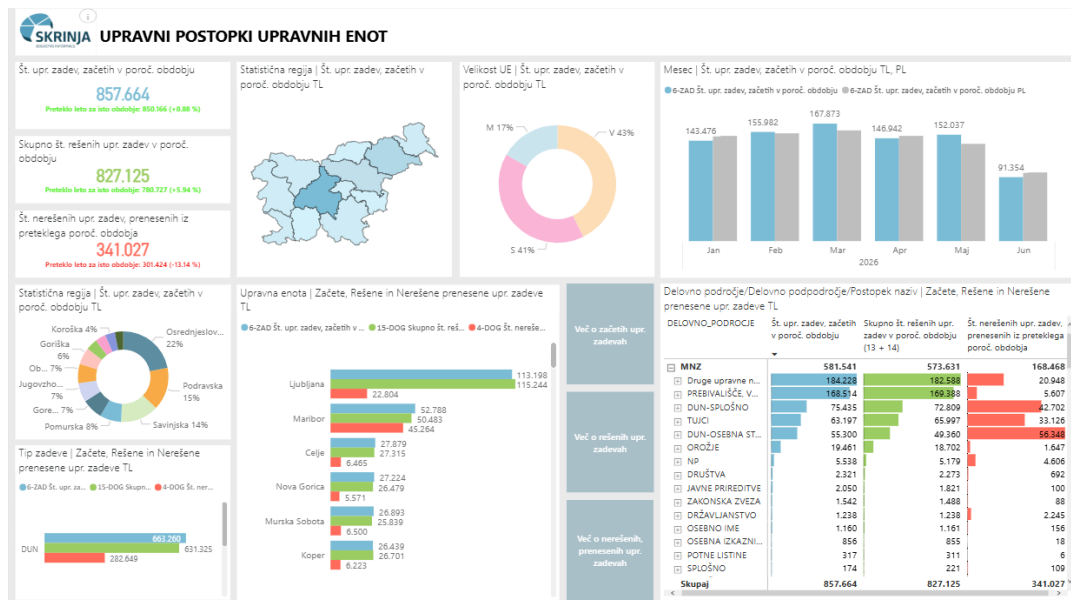
2.3. Podatkovni viri v sistemu

V sistem SKRINJA smo kot prvi podatkovni vir implementirali podatke o plačah v celotnem javnem sektorju (ISPAP), s katerimi upravlja Direktorat za javni sektor Ministrstva za notranje zadeve in javno upravo. Gre za kompleksen podatkovni vir (podatki o plačah več kot 180.000 javnih uslužbencev), kar je zahtevalo veliko prilagajanja na vseh ravneh razvoja, še posebej ob sočasnem razvoju platforme. Hkrati smo v sistem vpeljali podatkovni vir Skupne dimenzije, ki vključuje skupne šifrantе za uporabo na vseh podatkovnih virih, ki se zajemajo neposredno pri viru (PRS-AJPES, Tečajnica – Banka Slovenije...). V naslednjih letih smo v sistem SKRINJA pospešeno uvajali nove podatkovne vire (projekte) tako, da so danes v uporabi naslednji podatkovni viri:

- Ministrstvo za notranje zadeve in javno upravo Republike Slovenije:
- oddana javna naročila (več kot 11 % BDP), upravni postopki upravnih enot (več kot 1 milijon dokumentov letno), plače v javnem sektorju (več kot 180.000 javnih uslužbencev) ter kadrovske evidence organov državne uprave (več kot 15.000 zaposlenih);
- Ministrstvo za kmetijstvo Republike Slovenije: podatki o izvajanju skupne kmetijske politike (več kot 80.000 kmetijskih gospodarstev, več kot 80 % vseh kmetij);
- Tržni inšpektorat Republike Slovenije: podatki o inšpekcijskih postopkih (približno 17.000 letno);
- Zdravstveni inšpektorat Republike Slovenije: podatki o inšpekcijskih postopkih (približno 22.000 zadev letno);
- Ministrstvo za kulturo Republike Slovenije: podatki o javnih razpisih in javnih pozivih, v okviru katerih se razdelijo sredstva za uresničevanje javnega interesa na področju kulture.

Ob uvedbi posameznega podatkovnega vira upravljaavec podatkovnega vira odobri dostop do podatkov uporabnikom za uporabo v omrežju HKOM. Po izkušnjah je v začetni fazi teh uporabnikov le nekaj naprednih analitikov iz posameznega organa, ki so sodelovali pri razvoju. V državni upravi namreč še vedno opažamo veliko nezaupanje glede posredovanja in analize podatkov. V nadaljevanju pa se krog uporabnikov ustrezno razširi, saj upravljalci virov bolj zaupajo v pravilnost prikazanih (svojih) podatkov. Tako ima podatkovni vir Upravni postopki upravnih enot že skoraj 300 uporabnikov (vodstvo ministrstva, načelniki, vodje NOE, analitiki), vir Inšpekcijski

postopki Tržnega inšpektorata že več kot 30 uporabnikov – inšpektorjev, vir Inšpekcijski postopki Zdravstvenega inšpektorata več kot 90 uporabnikov – inšpektorjev in vir Skupna kmetijska politika Ministrstva za kmetijstvo že več kot 50 uporabnikov (zaposleni na MKM). Kot zadnja sta bila v juniju letos v sistem SKRINJA vključena CKEDU in JRP-MK, ki tudi že širita krog uporabnikov.



Slika 1: Primer nadzorne plošče s ključnimi kazalniki uspeha

2.4. Nadaljnji razvoj analitike

Trenutno razvijamo oziroma v sistem uvajamo in nadgrajujemo več podatkovnih virov:

- Sprejemanje zakonodaje v Sloveniji Službe Vlade Republike Slovenije za zakonodajo (SZS-SVZ),
- Upravni postopki Centrov za socialno delo,
- Upravni postopki upravnih enot (UPP) – širitev podatkovnega skladišča s podatki aplikacije eGraditve Ministrstva za naravne vire in prostor,
- Oddana javna naročila Ministrstva za notranje zadeve in javno upravo – nadgradnja zaradi spremembe metodologije pri zajemu podatkov.

Načrtujemo, da bomo v naslednjem štiriletnem obdobju v sistem SKRINJA uvedli dodatne podatkovne vire različnih državnih organov, ki so izkazali interes. Na podlagi poizvedbe med državnimi organi smo preverili, da je zanimanje za uvedbo podatkovnih virov v sistem SKRINJA veliko. To potrjuje, da se je tudi v državni upravi okrepila analitična kultura in da je prepoznana pomembnost predpripravljenih (avtomatizirane) analitike z realnimi podatki za hitrejša in kakovostnejša odločanja.

3 Odprta SKRINJA

3.1. Namen in cilji

Odprta Skrinja predstavlja nadgradnjo platforme SKRINJA, katere namen je zagotoviti standardizirano pripravo, upravljanje in objavo odprtih podatkov. Razvoj modula izhaja iz potrebe po učinkovitejšem upravljanju podatkov, zmanjšanju podvajanja postopkov priprave podatkovnih zbirk ter zagotavljanju enotnega in zaupanja vrednega vira podatkov za interno analitiko in javno objavo.

Tradicionalni pristopi k objavi odprtih podatkov pogosto temeljijo na pripravi ločenih podatkovnih zbirk in ročnih postopkih izvoza podatkov iz operativnih informacijskih sistemov. Takšen način povečuje možnost napak, otežuje redno posodabljanje podatkov in povzroča neskladja med internimi analitičnimi podatki ter javno objavljenimi informacijami.

Cilj razvoja modula Odprta Skrinja je bil vzpostaviti rešitev, ki omogoča pripravo odprtih podatkov neposredno iz podatkovnega skladišča platforme SKRINJA. Na ta način odprti podatki postanejo naravni rezultat procesa upravljanja podatkov in ne predstavljajo ločenega informacijskega sistema. Rešitev zagotavlja avtomatizirano pripravo podatkov, zagotavljanje kakovosti podatkov in njihovo redno objavo za ponovno uporabo.

3.2. Arhitektura modula

Modul Odprta Skrinja je zasnovan kot sestavni del horizontalne podatkovno-analitične platforme SKRINJA. Temelji na podatkovnem skladišču Oracle, ki združuje podatke iz različnih operativnih informacijskih sistemov ministrstva. Pred vključitvijo v podatkovno skladišče se podatki obdelajo v postopkih ETL, v katerih se izvedejo validacija, transformacija in poenotenje podatkovnih struktur.

Podatkovno skladišče predstavlja enotni vir resnice, zato modul za pripravo odprtih podatkov uporablja iste konsolidirane podatke kot poslovna analitika. Takšna arhitektura zagotavlja, da so vsi uporabniki – interni analitiki in zunanji uporabniki odprtih podatkov – usmerjeni na isti podatkovni vir. Tako je zagotovljena tudi popolna usklajenost med javno objavljenimi podatki in internimi analitičnimi podatki, ki jih uporabljajo strokovne službe organa.

Za posamezno podatkovno zbirko se določijo pravila izbora podatkov, način njihove predstavitve, pogostost osveževanja ter pripadajoči metapodatki. Po pripravi podatkov se zbirke samodejno objavijo prek uporabniškega portala, kjer so dostopne za pregledovanje, iskanje in nadaljnjo uporabo. Takšna zasnova odpravlja potrebo po vzdrževanju ločenih podatkovnih zbirk za javno objavo, zmanjšuje podvajanje podatkov in poenostavlja upravljanje celotnega življenjskega cikla odprtih podatkov.

Ker odprti podatki predstavljajo javno dostopen vir informacij, je zagotavljanje njihove kakovosti ključni del celotnega procesa. Modul uporablja enake postopke validacije kot preostali deli platforme SKRINJA, kar zagotavlja doslednost, popolnost in sledljivost objavljenih podatkov.

Predstavljena arhitektura omogoča, da odprti podatki niso obravnavani kot ločena funkcionalnost, temveč kot naravna razširitev upravljanja podatkov v platformi SKRINJA. Uporaba enotnega podatkovnega skladišča zmanjšuje podvajanje podatkov, poenostavlja vzdrževanje sistema in zagotavlja usklajenost med internimi analizami ter javno objavljenimi podatki.

Modularna zasnova omogoča enostavno vključevanje novih podatkovnih zbirk, prilagajanje spremembam zakonodaje in razširjanje funkcionalnosti brez večjih posegov v osnovno arhitekturo platforme. Predstavljena rešitev tako predstavlja pomemben korak k učinkovitejšemu upravljanju podatkov, večji preglednosti delovanja javnega sektorja in spodbujanju ponovne uporabe informacij javnega sektorja.

3.3. Uporabniški vidik in primer uporabe

Modul je namenjen dvema skupinama uporabnikov. Prvo skupino predstavljajo skrbniki podatkov, ki upravljajo podatkovne zbirke, določajo metapodatke ter spremljajo postopke objave in osveževanja podatkov. Celoten postopek upravljanja poteka v enotnem uporabniškem okolju platforme SKRINJA.

Drugo skupino predstavljajo končni uporabniki odprtih podatkov. Ti lahko prek spletnega portala pregledujejo objavljene podatkovne zbirke in podatke prenašajo v standardnih odprtih formatih za nadaljnjo uporabo. Takšen način dostopa omogoča ponovno uporabo podatkov raziskovalcem, podjetjem, javnim institucijam, medijem in širši javnosti.

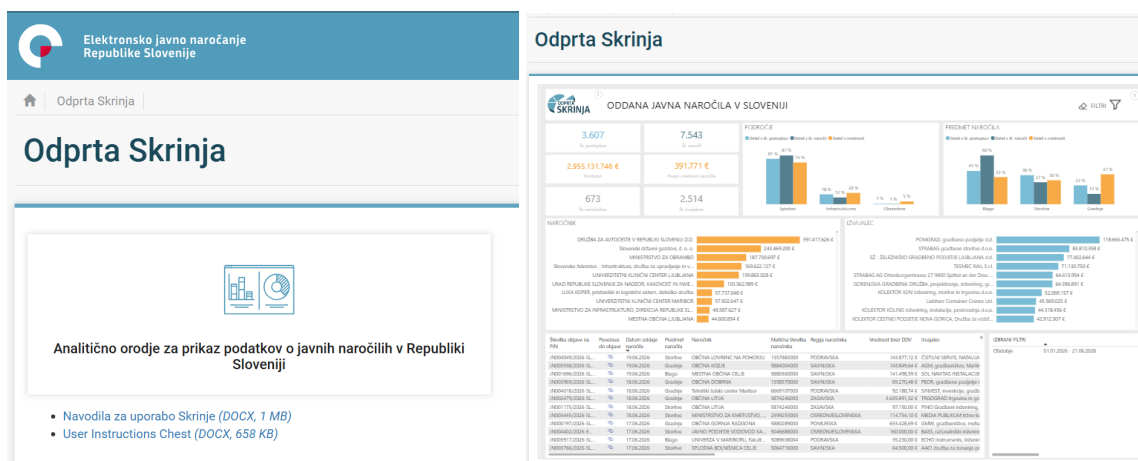
Kot prvi produkcijski primer uporabe modula Odprta Skrinja je bila vzpostavljena rešitev za objavo podatkov o oddanih javnih naročilih v Republiki Sloveniji v okviru Portala za elektronsko javno naročanje (e-JN). Rešitev je nadomestila prejšnjo aplikacijo Statist in omogočila sodobnejši način dostopa do podatkov o javnem naročanju.

Podatki o oddanih javnih naročilih se pripravljajo neposredno iz podatkovnega skladišča SKRINJA, kjer so predhodno konsolidirani in validirani. Za objavo se uporabljajo vnaprej določena pravila, ki določajo izbor atributov, način agregacije podatkov, pogostost osveževanja in pripadajoče metapodatke. Po pripravi se podatkovni nabor samodejno objavi v modulu Odprta Skrinja, kar odpravlja potrebo po ločenem izvozu ali ročni pripravi podatkov.

Končni uporabniki lahko prek portala e-JN pregledujejo podatke z uporabo različnih filtrov, kot so naročnik, ponudnik, predmet naročila, CPV koda, vrsta postopka in časovno obdobje. Poleg interaktivnih pregledov so na voljo tudi preglednice z možnostjo izvoza podatkov za nadaljnjo analizo, kar omogoča njihovo uporabo raziskovalcem, nadzornim institucijam, gospodarskim subjektom in širši javnosti.

Predstavljeni primer potrjuje, da je mogoče z uporabo enotnega podatkovnega skladišča učinkovito podpreti tako notranje analitične procese kot tudi odprto objavo podatkov. Modul Odprta Skrinja tako ne predstavlja zgolj portala za objavo odprtih podatkov, temveč integrirano komponento horizontalne podatkovno-analitične platforme SKRINJA, ki zagotavlja avtomatizirano pripravo, kakovost, sledljivost in ponovno uporabo podatkov iz enotnega vira.

Odprta Skrinja: <https://ejn.gov.si/skrinja.html>



Slika 2: Odprta SKRINJA na Portalu e-JN

3.4. Nadaljnji razvoj

Uvedba modula na področju javnega naročanja je pokazala, da je mogoče odprte podatke učinkovito objavljati neposredno iz poslovno-inteligenčne platforme brez razvoja ločenih rešitev za posamezne podatkovne vire. Ker je modul zasnovan generično, ga je mogoče uporabiti tudi za druge podatkovne vire, vključene v sistem SKRINJA, ki želijo podatke javno objaviti.

Uporaba rešitve bo organom državne uprave omogočala enostavno pripravo in objavo kakovostnih odprtih podatkov ter s tem prispevala k večji preglednosti delovanja javnega sektorja, učinkovitejši ponovni uporabi podatkov in nadaljnji digitalni preobrazbi javne uprave.

4 Napovedna analitika

4.1. Namen in cilji

Napovedna analitika predstavlja eno ključnih nadgradenj platforme SKRINJA, katere namen je razširiti klasično poslovno analitiko z metodami umetne inteligence in strojnega učenja. Medtem ko poslovna analitika odgovarja na vprašanje, *kaj se je zgodilo*, napovedna analitika omogoča oceno, *kaj se bo verjetno zgodilo* ter tako podpira pravočasno in podatkovno podprto odločanje.

Pri razvoju modula je bilo izhodišče, da mora biti rešitev popolnoma vključena v obstoječo podatkovno-analitično platformo. Cilj ni bil razviti samostojne aplikacije za strojno učenje, temveč vzpostaviti produkcijsko okolje, ki omogoča pripravo podatkov, razvoj, vrednotenje in izvajanje napovednih modelov ter njihovo uporabo v vsakodnevnih analitičnih procesih.

4.2. Arhitektura modula

Napovedna analitika je zasnovana kot modularna komponenta platforme SKRINJA. Vhodne podatke predstavljajo konsolidirani podatki podatkovnega skladišča Oracle, ki nastanejo po postopkih zajema, validacije, čiščenja in transformacije podatkov iz različnih operativnih informacijskih sistemov. Podatkovno skladišče predstavlja enoten vir resnice, kar zagotavlja enotno interpretacijo podatkov in sledljivost vseh analitičnih rezultatov.

Za razvoj in izvajanje napovednih modelov je bilo izbrano okolje Python, saj omogoča uporabo uveljavljenih knjižnic za strojno učenje, učinkovito upravljanje življenjskega cikla modelov ter dobro integracijo z obstoječo podatkovno platformo. Napovedni modeli se izvajajo nad pripravljenimi podatki iz podatkovnega skladišča, rezultati pa se po zaključeni obdelavi zapišejo nazaj v podatkovno skladišče kot nov analitični sloj.

Na ta način napovedi postanejo sestavni del poslovnih podatkov in so uporabnikom dostopne prek večdimenzionalnih modelov SQL Server Analysis Services (SSAS) ter vizualizirane v okolju Microsoft Power BI. Takšna arhitektura omogoča ponovno uporabo rezultatov napovednih modelov v vseh analitičnih rešitvah platforme ter enostavno vključevanje novih modelov brez sprememb osnovne arhitekture.

4.3. Razvoj napovednih modelov

Razvoj napovednega modula je potekal po standardni metodologiji strojnega učenja, ki vključuje pripravo podatkov, razvoj kandidatnih modelov, njihovo primerjalno vrednotenje ter izbor najprimernejše rešitve za produkcijsko uporabo.

Kot pilotni primer uporabe napovedne analitike je bil razvit model za področje kmetijstva, katerega namen je oceniti verjetnost, da bo posamezna kmetija v naslednjem letu prenehala z dejavnostjo. Model trenutno zajema več

kot 100.000 kmetij. Pri tem upošteva pretekle podatke in značilnosti kmetije — na primer, kako dolgo deluje, kako redno oddaja vloge, kakšen je prihodek, obseg površin, naklon površin in podobno. Namen modela ni le opis stanja, temveč zagotavljanje zgodnjih opozoril, ki pristojnim institucijam omogočajo pravočasno načrtovanje in izvajanje ustreznih ukrepov.

Delovanje je samodejno. Ob rednem osveževanju podatkov v skladišču se v ozadju posodobijo tudi napovedi. Uporabnikom ni treba modela zagnati sami — na nadzornih ploščah so ocene tveganja že pripravljene.

Pri razločljivosti nam ni dovolj, da model le pove »visoko« ali »nizko« tveganje. Na nadzornih ploščah zato rezultate pokažemo tako, da jih je mogoče razumeti in preveriti. Za vsako kmetijo je na voljo verjetnost opustitve dejavnosti, hkrati pa so kmetije razvrščene v razrede tveganja (npr. nizko, srednje, visoko), da lahko uporabnik hitro vidi, kje je treba ukrepati najprej. Poleg ocene prikažemo tudi ključne značilnosti kmetije, ki so v modelu najbolj povezane s tveganjem — na primer, kako dolgo kmetija že deluje, koliko časa je minilo od zadnje zbirne vloge, kakšen je prihodkovni položaj ali obseg površin. Tako odločevalec ne vidi le številke, temveč tudi kontekst, ki pomaga razumeti uvrstitev v razred. Cilj je, da napoved ni »črna škatla«, ampak orodje, ki podpre odločanje z jasnimi, preglednimi informacijami. V nadaljevanju načrtujemo še bogatejše razlage in bolj intuitivne prikaze.

Model je bil integriran v platformo SKRINJA kot del rednih analitičnih postopkov. Življenjski cikel modela obsega pripravo vhodnih podatkov, izvajanje modela, zapis rezultatov v podatkovno skladišče ter prikaz napovedi v analitičnih poročilih. Takšna zasnova omogoča periodično ponovno učenje modela ob razpoložljivosti novih podatkov in njegovo nadaljnjo izboljšavo brez sprememb osnovne arhitekture sistema.

Pomembna značilnost rešitve je, da napovedni model ni samostojna aplikacija, temveč sestavni del podatkovno-analitične platforme. Rezultati napovedi so zato neposredno povezani z drugimi poslovnimi kazalniki in jih je mogoče uporabljati v vseh obstoječih analitičnih poročilih.

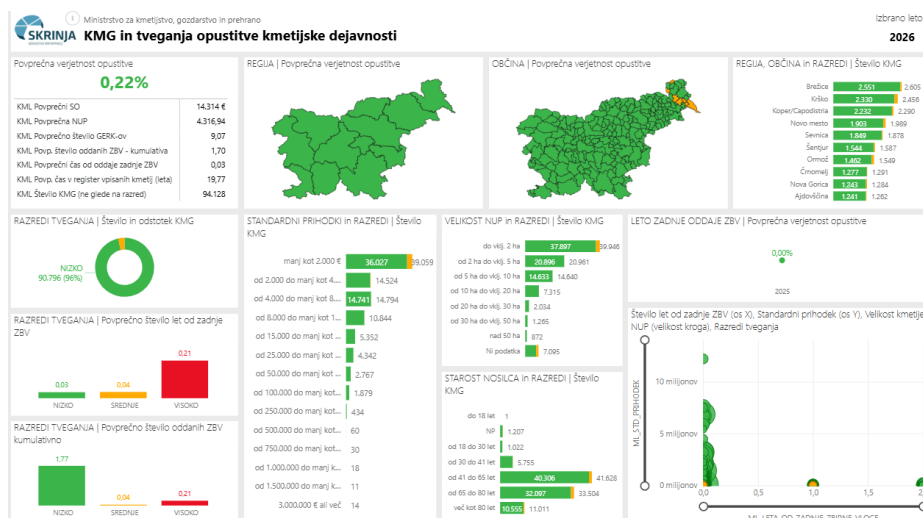
4.4. Uporabniški vidik in primer uporabe

Modul je zasnovan za strokovne uporabnike, ki za njegovo uporabo ne potrebujejo posebnega znanja s področja strojnega učenja. Uporabnik v analitičnem okolju izbere ustrezen napovedni scenarij, sistem pa samodejno izvede pripravo podatkov, uporabo produkcijskega modela in izračun napovedi.

Rezultati so prikazani v interaktivnih poročilih Microsoft Power BI skupaj z drugimi poslovnimi kazalniki. Uporabniki lahko spremljajo napovedane vrednosti, primerjajo dejanske in napovedane rezultate, prepoznavajo skupine z večjim tveganjem ter ocenjujejo možne učinke posameznih ukrepov. Napovedna analitika tako postane sestavni del vsakodnevnega odločanja in predstavlja podporo strokovnim uporabnikom, ne pa nadomestila njihove presoje.

Kot prvi produkcijski primer uporabe je bil razvit model za napovedovanje tveganja opuščanja kmetijskih gospodarstev. Model temelji na večletnih administrativnih podatkih ter izvedenih značilkah, ki opisujejo poslovanje in razvoj posameznega kmetijskega gospodarstva. Rezultat modela je ocena verjetnosti opustitve dejavnosti oziroma razvrstitev kmetijskih gospodarstev v razrede tveganja.

Primer potrjuje, da je mogoče z uporabo obstoječega podatkovnega skladišča in metod strojnega učenja razviti produkcijske napovedne modele, ki se brez sprememb osnovne arhitekture vključijo v platformo SKRINJA. Platforma tako presega tradicionalno opisno analitiko ter uporabnikom omogoča tudi napovedovanje prihodnjih dogodkov in pravočasno podporo odločanju na podlagi podatkov.



Slika 3: Primer nadzorne plošče z napovedno analitiko

4.5. Nadaljnji razvoj

Uvedba modula na področju Skupne kmetijske politike je potrdila ustreznost izbrane arhitekture ter pokazala, da je mogoče napovedne modele učinkovito vključiti v obstoječo poslovno-inteligenčno platformo kmetij brez sprememb uporabniškega okolja.

Ker je modul zasnovan generično, ga je mogoče uporabiti tudi na drugih področjih podatkovnih skladišč sistema SKRINJA. Nadaljnji razvoj bo usmerjen v širitev nabora napovednih modelov, uporabo dodatnih metod strojnega učenja, spremljanje kakovosti modelov skozi njihov življenjski cikel ter vključevanje novih podatkovnih virov. Za to pa bo treba okrepiti tudi infrastrukturo: trenutna rešitev deluje na običajnih procesorjih (CPU); ob večjem obsegu podatkov, več sočasnih modelih in zahtevnejših nalogah pa bodo potrebne tudi grafične kartice (GPU) — za hitrejšo učenje, ocenjevanje velikega števila zapisov in, odvisno od izbrane smeri, tudi za lokalno izvajanje jezikovnih modelov v državnem okolju.

V prihodnje načrtujemo vključitev tudi generativno analitiko (npr. vprašanja v običajnem jeziku o kazalnikih, kratki povzetki, pojasnila mer, iskanje po dokumentaciji). Točen način vključitve še preverjamo.

Dolgoročni cilj je vzpostaviti skupno storitev napovedne analitike v sistemu SKRINJA, ki bo organom državne uprave omogočala hitro uvedbo naprednih analitičnih rešitev brez razvoja lastne infrastrukture. S tem bo sistem SKRINJA poleg poslovne inteligence in odprtih podatkov zagotavljal tudi podporo podatkovno podprtemu napovedovanju prihodnjih trendov ter učinkovitejšemu oblikovanju javnih politik.

5 Zaključek

Prispevek predstavlja primer dobre prakse na področju poslovne inteligence (BI – Business Intelligence) in uporabo umetne inteligence za razvoj in nadgradnjo poslovno-inteligenčne platforme SKRINJA z dvema ključnima funkcionalnostma: modulom za odprto objavo podatkov (Odprta Skrinja) in modulom napovedne analitike. Predstavljena arhitektura omogoča enotno upravljanje podatkov, pripravo analitičnih vsebin, razvoj napovednih modelov ter njihovo uporabo v produkcijskem okolju brez vzpostavitve ločenih analitičnih rešitev. Uporabnost obeh funkcionalnosti je bila potrjena na konkretnih primerih oziroma podatkih oddanih javnih naročil

pri modulu Odprta Skrinja in podatkih Skupne kmetijske politike za napovedovanje opuščanja kmetijske dejavnosti v prihodnjih letih.

Predstavljena rešitev potrjuje, da lahko javna uprava z nadgradnjo obstoječih podatkovnih platform učinkovito poveže poslovno analitiko, odprte podatke in umetno inteligenco v enotno okolje za podporo odločanju. Modularna zasnova platforme omogoča nadaljnji razvoj novih analitičnih storitev in napovednih modelov ter predstavlja dobro osnovo za prihodnjo digitalno preobrazbo na področju podatkovnega upravljanja.

Literatura

- [1] Interno gradivo Ministrstva za notranje zadeve in javno upravo, Ljubljana 2026.
- [2] <https://nio.gov.si/products/skrinja%2B20%2Bsistem%2Bposlovne%2Banalitike?release=>, Portal NIO: SKRINJA – Poslovna inteligenca (BI) v državni upravi, obiskano 11. 7. 2026.
- [3] Marija Gorenc Novak & Dejan Velušček, 2016. "Prediction of stock price movement based on daily high prices," Quantitative Finance, Taylor & Francis Journals, vol. 16(5), pages 793-826, May.

Agentni sistem RAG za podporo zaposlenim v večjih podjetjih

Rok Hladin

Telekom Slovenije, d. d., Ljubljana, Slovenija
rok.hladin@telekom.si

Predstavljamo zasnovo, izvedbo in vrednotenje agentnega sistema RAG, ki zaposlenim v velikem telekomunikacijskem podjetju omogoča pogovorno iskanje informacij po bazi znanja, zgrajeni iz vsebin javne spletne strani. Sistem sestavljata cevovod za zajem podatkov, ki spletne vsebine pretvarja v vektorske vložitve v bazi Azure AI Search, in agentna aplikacija, zgrajena z ogradjem LlamaIndex, ki po vzorcu ReAct sama odloča o hibridnem iskanju ter vsak odgovor podpre z navedbo virov. Poleg oblačnega jezikovnega modela sistem prek prehoda Bifrost za umetno inteligenco uporablja lokalno gostujoče modele, pri katerih podatki ne zapuščajo infrastrukture podjetja. Na naboru 105 realnih vprašanj testnih uporabnikov delež uspešnih odgovorov po širši opredelitvi, ki šteje tudi delno pravilne in upravičeno zavrnjene odgovore, znaša 94,3 %, delež v celoti pravilnih pa 67,6 %; prvi del odgovora uporabnik običajno prejme po približno petih sekundah. Anketa med testnimi uporabniki potrjuje prihranek časa, kot glavna izziva pa izpostavlja pokritost baze znanja in zaupanje v odgovore.

Ključne besede:

agentni sistemi

s poizvedovanjem obogateno generiranje (RAG)

veliki jezikovni modeli

vektorsko iskanje

LlamaIndex

1 Uvod

Zaposleni v večjih podjetjih se pri vsakodnevnem delu opirajo na obsežen in nenehno spreminjajoč se nabor informacij o produktih, storitvah, postopkih in pogodbenih pogojih. V Telekomu Slovenije so te informacije razpršene po več virih, od javne spletne strani do internih navodil in dokumentov, iskanje po njih pa je ročno: zaposleni mora sam presoditi, kateri vir informacijo vsebuje, jo v njem poiskati in preveriti njeno ažurnost. Za nekatere skupine zaposlenih je tak način dela posebej obremenjujoč. Zaposleni v klicnem centru potrebuje točen odgovor med telefonskim pogovorom z uporabnikom, prodajalec na prodajnem mestu pa mora pogoje paketa preveriti, medtem ko uporabnik čaka.

Veliki jezikovni modeli [1] so pokazali, da je dostop do informacij prek pogovora v naravnem jeziku za uporabnike izjemno učinkovit, vendar njihova neposredna uporaba v poslovnem okolju trči ob dve omejitvi: nagnjenost k halucinacijam ter znanje, omejeno na učne podatke z določenim časovnim presekom. Obe omejitvi rešuje s poizvedovanjem obogateno generiranje (angl. *Retrieval-Augmented Generation*, RAG) [2], pri katerem model pred generiranjem odgovora prejme relevantne odlomke iz zunanje baze znanja. Agentni sistemi ta pristop nadgradijo tako, da jezikovnemu modelu prepustijo odločanje o poteku reševanja naloge, na primer o tem, ali in kolikokrat bo iskal po bazi znanja ter s kakšno poizvedbo [3].

V prispevku predstavljamo izkušnje z zasnovo, izvedbo in vrednotenjem takega sistema v realnem okolju Telekoma Slovenije. Tehnična izvedba pri tem ni sama sebi namen; osrednje vprašanje je, ali tak sistem zaposlenim pri delu dejansko pomaga. Prispevek zato poleg arhitekture in izvedbenih odločitev predstavlja tudi rezultate vrednotenja na realnih vprašanih testnih uporabnikov, meritve odzivnih časov in povratne informacije uporabnikov, ki so sistem uporabljali pri svojem delu.

2 Od klasičnega RAG do agentnega sistema

Tipičen sistem RAG deluje v treh korakih: dokumenti iz baze znanja se razdelijo na odlomke, pretvorijo v vektorske vložitve in shranijo v vektorsko podatkovno bazo; ob uporabnikovi poizvedbi se iz baze pridobijo pomensko najpodobnejši odlomki; ti se skupaj s poizvedbo vključijo v poziv jezikovnemu modelu, ki generira odgovor [2]. Odgovori so tako utemeljeni na dejanskih virih, bazo znanja pa je mogoče posodabljalati neodvisno od modela.

Klasični RAG sledi fiksному zaporedju: ena poizvedba, eno iskanje, eno generiranje, zato odpove pri sestavljenih vprašanih, ki zahtevajo informacije iz več virov, pri nejasnih vprašanih, ki bi jih bilo treba preoblikovati, in pri vprašanih, ki iskanja sploh ne potrebujejo. Agentni sistem te omejitve rešuje tako, da odločitve o poteku (ali, kolikokrat in s kakšno poizvedbo iskati) prepusti modelu; tok izvajanja s tem ni več vnaprej določen, temveč odvisen od sprotne odločitve modela. Med agentnimi vzorci je za obravnavani sistem osrednji vzorec ReAct [3], ki združuje sklepanje in izvajanje dejanj v iterativni zanki: model izmenično oblikuje sklepni korak, izvede dejanje (na primer klic orodja za iskanje) in prejme rezultat dejanja, cikel pa se ponavlja, dokler model ne oceni, da ima dovolj informacij za odgovor. Sorodna vzorca Self-RAG [4] in CRAG [5] dodajata samorefleksijo in mehanizme za izboljševanje kakovosti pridobljenega konteksta.

Za izvedbo smo primerjali štiri zrela odprtokodna ogrodja: LangChain [6], LlamaIndex [7], Haystack 2.x [8] in Semantic Kernel [9]. Šlo je za interno kvalitativno presojo po vnaprej določenih merilih: integracije s ciljnim storitvami (Azure OpenAI, lokalni modeli, vektorske shrambe), podpora zajemu in pripravi dokumentov, funkcionalnosti RAG (hibridno iskanje, ponovno razvrščanje, agentni delovni tokovi), vzdrževanost in licenca. Izbrali smo LlamaIndex, ker je obravnavani sistem predvsem podatkovno in dokumentno usmerjen: ogrodje je med primerjanimi možnostmi ponujalo najboljšo podporo za dokumentne bralnike, transformacije, metapodatke in indeksiranje, širok nabor konektorjev za ciljne shrambe ter podporo agentnim vzorcem. Njegova omejitev je, da na višjih ravneh abstrakcije včasih skrije del izvedbene logike, zato je za produkcijsko rabo pomembno dosledno sledenje izvajanja.

3 Zahteve in kontekst uporabe

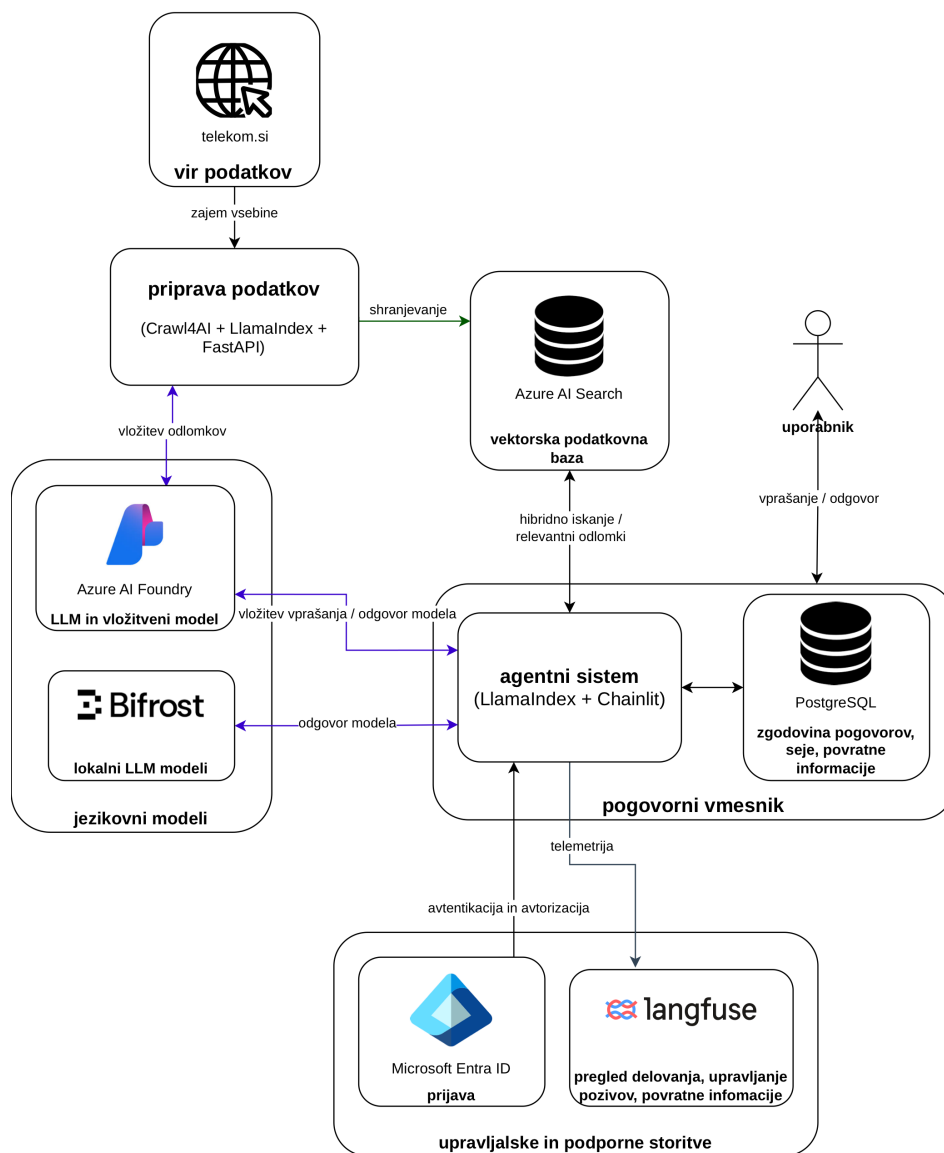
Primarne uporabniške skupine sistema so zaposleni v klicnem centru, prodajnih centrih in zalednih službah ter zunanji prodajni partnerji. Skupine se razlikujejo po obsegu potrebnih informacij in načinu dela: zaposleni v klicnem centru potrebujejo hitre odgovore med telefonskim pogovorom, zaledne službe pa podrobnejše postopkovne informacije.

Iz analize dela uporabniških skupin smo izpeljali ključne funkcionalne zahteve: odgovarjanje na vprašanja v slovenskem jeziku na podlagi vsebin iz baz znanja, obvezno navedbo vira v vsakem odgovoru, prijavo prek Microsoft Entra ID, beleženje zgodovine pogovorov, telemetrijo za spremljanje delovanja, pretočno prikazovanje odgovora med generiranjem ter jasno sporočanje omejitev, kadar sistem relevantnih vsebin ne najde. Zadnji dve zahtevi skupaj gradita zaupanje v sistem: odgovori so preverljivi prek virov, ob manjkajočih informacijah pa sistem omejitev prizna, namesto da bi vrnil nezanesljiv odgovor. Med nefunkcionalnimi zahtevami so podpora do 50 hkratnih in približno 200 dnevni uporabnikov ter hramba vseh podatkov v Evropski uniji.

Primarni vir podatkov v prvi fazi je javno spletno mesto telekom.si s približno 3.000 stranmi, ki vsebuje informacije o produktih, storitvah, cenah in pogojih. Kot razširitev je načrtovana vključitev interne baze znanja Telekoma Slovenije, ki temelji na istem sistemu za upravljanje vsebine (angl. *content management system*, CMS) in zato dopušča enak pristop k zajemu; baza v času pisanja še ni pripravljena za uporabo, dostop do njenih vsebin pa bo pogojen z uporabniško skupino. Kot merilo uspešnosti sistema smo opredelili delež uspešno odgovorjenih vprašanj in zadovoljstvo uporabnikov.

4 Arhitektura in implementacija

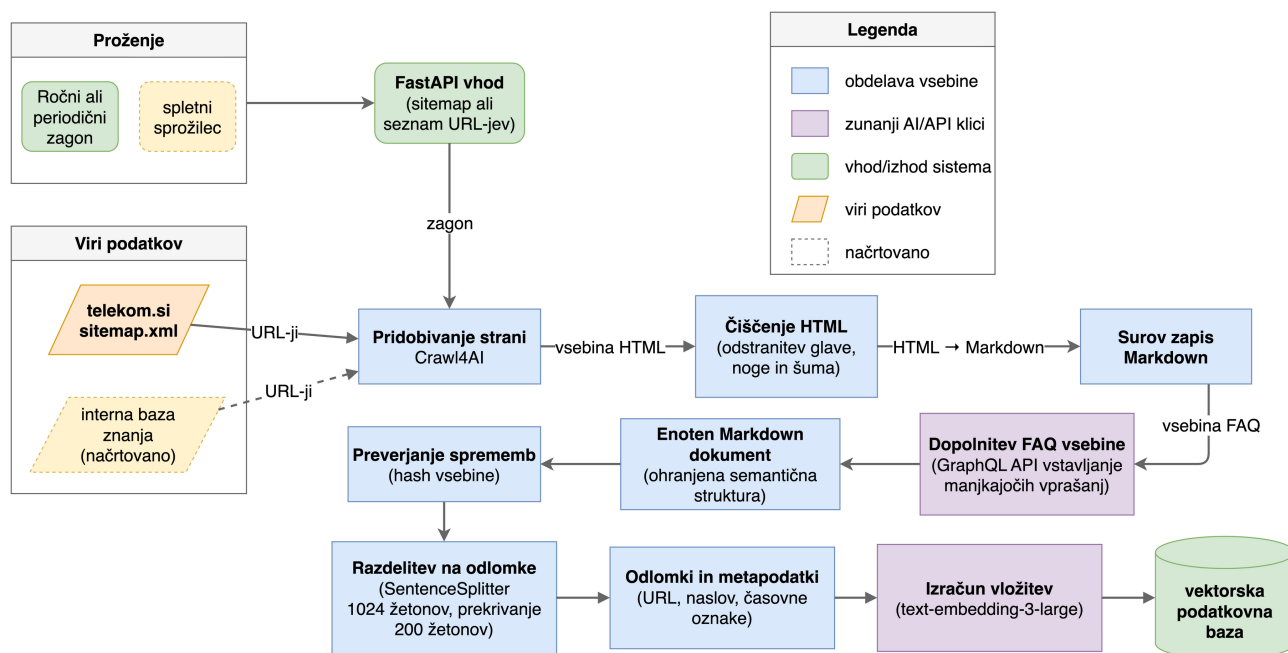
Arhitektura sistema je razdeljena na dva podsistema, ki sta povezana prek vektorske baze: cevovod za zajem podatkov, ki spletne vsebine pridobiva, obdeluje in zapisuje v vektorsko bazo, ter agentno pogovorno aplikacijo, ki ob uporabnikovem vprašanju išče po vektorski bazi in z jezikovnim modelom oblikuje odgovor. Taka delitev omogoča, da se osveževanje znanja in odgovarjanje izvajata neodvisno in z različnima frekvencama. Vse lastne komponente (cevovod, agentna aplikacija in baza PostgreSQL) tečejo kot ločeni »podi« na platformi OpenShift znotraj infrastrukture podjetja; Azure AI Search, Azure AI Foundry, Microsoft Entra ID in Langfuse so zunanje storitve. Celotno arhitekturo prikazuje slika 1.



Slika 1: Arhitektura agentnega sistema za podporo zaposlenim.

4.1. Cevovod za zajem in obdelavo podatkov

Cevovod za zajem podatkov je samostojna aplikacija, ki spletne vsebine pretvarja v obliko, primerno za semantično iskanje; njegov potek prikazuje slika 2. Seznam URL-jev pridobi iz datoteke sitemap.xml, ki v praksi predstavlja kanonični seznam objavljenih strani. Strani HTML prenese in pretvori v zapis Markdown s knjižnico Crawl4AI [10], ki je namenjena prav pripravi spletnih vsebin za jezikovne modele. Zapis Markdown ohranja semantično strukturo dokumenta (naslove, sezname, povezave) in je hkrati dovolj zgoščen za vključitev v kontekst jezikovnega modela.



Slika 2: Cevovod za zajem in obdelavo podatkov.

Surov zapis Markdown ni neposredno primeren za vektorsko iskanje, zato sledi korak čiščenja in obogatitve. Glavo in nogo strani, ki vsebujeta predvsem navigacijske elemente in se ponavljata na vseh straneh, odstranimo, saj v vektorski bazi povzročata šum. Posebno obravnavo so zahtevali razdelki s pogosto zastavljenimi vprašanji: ker so v sistemu CMS vprašanja in odgovori shranjeni v ločenih strukturah, pretvorba v Markdown zajame le odgovore, vprašanja pa izgubi. Manjkajoča vprašanja zato pridobimo neposredno prek vmesnika GraphQL, ki ga izpostavlja sistem CMS, in jih programsko vstavimo nad pripadajoči odgovor, s čimer par vprašanje-odgovor ostane povezan v istem odlomku in ga je mogoče najti prek poizvedb v naravnem jeziku.

Pripravljene dokumente razdelimo na odlomke z razdelilnikom *SentenceSplitter* iz ogrodja LlamaIndex z velikostjo 1024 žetonov in prekrivanjem 200 žetonov; meje odlomkov so poravnane s stavčnimi mejami. Velikost odlomka določa kompromis med natančnostjo iskanja in celovitostjo konteksta: pri manjših odlomkih se vsebinsko zaokrožene enote, kot so navodila po korakih, razbijejo na več delov, pri večjih pa vložitev postane manj razločevalna. Izbrana velikost praviloma v celoti zajame posamezen razdelek o storitvi, paketu ali postopku, vključno s pari vprašanje-odgovor. Vložitve odlomkov izračunamo z modelom *text-embedding-3-large* prek storitve Azure AI Foundry, ki podpira tudi neangleška besedila, in jih z operacijo *upsert* zapišemo v indeks Azure AI Search, kar omogoča ponovne zagone cevovoda brez ustvarjanja dvojnikov.

Cevovod je zapakiran v aplikacijo FastAPI in izpostavlja dve operaciji: celovit zagon nad celotno datoteko sitemap.xml in zagon nad eksplicitnim seznamom URL-jev za ciljno osvežitev. V trenutni postavitvi se proži ročno, ista zasnova pa omogoča tudi dogodkovno proženje, pri katerem sistem CMS ob spremembi vsebine prek spletnega sprožilca (angl. *webhook*) pokliče operacijo za ciljno osvežitev, tako da vektorska baza skoraj sproti sledi spremembam vsebin.

4.2. Vektorska baza in hibridno iskanje

Kot vektorsko bazo uporabljamo Azure AI Search, ki je del ekosistema Azure, v katerem že deluje preostali del sistema, in podpira hibridno iskanje brez zunanjih komponent. Iskanje kombinira rezultate polnobesedilnega iskanja BM25 [11] in vektorskega iskanja z algoritmom RRF (angl. *reciprocal rank fusion*) [12]. Kombinacija je za obravnavano domeno pomembna, ker zaposleni pogosto sprašujejo po specifičnih imenih tarifnih paketov, kraticah in kodah storitev, kjer je odločilno natančno besedno ujemanje, vektorska komponenta pa pokrije parafrazirana vprašanja. Semantičnega ponovnega razvrščanja zadetkov ne uporabljamo, ker njegov prispevek pri

trenutnem obsegu podatkov ne odtehta dodatne zakasnitve in stroškov; ostaja kandidat za nadaljnje izboljšave ob bistveni širitvi vsebin.

4.3. Sloj jezikovnih modelov

Agentna aplikacija do jezikovnih modelov ne dostopa neposredno, temveč prek skupnega sloja, ki povezuje dva ponudnika: oblačno storitev Azure AI Foundry, prek katere je na voljo GPT-5, in lokalno gostujoče odprtokodne modele, ki se izvajajo znotraj infrastrukture podjetja. Slednje izpostavljamo prek prehoda Bifrost za umetno inteligenco [13], ki več lokalnih modelov združi pod enotnim vmesnikom, združljivim z vmesnikom OpenAI API, tako da je modele mogoče zamenjati brez posegov v aplikacijo. V času pisanja so bili testirani trije modeli: lokalni model Gemma 4 31B, ki je privzeti, DeepSeek V4 Flash ter GPT-5; uporabnik lahko model izbere za posamezni pogovor.

Uvedba lokalnih modelov ima dva poglobitna razloga. Prvi je zasebnost podatkov: pri lokalno gostujočih modelih vsebina vprašanj in pridobljenih odlomkov ne zapusti infrastrukture podjetja, kar je za sistem, ki bo obdeloval interne podatke, pomembna prednost. Drugi je zmanjšanje stroškov: lokalna inferenca odpravi plačilo po številu žetonov, kar je pri predvideni dnevni uporabi gospodarno. GPT-5 ohranjamo kot najzmogljivejšo možnost za primere, ko razumevanje jezika in sledenje navodilom odtehtata omenjeni prednosti.

4.4. Agentna aplikacija

Agentna aplikacija, zgrajena na ogrodju LlamaIndex, podpira dva načina odgovarjanja, med katerima uporabnik izbira v vmesniku. Oba uporabljata isto orodje za hibridno iskanje in isti jezikovni model, razlikujeta pa se v tem, koliko nadzora nad iskanjem prepustita modelu. Hitri način RAG izvede natanko en klic orodja s poizvedbo, ki je enaka uporabnikovem vprašanju, in iz pridobljenih odlomkov sintetizira odgovor; ker je zaporedje korakov vnaprej določeno, je hitrejši in primeren za kratka, neposredna vprašanja. Agentni način deluje po vzorcu ReAct [3]: model se sam odloča, ali in kolikokrat bo orodje uporabil ter s kakšno poizvedbo. Pri sestavljenih vprašanjih lahko izvede več zaporednih poizvedb ali neustrezno poizvedbo preoblikuje, pri vprašanjih, ki nadaljujejo prejšnji pogovor, pa odgovori brez iskanja. Privzeti način delovanja je agentni.

Sistemiški poziv usmerja agenta k podajanju odgovorov v slovenščini, podpiranju vsakega odgovora z viri in poštenemu priznanju, kadar informacij v bazi znanja ni. Poziv ni vgrajen v izvorno kodo, temveč je upravljan kot različica v orodju Langfuse, kar omogoča iterativno izboljševanje navodil agenta brez ponovne namestitve aplikacije. Sistem podpira več pozivov, prilagojenih vlogi uporabnika: poleg splošnega asistenta je na voljo poziv za tehnike na terenu, ki odgovore oblikuje jedrnato in postopkovno. Agent vire navaja na dva načina: pri krajših odgovorih kot povezave neposredno v besedilu, pri daljših pa kot številčne citate, ki jih vmesnik preslika v ločen razdelek z viri pod odgovorom; obe obliki zagotavljata sledljivost vsake navedbe do izvirnega naslova URL.

4.5. Pogovorni vmesnik in podporne storitve

Za pogovorni vmesnik smo izbrali ogrodje Chainlit [14], ki je odprtokodno, neposredno združljivo z ogrodjem LlamaIndex in ponuja izkušnjo v slogu pogovornih asistentov, na katero so uporabniki navajeni iz drugih orodij. Vmesnik omogoča pregled in nadaljevanje preteklih pogovorov, nad vsakim odgovorom pa prikaže korake agenta s klici orodja za iskanje in uporabljenimi poizvedbami, kar razkriva, na podlagi česa je bil odgovor oblikovan. V nastavitvah pogovora uporabnik izbere jezikovni model in sistemiški poziv glede na svojo vlogo, nad pogovorom pa preklaplja med načinoma odgovarjanja.

Zgodovina pogovorov, seje in povratne informacije uporabnikov se hranijo v bazi PostgreSQL. Avtentikacija poteka prek storitve Microsoft Entra ID, obstoječega sistema za enotno prijavo v podjetju; skupinsko članstvo uporabljamo za avtorizacijo na ravni aplikacije, dostop pa je omejen na člane namenske skupine. Sistem omogoča nastavljivo beleženje tehničnih sledi izvajanja v orodju Langfuse [15], ki ga je mogoče glede na okolje in namen uporabe vključiti ali izključiti. V fazi razvoja in vrednotenja se beležijo izključno tehnične vsebine, potrebne za

analizo delovanja in kakovosti sistema; sledi ne vsebujejo podatkov naročnikov ali drugih osebnih podatkov ter se uporabljajo zgolj za odkrivanje napak in nadaljnje izboljševanje sistema.

Izbor uporabljenih platform, ogrodij in podpornih komponent je temeljil predvsem na njihovi razširjenosti, podpori, združljivosti s ciljnim okoljem in primernosti za hitro prototipiranje. Z izjemo ogrodij RAG tehnološke alternative niso bile predmet poglobljene primerjalne analize, zato se lahko posamezne komponente pred produkcijsko uvedbo ponovno presodijo in po potrebi zamenjajo.

5 Vrednotenje

5.1. Metodologija

Vrednotenje ni zasnovano kot primerjalna študija algoritmov, temveč kot pragmatična presoja uporabnosti na podlagi realnih vprašanj in izkušenj dejanskih uporabnikov. Opira se na štiri vire podatkov: ročno vrednotenje odgovorov na naboru realnih vprašanj, meritve odzivnih časov iz sledi izvajanja v Langfuse, neposredne povratne informacije uporabnikov in anketo med testnimi uporabniki.

Sistem je pred vrednotenjem približno dva tedna uporabljala skupina sedmih testnih uporabnikov, večinoma iz klicnega centra, njihova vprašanja pa so se beležila v Langfuse. Odgovorov iz tega obdobja za vrednotenje nismo uporabili neposredno, saj pogoji uporabe niso bili nadzorovani: uporabniki so pogosto mešali vsebinsko nepovezana vprašanja v istem pogovoru ali za sestavljena vprašanja izbrali hitri način RAG. Zabeležena vprašanja smo zato ponovno izvedli v nadzorovanih pogojih: vsako v novem pogovoru, brez predhodnega konteksta, v agentnem načinu s privzetim modelom in splošnim sistemskim pozivom. Iz izvoza smo odstranili podvojena in nevsebinska vprašanja, navezujoča se vprašanja preoblikovali v samozadostno obliko, vprašanja brez odgovora v bazi znanja pa namenoma ohranili, saj preverjajo, ali sistem pomanjkanje informacij prizna. Končni nabor obsega 105 vprašanj s področij mobilnih in fiksnih paketov, gostovanja v tujini, postopkov, opreme in storitev.

Vsak odgovor smo uvrstili v eno od petih kategorij: pravilen, delno pravilen, nepravilen, upravičena zavrnitev (sistem sporoči, da nima dovolj informacij, in odgovora v bazi znanja dejansko ni) in neupravičena zavrnitev (sistem odgovor zavrne, čeprav informacija v bazi obstaja). Upravičena zavrnitev je zaželeno vedenje, saj je priznanje omejitve za uporabnika koristnejše od nezanesljivega odgovora. Delež uspešnih odgovorov opredelimo kot delež vprašanj s pravilnim, delno pravilnim ali upravičeno zavrnjenim odgovorom; ob njem poročamo še strožji delež, ki upošteva samo pravilne odgovore. Pravilnost smo presojali glede na dejansko vsebino strani telekom.si v času vrednotenja, ne zgolj glede na vir, ki ga je sistem navedel, s čimer se izognemo krožnosti pri napačno pridobljenih straneh. Odgovore je vrednotil en ocenjevalec, avtor sistema; omejitve delno blažijo vsebinske pripombe uporabnikov, ki so bile pri presoji uporabljene kot dodatna referenca.

5.2. Uspešnost odgovorov

Porazdelitev ocen prikazuje tabela 1. Od 105 vprašanj je sistem na 71 odgovoril pravilno, na 14 delno pravilno, 14 vprašanj je upravičeno zavrnil, 5 neupravičeno, en odgovor pa je bil nepravilen. Delež uspešnih odgovorov znaša 94,3 %, po strožji opredelitvi pa 67,6 %.

Struktura neuspešnih odgovorov je povednejša od samih deležev. Očitno napačen odgovor, torej tak, ki bi uporabnika zavedel z napačno informacijo, je bil en sam: na vprašanje o cenovnem območju gostovanja za Karibe je sistem samozavestno in z viri opisal zakupe prenosa podatkov, torej odgovoril na drugo vprašanje. Primer ponazarja najnevarnejšo obliko napake: odgovor je tekoč, strukturiran in podprt z viri, zato deluje verodostojno. Delno pravilni odgovori so praviloma nepopolni, ne napačni, in manjkajoče informacije pošteno označijo. Neupravičene zavrnitve nastanejo, kadar korak iskanja ne pridobi relevantne strani, praviloma pri kratkih poizvedbah z imeni ponudb, ki se prekrivajo z drugimi vsebinami; agent je pri tem ravnal skladno z navodili, saj si odgovora ni izmislil, temveč je ponudil sorodne zadetke in prosil za natančnejši opis. Upravičene zavrnitve

večinoma izvirajo iz vprašanj, ki zahtevajo osebne podatke strank ali interne vsebine, torej informacije, ki jih trenutna baza znanja načrtno ne vsebuje; sistem je omejitev priznal in uporabnika usmeril na ustrezne portale.

Tabela 1: Rezultati ročnega vrednotenja odgovorov na naboru 105 vprašanj.

Ocena	Število vprašanj	Delež [%]
pravilen	71	67,6
delno pravilen	14	13,3
nepravilen	1	1,0
upravičena zavrnitev	14	13,3
neupravičena zavrnitev	5	4,8
delež uspešnih odgovorov	99	94,3

5.3. Odzivni časi

Odzivne čase smo izmerili iz sledi izvajanja v Langfuse za vseh 105 vprašanj; rezultate povzema tabela 2. Poročamo mediano in razpon med 5. in 95. percentilom, saj je porazdelitev desno asimetrična. Ker se odgovor uporabniku prikazuje pretočno, merimo tudi čas do prvega žetona odgovora (angl. *time to first token*) kot mero zaznane odzivnosti.

Tabela 2: Odzivni časi po korakih izvajanja agenta za 105 vprašanj iz nabora za vrednotenje.
Meji razpona sta 5. in 95. percentil.

Korak	Mediana [s]	Razpon [s]
sklepalni koraki (odločitev in poizvedba)	0,6	0,5–0,9
iskanje po vektorski bazi	0,9	0,7–1,2
končno oblikovanje odgovora	11,7	4,6–21,4
skupaj	13,8	6,0–23,0
čas do prvega žetona odgovora	5,3	4,3–6,2

Skupni čas skoraj v celoti določa generiranje besedila v jezikovnem modelu; odločanje agenta in iskanje po vektorski bazi skupaj tipično zahtevata približno 1,5 sekunde. Agentna zasnova in poizvedovanje po bazi znanja k skupnemu odzivnemu času prispevata razmeroma majhen delež, zato bi se morebitne optimizacije morale osredotočiti na generiranje, na primer z izbiro hitrejšega modela ali s krajšimi odgovori. Agent je pri 97 vprašanjih orodje za iskanje klical enkrat, pri treh dvakrat, pri petih pa je odgovoril brez iskanja, praviloma tedaj, ko je vprašanje zahtevalo podatke, ki jih baza znanja načrtno ne vsebuje; to potrjuje, da število iskanj prilagaja vprašanju. Z vidika uporabnika je pomembnejši čas do prvega žetona: uporabnik začne odgovor brati po približno petih sekundah, celoten odgovor pa je tipično izpisan v slabih štirinajstih sekundah. Rezultati nakazujejo, da je taka odzivnost sprejemljiva tudi za delo v klicnem centru; skladno s tem je pet od sedmih anketirancev poročalo o krajšem ocenjenem času iskanja informacij kot pred uvedbo sistema, nihče pa o daljšem.

5.4. Povratne informacije uporabnikov

Splošno oceno uporabnosti smo zajeli z anonimno anketo med testnimi uporabniki, ki jo je do zaključka pisanja izpolnilo sedem uporabnikov. Zaradi majhnega vzorca rezultatov ne posplošujemo; poročamo porazdelitve in mediane. Anketiranci sistem uporabljajo za preverjanje informacij o paketih, zakupih, cenikih, postopkih in pogojih, torej natanko za naloge, ki jih je zajem zahtev predvidel. Pred uvedbo sistema so odgovore iskali po več virih hkrati (spletna stran, interni portal, ceniki, elektronska pošta, sodelavci); čas iskanja odgovora je šest anketirancev ocenilo na 1 do 5 minut, eden na 5 do 15 minut, s sistemom pa pet anketirancev odgovor najde v manj kot minuti, dva pa v 1 do 5 minutah. Pri petih anketirancih se je ocenjeni čas iskanja torej skrajšal, pri nobenem pa podaljšal.

Tabela 3: Porazdelitev odgovorov na trditve ankete
(1 pomeni sploh se ne strinjam, 5 popolnoma se strinjam), N = 7.

Trditev	1	2	3	4	5	Mediana
Sistem mi je pogosto vrnil uporaben odgovor.	0	0	3	4	0	4,0
Navedeni viri so podpirali odgovor.	0	1	1	4	1	4,0
Odgovorom sistema zaupam.	1	2	3	1	0	3,0
Sistem mi prihrani čas pri iskanju informacij.	0	0	2	4	1	4,0
Sistem bi priporočil sodelavcem.	0	1	1	4	1	4,0

Kot prikazuje tabela 3, so uporabnost odgovorov, ustreznost virov, prihranek časa in pripravljenost priporočiti sistem ocenjeni zmerno pozitivno z mediano 4,0, najnižje pa je ocenjeno zaupanje v odgovore, z mediano 3,0 in s šestimi od sedmih ocen na sredini lestvice ali pod njo. Ta razkorak je vsebinsko pomenljiv in skladen s strukturo napak iz tabele 1: čeprav so vsebinske napake redke, so uporabniki med testiranjem naleteli nanje in na vsebinske vrzeli, vsaka taka izkušnja pa zahteva ponovno preverjanje tudi sicer pravih odgovorov. Odgovori na odprta vprašanja razkorak pojasnijo: kot prednosti uporabniki navajajo hitrost, preglednost odgovorov in združevanje informacij, kot pomanjkljivosti pa zahtevnejša sestavljena vprašanja, nepopolne sezname možnosti, zastarele podatke in vsebinske vrzeli. Med zelenimi izboljšavami prevladujejo interne vsebine, ki jih trenutna baza znanja ne pokriva. Takšno stališče povzema odgovor ene od anketirank: „AI chat kot tak je vrhunska rešitev, ker prideš zelo hitro do odgovora, dodelati je treba samo bazo, iz katere črpa informacije.“

6 Zaključek

V prispevku smo predstavili zasnovo, izvedbo in vrednotenje agentnega sistema RAG za podporo zaposlenim v Telekomu Slovenije. Sistem sestavljata cevovod za zajem podatkov, ki vsebine javne spletne strani pretvarja v vektorske vložitve v bazi Azure AI Search, in agentna aplikacija na ogrodju LlamaIndex, ki po vzorcu ReAct sama odloča o hibridnem iskanju in vsak odgovor podpre z viri. Poleg oblačnega modela sistem prek prehoda Bifrost uporablja lokalno gostujoče jezikovne modele; pri njihovi uporabi vsebina vprašanj in pridobljenih odlomkov ne zapusti infrastrukture podjetja, na kateri tečejo tudi vse lastne komponente sistema.

Vrednotenje na 105 realnih vprašanjih testnih uporabnikov je po vnaprej določeni širši opredelitvi uspešnosti pokazalo 94,3 % uspešnih odgovorov, po strožji opredelitvi, ki upošteva samo v celoti pravilne odgovore, pa 67,6 %. Očitno napačen odgovor je bil en sam. Čas do prvega žetona je znašal približno pet sekund, ocena uporabnikov pa je bila zmerno pozitivna, pri čemer je bil glavni zadržek zaupanje v odgovore. Izkušnje potrjujejo tri praktične ugotovitve, prenosljive na sorodne sisteme: da agentno odločanje in iskanje k odzivnemu času prispevata razmeroma majhen delež v primerjavi z generiranjem, da je za sprejem sistema pomembnejša pokritost baze znanja

kot izpopolnjevanje samega odgovarjanja ter da uporabniki brez uvajanja funkcionalnosti, kot je izbira načina odgovarjanja, ne uporabljajo optimalno. Smeri nadaljnega dela izhajajo neposredno iz teh ugotovitev: vključitev internih virov, predvsem Telekomove interne baze znanja, skupaj z nadzorom dostopa po uporabniških skupinah, dogodkovno proženje zajema prek spletnega sročilca sistema CMS, sistematična primerjava razpoložljivih jezikovnih modelov in samodejno vrednotenje na pripravljenem naboru vprašanj ter vključitev v okolje Microsoft Teams.

Literatura

- [1] BROWN Tom B. in drugi, "Language Models are Few-Shot Learners", Advances in Neural Information Processing Systems, 2020.
- [2] LEWIS Patrick in drugi, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", Advances in Neural Information Processing Systems, 2020.
- [3] YAO Shunyu, ZHAO Jeffrey, YU Dian, DU Nan, SHAFRAN Izhak, NARASIMHAN Karthik, CAO Yuan, "ReAct: Synergizing Reasoning and Acting in Language Models", International Conference on Learning Representations (ICLR), 2023.
- [4] ASAI Akari, WU Zeqiu, WANG Yizhong, SIL Avirup, HAJISHIRZI Hannaneh, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection", International Conference on Learning Representations (ICLR), 2024.
- [5] YAN Shi-Qi, GU Jia-Chen, ZHU Yun, LING Zhen-Hua, "Corrective Retrieval Augmented Generation", arXiv:2401.15884, 2024.
- [6] LangChain Documentation, <https://docs.langchain.com/>, obiskano 12. 7. 2026.
- [7] LlamaIndex Documentation, <https://developers.llamaindex.ai/python/framework/>, obiskano 12. 7. 2026.
- [8] Haystack Documentation, <https://docs.haystack.deepset.ai/>, obiskano 12. 7. 2026.
- [9] Semantic Kernel Documentation, <https://learn.microsoft.com/en-us/semantic-kernel/>, obiskano 12. 7. 2026.
- [10] Crawl4AI: Open-Source LLM-Friendly Web Crawler and Scraper, <https://github.com/unclecode/crawl4ai>, obiskano 12. 7. 2026.
- [11] ROBERTSON Stephen, ZARAGOZA Hugo, "The Probabilistic Relevance Framework: BM25 and Beyond", Foundations and Trends in Information Retrieval, letnik 3, številka 4, 2009, str. 333–389.
- [12] CORMACK Gordon V., CLARKE Charles L. A., BÜTTCHER Stefan, "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods", Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2009.
- [13] Bifrost: Open-Source LLM Gateway, <https://github.com/maximhq/bifrost>, obiskano 12. 7. 2026.
- [14] Chainlit: Build Conversational AI Applications, <https://docs.chainlit.io/>, obiskano 12. 7. 2026.
- [15] Langfuse: Open-Source LLM Engineering Platform, <https://langfuse.com/docs>, obiskano 12. 7. 2026.

Ko uporabnik ni več samo človek: od priporočil do avtonomnih agentov v poslovnih aplikacijah

Boštjan Grašič,¹ Gal Badrov,¹ Hanan Mešić,¹ Saša Grašič²

¹ Alloxy, informacijske rešitve, d.o.o., Maribor, Slovenija

bostjan.grasic@alloxy.io, gal.badrov@alloxy.io, hanan.mesic@alloxy.io

² Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija

sasa.grasic@um.si

Umetna inteligenca (UI) postaja nepogrešljiv sestavni del poslovnih aplikacij, vendar njena vgradnja v poslovno-kritične sisteme zahteva bistveno drugačen pristop od preprostega posredovanja podatkov velikim jezikovnim modelom. Prispevek na podlagi praktičnih izkušenj z razvojem platforme za upravljanje virov v projektno usmerjenih storitvenih podjetjih predstavi sistematičen pristop k vgradnji UI. Najprej opredelimo spekter vključevanja UI v poslovne procese — od priporočil, ki jih uporabnik potrjuje, prek pol-avtonomnih agentov do avtonomnih agentov — in za vsako stopnjo avtonomije določimo ustrezne mehanizme upravljanja ter pokažemo, kje se ti pokrivajo z zahtevami Akta EU o umetni inteligenci (EU AI Act). Nadalje naslovimo tezo, da bomo informacijske sisteme morali graditi tudi za agente UI, ki bodo do njih dostopali programske, na primer prek protokola MCP, in da morajo biti varovalke vgrajene v sama orodja, ne le v agente. Sledi praktična primerjava sedmih ogrodiv za gradnjo agentov in utemeljitev naše izbire, nato pa arhitektura agentne plasti, kot smo jo zgradili v platformi Alloxy: dvoplastna avtorizacija orodij, strukturiran dolgoročni spomin, vzorec človeške potrditve pred dejanji s posledicami, deterministično usmerjanje modelov, opazljivost v lastni infrastrukturi in preverljivo kaskadno brisanje. Zaključimo s tremi konkretnimi agenti in prenosljivimi spoznanji za razvijalce in arhitekte poslovnih rešitev.

Ključne besede:

agenti umetne inteligence

poslovne aplikacije

EU AI Act

Model Context Protocol

upravljanje virov

1 Uvod

Umetna inteligenca v poslovnih aplikacijah ni več zgolj funkcija, ki jo dodamo obstoječemu sistemu — kot so bili to pred nekaj leti pametno iskanje, klasifikacija dokumentov ali priporočilni algoritem. Postaja akter, ki v imenu uporabnika ali samostojno izvaja korake v poslovnem procesu: pridobiva podatke, jih povezuje, pripravlja predloge in — če mu to dovolimo — izvaja spremembe. Ta premik od »UI kot funkcije« k »UI kot akterju« odpira nov razred vprašanj, ki jih klasična arhitektura poslovnih aplikacij ne naslavlja: kdo je odgovoren za akcijo, ki jo je sprožil agent, kako jo razložimo uporabniku, kako jo omejimo, kako jo revidiramo — in kako vse to uskladimo z regulatornim okvirom, ki je medtem stopil v veljavo.

Vgradnja UI v poslovno-kritične sisteme se zato bistveno razlikuje od preprostega posredovanja podatkov velikemu jezikovnemu modelu (LLM). Potreben je sistematičen arhitekturni pristop, ne le »prompt in izpis«. Prav tako se je izkazalo, da vgradnje ni mogoče obravnavati kot enoten projekt: različni uporabniški scenariji znotraj iste aplikacije imajo bistveno različno tveganje in različne posledice napačne odločitve.

Prispevek je zavestno praktičen, ne le konceptualen: poleg ogrođja za odločanje o stopnji avtonomije predstavimo tudi konkretno primerjavo ogrođij za gradnjo agentov UI, utemeljitev naše izbire in arhitekturo, ki smo jo na tej podlagi zgradili. Izkušnje izhajajo iz razvoja platforme Alloxy — rešitve za upravljanje in načrtovanje virov v projektno usmerjenih storitvenih podjetjih z 30 do 250 zaposlenimi. Tehnološko gre za večnajemniško (multi-tenant) rešitev SaaS na skladu TypeScript in Node.js s podatkovno bazo PostgreSQL, gostovano v EU; prav večnajemniškost, EU rezidenčnost podatkov in narava domene so najbolj oblikovali naše arhitekturne odločitve.

Prispevek je členjen takole. Drugo poglavje predstavi spekter avtonomije, mehanizme upravljanja za vsako stopnjo in njihovo prekrivanje z zahtevami Akta EU o umetni inteligenci, iz česar izpeljemo tehnične zahteve. Tretje poglavje obravnava agenta kot nov tip uporabnika sistema, četrto prinaša primerjavo ogrođij in utemeljitev naše izbire, peto arhitekturo agentne plasti v platformi Alloxy, šesto tri konkretne agente, sedmo pa strne prenosljiva spoznanja.

2 Spekter vključevanja UI v poslovne procese

2.1. Zakaj iterativni pristop po scenarijih

Vgradnje UI se ni mogoče lotiti kot enega samega projekta s fiksnim obsegom. Priporočen pristop je, da najprej identificiramo posamezne uporabniške scenarije — v platformi Alloxy na primer dodelitev vira na projekt, poizvedovanje prostih kapacitet in potrjevanje zahtevkov — in za vsakega posebej opredelimo stopnjo avtonomije, varovalke ter regulatorno klasifikacijo. Prednost je hitrejšo učenje na scenarijih z nizkim tveganjem, preden UI vgradimo v procese, ki vplivajo na ljudi ali na denar; kot pokažemo v razdelku 2.3, se prav ob tej ločnici odloča tudi regulatorna klasifikacija.

2.2. Tri stopnje avtonomije

Spekter vključevanja UI v poslovne procese smo razdelili na tri stopnje.

UI kot razširitev človeških sposobnosti. Sistem analizira podatke in poda priporočilo, uporabnik pa ga eksplicitno potrdi, preden se karkoli zgodi — sistem na primer predlaga najprimernejšega sodelavca za projekt, vodja oddelka pa predlog potrdi ali zavrne. Tveganje napake je omejeno, ker človek ostane zadnja instanca odločanja.

Pol-avtonomni agenti. Agent samostojno pridobiva podatke iz več virov, jih poveže in pripravi konkreten predlog za akcijo, ne le informacije. Uporabnik odločitve ne gradi od začetka, temveč pregleda in odobri, zavrne ali korigira že pripravljen predlog — agent na primer zazna prosto kapaciteto in konflikt v razporedu ter predlaga prerazporeditev.

Avtonomni agenti. Agent v ozadju izvaja optimizacijo brez neposredne udeležbe uporabnika pri vsakem koraku in mu predstavi le končni, že izračunan predlog. Vključenost človeka (angl. human-in-the-loop) se premakne z »vsak korak« na »končna odobritev«, pri manj kritičnih odločitvah pa lahko celo na naknadni pregled.

Stopnje niso strogo ločene kategorije, temveč kontinuum: isti scenarij lahko sčasoma premikamo po spektru, ko zaupanje v sistem raste. Pomembno razlikovanje, ki ga razvijemo v nadaljevanju, pa je, da stopnja avtonomije in regulatorna teža scenarija nista isto — nizka avtonomija ne pomeni samodejno nizkega regulatornega tveganja.

2.3. Mehanizmi upravljanja in njihovo prekrivanje z EU aktom o umetni inteligenci

Za vsako stopnjo avtonomije je treba določiti ustrezne mehanizme upravljanja. Ti niso dodatek k arhitekturi, temveč njen sestavni del — in v veliki meri hkrati tudi pravna zahteva.

Sledljivost in revizijska sled. Beležiti je treba, kateri podatki so bili uporabljeni, kateri model ali pravilo je generiralo predlog, kdo (človek ali agent) je akcijo sprožil in kdo jo je potrdil. Akt o umetni inteligenci [1] to za visoko tvegane sisteme zahteva kot samodejno beleženje skozi celoten življenjski cikel sistema (člen 12).

Nadzor nad akcijami in vključevanje človeka. Sem sodijo eksplicitne meje delovanja, mehanizmi razveljavitve ter časovna in obsegovna omejitve avtonomnega delovanja. Na prvi stopnji človek odloča o vsaki akciji, na drugi o vsakem predlogu, na tretji periodično ali izjemoma; ključno vprašanje torej ni, ali vključiti človeka, temveč na kateri točki procesa in s kakšno pogostostjo. Akt o umetni inteligenci to v členu 14 formalizira kot zahtevo, da je človeški nadzor sorazmeren s tveganjem, stopnjo avtonomije in kontekstom uporabe.

Kdaj scenarij postane pravno »visoko tvegan«. Za našo domeno je najbolj neposredno relevantna Priloga III, točka 4(a) — sistemi, namenjeni zaposlovanju, dodeljevanju nalog in upravljanju delavcev. Scenarij, ki se nam zdi »le priporočilo za razporeditev«, lahko pade v to kategorijo, ker vpliva na to, kdo dobi katero delo; za take sisteme veljajo dodatne obveznosti od sistema obvladovanja tveganj in kakovosti podatkov do tehnične dokumentacije, beleženja, človeškega nadzora in kibernetske varnosti (členi 9 do 15) [3].

Preglednost. Člen 50 zahteva, da uporabnik razume, da komunicira z UI. Pri agentih to ni omejeno na neposrednega uporabnika: če agent v imenu uporabnika spremeni razpored, ki zadeva druge osebe, so te »prizadete osebe«, čeprav z agentom niso nikoli neposredno komunicirale [2]. To je pomembno prav za scenarije, kjer agent posega v razpored več ljudi hkrati.

Časovnica uveljavljanja se je zaradi zakonodajnega paketa »AI Omnibus« večkrat premikala. V času pisanja velja, da so prepovedane prakse v veljavi od februarja 2025, obveznosti ponudnikov modelov za splošne namene od avgusta 2025, obveznosti preglednosti iz člena 50 od avgusta 2026, obveznosti za visoko tvegane sisteme pa so premaknjene na december 2027 oziroma avgust 2028 za sisteme, vgrajene v regulirane izdelke [3]. Odloga pri tem ni mogoče razumeti kot razloga za odlašanje: zahteve po sledljivosti in nadzoru je bistveno ceneje vgraditi v arhitekturo takoj kot naknadno v delujoč produkcijski sistem.

Tabela 1: Stopnje avtonomije, pripadajoči mehanizmi upravljanja in regulatorne zahteve.

Stopnja avtonomije	Vključenost človeka	Zahtevana sledljivost	Regulatorna teža
Razširitev človeka	Potrditev vsake akcije	Predlog, razlaga, odločitev	Odvisna od namena, ne od stopnje
Pol-avtonomni agent	Potrditev vsakega predloga	Zgoraj + viri podatkov in klici orodij	Praviloma člen 12 in 14
Avtonomni agent	Končna odobritev ali naknadni pregled	Popolna sled izvajanja in odločitev	Člen 12, 14 in praviloma Priloga III

2.4. Iz ogrodja izpeljane tehnične zahteve

Iz zgornjega ogrodja in iz našega poslovnega konteksta smo izpeljali konkreten nabor zahtev za agentno plast. Te zahteve so hkrati merila, po katerih smo v četrtem poglavju primerjali ogrodja.

Tabela 2: Zahteve za agentno plast in njihov izvor.

Oznaka	Zahteva	Izvor
Z1	TypeScript in Node.js, vgradljivost v obstoječi zaledni sistem	obstoječi tehnološki sklad
Z2	Lastna orodja nad internimi vmesniki in bazo	domenska narava rešitve
Z3	Dostop do orodij glede na vlogo in naročniški paket	nadzor nad dejanji (2.3)
Z4	Stroga izolacija med najemniki	zahteva B2B, GDPR
Z5	Usmerjanje na modele glede na zahtevnost naloge	stroški in kakovost
Z6	Kratkoročna zgodovina in dolgoročni spomin pogovorov	uporabniška izkušnja
Z7	Človeška potrditev pred dejanji s posledicami	razdelek 2.2 in 2.3, člen 14
Z8	Revizijsko beleženje vsakega klica LLM in orodja	razdelek 2.3, člen 12
Z9	Rezidenčnost podatkov v EU	GDPR in Akt o umetni inteligenci
Z10	Samogostljivost brez obveznega oblačnega ponudnika	Z9 in neodvisnost
Z11	Odprta koda brez licenčnine na uporabnika ali sled	ekonomika rešitve SaaS
Z12	Produksijska zrelost in obvladljiva vezanost na ogrodje	poslovno tveganje

3 Poslovne aplikacije za dva tipa uporabnikov

3.1. Agent kot nov tip uporabnika sistema

Osrednja teza tega poglavja je, da bomo informacijske sisteme v prihodnosti morali graditi ne zgolj za človeške uporabnike, temveč tudi za agente UI. To pomeni, da bodo poslovne aplikacije morale ponuditi dva vzporedna dostopa: uporabniški vmesnik za ljudi in programski vmesnik za agente, na primer prek protokola MCP (Model Context Protocol) [4]. Ta ponuja standardiziran način, kako zunanji agent odkrije in kliče funkcionalnosti aplikacije, ne da bi poznal njeno notranjo implementacijo.

Posledica je, da aplikacija dobi drugega »uporabnika«, ki ne klika po zaslonih, temveč neposredno kliče funkcije in orodja. S tem odpade celotna vrsta implicitnih omejitev grafičnega vmesnika: predpisan vrstni red korakov, potrditveni dialogi in vizualni kontekst, ki pove, da je predlagana sprememba v konfliktu z drugo.

3.2. Zakaj agentom ne moremo izpostaviti enakih zmogljivosti kot ljudem

Človek v grafičnem vmesniku dobi implicitne varovalke — vidi kontekst, sledi predpisanemu toku in naleti na potrditvena okna. Agent, ki kliče orodje neposredno, teh omejitev nima: naredi natanko to, kar orodje dopušča, ne glede na to, ali je to v danem trenutku smiselno. Za agente je zato treba eksplicitno opredeliti, katera orodja so na voljo, s kakšnimi parametri in v kakšnih mejah ter katera dejanja zahtevajo potrditev.

Iz tega sledi ključno arhitekturno načelo tega prispevka: varovalke morajo biti vgrajene v sama orodja, ne le v logiko agenta. To ni zgolj dobra inženirska praksa. Strokovna literatura o skladnosti agentnih sistemov z Aktom o umetni inteligenci [5] izrecno opozarja, da navodilo modelu v pozivu — na primer »ne izvedi rezervacije brez potrditve« — ni varnostna kontrola, ker ga lahko model prezre zaradi napačne interpretacije, vrivanja navodil (angl. prompt injection) ali preprosto nepredvidenega vedenja. Zahtevana je uveljavitev omejitev na ravni orodja oziroma vmesnika. Isto načelo neodvisno izpostavlja tudi nabor tveganj za agentne aplikacije organizacije OWASP [6], ki med najresnejšimi tveganji navaja prav zlorabo orodij in postopno razširitev pravic prek zaporednih klicev.

3.3. Interni in zunanji agenti

Pristop k upravljanju se bistveno razlikuje glede na to, ali agenta razvijamo sami ali do naših orodij dostopa tuj agent. Pri internih agentih poznamo njihovo logiko, pozive in omejitve ter jih lahko vnaprej testiramo na znanih scenarijih. Pri zunanjih agentih, ki dostopajo prek protokola MCP, ne poznamo niti njihove notranje logike niti namena klicatelja niti konteksta, v katerem delujejo, in ne moremo vnaprej testirati vseh možnih načinov klicanja naših orodij. Upravljanje se mora zato preseliti iz »nadzora nad agentom« v »nadzor nad orodjem«: orodje mora biti varno ne glede na to, kdo in zakaj ga kliče. Tveganja, ki jih je treba predvideti, so napačna interpretacija namena uporabnika, verižno klicanje več orodij, ki posamezno delujejo smiselno, skupaj pa povzročijo neželen učinek, poskusi razširitve obsega delovanja prek zaporednih klicev ter preobremenitev sistema z avtomatiziranimi klici.

Praktični protiukrepi, ki jih konkretiziramo v šestem poglavju, so omejitve obsega, potrditveni koraki za kritična dejanja, način poskusne izvedbe (angl. dry-run) pred dejansko spremembo, omejitve hitrosti klicanja in dosledno načelo minimalnih potrebnih pravic.

4 Primerjava ogrodij za gradnjo agentov UI

4.1. Namen in metodologija primerjave

Ko smo se v Alloxyju odločili, da bomo agentne zmogljivosti razvijali sistematično, smo se znašli pred odločitvijo, ki je bila hkrati tehnološka in arhitekturna: ali agentno plast zgraditi sami neposredno nad klici LLM ali prevzeti eno od obstoječih ogrodij. Lastna implementacija je privlačna, ker daje popoln nadzor in nič vezanosti, a pomeni, da je treba samostojno rešiti zanko klicanja orodij, upravljanje zgodovine, perzistenco, sledenje in obravnavo napak — kar je prvih nekaj sto vrstic vsakega agentnega projekta, ki jih je nato treba tudi vzdrževati. Zato smo pred odločitvijo sistematično pregledali obstoječa ogrodja.

Merila so izpeljana neposredno iz zahtev Z1 do Z12 iz tabele 2. Poudariti velja, da so regulatorne zahteve (Z8 in Z9) v tej primerjavi enakovredna tehnična merila, ne naknaden dodatek — kar je ena od praktičnih posledic ogrodja iz drugega poglavja. Ocenili smo sedem ogrodij; izbor odraža stanje ekosistema v času odločitve, ki se hitro spreminja [14].

4.2. Pregled posameznih ogrodij

OpenAI Agents SDK [7] je bil naše izhodišče. Je dobro dokumentiran in podpira klicanje orodij, človeško potrditev in sledenje, a je tesno vezan na enega ponudnika modelov, nima večnajemniškega ločevanja spomina in zgodovine, opazljivost pa je na voljo le prek platforme ponudnika, kar je z vidika zahteve Z9 problematično. LangChain [8] je bil v času odločanja najbolj razširjen, a njegova visoka cena abstrakcije se redko lepo preslika na domensko specifično logiko agenta, Python pa ostaja primarno okolje; upravičen je na začetku projekta, ne kot temelj zrele domenske rešitve. LangGraph [11] iz istega ekosistema je najzrelejša možnost za kompleksne grafovsko modelirane tokove, a za agenta z veliko orodji in preprosto zanko dodaja kompleksnost brez koristi. Google ADK [10] je zmogljiv in odprtokoden, vendar v praksi optimiziran za en oblak, kar je pri gostovanju v EU zunaj njega pomemben zadržek. Letta [12] postavlja spomin v ospredje, a je celovit izvajalnik in ne vstavljiva plast — njegov prevzem pomeni prevzem celotnega modela agenta. Vercel AI SDK [9] smo že uporabljali in ga odlikuje odlična podpora za TypeScript ter neodvisnost od ponudnika, a gre primarno za plast za klice modelov: manjkajo delovni tokovi, upravljanje spomina, večnajemniško ločevanje in evalvacije.

Mastra [13] je ogrodje, zasnovano primarno za TypeScript, z različico 1.0 iz januarja 2026 in temelji na ogrodju Vercel AI SDK. Eno samo ogrodje pokriva agente, spomin, orodja, delovne tokove, človeško potrditev in opazljivost, ključna zmogljivost za naš kontekst pa je dinamično razreševanje orodij glede na kontekst zahteve: orodja, ki niso v vrnjenem naboru, so v celoti odsotna iz konteksta modela. Strežniški adapterji omogočajo vgradnjo v obstoječi zaledni sistem brez ločene storitve, kot hramba zadostuje že obstoječa baza PostgreSQL, licenca Apache 2.0 pa pomeni samogostljivost in odsotnost licenčnine. Med slabostmi je treba izpostaviti mladost ogrodja ter dejstvo, da izolacija na ravni najemnika ni samodejna, prav tako niso vgrajeni proračuni stroškov po najemniku, razkritja po členu 50 pa so stvar uporabniškega vmesnika.

Tabela 3: Primerjava ogrodij glede na zahteve Z1 do Z12.

Merilo	OpenAI SDK	LangChain	Vercel AI SDK	Google ADK	LangGraph	Mastra
Domoroden za TypeScript	delno	ne	da	delno	delno	da
Register lastnih orodij	da	da	da	da	da	da
Omejevanje orodij (vloga, paket)	ročno	ročno	ročno	ročno	ročno	domorodno
Večnajemniška izolacija	ročno	ročno	ročno	ročno	ročno	delno
Usmerjanje modelov	omejeno	da	da	da	da	da
Upravljanje pogovorov	ročno	da	delno	da	da	da
Dolgoročni spomin	ne	prek vtičnikov	ne	delno	delno	da
Stiskanje zgodovine	ne	delno	ne	delno	delno	da
Človeška potrditev	da	delno	delno	da	da	da
Vgrajena opazljivost	delno	prek orodja	delno	delno	prek orodja	da
Kaskadno brisanje po GDPR	ne	ne	ne	ne	ne	da
Samogostljivost	da	da	da	delno	da	da
Rezidenčnost v EU	da	da	da	tvegano	da	da
Vgradnja v obstoječi sistem	da	da	da	delno	da	da
Odprta koda brez licenčnine	ne	da	da	da	da	da
Produkcijska zrelost	visoka	visoka	visoka	srednja	visoka	srednja
Tveganje vezanosti	visoko	visoko	nizko	srednje	visoko	srednje

4.3. Zakaj smo izbrali Mastro

Odločili smo se za Mastro kot orkestracijsko plast, pri čemer smo Vercel AI SDK ohranili kot plast za klice LLM pod njo. To nam omogoča prilagodljivost pri izbiri ponudnika modelov LLM, ki ga lahko enostavno zamenjamo brez spremembe izvorne kode, hkrati pa nam omogoča, da našim strankam ponudimo, da uporabijo lastne modele LLM. OpenAI Agents SDK je bil v celoti nadomeščen, LangChain, Google ADK in LangGraph pa nismo sprejeli. Odločilni argumenti so bili, po vrsti pomembnosti za naš kontekst, naslednji.

Prvič, Mastra je edino ogrodje z domorodnim omejevanjem orodij glede na vlogo in naročniški paket (Z3). Pri vseh ostalih bi ta mehanizem morali zgraditi ročno, pri nas pa je nosilec ključne zahteve iz razdelka 3.2. Konkretno to pomeni, da agent dostopa izključno do podatkov in dejanj, do katerih ima dostop trenutno prijavljeni uporabnik glede na svojo vlogo, in izključno do zmogljivosti, ki jih ima njegova organizacija glede na svoj naročniški paket. Agent torej ni privilegiran akter s širšim dostopom kot človek, v imenu katerega deluje. Vodja oddelka, ki v uporabniškem vmesniku ne vidi finančnih podatkov projekta, jih ne bo dobil niti prek asistenta — ne zato, ker bi model to zavrnil, temveč zato, ker orodje, ki bi jih vrnilo, sploh ne pride v njegov kontekst.

Drugič, vgrajena opazljivost s hrambo v lastni bazi (Z8, Z9). Alternativa bi bila zunanja storitev SaaS za sledenje, kar je zaradi rezidenčnosti v EU izključeno, ali lastnoročno pisanje revizijskih sledi. Licenca Apache 2.0 pri tem pomeni tudi, da opazljivost ni metrirana po revizijskih sledeh (Z11) — pri metriranem modelu bi se strošek skaliral skupaj z uporabo.

Tretjič, vgradnja v obstoječi zaledni sistem brez ločene storitve (Z1). To pomeni manj operativne kompleksnosti, eno samo avtentikacijsko površino in eno samo bazo. Kaskadno brisanje spomina ob izbrisu uporabnika je pri tem del ogrodja in ne naknadno opravilo.

Kljub izbiri ogrodja smo morali sami zgraditi izolacijo na ravni najemnika, proračune stroškov po najemniku, razkritja po členu 50 v uporabniškem vmesniku ter — kot se je izkazalo med načrtovanjem — tudi lastno plast dolgoročnega spomina in lasten mehanizem človeške potrditve; zakaj smo se pri zadnjih dveh odločili proti vgrajenim rešitvam, utemeljujemo v razdelkih 5.4 in 5.5.

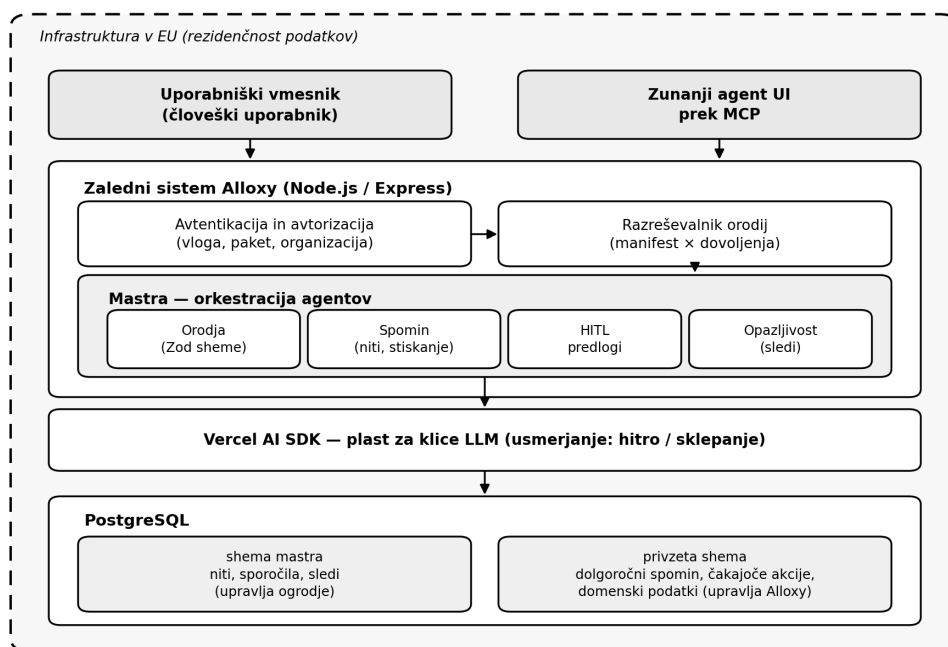
5 Arhitektura agentne plasti v platformi Alloxy

5.1. Izhodišče in cilj migracije

Prvotna implementacija asistenta je tekla na OpenAI Agents SDK: en usmerjevalni agent s trdo zapisanim modelom, pet orodij, stanje pogovora pri ponudniku modela, brez lokalne perzistence in brez sledenja. Cilj migracije ni bil zgolj »zamenjati ogrodje«, temveč prenesti avtorizacijo, spomin, usmerjanje modelov, opazljivost in brisanje po GDPR iz ročno pisanega lepila v konfigurabilne platformne zmogljivosti.

5.2. Namestitveni model in hramba

Mastra teče v procesu obstoječega zalednega sistema, ne kot ločena storitev, saj bi samostojni strežnik ogrodja pomenil drugo namestljivo enoto in drugo avtentikacijsko površino. Vsa hramba, ki jo upravlja ogrodje — niti pogovorov, sporočila, viri spomina in revizijske sledi — živi v ločeni shemi v obstoječi bazi PostgreSQL, medtem ko privzeta shema ostane v domeni naše plasti za objektno-relacijsko preslikavo; migracijski tok te plasti namreč ne sme nikoli povleči vase tabel, ki jih upravlja ogrodje. Nove tabele pod našim upravljanjem uvajamo le tam, kjer podatkovni model in podatke upravlja Alloxy: konfiguracija usmerjanja, dolgoročni spomin in zapis čakajočih akcij. Arhitektura je prikazana na sliki 1.



Slika 1: Arhitektura agentne plasti v platformi Alloxy.

5.3. Avtorizacija orodij: dve plasti brez zaupanja modelu

To je arhitekturno najpomembnejši del sistema in neposredna izvedba načela iz razdelka 3.2. Kontekst zahteve nosi v vsak klic agenta naročniški paket, učinkovita dovoljenja ter identiteto organizacije in zaposlenega. Vsako orodje v manifestu deklarira zahtevana dovoljenja kot polje polj, pri čemer zunanja raven pomeni konjunkcijo, notranja pa disjunkcijo — branje razporeditev na primer zahteva dovoljenje za pregled razporeditev vseh zaposlenih ali zgolj za pregled lastnih razporeditev. Ključna odločitev je bila, da ponovno uporabimo obstoječe domenske ključe dovoljenj in ne uvedemo vzporednega »dovoljenjskega modela za UI«; za celotno migracijo ni bil uveden noben nov ključ, saj vsak vzporeden model sčasoma neizogibno zdrsne od izvirnega.

Razreševalnik orodij preseje manifest skozi isti evalvator dovoljenj, ki ga uporablja preostala aplikacija, in to sveže ob vsaki zahtevi, tako da se sprememba pravic sredi seje odrazi že pri naslednji. Neavtorizirano orodje ni nikoli posredovano agentu in zato nikoli ne pride v kontekst modela; model zanj ne ve, zato ga ne more ponuditi, se zanj

opravičiti ali tiho spodleteti. Alternativo, pri kateri bi orodje ob klicu vrnilo sporočilo »tega ne smete«, smo zavrnil, ker razkrije obstoj orodja in posredno strukturo pravic v organizaciji.

Druga plast je ponovna avtorizacija v vsaki funkciji: vsaka izvedbena funkcija orodja prejme organizacijo in obseg ter vedno filtrira po njej, tudi če je razreševalnik klic že spustil skozi, saj bi bilo zanašanje zgolj nanj enojna točka odpovedi. Kot dodatni varovalki delujeta (a) projekcija polj: občutljiva finančna polja so v naboru dovoljenih le, če klicatelj hkrati drži ustrezni finančni ključ, ter (b) filtriranje podatkov na nivoju orodja: agent dobi le tiste podatke, do katerih ima dostop uporabnik. Stranski učinek je manjša poraba žetonov.

5.4. Spomin: kaj prepustimo ogrodju in kaj ne

Ogrodju prepuščamo perzistenco niti pogovora in stiskanje delovnega konteksta, tako da dolgi pogovori ne prelijejo kontekstnega okna modela; ročno rezanje zgodovine je iz kode odstranjeno. Dolgoročni spomin pa smo zgradili sami: namesto da bi se zanesli na nestrukturirane spominske zapise ogrodja, ob koncu seje zaženemo lasten ekstrakcijski cevovod — klic modela s strukturiranim izhodom — ki zapiše tipizirane vrstice v našo tabelo. Taksonomija dolgoročnega spomina je namreč produktno sredstvo, ki napaja diferenciator asistiranega razporejanja: mora biti strukturirana, poizvedljiva po kategoriji in preverljivo izbrisljiva, česar neprozoren zapis, ki ga upravlja model, ne omogoča.

Taksonomija ima štiri kategorije: preference (izražene omejitve pri razporejanju, na primer da se na določeno stranko ne dodeljujejo pripravniki ali da je nekdo ob petkih zaposlen polovično), cilji (ponavljajoči se nameni, na primer redno prizadevanje za popolnitev ekipe na projektu), leče (privzete ekipe, projekti ali filtri, h katerim se uporabnik vrača) in utemeljitve (razlog za sprejeto odločitev, pri čemer transakcijski zapis že živi v domenski bazi). Pomembno je tudi, česa namenoma ne shranjujemo: prehodnih vprašanj in odgovorov ter podatkov, ki že obstajajo v domenskih tabelah. Spomin ni druga kopija baze — če bi to postal, bi se neizogibno razšel z njo.

Prožilci konca seje so začetek novega pogovora, zaprtje okna asistenta, konec seje aplikacije in odjava, kar nastopi prej. Ker nobenega od teh dogodkov ni mogoče zanesljivo ujeti v vseh primerih, imamo kot varovalo periodično opravilo, ki obdela nedejavne niti; ekstrakcija je idempotentna na nit, tako da prekrivanje prožilcev ne povzroči podvojenih zapisov.

5.5. Človeška potrditev: predlagaj, shrani, potrdi, izvedi

Ogrodje ponuja domoroden mehanizem zaustavitve in nadaljevanja izvajanja pred dejanjem s posledicami. Kljub temu smo izbrali lasten vzorec: orodje pripravi predlog, ta se shrani kot vrstica v namenski tabeli, uporabnik ga potrdi ali zavrne prek ločene končne točke, in šele nato se sprememba izvede. Razloga sta bila dva. Prvi je manjša kompleksnost: asistent je navaden agent brez delovnih tokov, domorodna zaustavitev pa bi zahtevala, da zapisovalne akcije postanejo delovni tokovi in da se žetoni za nadaljevanje pretakajo skozi transport klepeta. Drugi je regulatorno ujemanje: pričakovanja glede sledljivosti iz člena 12 ter odgovornosti in izbrisa po GDPR nagrajujejo prvorazreden, poizvedljiv in pripisljiv zapis z nadzorovano hrambo, kar namenska tabela daje, hramba stanja izvajanja ogrodja pa ne. To ni varnostna trditev — tudi pri domorodni zaustavitvi je mutacija po nadaljevanju deterministična koda; razlika je izključno v kakovosti revizijske sledi.

Vsebina potrditvene kartice ni formalnost. Vsak predlog mora nositi prizadeto osebo ali entiteto, človeku berljiv prikaz stanja pred spremembo in po njej ter nastali učinek; golo vprašanje »Potrdite?« je v našem sistemu napaka in ne funkcionalnost, saj je prav to vsebina človeškega nadzora, ki jo pričakujeta člen 14 Akta o umetni inteligenci in člen 22 Splošne uredbe o varstvu podatkov. Isti tok velja za vsako zapisovalno orodje, vključno z orodji za odobritve — potrditev odobritve je namreč sama po sebi mutacija. Vrstica čakajočega dejanja je hkrati revizijski zapis, ki je kljub trajni hrambi vključen v kaskado brisanja.

5.6. Usmerjanje modelov

Vsi trdo zapisani modeli so odstranjeni iz agentne poti; namesto njih tipizirana in ob zagonu validirana konfiguracija preslika razred zahteve — hitro ali sklepanje — v ponudnika in model. Neveljavna konfiguracija poruši zagon in ne posamezne zahteve, tako da napako odkrijemo ob namestitvi. Konfiguracija živi v kodi in ne v bazi, verzioniranje in vračanje nazaj pa pridejo iz zgodovine različic.

Razvrščanje zahteve v razred je determinističen predklasifikator in ne dodaten klic modela. Deluje po prednostnem vrstnem redu: ročni globalni pribitek kot operativno stikalo, oznaka na posameznem orodju (razvrščanje kandidatov na primer vedno teče na močnejšem modelu), jezikovno občutljivo ujemanje ključnih besed in nazadnje privzeta vrednost. Tak pristop je poceni, razločljiv in nastavljen, medtem ko bi klic modela za odločanje o modelu dodal strošek in latenco vsakemu sporočilu. Konfiguracija nosi tudi nabor odobrenih regij, tako da usmerjanje ne more izbrati ponudnika zunaj odobrenega nabora.

5.7. Opazljivost in izpolnjevanje zahtev GDPR

Vsak klic modela in vsak klic orodja oddaja revizijsko sled z modelom, porabo žetonov, latenco in stroškom; sledi se hranijo v naši bazi, brez pošiljanja v zunanjo storitev za sledenje, saj je rezidenčnost v EU trda omejitev. Vsaka sled nosi identiteto uporabnika in najemnika, agenta, razred usmerjanja in dejansko uporabljen model, kar omogoča razčlenbo porabe po vseh teh dimenzijah. Neuspeh klic prav tako proizvede sled z zabeleženim neuspehom in delno porabo.

Kaskada brisanja po GDPR zajema vse domene hrambe, povezane z UI: niti in sporočila, vire spomina, dolgoročni spomin, čakajoče akcije in revizijske sledi. Vsak korak vrne število izbranih zapisov, preverjena poizvedba pa mora vrniti nič, preden se brisanje označi za dokončano — brisanje torej ni domnevano, temveč dokazano. Kaskada je vpeta v brisanje uporabnika in organizacije, pri čemer mehko brisanje osebe še vedno trdo počisti njene podatke, povezane z UI.

6 Trije agenti, s katerimi smo podprli poslovne procese

6.1. Pametno priporočanje virov

Prvi scenarij je dodeljevanje sodelavcev na projekte ob upoštevanju razpoložljivosti, kompetenc, preferenc in poslovnih omejitev. Najpomembnejša ugotovitev je bila, da mora biti sam predlog determinističen: vodja, ki dvakrat postavi isto vprašanje, mora dobiti primerljiv odgovor, predlog, ki ga ni mogoče razložiti, pa v postopku, ki vpliva na razporeditev dela med ljudi, ni uporaben.

Zato je bilo treba natančno načrtovati, kje v procesu uporabiti katero tehnologijo. Veliki jezikovni model uporabljamo na ravni šifranta, ne na ravni posameznega predloga: z njegovo pomočjo izračunamo faktor zamenljivosti med vlogami — na primer, v kolikšni meri lahko razvijalec za celoten sklad nadomesti razvijalca uporabniškega vmesnika — in podobno oceno sorodnosti med veščinami. Ti faktorji se izračunajo vnaprej in shranijo, tako da so ob predlogu na voljo kot podatek in ne kot sprotna presoja modela.

Sam predlog zaposlenega za določen projekt pa poteka povsem deterministično na podlagi več dejavnikov: primernosti vloge, ujemanja veščin, razpoložljivosti, predhodnega sodelovanja s projektno ekipo in stroška. Uteži se prilagajajo glede na preference organizacije — nekatere dajejo prednost kontinuiteti ekipe, druge stroškovni učinkovitosti. Rezultat je ponovljiv in razločljiv predlog, pri katerem je vsak dejavnik mogoče prikazati kot razlog za uvrstitev kandidata.

Veliki jezikovni model ima v tem scenariju še eno vlogo, in sicer sintezo razlage. Deterministični rezultat je nabor številskih ocen po faktorjih, kar je za sistem uporabno, za uporabnika pa slabo berljivo. Model iz teh ocen sestavi kratko utemeljitev v naravnem jeziku, ki izpostavi dejavnike, ki so prevladali, in tiste, ki so bili sporni — na primer da kandidat vlogo pokriva posredno prek zamenljivosti in je z ekipo že sodeloval, a da je njegova razpoložljivost v

drugem tednu obdobja tesna. Model pri tem ne spreminja razvrstitve in ne dodaja novih dejstev; njegov vhod so izključno izračunane ocene. Če bi model smel vplivati na razvrstitev, bi izgubili ponovljivost, ki je bila izhodiščna zahteva.

Sorodnost veščin smo prvotno računali iz vektorskih vložitev, kar je dajalo netočne zadetke: veščine, ki si le delijo besedišče, so bile označene kot sorodne, resnično zamenljive pa spregledane. Zamenjali smo jih s predizračunano tabelo parne podobnosti, torej z istim pristopom kot pri zamenljivosti vlog. Merilo uspeha je bilo povsem konkretno: veščini »React« in »ReactJS« morata biti prepoznani kot tesno ujemanje, veščini »React« in »Angular« pa ne, čeprav sta si besedilno blizu in v vektorskem prostoru pogosto sosednji.

Stopnja avtonomije tega scenarija je prva — sistem predlaga, vodja potrdi. Kljub temu se dotika dodeljevanja nalog in upravljanja delavcev, kar po Prilogi III, točki 4(a) potencialno spada med visoko tvegana področja uporabe. To je konkreten dokaz razlikovanja iz razdelka 2.2: nizka avtonomija ne pomeni nizkega regulatornega tveganja, ker je slednje odvisno predvsem od namena uporabe. Prav zato so razločljivost predloga, revizijska sled in obvezna človeška potrditev tu več kot dobra praksa.

6.2. Poizvedovanje razpoložljivosti in rezervacija virov

Drugi scenarij odgovarja na vprašanja tipa »kdo je naslednji mesec vsaj dvajset ur na teden prost in obvlada določeno tehnologijo« ter omogoča posledično rezervacijo. Pri njem smo prišli do treh prenosljivih spoznanj.

Prvo spoznanje je, da mora izračun izvesti orodje in ne asistent. Prvotno bi asistent sam ugibal odstotek razpoložljivosti iz zahtevanih ur, kar je nedeterministično in — kar je pomembneje — napačno za zaposlene s krajšim delovnim časom, ker se tedenska kapaciteta razlikuje po osebi. Orodje zato prejme zahtevane ure eksplicitno in pretvorbo izvede samo, za vsakega zaposlenega glede na njegovo lastno tedensko kapaciteto. Splošno pravilo: kjerkoli obstaja deterministična formula, naj jo izvede koda in ne model.

Drugo spoznanje je zahteva po enem samem viru resnice. Agent za razpoložljivost je sprva računal sproti na podlagi operativnih tabel. Rezultati se niso povsem skladali s podatki iz poročevalskih dejstev, zato je lahko ista oseba v odgovoru agenta veljala za prosto, dejansko pa ni bila na voljo zaradi dopusta ali skrajšanega delovnega časa. Rešitev je bila preusmeriti orodje na iste poročevalske podatke, s čimer smo brez dodatnega dela dobili tudi upoštevanje praznikov in odobrenih odsotnosti.

Tretje spoznanje je, da se negotov podatek prikaže in ne vračuna. Še neodobrene odsotnosti se ne odštejejo od razpoložljivosti, temveč se izpišejo kot opozorilo ob zaposlenem, uporabnik pa odloči, kako jih upoštevati. Vgraditi špekulativne podatke v uradno številko bi naredilo poročila nestabilna ob vsakem neodobrenem zahtevku, in to za uporabnika nevidno.

Prav ta scenarij najboljše ponazori tezo iz tretjega poglavja, saj isto orodje deluje v dveh bistveno različnih kontekstih. Znotraj aplikacije je klic pod neposrednim nadzorom: uporabnik je avtenticiran, vidi kontekst, dejanje je del znanega toka. Pri izpostavitvi prek protokola MCP pa isto orodje kliče zunanji agent, nad katerim nimamo nadzora; tveganja so konflikti v razporejanju, neželene spremembe plana, rezervacije, ki ne upoštevajo pravil, vidnih le v uporabniškem vmesniku, ter posreden vpliv na tretje osebe, katerih razpored se spremeni, ne da bi vedele, da je spremembo sprožil avtonomni agent. Varovalke, ki smo jih uvedli, so obvezen potrditveni korak po istem vzorcu kot v razdelku 5.5, omejitev časovnega okna in obsega, odgovor v načinu poskusne izvedbe ter beleženje vseh poskusov klicev, vključno z zavrnjenimi. Zadnje se je izkazalo za nepričakovano koristno: zavrnjeni klici so koristen pokazatelj napak v logiki agenta in morebitnih poskusov zlorabe.

6.3. Avtonomni optimizacijski agent

Tretji scenarij je avtonomni optimizacijski agent, ki periodično analizira stanje virov in pripravi predloge za izboljšanje razporeditve. Gre za najvišjo stopnjo avtonomije v našem naboru, saj agent ni sprožen z uporabnikovim vprašanjem, temveč se zažene sam.

Agent v analizo zajame tri skupine vhodnih podatkov: izkoriščenost virov, torej kdo je v obravnavanem obdobju preobremenjen in kdo premalo zaseden (iz istih poročevalskih podatkov kot scenarija 6.1 in 6.2), vpliv načrtovanih odsotnosti na kapaciteto, saj odobrena odsotnost ključne osebe sredi projekta ustvari vrzel, ki jo je bolje predvideti vnaprej, ter maržo posameznih dodelitev, ker prerazporeditev, ki uravnovesi obremenitev, a znatno poslabša donosnost projekta, ni izboljšava.

Agent najprej identificira problematične vzorce — preobremenitve, nepokrite vrzeli, dodelitve z neugodnim razmerjem med stroškom in vlogo — nato pa za vsakega pripravi konkreten predlog: prerazporeditev ur z enega projekta na drugega, zamenjavo dodeljene osebe ali opozorilo, da za vrzel ni ustreznega kandidata. Vsak predlog je opremljen z izračunanim učinkom, tako da uporabnik vidi tudi, kaj se s tem izboljša in kaj morda poslabša. Bistveno je, da agent kljub najvišji stopnji avtonomije ne izvede nobene spremembe brez končne človeške potrditve — teče skozi isti vzorec kot vsa druga zapisovalna orodja. Avtonomija se torej nanaša na to, kdaj in kako agent pride do predloga, ne na to, ali ga sme izvesti.

7 Zaključek

Vgradnja UI v poslovne aplikacije ni binarna odločitev med tem, ali umetno inteligenco imamo ali ne, temveč niz odločitev o stopnji avtonomije za vsak scenarij posebej. Vsaka stopnja zahteva svoj nabor mehanizmov upravljanja, ti pa ne smejo biti zgolj dodatek, temveč sestavni del arhitekture.

Prvo prenosljivo spoznanje je, da je izbira ogrodka upravljalška odločitev in ne le tehnološka: merila, ki so pri nas prevesila tehtnico — omejevanje orodij glede na pravice uporabnika, opazljivost v lastni infrastrukturi in kaskadno brisanje — izhajajo neposredno iz regulatornih in poslovnih zahtev. Drugo je, da varovalke sodijo v orodja in ne v poziv: neavtorizirano orodje ne sme priti v kontekst modela, avtorizacija se preverja dvakrat, deterministične izračune izvaja koda in ne veliki jezikovni model, negotovi podatki pa se prikažejo in ne vračunajo.

Tretje spoznanje je, da mora biti človeški nadzor vsebinski: potrditvena kartica brez prikaza, kdo je prizadet in kakšno je stanje pred spremembo in po njej, je videz nadzora in ne nadzor. Četrto pa je, da je regulatorna klasifikacija scenarija odvisna predvsem od namena in področja uporabe, ne od stopnje avtonomije, zato je regulatorni pregled treba izvesti za vsak scenarij posebej.

Predstavljeno ogrodje je po naši oceni prenosljivo na druge domene. Med odprtimi smermi nadaljnjega dela izpostavljam standardizacijo varovalk za dostop prek protokola MCP, avtomatizirano evalvacijo agentnih scenarijev pred produkcijsko uvedbo, metrike zaupanja za premik scenarija po spektru avtonomije ter večagentno zasnovo, pri kateri agenti kličejo druge agente. Zadnja odpira vprašanje, ki ga tu še nismo naslovili: kako se avtorizacija in sledljivost obnašata v verigi agentov, kjer vsak deluje v imenu prejšnjega.

Literatura

- [1] Evropski parlament in Svet EU, Uredba (EU) 2024/1689 o določitvi harmoniziranih pravil o umetni inteligenci (Akt o umetni inteligenci), UL EU L 2024/1689, 12. 7. 2024. <https://eur-lex.europa.eu/legal-content/SL/TXT/?uri=CELEX:32024R1689>, obiskano 13. 7. 2026.
- [2] Evropska komisija, Guidelines on transparency obligations for providers and deployers of certain AI systems (Article 50 AI Act), julij 2026. <https://digital-strategy.ec.europa.eu/en/policies/guidelines-transparency-ai-generated-content>, obiskano 13. 7. 2026.
- [3] Evropska komisija, Guidelines for providers and deployers of AI high-risk systems. <https://digital-strategy.ec.europa.eu/en/policies/guidelines-ai-high-risk-systems>, obiskano 13. 7. 2026.
- [4] Anthropic, Model Context Protocol, specifikacija. <https://modelcontextprotocol.io>, obiskano 13. 7. 2026.
- [5] NANNINI Luca et al., "AI Agents Under EU Law: A Compliance Architecture for AI Providers", arXiv:2604.04604, april 2026. <https://arxiv.org/abs/2604.04604>, obiskano 13. 7. 2026.
- [6] OWASP Foundation, Agentic Security Initiative: Top 10 for Agentic Applications, december 2025. <https://owasp.org/www-project-agentic-security-initiative/>, obiskano 13. 7. 2026.
- [7] OpenAI, Agents SDK. <https://openai.github.io/openai-agents-python/>, obiskano 13. 7. 2026.
- [8] LangChain. <https://python.langchain.com/docs/introduction/>, obiskano 13. 7. 2026.
- [9] Vercel, AI SDK. <https://ai-sdk.dev/docs>, obiskano 13. 7. 2026.
- [10] Google, Agent Development Kit. <https://google.github.io/adk-docs/>, obiskano 13. 7. 2026.
- [11] LangChain, LangGraph. <https://langchain-ai.github.io/langgraph/>, obiskano 13. 7. 2026.
- [12] Letta. <https://docs.letta.com/>, obiskano 13. 7. 2026.
- [13] Mastra, TypeScript agent framework. <https://mastra.ai/docs>, obiskano 13. 7. 2026.
- [14] Langfuse, Comparing Open-Source AI Agent Frameworks, 2026. <https://langfuse.com/blog/2025-03-19-ai-agent-comparison>, obiskano 13. 7. 2026.

Od dveh generalov do dveh storitev

Matjaž Prtenjak

Endava, d. o. o., Ljubljana, Slovenija
matjaz.prtenjak@endava.com

V svetu umetne inteligence se hitro ujamemo v zanko, da zaradi njene pomoči razvijamo preveč kompleksne sisteme. V tem duhu in duhu spleta ter neomejenih možnosti tako zagrabimo tudi idejo mikrostoritev. Sistem razdelimo na manjše storitve, ki imajo praviloma lastništvo nad svojimi podatki, komunikacija med njimi pa poteka prek omrežja, oddaljenih klicev ali sporočil. Rešitev je videti dobra, naravna in vsi jo priporočajo, pri tem pa zamolčijo kompleksnost. Klic funkcije v dinamični knjižnici zamenjamo s komunikacijo po nezanesljivem omrežju. Lokalno transakcijo nadomestimo s porazdeljeno, preprost vrstni red izvajanja pa z dogodki, ki lahko pridejo večkrat, nikoli ali pa celo v povsem napačnem vrstnem redu. Izkazuje se, da mikrostoritve niso tehnična rešitev, temveč organizacijska rešitev za dovolj velike organizacije.

Ključne besede:

mikrostoritve

distribuirani sistemi

modularni monolit

sporočilni sistemi

kompensacijska dejanja

1 Uvod – dva generala

Dva generala sta osnovna premisa članka, ki ponazarja ključne težave distribuirane arhitekture.

S svojima vojskama sta na nasprotnih straneh mesta in lahko zmagata samo, če napadeta usklajeno. Komunicirata lahko le s sli, ki pa jih lahko sovražnik prestreže (oziroma katerih sporočila se lahko izgubijo).

- Prvi general pošlje sporočilo: "Napadli bomo opolnoči." – vendar ne ve, ali je drugi general prejel sporočilo, zato čaka na potrditev.
- Drugi pošlje odgovor: "Dogovorjeno." – vendar ne ve, ali je prvi general prejel potrditev, zato čaka na potrditev potrditve.
- Prvi general pošlje odgovor na odgovor: "Prejel potrdilo!", vendar spet ne ve, ali je drugi general prejel potrditev potrditve, zato čaka.
- Drugi general pošlje sporočilo: ...

Veriga potrditev se lahko nadaljuje v neskončnost, popolne gotovosti pa ne dosežemo. Problem koordiniranega napada je bil formalno obravnavan že v zgodnjih delih o komunikacijskih protokolih [1].

Lahko pa zadevo še bolj zapletemo.

Problem »bizantinskih generalov« [2] opisuje še zahtevnejši scenarij, v katerem lahko posamezen udeleženec ali komponenta deluje napačno oziroma zlonamerno in različnim udeležencem pošilja nasprotujoče si informacije. Tu ne govorimo več samo o nezanesljivosti dostave, temveč tudi o nezanesljivosti vsebine in vedenja udeležencev.

2 Ko se preprosti klic spremeni v dogovarjanje

Pri monolitni aplikaciji je lahko veliko stvari presenetljivo dolgočasnih. Metoda pokliče drugo metodo, knjižnica vrne rezultat, transakcija se potrdi ali zavrne, izjema pa se običajno pojavi v istem procesu in v istem programu.

To seveda ne pomeni, da je monolitna aplikacija privzeto enostavna. Kot vemo, se namreč navadno izkaže prav nasprotno. Monolitne aplikacije so lahko presenetljivo kompleksne in zapletene. Mnogokrat postanejo skupek programske kode, ki je medsebojno tako prepletena, da se je nihče ne upa več dotakniti [3].

Imajo pa monolitne aplikacije eno zelo pomembno lastnost: **vsa komunikacija poteka lokalno**. Če modul A pokliče modul B, ne razmišljamo o:

- izgubljenem paketu,
- ponovnem pošiljanju sporočila,
- potrditvi prejemnika,
- tem, ali je prejemnik procesiral zahtevo, preden je padla povezava.

Mikrostoritve pa vse te predpostavke obrnejo na glavo:

Monolitna aplikacija	Mikrostoritve
Klic funkcije je lokalni in neposreden	Komunikacija poteka prek HTTP-ja, RPC-ja ali sporočil
Ena lokalna transakcija	Več lokalnih transakcij in usklajevanje med storitvami
Skupna podatkovna baza (lahko celo del istega procesa!)	Storitve imajo praviloma ločeno lastništvo nad podatki
Napake in ponovitve so običajno lažje obvladljive	Sporočila so lahko podvojena, zakasnjena, preurejena ali izgubljena, odvisno od jamstev sistema

Mikrostoritve so distribuirani sistemi z vsemi težavami, ki jih slednji prinašajo. Mikrostoritve niso zahtevne zato, ker uporabljamo zveneča tehnična imena, kot so SAGA, Inbox, Outbox ali pa Eventual Consistency. Mikrostoritve so zahtevne zato, ker je **nemogoče popolnoma odstraniti negotovost, zakasnitev, podvajanje in delne odpovedi**.

3 Od vsakdanjih metafor do tehničnih problemov

- *Potujete ... Hotel, letalo in avtomobil ste rezervirali neodvisno na treh spletnih straneh. Na letališču dobite sporočilo, da je rezervacija hotela zavrnjena. Kaj storiti? Prekiniti potovanje, potovati in upati, da najdete drugi hotel?*

Če rezerviramo hotel, letalo in avtomobil na treh ločenih spletnih mestih, pričakujemo eno potovanje, sistemsko pa imamo tri ločene poslovne transakcije. Če rezervacija hotela odpove po tem, ko sta letalo in avtomobil že potrjena, ni več preproste možnosti "rollback". Sistem lahko prekliče let, odpove avtomobil ali ponudi drug hotel, vendar je to poslovna odločitev, ne samo tehnična operacija.

To je problem distribuiranih transakcij.

- *Kolesarite in imate hudo nesrečo. Vzamete mobilni telefon in pokličete pomoč... oglasi se operater in začnete govoriti, kje ste, kaj se je zgodilo ..., nato pa vidite, da se je telefon vmes ugasnil. Kaj storiti? Ali je operater razumel, da ste imeli nesrečo? Kdaj je telefon nehal delovati?*

Morda je operater razumel lokacijo in naravo nesreče, morda samo prvo polovico stavka, morda ničesar.

V mikrostoritvah se to pokaže kot delno izvedeno dejanje: ena storitev je nekaj zapisala, druga je dobila samo del podatkov, tretja pa se je odzvala prepozno. Če aplikacija tak dogodek samo ponovi, lahko stvar popravi ali pa naredi dodatno škodo. Zato morajo biti ponovitve načrtovane, operacije pa po možnosti idempotentne.

- *Nujno morate na sestanek iz Maribora v Ljubljano. Na avtocesti so hudi zastoji. Kaj storiti? Poskusiti vseeno na avtocesti ali iti po stari cesti?*

Če stojimo v zastoju med Mariborom in Ljubljano, izbiramo med slabo in nepopolno informacijo: ostati na avtocesti ali iti po stari cesti.

Tudi mikrostoritve živijo v okolju, kjer je omrežje včasih počasno, posrednik sporočil preobremenjen ali pa podatkovna baza ne deluje.

Vzorec ponovnih poskusov (*»retry«*) poudarja, da so prehodne napake v oblačnih in distribuiranih okoljih normalne ter da mora aplikacija odpovedi obravnavati z ustrezno zakasnitvijo in omejenim številom poskusov. Vendar *ponovni poskus* ni univerzalno zdravilo. Če vse storitve hkrati agresivno ponavljajo zahteve, lahko iz prehodne napake naredijo trajnejšo preobremenitev.

- *Od stranke prejmete naročilo po elektronski pošti in naročilo za iste izdelke tudi po navadni pošti. Ali je to isto naročilo ali moram dvakrat dostaviti iste izdelke?*

Če stranka pošlje naročilo po elektronski in navadni pošti, je poslovno vprašanje, ali gre za isto naročilo ali za dve naročili.

Tehnično podobno vprašanje se pojavi pri sporočilih: posrednik sporočil lahko iz različnih razlogov dostavi isto sporočilo več kot enkrat. Tudi če je posrednik zanesljiv, se lahko potrošnik sesuje po obdelavi, vendar pred potrditvijo.

Po ponovnem zagonu dobi isto sporočilo znova. Rešitev ni slepo zaupanje infrastrukturi, temveč deduplikacija na poslovni ravni: enolični identifikatorji.

- *Na telefon dobite sporočilo DHL-a: „Paket dostavljen“. Greste pred hišo in iščete, kje je kurir pustil paket, nakar dobite novo sporočilo: „Paket prevažamo“. Katero sporočilo je pravilno? Ali je paket vseeno dostavljen, pa ga je nekdo ukradel?*

Če prejmemo najprej obvestilo "paket dostavljen", čez minuto pa "paket prevažamo", imamo problem vrstnega reda dogodkov.

V centraliziranem sistemu je vrstni red pogosto samoumeven: ena transakcija se zgodi pred drugo. V distribuiranem sistemu pa lahko dogodek potuje skozi vrsto, ponovno dostavo, transformacijo, shrambo in potrošnika, ki deluje z drugačno hitrostjo.

Zato se moramo vprašati, ali dogodki vsebujejo različico, zaporedno številko ali časovno oznako, in ali zna prejemnik ignorirati starejši dogodek.

- *Greste na upravno enoto, da bi spremenili naslov...*

Če na upravni enoti spremenimo stalni naslov, pa zdravstveni dom še nekaj časa pošilja pošto na stari naslov, doživimo neusklajenost podatkov. Včasih to ni katastrofa, včasih pa je nesprejemljivo.

Glavna arhitekturna odločitev ni, ali si neusklajenost želimo, temveč, kje jo lahko sprejmemo, kje jo moramo zmanjšati in kje je sploh ne smemo uporabiti.

4 Mikrostoritve so organizacijska, ne samo tehnična arhitektura

Mikrostoritve so postale priljubljene zato, ker jih uporabljajo velika podjetja. In če jih uporabljajo velika podjetja, potem morajo biti »dobre! Toda to je nevarna poenostavitev.

Velika podjetja imajo veliko razvijalskih ekip, ločene domene, kompleksne postopke, različne tehnologije in potrebe, saj se deli organizacije premikajo neodvisno drug od drugega. Tam mikrostoritve pogosto rešujejo problem koordinacije med ljudmi. V manjšem sistemu pa lahko isti pristop ustvari več komunikacijskih težav, kot jih reši.

Conwayevo pravilo pravi, da organizacije oblikujejo sisteme, ki odražajo njihove komunikacijske strukture [4]. To je eden najpomembnejših razlogov, zakaj mikrostoritve pogosto delujejo v velikih organizacijah: meje med storitvami lahko sledijo mejam med enotami. To ni samo tehnični opis. Je tudi organizacijski opis.

Samostojna storitev ima največ smisla, če obstaja tudi samostojen lastnik: enota, ki razume domeno, razvija funkcionalnost, upravlja podatke, spremlja produkcijo in se odziva na incidente. Če organizacija teh sposobnosti nima, mikrostoritve ne bodo zmanjšale kaosa; samo razpršile ga bodo po omrežju.

To je tudi razlog, zakaj je trditev "mikrostoritve rešujejo skalabilnost" premalo natančna.

Da, posamezno storitev lahko skaliramo neodvisno.

Da, težko komponento lahko ločimo od manj obremenjenih delov.

Toda tudi monolit lahko horizontalno skaliramo, če le ni preslabo zasnovan, in lahko interno uporablja asinhrono obdelavo, predpomnjenje in delitev podatkovnih obremenitev. Mikrostoritve rešujejo specifične probleme skaliranja, ne pa vseh problemov skaliranja. Pogosto bolj kot strežniško skalabilnost rešujejo organizacijsko skalabilnost.

Zato je praktično vprašanje drugačno: ali je težava v tem, da se aplikacija tehnično ne more skalirati, ali v tem, da se ljudje ne morejo več uskladiti?

Če je glavni problem usklajevanje več ekip, dolgi cikli izdaj in nejasno lastništvo domen, so mikrostoritve lahko smiselne.

Če je problem predvsem neurejena koda, slabe meje modulov in manjkajoči testi, mikrostoritve ne bodo pomagale. Takrat bomo iz slabe monolitne aplikacije naredili še slabše mikrostoritve.

5 Saga: ko rollback ni več ena sama operacija

Klasična ACID transakcija [5] deluje dobro, dokler je meja transakcije dovolj majhna in pod nadzorom enega sistema. V mikrostoritvah pa ima vsaka storitev pogosto svojo bazo in svoj model podatkov.

Globalna transakcija prek več storitev je mogoča, vendar običajno poveča soodvisnost med storitvami, zahteve do infrastrukture in občutljivost sistema na nedosegljivost posameznih komponent. Zato se za številne daljše poslovne procese uporablja vzorec Saga [6]. Izvirni koncept sage je pravzaprav zaporedje transakcij, kjer se ob delni napaki izvedejo kompenzacijske transakcije. To so transakcije, ki popravijo ali izničijo predhodne korake.

V sodobnih mikrostoritvah saga pomeni nekaj podobnega: namesto ene velike transakcije izvedemo več lokalnih transakcij. Vsak korak potrdi svoje spremembe, ob napaki pa sistem izvede poslovno smiselne kompenzacije.

Pri monolitni aplikaciji lahko takšen proces pogosto izvedemo v eni transakciji ACID, zato velja načelo »vse ali nič«. V realnem svetu pa je lahko odpoved leta dražja kot iskanje drugega hotela. Zato kompenzacija ni vedno matematični "undo". Je poslovni postopek. Če ni hotela, lahko ponudimo drug hotel, delni dobropis, človeško intervencijo ali odpoved celotnega paketa. Vzorec kompenzacijskih transakcij (*«compensating transactions»*) opozarja, da kompenzacija v distribuiranih sistemih ni nujno preprosto vračanje korakov v obratnem vrstnem redu, temveč zahteva poslovna pravila.

Vzorec saga poznamo v dveh oblikah: orkestrirana saga in koreografirana saga. Pri orkestrirani sagi obstaja centralni orkestrator. Ta pozna tok procesa, pošilja ukaze posameznim storitvam, spremlja stanje in ob napaki sproži kompenzacije. Prednost je preglednost. Lažje je videti, kje je proces, lažje je dodati nadzor in lažje je razložiti poslovni tok. Slabost je, da orkestrator postane dodatna komponenta, ki jo moramo razvijati, testirati, skalirati in opazovati. Če je slabo zasnovan, lahko postane enotna točka odpovedi (*«single point of failure»*) ali centralna poslovna logika, ki zmanjšuje avtonomijo storitev.

Pri koreografirani sagi centralnega dirigenta ni. Storitve objavljajo dogodke, druge storitve jih poslušajo in nadaljujejo svoje korake. Tak pristop je bolj naraven v dogodkovno vodenih sistemih (*«event-driven systems»*) in ohranja več avtonomije. Vendar ima drugačno ceno: tok procesa je razpršen po več storitvah. Hitro lahko pridemo do množice dogodkov, ki jim sploh ne moremo več slediti in se jih bojimo dotikati. Iz slabe programske kode v monolitni aplikaciji pridemo do nepregledne množice dogodkov, ki jih več ne obvladamo, ker nihče več ne ve, kaj je sprožilo kaj in kateri dogodek je še vedno relevanten... in spet smo pri slabi aplikaciji!

Praktično pravilo je naslednje: če je poslovni proces dolg, kritičen, ima jasen življenjski cikel in ga je treba nadzorovati, je orkestracija pogosto razumljivejša.

Če so dogodki naravni del domene, so koraki šibko sklopljeni in ni enega jasnega lastnika procesa, je koreografija lahko primerna.

V obeh primerih pa saga ni čarobna zamenjava za transakcije. Je eksplicitna odločitev, da konsistentnost dosežemo s procesom, opazovanjem, idempotentnostjo in kompenzacijo.

6 Outbox, inbox in dve hudi težavi: *zombie record* in *ghost message*

Storitev mora pogosto narediti dve stvari: spremeniti podatke v svoji bazi in objaviti sporočilo v sporočilno vrsto.

Če najprej zapiše v bazo in nato pade pred objavo sporočila, dobimo zapis, za katerega drugi deli sistema nikoli ne izvedo. Takšnim zapisom pravimo »*zombie record*«: zapis obstaja, vendar proces, ki bi moral slediti, ni zaživel.

Če pa storitev najprej objavi sporočilo in nato pade pred potrditvijo transakcije v bazi, lahko drugi sistemi reagirajo na dogodek, ki ga baza ne vsebuje. Temu lahko rečemo »*ghost message*«: sporočilo obstaja, poslovno dejstvo pa ne.

Vzorec Outbox rešuje prav ta razkorak. Storitve v isti lokalni transakciji zapiše poslovno spremembo in vrstico v tabelo Outbox. Ločen proces nato bere Outbox in sporočila objavlja v sporočilno vrsto.

Vzorec Outbox ne zagotavlja samodejno dostave ali obdelave »točno enkrat«. Če proces za objavljane pade po uspešni objavi v posredniku, vendar pred označitvijo zapisa kot obdelanega, se lahko sporočilo objavi večkrat. Zato mora potrošniška storitev uporabljati mehanizem Inbox.

Vzorec Inbox je torej druga polovica zgodbe. Prejemnik mora vedeti, ali je sporočilo z določenim identifikatorjem že obdelal. Eden od praktičnih pristopov je tabela ProcessedMessages z enoličnim ključem, na primer kombinacijo SubscriberId in MessageId. Potrošnik ob začetku transakcije vstavi ID sporočila v tabelo obdelanih sporočil; če vnos zaradi primarnega ključa ne uspe, je bilo sporočilo že obdelano in ga lahko ignorira.

Vendar je pri tem ključna razlika med različnimi identifikatorji! MessageId je identiteta konkretnega tehničnega sporočila. BusinessId je identiteta poslovnega dogodka ali entitete. Poznamo pa jih seveda še več: *IssueId*, *ReservationId*, *PaymentId*, *OrderNumber* in podobni.

Če sistem generira nov identifikator pri vsakem ponovnem pošiljanju, deduplikacija po tem identifikatorju ne bo ujela poslovnega dvojnika. Zato mora arhitektura jasno določiti, kateri identifikator rešuje katero težavo. Vem, sliši se skrajno preprosto, a se v praksi izkaže, da temu ni ravno tako!

MessageId pomaga pri tehnični ponovni dostavi istega sporočila. BusinessId pomaga pri vprašanju, ali gre za isto poslovno naročilo, isto prijavo okvare ali isto rezervacijo.

7 Retry, timeout in circuit breaker: varovalke namesto panike

Napake v distribuiranem sistemu niso izjema, ampak del običajnega delovanja.

Povezava se prekine. Storitev potrebuje več časa. Sporočilna vrsta je začasno nedosegljiva. Podatkovna baza naredi »*failover*«. Zato so časovne omejitve (»*timeout*«) in ponovni poskusi osnovna higiena. Brez časovnih omejitev lahko nit čaka predolgo in porabi vire, ki jih potrebujejo drugi deli sistema. Brez ponovnih poskusov pa lahko kratka prehodna napaka povzroči nepotreben poslovni neuspeh.

Toda ponovni poskus mora biti varen. Če ponavljamo branje podatkov, je tveganje majhno. Če ponavljamo plačilo, rezervacijo ali pošiljanje elektronske pošte, pa je to nevarno. Pred tem moramo zagotoviti idempotentnost.

Ponovni poskusi z naraščajočim časovnim zamikom in naključnim odmikom (»*backoff*«, »*jitter*«) zmanjšajo možnost, da bi vse instance sistema ponavljale zahteve v istem ritmu. Ponovne poskuse uporabljajmo predvsem pri prehodnih napakah. Pri trajnih napakah ali neveljavnih zahtevah ponavljanje običajno nima smisla.

Tu moramo namreč biti pazljivi. Predstavljajmo si veliko odjemalcev, ki poskušajo doseči strežnik, ki pa ima določene težave. In ravno, ko se strežnik »pobere«, ga spet zasujejo vsi odjemalci, zato strežnik spet »počepne«. Ponovni poskusi morajo torej imeti naključno dolg vmesni čas čakanja in ta čas naj postopoma raste.

Odklopnik (*»circuit breaker«*) rešuje drugo plat tega problema. Če odvisna storitev očitno ne deluje, nima smisla, da jo vsaka zahteva znova obremenjuje. Odklopnik po določenem pragu napak zadevo prekine in klice začasno zavrne ali preusmeri.

Kasneje v napol odprtem stanju (*»half-open«*) dovoli omejeno število poskusov, da preveri, ali se je storitev popravila. Odklopnik preprečuje ponavljajoče izvajanje operacije, ki bo verjetno neuspešna, in s tem pomaga preprečiti kaskadne napake.

Pregledi zdravja storitve (*»health checks«*) so naslednji del te zgodbe. Storitev mora znati povedati, ali deluje in ali je pripravljena sprejemati promet. Končna točka za preverjanje zdravja ni samo privlačen naslov URL *»/health«*. Je dogovor med aplikacijo, orkestratorjem, uravnavalnikom obremenitev (*»load balancer«*) in nadzornikom.

Vse omenjene tehnike – časovna omejitev, ponovni poskus, odklopnik in preverjanje zdravja – morajo biti usklajene. Kratke časovne omejitve brez ponovnih poskusov lahko povzročijo preveč napak. Prepogosti ponovni poskusi brez naključnih zakasnitev pa lahko povzročijo kaskadno odpoved sistemov zaradi prevelikega prometa.

Zato jih ne smemo obravnavati kot knjižnične nastavitve, temveč kot del arhitekturnega dogovora: katere operacije se smejo ponoviti, koliko časa smejo trajati, kdaj odpovemo hitro in kako operaterji vidijo stanje sistema.

8 Vrstni red, verzije, stanja in eventualna konsistentnost

Dogodki v mikrostoritvah pogosto ne pridejo v vrstnem redu, ki ga pričakujemo. Vzrok je lahko sporočilna vrsta, več pošiljateljev, paralelizem, ponovna dostava, različne poti dogodkov ali preprosto različen čas obdelave. Če prejemnik nekritično sprejme vsak dogodek, lahko starejši dogodek prepiše novejšo stanje. Zato mora dogodek nositi dovolj informacij, da prejemnik sprejme pravilno odločitev.

Najpogostejši mehanizmi so verzije, zaporedne številke in grafi stanja (*»state machine«*). Če ima entiteta verzijo, lahko potrošnik dogodek verzije X ignorira, če je že obdelal dogodek verzije X+1.

Če ima tok dogodkov zaporedno številko, lahko zazna luknjo in sproži ponovno sinhronizacijo. Če ima proces eksplicitna stanja, se lahko sistem odloči, da prehod iz "Dostavljeno" nazaj v "V prevozu" ni veljaven, razen če gre za poseben poslovni scenarij.

Konsistentnost dogodkov je pri tem pogosto nujna, vendar mora biti vidna. Slabo je, če uporabnik misli, da je sprememba naslova takoj veljavna v vseh sistemih, sistem pa v resnici potrebuje minute ali ure, da sinhronizira podatkovne baze. Bolje je, da domena jasno pove, kaj je vir resnice in kdaj so drugi pogledi posodobljeni.

Pri usklajevanju stanj so koristna tudi periodična opravila usklajevanja (*reconciliation jobs*). Ta primerjajo stanje med sistemi, odkrijejo odstopanja in jih usmerijo v samodejno ali ročno popravljanje. Če izhodno sporočilo ni bilo objavljeno, ga lahko opravilo pozneje ponovno objavi. Če potrošnik dogodka ni obdelal, se lahko obdelava ponovi. Če se stanje dveh sistemov še vedno razlikuje, opravilo odstopanje evidentira in sproži ustrezen popravilni postopek.

Pomembno je, da takšna opravila niso priznanje poraza. So priznanje, da distribuirani sistem ne more temeljiti na popolni gotovosti!

9 Single point of failure: mikrororitve ne odpravijo centralnih odvisnosti

Pogosta iluzija je, da mikrororitve samodejno pomenijo večjo odpornost. V resnici lahko hitro ustvarimo sistem, kjer je vse razdeljeno, razen ene kritične točke. Lahko imamo deset storitev, vse pa potrebujejo isto sporočilno vrsto ali pa istega SAGA koordinatorja. Lahko imamo ločene baze, vse pa uporabljajo isti DNS ali isto Kubernetes gručo...

Zato je treba enotno točko odpovedi obravnavati širše kot samo "ena baza" ali "ena sporočilna vrsta". Gre za vsako komponento, brez katere sistem ne more opravljati ključne poslovne funkcije. Včasih je rešitev redundanca: gruča, replike, visoka razpoložljivost, rezervna pot. Včasih je rešitev degradacija: če priporočilni sistem ne deluje, lahko osnovna funkcionalnost vseeno deluje. Včasih je rešitev izolacija: napaka v poročilih ne sme ustaviti obdelave kritičnih prijav.

Zanesljiva izmenjava sporočil zahteva več kot samo njihovo shranjevanje. Pomembno je razlikovati med tem, ali sporočilo preživi ponovni zagon sistema, ali je posrednik trenutno dosegljiv in kako sistem razume uspešno obdelavo sporočila. Obstojna sporočila, trajne čakalne vrste ter potrditve pošiljanja in prejema zmanjšujejo tveganje izgube podatkov.

Kljub temu mora sistem ob nedosegljivosti posrednika določiti, ali bo sporočilo shranil lokalno v odhodni predal in ga poslal pozneje, ali pa bo uporabniku vrnil napako. Prav tako mora določiti, kaj se zgodi, če storitev, ki naj bi sporočilo obdelala, ne deluje: sporočilo lahko nekaj časa čaka, ali pa se sproži eskalacijski postopek.

Orkestrirana saga lahko postane ranljiva, če orkestrator izgubi stanje in po ponovnem zagonu ne more ugotoviti, kateri koraki so bili že izvedeni. Pomembno je tudi, da ponovna izvedba istega ukaza ne povzroči neželenih učinkov. Kljub temu koreografija ni nujno varnejša, saj lahko pomembno odvisnost skrije drugje, na primer v posredniku sporočil, skupnih pravilih za dogodke ali znanju posamezne ekipe.

Pri izbiri arhitekture zato ni najpomembnejše vprašanje, ali imamo centralno ali decentralizirano rešitev. Pomembneje je razumeti, kje so odvisnosti, kdo jih vzdržuje in kaj se zgodi, ko odpovejo.

10 Predlog: ne zapletajte po nepotrebnem

Mikrororitve so uporabne, kadar rešujejo resničen problem avtonomije, lastništva in neodvisnega uvajanja. Če imamo tri razvijalske ekipe, to še ne pomeni, da mora vsaka ekipa dobiti svojo mikrororitev, svojo bazo, svoje sporočilne pogodbe, svoje incidente in svojo omrežno zakasnitev.

Pogosto je boljši prvi korak modularni monolit. To je sistem z jasno ločenimi moduli, notranjimi mejami, ločenimi imenskimi prostori, jasnim lastništvom kode in po možnosti ločenimi shemami ali vsaj strogo nadzorovanimi dostopi do podatkov. Moduli lahko komunicirajo prek dobro definiranih vmesnikov ali notranjih dogodkov, vendar ostanejo v istem procesu. Če se kasneje izkaže, da mora nek modul živeti samostojno, ga lažje izločimo, ker so meje že narisane.

Ali so NuGet/npm/pip/maven/... paketi mikrororitve? Seveda niso. Vendar nas prisilijo v razmislek.

Knjižnica je lahko samostojna komponenta, ima lastnika, verzije, javni API in testni paket. Razlika je, da jo kličemo lokalno in pošljemo skupaj z aplikacijo. Če je cilj samo razdeliti odgovornost med razvojne ekipe, je knjižnica pogosto dovolj. Če je cilj ločeno skaliranje, ločeno uvajanje, ločen tehnološki sklad ali različna operativna odgovornost, je storitev bolj smiselna.

Storitve so komponente zunaj procesa in oddaljeni klici so dražji kot lokalni klici, zato morajo biti vmesniki bolj grobozrnati. To je praktičen kriterij. Če bodo nove mikrororitve samo nadomestile vsak lokalni klic s klicem REST, bomo dobili počasnejši in bolj krhek sistem. Če pa storitev predstavlja jasno poslovno zmožnost, ima svoje podatke, svoj življenjski cikel in svojo ekipo, je komunikacijski strošek lahko upravičen.

11 Zaključek: arhitektura kot odločitev o stroških

Mikrostoritve same po sebi niso ne dobre ne slabe; njihova uvedba je predvsem odločitev o tem, katere kompromise je organizacija pripravljena sprejeti.

Pri monolitni arhitekturi moramo sprejeti skupno uvajanje celotne aplikacije, večji obseg kode in nevarnost, da notranje meje med funkcionalnostmi niso ustrezno določene.

Pri mikrostoritvah se ti izzivi prenesejo na omrežno komunikacijo, sporočilne sisteme, sčasoma doseženo konsistentnost podatkov ter zahtevnejše upravljanje in odpravljanje napak. Delovanje sistema je zato tudi manj predvidljivo.

Modularni monolit poskuša začasno združiti prednosti obeh svetov: jasne meje brez distribuirane kompleksnosti.

Pri uporabi mikrostoritev je smiselno upoštevati naslednja pravila:

1. Mikrostoritev ne izbirajmo zgolj zato, ker jih uporabljajo velika podjetja.
2. Aplikacije ne razdelimo na posamezne storitve, dokler ne razumemo domenskih meja in odgovornosti.
3. Poslovno pomembnih dogodkov ne objavljajmo brez vzorca Outbox oziroma zanesljivega lokalnega shranjevanja odhodnih sporočil.
4. Pri obdelavi sporočil zagotovimo idempotentnost, da ponovna obdelava ne povzroči neželenih učinkov.
5. Ponovnih poskusov ne izvajajmo brez časovnih omejitev, omejenega števila poskusov in ustreznega zamika med njimi.
6. Sage ne uvajajmo, dokler ne razumemo kompenzacijskih dejanj, potrebnih ob neuspehu posameznega koraka.
7. Zavedajte se, da bo prišlo do neusklajenosti podatkov in to sprejmite ter ustrezno ukrepajte že vnaprej.

Najboljša arhitektura ni tista z največ vzorci. Najboljša arhitektura je tista, pri kateri so vzorci uporabljeni tam, kjer rešujejo konkreten problem. Včasih je to saga. Včasih je to Outbox. Včasih je to samo dobro oblikovana knjižnica v istem procesu.

Arhitektura ni tekmovanje v številu škatlic na diagramu!

Literatura

- [1] Two Generals' problem. Wikipedia. obiskano 8. 7. 2026, https://en.wikipedia.org/wiki/Two_Generals%27_Problem
- [2] Byzantine fault, Wikipedia. obiskano 8. 7. 2026, https://en.wikipedia.org/wiki/Byzantine_fault
- [3] Spaghetti code, Wikipedia. obiskano 8. 7. 2026, https://en.wikipedia.org/wiki/Spaghetti_code
- [4] Conway, M. E. (1968). How do committees invent? Datamation, 14(4), 28-31.
https://www.melconway.com/Home/Committees_Paper.html
- [5] ACID, Wikipedia. Obiskano 08.07.2026, <https://en.wikipedia.org/wiki/ACID>
- [6] Long-running transaction, Wikipedia. Obiskano 08.07.2026, https://en.wikipedia.org/wiki/Long-running_transaction

Zrelost cevovodov neprekinjene dostave informacijskih rešitev

Luka Četina, Luka Pavlič

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
luka.cetina@um.si, luka.pavlic@um.si

DevOps se je uveljavil kot pomemben pristop k sodobnemu razvoju informacijskih rešitev, saj povezuje razvojne in operativne vidike ter organizacijam omogoča hitrejši, zanesljivejši razvoj. Ker DevOps ne predstavlja enotne metode, organizacije prakse uvajajo postopoma in različno, težko pa ocenijo, kako zreli so njihovi procesi in katere izboljšave so najbolj nujne. Obstoječi modeli zrelosti DevOps pri tem pomagajo, a so praviloma usmerjeni na organizacijsko raven, kulturo in procese, ocenjevanje pa temelji na samooceni s splošno opisanimi kriteriji, kar otežuje primerljivost ocen med ekipami. Manj pozornosti namenjajo konkretni tehnični izvedbi avtomatizacije – cevovodom neprekinjene dostave, prek katerih se velik del DevOps praks dejansko izvaja. Prispevek predstavlja preliminarno zasnovo modela za ocenjevanje zrelosti CI/CD cevovodov, ki temelji na kriterijih, preverljivih neposredno v konfiguraciji cevovoda. Obrazec zajema šest razsežnosti (avtomatizacija gradnje in integracije, testiranje in analiza kode, avtomatizacija izdaje in namestitve, infrastruktura kot koda, spremljanje izvajanja cevovoda ter varnost dobavne verige), vsaka s petimi kriteriji, ocenjenimi s 0 do 2 točkama. Skupna ocena (0–60 točk) se preslika v pet ravni zrelosti, razčlenitev po razsežnostih pa pokaže, kje je cevovod dodelan in kje zaostaja. Predlagani model predstavlja dopolnilo obstoječim modelom zrelosti DevOps, saj omogoča sistematično, ponovljivo in potencialno avtomatizirano ocenjevanje tehnične zrelosti posameznega CI/CD cevovoda.

Ključne besede:

CI/CD cevovodi

zrelost cevovodov

DevOps

model zrelosti

neprekinjena dostava

1 Uvod

Prakse neprekinjene integracije, dostave in namestitve (angl. continuous integration, continuous delivery, continuous deployment – CI/CD) so danes med temeljnimi gradniki sodobnega razvoja informacijskih rešitev. Neprekinjena integracija zahteva pogosto in avtomatizirano združevanje kode in tako zmanjšuje tveganje integracijskih napak, neprekinjena dostava pa ta koncept nadgradi z avtomatiziranim testiranjem in preverjanjem kakovosti, tako da je aplikacija vedno pripravljena na produkcijsko namestitev [1]. Neprekinjena namestitev gre še korak dlje in produkcijsko namestitev izvede samodejno, brez ročnega posredovanja [1]. Te prakse se izvajajo prek CI/CD cevovodov, ki avtomatizirajo aktivnosti, kot so gradnja, testiranje, analiza kode in namestitev [2].

CI/CD cevovodi so pomemben del metodologije DevOps, saj povezujejo razvojne in operativne vidike ter organizacijam omogočajo hitrejši, zanesljivejši in manj naporen razvoj [3]. Avtomatizacija teh aktivnosti pospešuje izdajne cikle, izboljšuje kakovost programske opreme in povečuje produktivnost razvijalcev [4]. Kljub tem koristim pa vzpostavitev in razvoj cevovodov v praksi neredko ostajata precej nenačrtovana, saj se cevovod pogosto razvija glede na trenutne potrebe, namesto da bi sledil strateškemu načrtu [5].

Da bi lahko razvoj cevovoda načrtovali in ga postopoma izboljševali, je treba najprej ugotoviti, v kakšnem stanju cevovod dejansko je [6]. Vendar za ocenjevanje njegove zrelosti ni uveljavljenega enotnega pristopa, saj ni jasno opredeljeno, katere značilnosti mora cevovod izpolnjevati, da bi ga lahko šteli za zrelega oziroma skladnega z načeli neprekinjene dostave [7]. Obstoječi modeli zrelosti DevOps sicer ponujajo okvir za ocenjevanje uvajanja DevOps praks, vendar praviloma obravnavajo organizacijo ali razvojno ekipo kot celoto. Njihovo ocenjevanje temelji predvsem na intervjujih ali samoocenjevalnih vprašalnikih [8], zato so primerni za presojo organizacijskih procesov, kulture in sodelovanja, bistveno manj pa za objektivno oceno tehnične izvedbe posameznega CI/CD cevovoda.

Ta prispevek predstavlja preliminarno zasnovo modela za ocenjevanje zrelosti CI/CD cevovodov, ki temelji na kriterijih, razvidnih neposredno iz konfiguracije cevovoda. Za razliko od obstoječih modelov zrelosti DevOps, ki ocenjujejo predvsem organizacijsko raven, predlagani pristop omogoča ocenjevanje tehnične zrelosti posameznega cevovoda brez uporabe intervjujev ali samoocenjevalnih vprašalnikov. Takšna ocena ekipam omogoča prepoznavanje konkretnih tehničnih vrzeli, spremljanje napredka skozi čas ter primerjavo cevovodov na enotni osnovi. V prispevku predstavimo razsežnosti predlaganega modela, način ocenjevanja in točkovanja ter njegovo uporabo na primeru hipotetičnega cevovoda. Ker gre za konceptualno zasnovo modela, predstavljeni kriteriji in pragovi še niso empirično validirani ter predstavljajo izhodišče za nadaljnji razvoj.

2 Ozadje in sorodna dela

2.1. CI/CD cevovodi kot koda

CI/CD cevovod je zaporedje avtomatiziranih korakov, ki se sprožijo ob dogodkih v razvojnem procesu (npr. nalaganju kode ali odprtju zahteve za združitev) in izvedejo aktivnosti, kot so gradnja, testiranje, analiza kode, izdaja in namestitev [2], [9]. Ključna lastnost sodobnih cevovodov je, da so vse pogostejše definirani deklarativno, v datotekah, ki se hranijo skupaj s programsko kodo (npr. GitHub Actions, GitLab CI, Azure DevOps) [5]. Cevovod tako postane samostojen izdelek, ki ga je mogoče verzionirati, pregledovati in sistematično ocenjevati na podlagi njegove konfiguracije [10], [5], [11]. Ta predstavlja strukturiran, strojno berljiv vir podatkov o tem, kaj cevovod vključuje oziroma avtomatizira.

2.2. Modeli zrelosti DevOps in njihove omejitve za oceno cevovoda

Vprašanje zrelosti sicer v okviru DevOps ni novo. Vzporedno s CI/CD cevovodi se je razvila vrsta modelov zrelosti DevOps, ki organizacijam pomagajo oceniti, kako celovito so DevOps prakse vpeljane [7]. Med bolj znanimi sta model Humble in Farley, ki zrelost povezuje s stopnjo avtomatizacije konfiguracije izdaj in namestitev [12], ter na sistematičnem pregledu literature zasnovan model Teixeira idr. [13], ki DevOps zrelost opredelijo prek

štirih razsežnosti: ljudje, proces, tehnologija in kultura. Sethupathy [14] je nedavno predstavil okvir zrelostnega modela, ki združuje elemente vitke (ang. lean) in agilne zrelosti z DevOps prakso.

Pregled večjega števila takih modelov [8], [15] pokaže nekaj ponavljajočih se značilnosti: modeli so praviloma večdimenzionalni, zajemajo tehnične, procesne in kulturne vidike, imajo približno pet ravni zrelosti, ocenjevanje pa največkrat temelji na samooceni prek vprašalnikov ali intervjujev. Kriteriji za posamezne ravni so pogosto opisani splošno (npr. "standardiziran proces"), kar otežuje objektivno in ponovljivo ocenjevanje. Merljivi kazalniki, kot jih predlagata DORA in Accelerate (pogostost namestitvev, čas do uveljavitve spremembe, čas okrevanja, delež neuspešnih sprememb) [16], [17], dajejo vpogled v učinkovitost dostave, a ne pojasnijo, katere avtomatizacijske zmožnosti cevovodu manjkajo. Namenjeni so predvsem merjenju rezultatov procesa razvoja programske opreme in ne neposredni oceni tehnične implementacije CI/CD cevovoda.

Podobne omejitve veljajo za pobude, usmerjene specifično v cevovode. Model zrelosti neprekinjene dostave podjetja AWS, prilagojen po zgledu CMM [18], ponuja okvirne stopnje zrelosti od osnovne avtomatizacije do optimizirane neprekinjene dostave, vendar brez konkretnih kriterijev za zaznavanje posameznih zmožnosti [19]. Podobno velja za varnostno usmerjeno pobudo SLSA, ki opredeljuje ravni zrelosti za sledljivost gradnje in celovitost dobavne verige, a se osredotoča le na varnost [20], ter za OpenSSF Scorecard, ki repozitorije samodejno ovrednoti glede na varnostne in vzdrževalne dobre prakse, brez opredeljenih razsežnosti zrelosti [21].

Tudi novejša sorodne raziskave večinoma ostajajo na organizacijski ravni: DevOps-RAF ocenjuje pripravljenost organizacije na DevOps prek agilnega razvoja, operativnega dela in kulture [22], model SAMAM naslavlja predvsem uvajanje ekstremnega programiranja [23], model ASPLA pa zrelost agilnih produktnih linij v avtomobilski industriji [24]. Akbar idr. [25] obravnavajo človeški dejavnik pri DevSecOps, Wang idr. [26] so pokazali povezavo med zrelostjo avtomatizacije testiranja in kakovostjo izdelka, a le za eno aktivnost cevovoda, Zarour idr. [27] pa so raziskali sprejemanje DevOps procesov, ne pa zrelosti tehnične izvedbe.

Skupni imenovalac navedenih del je torej, da zrelost obravnavajo na ravni organizacije, procesa ali posamezne prakse, ne pa na ravni CI/CD cevovoda, ki DevOps avtomatizacijo dejansko izvaja. Prav ta vrzel je izhodišče modela, predstavljenega v nadaljevanju.

3 Predlagani model zrelosti CI/CD cevovodov

Zrelost CI/CD cevovoda v tem prispevku opredeljujemo kot stopnjo, do katere cevovod dosledno, ponovljivo in preverljivo izvaja aktivnosti neprekinjene integracije, dostave in namestitve, usklajene z uveljavljenimi praksami DevOps. Za razliko od obstoječih modelov zrelosti DevOps [8], [12], [13], [14], predlagani model zrelost ocenjuje na ravni posameznega cevovoda, izključno na podlagi tega, kar je razpoznavno iz njegove konfiguracijske datoteke (npr. delovni tok GitHub Actions, datoteka GitLab CI .yaml ali cevovod Azure DevOps) [5]. Model je zasnovan kot ocenjevalni obrazec in vsebuje nabor konkretnih, preverljivih kriterijev, razporejenih v razsežnosti, ki jih ocenjevalec preveri na dejanskem cevovodu in za vsakega dodeli točke. Vsota točk daje oceno zrelosti razsežnosti in na koncu skupno oceno zrelosti cevovoda.

Podobno kot novejši modeli zrelosti DevOps, ki ocenjevanje gradijo na konkretnih zmožnostih in točkovanju namesto zgolj opisnih ravni [14], tudi predlagani model kriterije formulira tako, da je odgovor nanje mogoče preveriti, ne le subjektivno oceniti. Razlika je v tem, da model ostaja izključno na ravni cevovoda: ne vključuje kriterijev, ki bi zahtevali vpogled v organizacijsko kulturo, sodelovanje ali kadrovske vidike, saj ti niso razvidni iz konfiguracije.

V nadaljevanju opišemo razsežnosti in njihove kriterije ter pristop ponazorimo na primeru hipotetičnega cevovoda.

3.1 Razsežnosti modela

Izbira razsežnosti izhaja iz dveh virov: pregleda obstoječih modelov zrelosti DevOps [7], [8], [12], [13], [14], [15], [18], [22], [23], [24], [25], [26] in omejitve, da je vsak kriterij mogoče preveriti neposredno iz konfiguracije cevovoda.

Pregled pokaže, da so med najpogostejše vključenimi razsežnostmi avtomatizacija (v vseh modelih), kultura, sodelovanje in proces (v skoraj vseh), spremljanje (angl. monitoring), arhitektura oziroma infrastruktura ter kakovost/testiranje, medtem ko je varnost kot samostojna razsežnost prisotna le v manjšini modelov [14]. To kaže, da modeli zrelosti DevOps varnost dobavne verige sistematično podcenjujejo, čeprav so orodja za njeno avtomatizirano preverjanje (npr. SLSA [20], OpenSSF Scorecard [21]) že razmeroma uveljavljena in, za razliko od marsikaterega drugega vidika DevOps prakse, neposredno razvidna iz konfiguracije cevovoda. Varnost dobavne verige zato v predlaganem obrazcu dobi status samostojne razsežnosti.

Kakovost programske kode (npr. statična analiza, upoštevanje standardov kodiranja) v predlaganem obrazcu ni samostojna razsežnost, čeprav se v pregledanih modelih pojavlja pogostejše kot varnost [15]. Namesto tega je zajeta kot del razsežnosti "Testiranje in analiza kode". Tak pristop – kakovost kode kot sestavni del širše razsežnosti kakovosti oziroma testiranja – je uveljavljen tudi v več pregledanih modelih [8], [15].

Na podlagi navedenega predlagamo šest razsežnosti obrazca:

- *Avtomatizacija gradnje in integracije*: ali cevovod samodejno preverja in združuje spremembe kode.
- *Testiranje in analiza kode*: ali cevovod samodejno preverja funkcionalno pravilnost, pokritost s testi in kakovost izvirne kode.
- *Avtomatizacija izdaje in namestitve*: ali cevovod samodejno pripravi in namesti izdelek.
- *Infrastruktura kot koda*: ali je infrastruktura, na katero se namešča izdelek, definirana in nadzorovana skozi cevovod.
- *Spremljanje izvajanja cevovoda*: ali je izvajanje cevovoda mogoče spremljati in ali se ob napakah samodejno obvešča.
- *Varnost dobavne verige*: ali cevovod samodejno preverja varnost odvisnosti, artefaktov in samega procesa gradnje.

3.2. Kriteriji in način ocenjevanja

Za vsako od šestih razsežnosti predvidevamo nabor konkretnih kriterijev, ki jih je mogoče preveriti neposredno v konfiguracijski datoteki cevovoda, brez potrebe po dodatnem razgovoru ali dostopu do organizacije. Vsak kriterij se oceni s tremi možnimi vrednostmi:

0 točk - kriterij v konfiguraciji ni zaznan (korak, sprožilec ali mehanizem ne obstaja);

1 točka – kriterij je zaznan delno (npr. korak obstaja, vendar ne za vse veje/okolja, ali je nastavljen zgolj informativno brez vpliva na izvajanje);

2 točki – kriterij je v celoti izpolnjen (korak je prisoten, se izvaja dosledno in dejansko vpliva na potek oz. izid cevovoda).

Ocena razsežnosti je vsota točk njenih petih kriterijev (0–10 točk), skupna ocena cevovoda pa vsota šestih razsežnosti (0–60 točk). Primeri kriterijev za razsežnost "Avtomatizacija gradnje in integracije" so prikazani v tabeli 1.

Tabela 1: Primer kriterijev za razsežnost "Avtomatizacija gradnje in integracije"

Št.	Kriterij
1	Cevovod se samodejno sproži ob potisu kode ali zahtevi za združitev, ne le ročno.
2	Neuspešna gradnja samodejno prepreči združitev v ciljno vejo.
3	Izdelen artefakt se shrani sledljivo (npr. z oznako različice).
4	Odvisnosti se namestijo samodejno, z zaklenjenimi/fiksiranimi različicami (lockfile).
5	Gradnja je predpomnjena oz. optimizirana (cache korakov, vzporedni jobi).

Za lažjo interpretacijo se točke preslikajo v eno od petih opisnih ravni. Najprej za vsako razsežnost posebej, šele nato se te sestavijo v skupno oceno, saj bi slednja sama lahko prikrila, da je cevovod na eni razsežnosti zelo dodelan, na drugi pa komaj vzpostavljen.

3.3. Ravni zrelosti

Skupno število točk (0–60) se za lažjo interpretacijo preslika v pet opisnih ravni zrelosti, ki so po zgledu večine pregledanih modelov zrelosti DevOps oblikovane naraščajoče, od povsem ročnega oziroma priložnostnega stanja do optimiziranega cevovoda [7], [8], [13]:

Raven 1 – začetna (0–12 točk): cevovod obstaja zgolj v osnovni obliki ali ga (še) ni; večina aktivnosti, vključno s pregledom kode, se izvaja ročno zunaj cevovoda.

Raven 2 – osnovna (13–24 točk): cevovod avtomatizira gradnjo in del testiranja, druge razsežnosti (namestitve, infrastruktura, varnost, spremljanje) so še večinoma ročne ali odsotne.

Raven 3 – definirana (25–36 točk): večina razsežnosti je delno avtomatizirana, vendar neenakomerno – npr. testiranje in gradnja sta dodelana, medtem ko varnost dobavne verige ali spremljanje izvajanja zaostajata.

Raven 4 – napredna (37–48 točk): cevovod dosledno avtomatizira večino aktivnosti po vseh razsežnostih, vključno z varnostnimi preverjanji in obveščanjem ob napakah; ročni koraki so izjema.

Raven 5 – optimizirana (49–60 točk): cevovod celovito avtomatizira aktivnosti po vseh razsežnostih, uporablja napredne prakse, kot so postopne izdaje, samodejna povrnitev na prejšnjo različico in sledljivo poreklo artefaktov, ter samodejno zaznava napake in nepravilnosti med izvajanjem.

Enak razpon ravni je smiselno uporabiti tudi na ravni posamezne razsežnosti (0–10 točk, pragovi sorazmerno nižji: 0–2, 3–4, 5–6, 7–8, 9–10), kar omogoča, da se poleg skupne ocene identificira tudi razsežnosti, kjer je cevovod razmeroma šibkejši – kot že omenjeno, je to za praktično uporabo modela pogosto bolj informativno kot sama skupna ocena. Tabela 2 za vsako od šestih razsežnosti opisuje, kaj naj bi jo opredeljevalo na posamezni od petih ravni.

Pri tem najvišja raven ni nujno cilj za vsak cevovod oziroma vsako organizacijo. Za majhen projekt ali zgodnjo fazo razvoja je lahko smiselno ciljati na 2. ali 3. raven, saj so stroški vzpostavitve naprednejših praks (npr. postopnih izdaj, izvoza metrik v namensko nadzorno ploščo) lahko nesorazmerni s tveganjem, ki ga cevovod dejansko nosi. Za cevovode, ki namestitve izvajajo v produkcijsko okolje z visoko izpostavljenostjo (npr. plačilni sistemi, javno dostopne storitve z veliko uporabniki), pa je pričakovati, da bi morali stremeti vsaj proti 4. ravni, predvsem na razsežnostih varnosti dobavne verige in spremljanja izvajanja, kjer so tveganja največja. Podobno razlikovanje ciljne ravni glede na kontekst organizacije je uveljavljeno tudi pri obstoječih modelih zrelosti DevOps [7], [8].

Tabela 2: Opis petih ravni zrelosti po posamezni razsežnosti obrazca

Razsežnost	Raven 1: Začetna	Raven 2: Osnovna	Raven 3: Definirana	Raven 4: Napredna	Raven 5: Optimizirana
Avtomatizacija gradnje in integracije	gradnja ročna, brez samodejnega sprožilca.	samodejna gradnja ob spremembi kode; neuspeh ne prepreči združitve, odvisnosti niso zaklenjene.	uspešna gradnja je pogoj za združitev (build gate); odvisnosti so zaklenjene na fiksne različice.	gradnja je pospešena s predpomnjenjem in vzporednim izvajanjem; gradijo se le spremenjeni deli.	podpora gradnji za več okolij/konfiguracij hkrati prek skupnih predlog (pipeline-as-code).
Testiranje in analiza kode	testi (če obstajajo) potekajo ročno; statične analize ni.	samodejni testi enot in osnovna statična analiza, oba le informativna.	neuspeh testa ustavi cevovod (test gate); tudi kršitve pravil statične analize blokirajo združitev.	dodani integracijski/sistemski testi, prag pokritosti kode; strožja pravila statične analize (npr. varnostna) so obvezujoča.	testiranje na matriki okolij ali dodatni tipi testov; trend kakovosti kode se spremlja skozi čas.
Avtomatizacija izdaje in namestitve	izdaja in namestitev sta ročni.	cevovod samodejno pripravi artefakt (npr. vsebnik), namestitev ostaja ročna.	namestitev v testno okolje je avtomatizirana, v produkcijo pa zahteva ročen postopek.	namestitev v produkcijo je avtomatizirana ali zahteva le eno potrditev; artefakti so sledljivi v registru.	napredne strategije izdaje s samodejnim rollbackom ob napaki.
Infrastruktura kot koda	infrastruktura je konfigurirana ročno.	infrastruktura je opisana kot koda (npr. Terraform), ločeno od cevovoda, brez preverjanja.	cevovod preveri veljavnost infrastrukturne kode (lint/plan) pred uveljavitvijo, ki je še delno ročna.	uveljavitev sprememb infrastrukture je v celoti avtomatizirana; konfiguracija in skrivnosti so ločene od kode.	infrastrukturna koda je upravljana po različicah in sledljiva (remote state) ter gre skozi enak pregled kot aplikacijska koda.
Spremljanje izvajanja cevovoda	o izidu izvajanja se ekipa seznani le naključno.	samodejno obvestilo ob neuspehu (e-pošta, Slack), dnevniki se ne arhivirajo posebej.	status izvajanja je viden zunaj cevovoda (značka); dnevniki in poročila se hranijo kot artefakt.	cevovod izvaža metrike delovanja (čas, uspešnost) v namensko nadzorno ploščo.	zaznava običanih/nenavadno dolgih izvajanj in samodejno opozarjanje na odstopanja.
Varnost dobavne verige	odvisnosti/artefakti se ne preverjajo, skrivnosti so lahko v kodi.	skrivnosti so upravljane prek namenskega mehanizma orodja CI/CD, preverjanja ranljivosti še ni.	korak za preverjanje znanih ranljivosti v odvisnostih (SCA orodje).	preverjajo se tudi ranljivosti gradbenih artefaktov/vsebnikov in izvor paketov.	artefakti imajo sledljivo poreklo (npr. SBOM, podpis, poročilo skladno z okvirom SLSA [20]).

3.4. Primer ocenjevanja

Predlagani obrazec za primer uporabimo na hipotetičnem, a tipičnem cevovodu manjše spletne aplikacije v orodju za CI/CD (npr. GitHub Actions):

"Cevovod se samodejno sproži ob vsakem potisu kode; uspešna gradnja je pogoj za združitev, odvisnosti so zaklenjene, predpomnjenja ali vzporednega izvajanja pa ne uporablja. Izvajajo se testi enot in preverjanje sloga kode, integracijski testi in pragovi pokritosti niso vključeni. Cevovod zgradi vsebnik in ga označi z različico ter ga

samodejno namesti v testno okolje; namestitve v produkcijo zahteva ročno potrditev, naprednejših strategij izdaje pa ne uporablja. Infrastruktura je opisana v Kubernetes manifestih, ki pred uveljavitvijo niso samodejno preverjeni; skrivnosti so upravljane prek vgrajenega mehanizma orodja za CI/CD. Ob napaki cevovod pošlje obvestilo v Slack, ne izvaža pa metrik o delovanju niti ne zaznava običajnih izvajanj. Preverjanje ranljivosti odvisnosti, skeniranje artefaktov in beleženje porekla (SBOM) niso vključeni."

Na podlagi opisanega cevovoda posamezne razsežnosti ocenimo, kot prikazuje tabela 3.

Tabela 3: Ocena primera cevovoda po razsežnostih

Razsežnost	Ocena (od 10)
Avtomatizacija gradnje in integracije	8
Testiranje in analiza kode	4
Avtomatizacija izdaje in namestitve	7
Infrastruktura kot koda	4
Spremljanje izvajanja cevovoda	2
Varnost dobavne verige	2
Skupaj (od 60)	27

Skupna ocena cevovoda je tako 27 od 60 točk, kar ga uvršča v raven 3 – definirana. Ocena po razsežnostih (tabela 3) hkrati pokaže, kje je cevovod razmeroma dodelan (gradnja in integracija: 8/10) in kje precej zaostaja (spremljanje izvajanja in varnost dobavne verige: po 2/10). Taka razčlenitev je za ekipo uporabnejša od same skupne ocene, saj neposredno nakazuje, kam usmeriti naslednje izboljšave. V tem primeru najprej v uvedbo preverjanja ranljivosti odvisnosti in izvoz osnovnih metrik o delovanju cevovoda, saj gre za razmeroma enostavne posege z velikim vplivom na oceno.

3.5. Predviden način uporabe

Ideja predlaganega modela je, da ga lahko izpolni razvijalec ali vodja ekipe v razmeroma kratkem času, brez posebnega orodja, saj zadostuje vpogled v konfiguracijsko datoteko cevovoda in nastavitve orodja za CI/CD (npr. zaščita vej, register skrivnosti). Prepoznavanje kriterija je zasnovano kot preverjanje konkretnega, v konfiguraciji vidnega dejstva: npr. za kriterij "ali neuspešen test prepreči združitev kode" ocenjevalec preveri, ali je v nastavitvah zaščite veje vključeno zahtevano preverjanje stanja (angl. required status check) oziroma ali korak testiranja v konfiguraciji nima nastavljenih možnosti za prezrtje napake (npr. allow_failure). Predviden postopek obsega tri korake: za vsak kriterij se preveri, ali je izpolnjen v celoti, delno ali sploh ne (0, 1 ali 2 točki); točke se seštevajo po razsežnostih in skupno, kar da profil zrelosti cevovoda; na podlagi profila ekipa prepozna razsežnosti z najnižjo oceno in jih prednostno naslovi, saj profil pove ne le, "kako zrel" je cevovod, temveč tudi, kje so vrzeli.

4 Praktične implikacije in omejitve

Predlagani model predstavlja praktično orodje za sistematično ocenjevanje tehnične zrelosti CI/CD cevovodov. Za razliko od obstoječih modelov zrelosti DevOps omogoča neposredno oceno tehničnih zmožnosti posameznega cevovoda na podlagi njegove konfiguracije. Razvijalcem in ekipam tako omogoča hitro prepoznavanje področij, kjer avtomatizacija še ni zadostno razvita, ter lažje načrtovanje nadaljnjih izboljšav.

Ker model podaja oceno po posameznih razsežnostih in ne zgolj skupne ocene, omogoča natančnejšo identifikacijo prednosti in pomanjkljivosti posameznega cevovoda. Tak pristop lahko organizacije uporabijo pri primerjavi različnih projektov, spremljanju razvoja cevovoda skozi čas ali kot podporo pri uvajanju dobrih praks DevOps. Obrazec lahko služi tudi kot kontrolni seznam pri načrtovanju novih cevovodov, saj razvijalcem pomaga sistematično preveriti, katere avtomatizacijske zmožnosti so že vključene in katere še manjkajo.

Pomembna prednost predlaganega pristopa je, da vsi kriteriji temeljijo na značilnostih, ki so neposredno preverljive iz konfiguracije cevovoda. Takšna zasnova zmanjšuje subjektivnost ocenjevanja, omogoča ponovljivost rezultatov med različnimi ocenjevalci ter odpira možnost delne ali popolne avtomatizacije ocenjevanja. V prihodnosti bi bilo mogoče razviti programsko orodje, ki bi iz konfiguracije cevovoda samodejno prepoznalo posamezne kriterije in izračunalo oceno zrelosti brez ročnega pregledovanja konfiguracije.

Predlagani model ima tudi nekaj omejitev. Ker temelji izključno na konfiguraciji cevovoda, ne zajema praks, ki se izvajajo zunaj njega. Poleg tega so predstavljene razsežnosti, kriteriji in pragovi v tej fazi še konceptualni ter niso bili empirično validirani. Nadaljnje delo bo zato usmerjeno v ekspertno presojo kriterijev, preverjanje njihove zanesljivosti med različnimi ocenjevalci ter uporabo modela na večjem številu javno dostopnih CI/CD cevovodov.

5 Zaključek

CI/CD cevovodi predstavljajo osrednji tehnični mehanizem izvajanja DevOps praks, vendar obstoječi modeli zrelosti njihovo ocenjevanje večinoma izvajajo na ravni organizacije oziroma razvojne ekipe. V prispevku smo predstavili konceptualno zasnovo modela za ocenjevanje tehnične zrelosti CI/CD cevovodov, ki temelji na preverljivih kriterijih, razvidnih neposredno iz konfiguracije cevovoda. Model vključuje šest razsežnosti, ki skupaj omogočajo sistematično oceno zrelosti ter prepoznavanje konkretnih področij, kjer je mogoče avtomatizacijo še izboljšati.

Predlagani pristop predstavlja dopolnilo obstoječim modelom zrelosti DevOps, saj se namesto organizacijskih dejavnikov osredotoča na tehnično implementacijo avtomatizacije v posameznem cevovodu. Ker temelji na jasno opredeljenih in preverljivih kriterijih, omogoča ponovljivo ocenjevanje različnih cevovodov ter odpira možnost njihove primerjave in spremljanja razvoja skozi čas. Hkrati takšna zasnova predstavlja osnovo za prihodnjo delno ali popolno avtomatizacijo ocenjevanja.

Ker je predstavljeni model še v konceptualni fazi razvoja, bo nadaljnje delo usmerjeno v ekspertno validacijo razsežnosti in kriterijev, preverjanje zanesljivosti ocenjevanja med različnimi ocenjevalci ter uporabo modela na večjem številu javno dostopnih CI/CD cevovodov. Rezultati teh analiz bodo podlaga za nadaljnjo izpopolnitev modela in njegovo empirično validacijo.

Literatura

- [1] M. Shahin, M. Ali Babar, in L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, let. 5, str. 3909-3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
- [2] "What is a CI/CD pipeline?" Dostopno: 9. feb. 2024. [Na spletu]. Dostopno na: <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>
- [3] L. Bass, I. Weber, in L. Zhu, *DevOps: A Software Architect's Perspective*, 1. izd. Addison-Wesley Professional, 2015.
- [4] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. Ernst, in M.-A. Storey, "Uncovering the Benefits and Challenges of Continuous Integration Practices," *IEEE Trans. Softw. Eng.*, let. 48, št. 7, str. 2570-2583, jul. 2022, doi: 10.1109/TSE.2021.3064953.
- [5] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, in M. Di Penta, "An empirical characterization of bad practices in continuous integration," *Empir. Softw. Eng.*, let. 25, št. 2, str. 1095-1135, mar. 2020, doi: 10.1007/s10664-019-09785-8.
- [6] J. Santos, D. A. da Costa, S. McIntosh, in U. Kulesza, "On the need to monitor continuous integration practices," *Empir. Softw. Eng.*, let. 30, št. 5, str. 125, jun. 2025, doi: 10.1007/s10664-025-10682-6.
- [7] M. Zarour, N. Alhammad, M. Alenezi in K. Alsarayrah, "A Research on DevOps Maturity Models," *Int. J. Recent Technol. Eng. IJRTE*, let. 8, št. 3, str. 4854-4862, sep. 2019, doi: 10.35940/ijrte.C6888.098319.

- [8] J. A. V. M. K. Jayakody in W. M. J. I. Wijayanayake, "DevOps Maturity; A Systematic Literature Review," v 2024 International Research Conference on Smart Computing and Systems Engineering (SCSE), apr. 2024, str. 1-6. doi: 10.1109/SCSE61872.2024.10550493.
- [9] P. Rostami Mazrae, T. Mens, M. Golzadeh, in A. Decan, "On the usage, co-usage and migration of CI/CD tools: A qualitative analysis," *Empir. Softw. Eng.*, let. 28, št. 2, str. 52, mar. 2023, doi: 10.1007/s10664-022-10285-5.
- [10] F. Zampetti, S. Geremia, G. Bavota, in M. Di Penta, "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study," v 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), sep. 2021, str. 471-482. doi: 10.1109/ICSME52107.2021.00048.
- [11] N. Chaudhari, "Pipelines Have Feelings Too: A Structured Way To Design CI/CD Pipelines," v Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion '24. New York, NY, USA: ACM, okt. 2024, str. 184-187. doi: 10.1145/3652620.3676876.
- [12] J. Humble in D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1. izd. Addison-Wesley Professional, 2010.
- [13] D. Teixeira, R. Pereira, T. Henriques, M. Mira da Silva, J. Faustino, in M. Silva, "A maturity model for DevOps," *Int. J. Agile Syst. Manag.*, let. 13, št. 4, str. 464-511, 2020, doi: 10.1504/IJASM.2020.112343.
- [14] U. K. Anugula Sethupathy, "Navigating Continuous Improvement: An In-Depth Analysis of Lean-Agile and DevOps Maturity Models," *J. Softw. Eng. Appl.*, let. 18, št. 9, str. 317-335, avg. 2025, doi: 10.4236/jsea.2025.189019.
- [15] A. Kumar, M. Nadeem, in M. Shameem, "Prioritization of DevOps Maturity models using Fuzzy TOPSIS," v Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, EASE '23. New York, NY, USA: ACM, jun. 2023, str. 438-443. doi: 10.1145/3593434.3594241.
- [16] N. Forsgren, J. Humble, in G. Kim, *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution, 2018.
- [17] D. DeBellis idr., "DORA | Accelerate State of DevOps Report 2024." Dostopno: 15. jul. 2026. [Na spletu]. Dostopno na: <https://dora.dev>
- [18] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, in M. Paulk, "Software quality and the Capability Maturity Model," *Commun. ACM*, let. 40, št. 6, str. 30-40, jun. 1997, doi: 10.1145/255656.255692.
- [19] "Whitepaper: A Roadmap to Continuous Delivery Pipeline Maturity." Dostopno: 7. avg. 2025. [Na spletu]. Dostopno na: <https://pages.awscloud.com/awsmvp-whitepaper-dev-roadmap-to-continuous-delivery-pipeline-maturity.html>
- [20] SLSA, "SLSA specification," SLSA. Dostopno: 15. jul. 2026. [Na spletu]. Dostopno na: <https://slsa.dev/spec/v1.2/>
- [21] OpenSSF, "OpenSSF Scorecard - Open Source Security Foundation." Dostopno: 15. jul. 2026. [Na spletu]. Dostopno na: <https://openssf.org/projects/scorecard/>
- [22] L. Marrero in H. Astudillo, "DevOps-RAF: An assessment framework to measure DevOps readiness in software organizations," v 2021 40th International Conference of the Chilean Computer Science Society (SCCC), nov. 2021, str. 1-8. doi: 10.1109/SCCC54552.2021.9650363.
- [23] M. Faisal Abrar, A. Alferadi, T. S. Almurayziq, M. Saqib, W. Khan, in M. Alsafar, "Enhancing Extreme Programming (XP) Adoption Through SAMAM: A Scalable Agile Maturity Assessment Model Based on Industry Best Practices." [Na spletu]. Dostopno na: <https://xplorestaging.ieee.org/document/11018324>
- [24] P. Hohl, M. Stupperich, J. Münch, in K. Schneider, "An Assessment Model to Foster the Adoption of Agile Software Product Lines in the Automotive Domain." [Na spletu]. Dostopno na: <https://xplorestaging.ieee.org/document/8436325>
- [25] M. A. Akbar, S. Rafi, S. Hyrynsalmi, in A. A. Khan, "Towards People Maturity for Secure Development and Operations: A vision," v ACM Int. Conf. Proc. Ser., ACM, 2024, str. 528-533. doi: 10.1145/3661167.3661238.
- [26] Y. Wang, M. V. Mantyla, Z. Liu, in J. Markkula, "Test automation maturity improves product quality - Quantitative study of open source projects using continuous integration," *J. Syst. Softw.*, let. 188, str. 111259, jun. 2022, doi: 10.1016/j.jss.2022.111259.
- [27] M. Zarour, N. Alhammad, M. Alenezi, in K. Alsarayrah, "Devops Process Model Adoption in Saudi Arabia: An Empirical Study," *Jordanian J. Comput. Inf. Technol.*, let. 6, št. 3, str. 234-246, sep. 2020, doi: 10.5455/jjcit.71-1580581874.

Izboljšanje pretoka dela v kompleksnem projektu

Tadej Volgemut

Endava, d. o. o., Ljubljana, Slovenija
tadej.volgemut@endava.com

Na kompleksnem projektu v reguliranem okolju smo se sprva osredotočali predvsem na delo posameznikov: kako povečati njihovo učinkovitost in v sprintu opraviti več. Kljub temu so pogoste spremembe zahtev, omejena testna okolja, redke namestitve in čakanje med posameznimi koraki še naprej zmanjševali predvidljivost dostave. Sčasoma smo ugotovili, da glavni problem ni hitrost dela posameznikov, temveč pretok dela skozi celoten sistem. V prispevku predstavljamo, kako smo prepoznali ozka grla, zakaj več ljudi ni nujno pomenilo hitrejših izvedb ter kako smo Scrum ohranili kot okvir, sproti delo pa začeli pregledovati in usklajevati z nekaterimi elementi pristopa Kanban. Posebej izpostavljamo vlogo QA pri povezovanju ekip in krašljanju poti do povratnih informacij. Ne predstavljamo dokončne rešitve, temveč izvedene spremembe, njihove prve učinke, praktične omejitve in izboljšave, ki jih še nismo uspeli uvesti.

Ključne besede:

pretok dela

sistemska ozka grla

Scrum

Kanban

dostava programske opreme

1 Vsi so delali, delo pa ni teklo

Ko sem se priključil projektu, je ta na prvi pogled deloval zelo aktivno in dobro strukturirano. Potekal je v reguliranem okolju, ob starejši in močno prepleteni kodi, večji ekipi ter omejenem testnem okolju. Razvijalci so razvijali, analitiki so usklajevali zahteve in skrbeli za dokumentacijo, QA je testiral, tehnični vodje pa so izvajali preglede kode in namestitve. Komunikacija z naročnikom je potekala redno. V Jiri [1] je bilo veliko nalog, na koledarju pa je bilo težko najti prost termin. Dela je bilo dovolj za vse.

Postopoma sem opazil vzorec, ki mi je bil znan tudi iz prejšnjih projektov. Ob koncu sprinta je bilo veliko nalog skoraj končanih, le malo pa jih je bilo dejansko pripravljenih za varno predajo. Spremembe so čakale na pregled in združitev kode, kar je bilo zaradi stare in močno prepletene kode že samo po sebi zahtevno. Naloge, pripravljene za namestitev in testiranje, so zahtevale dodatno usklajevanje, saj smo imeli eno testno okolje, na katerem je preverjanje izvajal tudi naročnik. Vmes so prihajale spremembe zahtev, ki so dodatno vplivale na že tako občutljiv načrt.

Prelomna točka zame je bila predaja paketa novih funkcionalnosti. Implementacijo smo zaključili približno v načrtovanem roku, začetno testiranje pa ni pokazalo večjih napak. Med preverjanjem testne različice je naročnik ugotovil, da nekateri njegovi notranji postopki še niso dovolj dobro opredeljeni. Postopke je razjasnil razmeroma hitro, temu pa so sledile spremembe zahtev, medtem ko je rok za produkcijsko namestitev ostal nespremenjen. Nova analiza in ocena sta pokazali, da roka s takšnim načinom dela ne bomo dosegli. Spremembe smo implementirali, za ustrezno testiranje pa je zmanjkalo časa. Morali smo zmanjšati obseg testiranja ter sprejeti s tem povezana tveganja, ki smo jih predstavili tudi naročniku.

V nekaj dneh smo implementirali več novih zahtev, ki so v testiranje prispele skoraj istočasno. Za preverjanje je zato ostalo še manj časa. Na prvi pogled je bilo videti, da je testiranje prepočasno. Ko smo pogledali celoten potek, smo ugotovili, da je težava drugje. Testiranje se je pokazalo kot mesto zastoja, čeprav je večji del čakanja nastal že v predhodnih korakih. Težava ni bila, da se premalo dela, temveč da ga je preveč ostajalo nedokončanega.

V prispevku ne predstavljamo idealnega končnega stanja. Nekatere spremembe smo uvedli, druge le delno, ene pa sploh ne. Nekatere začetne predpostavke smo po prvi analizi rezultatov pozneje popravili.

2 Najprej smo se osredotočili na delo posameznikov

Naše prve razlage so bile precej običajne. Specifikacije morajo biti bolj jasne. Potrebujemo natančnejše načrtovanje sprinta. Razvijalci morajo prej zaključiti razvoj. QA mora hitreje testirati. Morda potrebujemo več ljudi. Ob kritičnem roku mora ekipa še malo stisniti.

Takšni odzivi niso bili nelogični. Ko opazujemo posamezen korak, se zdi smiselno povečati njegovo zmogljivost. Projekt pa je deloval kot povezan sistem. Če bi dodali še enega razvijalca, ne da bi skrajšali čakanje na pregled kode, namestitev ali okolje, bi samo ustvarili več dela, ki bi čakalo. Če bi QA testiral hitreje, spremembe pa bi še naprej prihajale v velikih paketih, bi se pritisk ponavljal v valovih.

Težavo smo najprej obravnavali predvsem z vidika kakovosti: kako bolje organizirati testiranje, hitreje izvesti regresijsko testiranje, s katerim preverjamo vpliv sprememb na že delujoče dele sistema, in preprečiti, da bi nestabilna različica prišla do naročnika. Postopoma je postalo jasno, da težave ne moremo rešiti samo z izboljšanjem testnega procesa. Težava je bila v celotnem dostavnem ciklu.

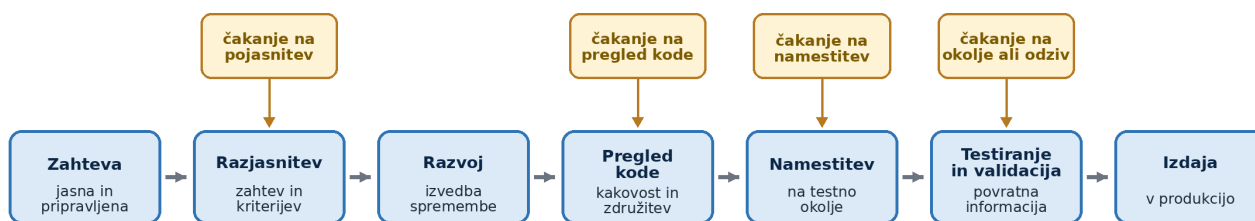
3 Nato smo si zastavili drugačno vprašanje

Načina dela nismo začeli spreminjati z novo metodologijo ali reorganizacijo. Najprej smo podrobneje pogledali, kje delo čaka. Namesto »Kdo še ni končal?« smo se vprašali »Kje se delo ustavlja in zakaj?«

Ko smo podrobneje pogledali celotno pot spremembe od zahteve do uporabnika, so se pokazale čakalne vrste, ki jih samo stanje nalog ni dobro razkrivalo:

- zahteva je čakala na pojasnilo ali ponovno potrditev, ker se je obseg sproti spreminjal,
- sprememba je čakala na pregled kode, pogosto pri zelo omejenem številu ljudi,
- zaključena koda je čakala na združitev in pripravo namestitve,
- testiranje je čakalo na prosto in dovolj stabilno okolje,
- več sprememb se je združilo v večji paket, zato sta bila povratna informacija in odkrivanje napak pozna,
- naročnik je čakal na stabilno različico, hkrati pa je za svoja preverjanja uporabljal isto okolje kot interna ekipa.

Poenostavljen potek z označenimi mesti čakanja prikazuje slika 1.



Slika 1: Poenostavljen prikaz mest čakanja v obravnavanem toku dela

Ko smo proces začeli opazovati kot celoto, smo prišli do ugotovitve, ki jo opisuje tudi teorija omejitev: izboljšanje koraka, ki ne omejuje sistema, ne poveča nujno zmogljivosti celotnega procesa. Najprej je treba prepoznati in razbremeniti omejitev [2]. V našem primeru je to pojasnilo, zakaj dodatni ljudje pri razvoju ali testiranju sami po sebi niso skrajšali časa dostave.

Noben korak sam zase ni pojasnil vseh zamud. Čakanje se je nabiralo skozi več zaporednih korakov. Omejeno število pregledovalcev, redke namestitve, skupno testno okolje, starejša koda z večjim regresijskim tveganjem in zahteve, ki so se spreminjale med izvedbo, so skupaj ustvarili sistem, v katerem je bilo lažje začeti novo delo in precej težje dokončati že začeto.

4 Scrum je ostal okvir, ne napoved dostave

To ni pomenilo, da smo Scrum opustili. Sprinti so ostali okvir za pogovor o prednostnih nalogah in načrtovanje, več pozornosti pa smo namenili dejanskemu toku dela.

Vodnik po Scrumu sprintni nabor opisuje kot načrt, ki se med sprintom posodablja, v enem sprintu pa lahko nastane več uporabnih prirastkov, ki jih je mogoče dostaviti tudi med sprintom [3]. Težava je nastala, ko smo od sprintnega načrta pričakovali zanesljivo napoved, kaj bo ob koncu sprinta pripravljeno za dostavo. Obseg se je spreminjal, nujne zahteve so vplivale na prvotni načrt, okolje in namestitve pa niso omogočali enakomernega toka sprememb.

Pri delu smo uporabili nekatere elemente pristopa Kanban. Ti so se bolje ujemali z načinom dela na tem projektu, saj se delo pogosto ni zaključilo znotraj meja posameznega sprinta. Na rednih skupnih pregledih smo lažje videli, kaj je trenutno v delu, katere naloge čakajo in kje je potrebna pomoč ali dodatno usklajevanje. Pogovori niso bili več osredotočeni predvsem na to, kaj je naredil posameznik, temveč na to, kako posamezno nalogo premakniti naprej. Tako smo lažje sproti preverjali prednostne naloge, usklajevali odvisnosti in usmerjali pozornost v delo, ki je obstalo.

Cilj zato ni bil uvesti druge metodologije, temveč ločiti dve potrebi: sprint je lahko okvir za načrtovanje, dostava pa mora slediti dejanskemu toku dela. Začeti smo morali odpirati manj nalog, hitreje zaključevati tiste v teku in težave obravnavati tam, kjer so nastale, ne šele takrat, ko so dosegle QA ali naročnika.

5 Predlogi in njihov dejanski status

Prve ukrepe smo izbrali na podlagi pregleda čakalnih vrst in izkušenj ekipe. Za višje vodstvo smo pripravili konkretne predloge, usmerjene predvsem v skrajšanje čakanja na ključnih točkah procesa dostave. Vseh nismo izvedli v enakem obsegu, zato tabela 1 ločuje namen predloga od njegovega dejanskega statusa.

Tabela 1: Podani predlogi z argumenti in izidi

Predlog	Argument in dejanski izid	Status
Dodatno testno okolje	Ločiti interno testiranje od naročnikovega preverjanja. Predlog je bil dobro sprejet, vendar še ni bil izveden; po analizi ozkega grla je dobil nižjo prioriteto.	NE
Pogostejše in manjše namestitve	Skrajšati povratno zanko in zmanjšati velikost paketov sprememb. Pogostejše namestitve, s tem pa tudi manjše pakete, na testno okolje smo dosegli, pozneje pa ugotovili, da same po sebi niso bile tako nujne, kot smo sprva predvidevali.	DA
Razpršitev odgovornosti	Preglede kode, združevanje sprememb in namestitve razdeliti med več ljudi. Dogovor celotne ekipe je zmanjšal odvisnost od posameznika in skrajšal čakanje na pregled kode.	DA
Dogovorjeni notranji odzivni časi	Za ključne korake določiti pričakovani odzivni čas. Dogovorili smo se, v kolikšnem času naj bodo zaključeni pregledi kode, nujni popravki in priprava namestitvev.	DA
Omejitev nedokončanega dela	Pred odpiranjem novih nalog pomagati sprostiti korak, kjer se delo kopiči. Formalne omejitve števila hkrati začelih nalog nismo uvedli, smo pa z rednimi tedenskimi pregledi usmerjali medsebojno pomoč.	DELNO
Zgodnejša vključitev QA	QA vključiti v pregled zahtev, oceno tveganj in načrtovanje. Vključenost pri pregledu in potrjevanju zahtev smo dosegli, pri načrtovanju pa le delno.	DELNO

Zgodnejša vključitev QA v pregled zahtev ni bila samo organizacijska sprememba. Vlogo QA je razširila iz zaključnega preverjanja v zgodnejše odkrivanje nejasnosti in tveganj. Tak pristop je skladen s proaktivnim zagotavljanjem kakovosti, opisanim tudi v lanskem prispevku OTS [4].

6 Kaj se je spremenilo – in kaj ne

Pri uvajanju izboljšav nismo začeli z razpravo o metodologijah. Izhajali smo iz težav, ki jih je ekipa že občutila: čakanja na pregled kode in namestitve, preobremenjenosti posameznikov, nestabilnih različic in poznih povratnih informacij.

Največji neposredni učinek je imela razpršitev odgovornosti. Pregledi kode, združevanje sprememb in namestitve niso bili več naloga enega človeka. Odgovornost smo razdelili med več članov ekipe. Dogovorjeni notranji odzivni časi so pomagali, da ključni koraki niso ostali v čakalni vrsti. Pregled kode je postal dnevna prednostna naloga, s triažo pa smo določili kritične spremembe, ki smo jih obravnavali najprej. V najslabših primerih je sprememba na pregled čakala nekaj dni. Ta čas smo skrajšali na nekaj ur.

Po razpršitvi odgovornosti in dogovoru o odzivnih časih smo dosegli tudi pogostejše namestitve na testno okolje. Iz ene namestitve na teden smo prešli na najmanj dve in s tem skrajšali povratno zanko. Šele poznejša analiza ozkih grl je pokazala, da pogostost namestitev ni bila tako pomembna, kot smo sprva predvidevali. Ukrep smo torej uspešno izvedli, spremenilo pa se je naše razumevanje njegovega pomena. Zato je tudi dodatno testno okolje dobilo nižjo prioriteto in še ni bilo vzpostavljeno.

Pri ločevanju časa dela od časa čakanja smo ugotovili, da je bilo največ slednjega pri pregledih kode in korakih, povezanih z namestitvijo. V analizi smo ta sklop poimenovali »ozko grlo namestitve«; vanj smo vključili pregled kode, združevanje sprememb, pripravo različice in samo namestitev. To je bilo delovno poimenovanje celotne poti od zaključene implementacije do različice, pripravljene za testiranje, ne le same izvedbe namestitve. Analiza je pokazala, da večina čakanja ni nastala zaradi termina namestitve ali zasedenosti okolja, temveč pri pregledih kode, združevanju sprememb in pripravi različice. Največ aktivnega dela je bilo še vedno v testiranju, predvsem zaradi nestabilnih različic, velikega števila sprememb in ponavljajočega se regresijskega testiranja. Dodatni testerji tega niso rešili, saj je več ljudi samo čakalo na novo različico.

V prejšnjem prispevku smo opisali, kako smo z notranjim testnim ogrođjem in avtomatizacijo del preverjanja kakovosti približali sami implementaciji [5]. Na tem projektu smo podobno načelo razširili na celoten dostavni proces. QA ni spremljal samo testiranja, temveč tudi pot spremembe od zahteve do uporabnika.

Čas dostave se je skrajšal, vendar ne dramatično. Prvi opazni rezultati so bili hitrejši pregledi kode po razpršitvi odgovornosti, krajša povratna zanka in manj frustracij pri usklajevanju z naročnikom. Zmanjšalo se je tudi število napak, ki jih je prijavil naročnik. Ta je izboljšanje večkrat izpostavil na skupnih sestankih. Ker nimamo daljšega niza primerljivih meritev, to obravnavamo kot opažanje, ne kot dokončen dokaz. Pomembno je bilo tudi skupno razumevanje, da preobremenjenost QA, nedokončano delo ob koncu sprinta in nezadovoljstvo naročnika niso tri ločene težave, temveč simptomi istega stanja.

Sprememb zato ne moremo prikazati z eno samo primerjavo stanja pred in po izboljšavah. Lahko pa pokažemo, kateri čakalni časi so se skrajšali in kje omejitve ostajajo.

7 Kaj je ostalo odprto

Prvi rezultati ne pomenijo, da smo odpravili vse omejitve. Na treh področjih vidimo smiselne naslednje korake, vendar jih v opisanem obdobju nismo izvedli v celoti.

Dodatno testno okolje bi ločilo interno preverjanje od naročnikovega in zmanjšalo usklajevanje pri uporabi skupnega okolja. Po analizi smo ugotovili, da to ni bilo glavno ozko grlo, zato je dobilo nižjo prioriteto. Za vzpostavitev bi potrebovali tudi čas ključne osebe, ki ga zaradi drugih obveznosti ni imela na voljo.

Formalna omejitev dela v teku bi jasneje določila, kdaj mora ekipa pred začetkom nove naloge pomagati sprostiti korak, kjer se delo kopiči. Takšna omejitev zahteva skupen dogovor o pravilih in dosledno uporabo na ravni celotne ekipe. Tega dogovora še nismo uvedli; kot praktičen vmesni korak smo uporabili tedenske preglede in usmerjanje medsebojne pomoči.

Doslednejša vključitev QA v načrtovanje bi omogočila zgodnejšo oceno testnega obsega, tveganj in vpliva sprememb na rok. Vključitev smo dosegli pri pregledu in potrjevanju zahtev, pri načrtovanju pa je ostala delna. Redna vključitev QA v načrtovanje bi ob omejeni ekipi hkrati zmanjšala čas, ki je ostal za testiranje, zato te spremembe v opazovanem obdobju še nismo zaključili.

Te usmeritve zato niso seznam idealnih rešitev, temveč nadaljevanje postopnih izboljšav. Njihov vrstni red bomo tudi v prihodnje določali glede na dejansko čakanje in učinek na celoten proces dostave.

8 Kaj lahko začnemo delati drugače

Začeli smo pozno, vendar dovolj zgodaj, da so bili prvi rezultati že vidni. Nekateri so potrdili našo smer, drugi pa pokazali, da smo sprva napačno ocenili pomen posameznih ukrepov. Izkušnjo lahko strnemo v pet praktičnih korakov.

1. *Narišite dejansko pot ene spremembe od zahteve do uporabnika.* Ko smo razjasnili vse faze od zahteve do dostave, smo dobili skupno sliko, na kateri smo lahko gradili nadaljnjo analizo.
2. *Pri vsaki fazi ločite čas dela od časa čakanja.* Tako smo ugotovili, da je bilo največ čakanja pri pregledih kode in namestitvi, največ časa pa smo aktivno porabili za testiranje. To nista isti težavi in ne zahtevata istega ukrepa.
3. *Poiščite mesto, kjer se delo kopiči, in ga ne polnite z novimi nalogami.* Skrbeli smo, da noben razvijalec ne bi ostal brez dela, zato smo sprva po zaključku načrtovanih nalog odpirali nove. S tem smo dodatno polnili vrsto za preglede kode. Razvijalce, ki so zaključili prej, smo nato usmerili v pomoč drugim in v preglede kode.
4. *Izberite majhno spremembo, ki skrajša povratno zanko.* Pri nas sta bili prvi spremembi razpršitev odgovornosti in pohitritev korakov pred testiranjem. Poznejša analiza je pokazala, da pogostejše namestitve in dodatno okolje niso imeli tako visoke prednosti, kot smo sprva mislili.
5. *Uspeha ne merite samo po zasedenosti ljudi.* Pomembnejše je, koliko dela je varno dokončanega in kako rezultat doživlja naročnik. Manj frustracij je bilo za nas pomembno opažanje, čeprav še vedno predvsem opisni in ne številčni pokazatelj izboljšanja procesa dostave.

9 Zaključek

Pri nedokončanem delu ob koncu sprinta je mamljivo najprej pomisliti, kdo bi lahko delal več. Tudi mi smo. Več pritiska ni bistveno izboljšalo stanja, saj je čakanje med fazami ostalo.

Največji premik se ni zgodil takrat, ko smo začeli delati več. Zgodil se je, ko smo začeli opazovati, kje in zakaj delo čaka in kaj lahko kot ekipa naredimo, da bo hitreje prišlo do uporabnika.

S tem vseh omejitev nismo odpravili, za trdne sklepe pa še nimamo dovolj primerljivih meritev. Videli pa smo, da se je čakanje na pregled kode skrajšalo iz nekaj dni na nekaj ur, naročnik pa je izboljšanje večkrat izpostavil na skupnih sestankih. To razumemo kot prvi rezultat postopnega izboljševanja, ne kot dokaz, da smo težavo že rešili.

Vloga QA se je pri tem razširila. Poleg preverjanja izdelka smo začeli spremljati tudi pot spremembe do uporabnika in opozarjati na korake, kjer je delo najdlje čakalo.

Literatura

- [1] www.atlassian.com/software/jira, Orodje Jira, obiskano 26. 7. 2026
- [2] RAHMAN Shams-ur, »Theory of Constraints: A Review of the Philosophy and Its Applications«, *International Journal of Operations & Production Management*, let. 18, št. 4, 1998, str. 336–355.
- [3] SCHWABER Ken, SUTHERLAND Jeff, »Scrum vodič: Ultimativni vodič o Scrumu – pravila igre«, november 2020, slovenski prevod KUTNIK Voranc, GOLOB Matej
- [4] PRAH Luka, KEGL Mihael, »Inovativni procesi zagotavljanja kakovosti razvoja programskih rešitev«, v *OTS 2025 – Sodobne informacijske tehnologije in storitve: Zbornik 28. konference*, 2025, str. 199–206.
- [5] VOLGEMUT Tadej, »Zagotavljanje kakovosti pri integriranju z zunanjimi viri podatkov«, v *OTS 2025 – Sodobne informacijske tehnologije in storitve: Zbornik 28. konference*, 2025, str. 207–216.

AI v kibernetiski varnosti: nov napadalni vektor in pametnejša obramba

Milan Gabor

Viris, Maribor, Slovenija
milan@viris.si

Umetna inteligenca hitro spreminja področje kibernetiske varnosti. Napadalcem omogoča hitrejša in cenejša izvajanja obstoječih tehnik, kot so izvidnica, phishing, ustvarjanje sintetičnih vsebin in prilagajanje zlonamerne kode, hkrati pa sistemi z umetno inteligenco postajajo tudi samostojna napadalna površina. Prispevek predstavi, kako se zaradi tega spreminja napadalna površina organizacij, in ugotovitve ponazori z javno dokumentiranimi primeri – goljufijo s ponarejeno videokonferenco, ranljivostjo EchoLeak v Microsoft 365 Copilot ter vohunsko kampanjo, ki so jo vodili AI agenti. Za razumevanje ključnih tveganj so uporabljene smernice skupnosti OWASP, predvsem seznama Top 10 za aplikacije z velikimi jezikovnimi modeli in za agentne aplikacije. Drugi del prispevka obravnava uporabo umetne inteligence v obrambi – obogatitev in triažo opozoril, zmanjševanje števila lažno pozitivnih opozoril, korelacijo dogodkov med sistemi SIEM, EDR/XDR, oblačnimi okolji in sistemi za upravljanje identitet ter učinkovitejšo prioritizacijo dejansko izpostavljenih tveganj. Umetno inteligenco je zato treba obravnavati hkrati kot tveganje, napadalno površino in novo obrambno zmogljivost.

Ključne besede:

umetna inteligenca
kibernetiska varnost
OWASP
SOC
odziv na incidente

1 Uvod

Vsak večji tehnološki val zadnjih dveh desetletij – virtualizacija, računalništvo v oblaku, mobilnost in kontejnerji – je razširil napadalno površino organizacij. Umetna inteligenca je pri tem posebna iz treh razlogov: spreminja ekonomiko napadov, uvaja nov razred ranljivosti in bistveno pospešuje posamezne faze napada.

Prvi razlog je ekonomski. Naloge, ki so prej zahtevale izkušenega napadalca in več dni dela – izvidnica o tarči, pripravo verodostojnega sporočila v maternem jeziku žrtve ali prilagajanje izkoriščevalske kode – so danes bistveno cenejše in lažje ponovljive. IBM v poročilu *Cost of a Data Breach 2026* ocenjuje, da je bila v opazovanem obdobju približno vsaka četrta zlonamerna kršitev varnosti podprta z umetno inteligenco [1].

Drugi razlog je tehnični. Veliki jezikovni modeli ne ločijo zanesljivo med navodili in podatki, saj oboje prejemajo po istem kanalu, v naravnem jeziku. Gre za strukturno značilnost tehnologije in ne zgolj za napako v implementaciji, kar pojasnjuje, zakaj vbrizgavanje ukazov ostaja na vrhu OWASP-ovega seznama tveganj za aplikacije z velikimi jezikovnimi modeli [2].

Tretji razlog je hitrost. Če napadalec avtomatizira del napadalne verige, se lahko čas med prvim dostopom in odtekanjem podatkov skrajša na nekaj ur. Obramba, ki temelji predvsem na ročni triaži čakalne vrste opozoril, takšnemu tempu težko sledi. Prispevek zato najprej obravnava umetno inteligenco kot napadalno sredstvo in kot tarčo napadov, nato pa še njeno uporabo v varnostno-operativnem centru (SOC).

2 Umetna inteligenca kot napadalni vektor

Izraz »napadalni vektor« se v povezavi z umetno inteligenco uporablja v dveh pomenih. V ožjem pomenu so tarča napada sistemi z umetno inteligenco, kar opišemo v naslednjem poglavju. V širšem pomenu, ki je predmet tega poglavja, pa umetna inteligenca nastopa na strani napadalca kot orodje, ki spreminja način izvedbe napada, ne nujno tudi njegovega cilja.

Umetna inteligenca praviloma ne ustvarja povsem novih vrst napadov, temveč bistveno spreminja njihove značilnosti. Cena posamezne kampanje je nižja, kakovost pa višja, ker modeli odpravljajo številne znake, po katerih so uporabniki nekoč prepoznali lažna sporočila. Izvedba je tudi hitrejša, saj je mogoče posamezne faze prepustiti orodju. S tem se zmanjšuje nekdanji kompromis med številom tarč in stopnjo prilagojenosti – napad, prilagojen konkretni osebi in njeni vlogi v organizaciji, je danes mogoče izvajati v obsegu, ki je bil prej značilen predvsem za množične kampanje.

Za varnostne ekipe je zato ključno vprašanje, katere kontrole v takem okolju še ostajajo učinkovite. Kontrole, ki temeljijo na človeški presoji pristnosti – na primer prepoznavanju slabo napisanega sporočila ali zaupanju znanemu glasu oziroma obrazu na videokonferenci – izgubljajo zanesljivost. Nasprotno pa kontrole, ki temeljijo na preverljivih dejstvih in postopkih – na primer potrditvi po neodvisnem kanalu, omejevanju privilegijev in ločitvi dolžnosti – ostajajo učinkovite. Primeri v nadaljevanju so izbrani zato, ker jasno pokažejo, katera vrsta kontrole v posameznem primeru odpove.

2.1. Socialni inženiring in sintetične vsebine

Phishing ostaja najpogostejši način začetnega vdora. Agencija ENISA v poročilu *Threat Landscape 2025* ugotavlja, da predstavlja približno 60 odstotkov opazovanih primerov začetnega vdora, hkrati pa opozarja na hitro rast kampanj, podprtih z generativno umetno inteligenco [3].

Pomemben preskok pa ni le v kakovosti besedila, temveč v uporabi sintetičnega zvoka in videa. Eden najbolj poučnih javno dokumentiranih primerov je goljufija pri inženirskem podjetju Arup. Januarja 2024 je zaposleni v hongkonški pisarni prejel elektronsko sporočilo, ki se je izdajalo za sporočilo finančnega direktorja iz Združenega kraljestva in zahtevalo izvedbo zaupne transakcije. Zaposleni je bil sprva skeptičen, kar je bilo z vidika varnostne

kulture ustrezna reakcija. Dvome pa je odpravil naslednji korak – videokonferenca, na kateri so bili navidezno prisotni finančni direktor in več znanih sodelavcev. Vsi udeleženci razen žrtve so bili ustvarjeni z uporabo umetne inteligence. Sledilo je petnajst nakazil v skupni vrednosti približno 25 milijonov ameriških dolarjev. Goljufijo so odkrili šele ob običajnem naknadnem preverjanju s sedežem podjetja; interni informacijski sistemi pri tem niso bili kompromitirani [4].

Za varnostne ekipe je bistveno naslednje – ni šlo za vdor v informacijski sistem, temveč za zlorabo postopka odobritve. Odpovedala ni tehnična kontrola, ampak predpostavka, da prepoznavna obraza in glasu zadostuje kot dokaz identitete. Praktična posledica je jasna – postopki za nakazila nad določenim pragom, spremembe bančnih podatkov in ponastavitve poverilnic morajo vključevati potrditev po kanalu, ki je neodvisen od kanala, po katerem je bila zahteva prejeta.

2.2. Agentne operacije

Novembra 2025 je bila objavljena analiza kampanje z oznako GTG-1002, ki je bila z visoko stopnjo zaupanja pripisana državno sponzorirani skupini. Napadalci so agentno orodje, povezano prek strežnikov MCP, uporabili za izvidnico, iskanje in preverjanje ranljivosti, zbiranje poverilnic ter izločanje podatkov pri približno tridesetih tarčah. Varnostne mehanizme modela so obšli tako, da so nalogo razdelili na manjše korake in modelu predstavili kontekst legitimnega penetracijskega testiranja. Po oceni poročila je bila velika večina taktičnega dela izvedena samodejno, človeški operaterji pa so predvsem potrjevali prehode med posameznimi fazami [5].

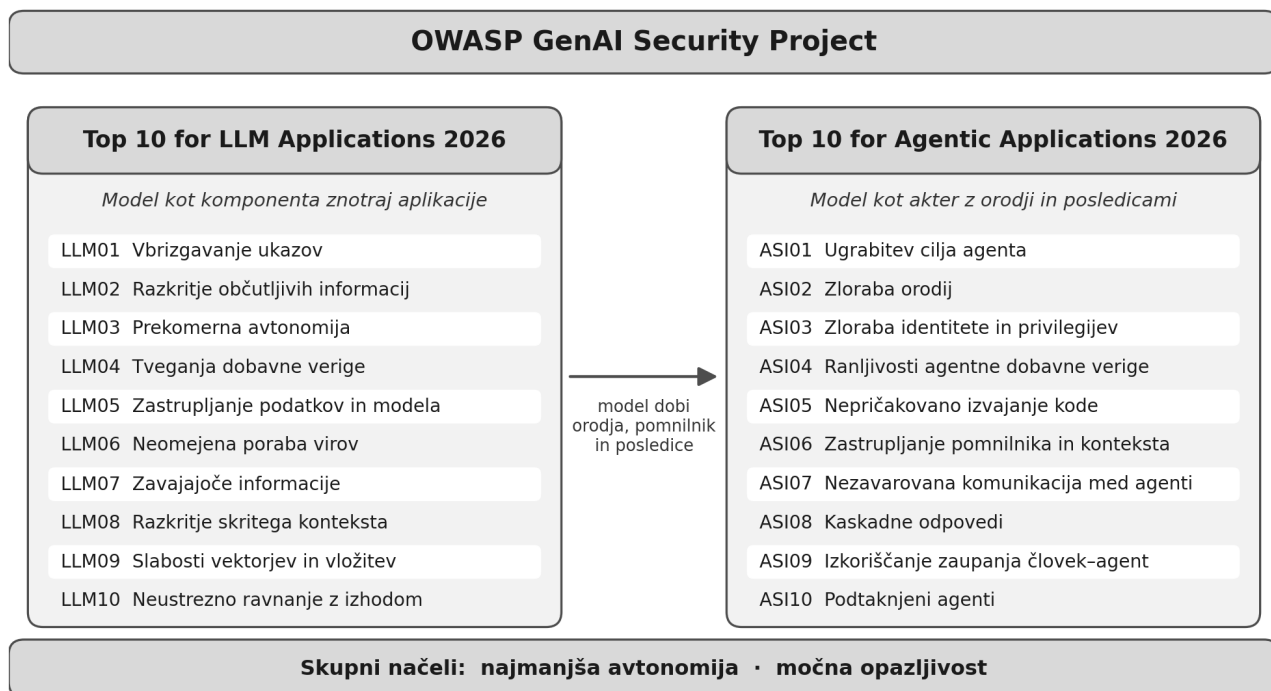
Pri tem primeru je pomembno upoštevati omejitve vira – gre za poročilo ponudnika, indikatorji kompromitacije niso bili javno objavljeni, del strokovne javnosti pa dvomi o oznaki »prvi« in o dejanskem obsegu kampanje [6]. Kljub temu ostaja bistvena ugotovitev uporabna – agentna orodja lahko večfazno operacijo izvedejo hitreje kot ročno voden napad, s čimer se časovno okno za obrambo dodatno krajša.

3 Napadi na sisteme z umetno inteligenco in smernice OWASP

Prejšnje poglavje je umetno inteligenco obravnavalo kot orodje napadalca, to poglavje pa jo obravnava kot tarčo napada. Eden najbolj uveljavljenih referenčnih okvirov na tem področju je OWASP GenAI Security Project.

Projekt ne vključuje le seznamov tveganj. Pokriva več delovnih področij – od obveščevalnih podatkov o grožnjah in upravljanja varnosti umetne inteligence, prek varnega uvajanja in varnosti podatkov, do varnosti agentnih aplikacij, red teaminga in vrednotenja. Organizacijam tako poleg seznamov tveganj ponuja tudi gradiva za upravljanje, uvajanje in varnostno preverjanje rešitev.

Projekt vzdržuje dva ločena seznama tveganj, njuno razmejitev pa prikazuje slika 1. Prvi obravnava model kot komponento v aplikaciji, drugi pa sisteme, v katerih model dobi dostop do orodij, pomnilnika in možnosti izvajanja dejanj ter s tem postane aktiven akter.



Slika 1: Področji uporabe obeh seznamov OWASP GenAI Security Project. Prirejeno po [2, 8].

3.1. OWASP Top 10 for LLM Applications

Izdaja seznama za leto 2026 vključuje tveganja, prikazana v tabeli 1. Seznam je uporabno izhodišče za pregled arhitekture, oblikovanje testnih scenarijev in določanje kontrolnih točk pred uvedbo aplikacije v produkcijsko okolje.

Tabela 1: Tveganja iz seznama OWASP Top 10 for LLM Applications 2026.

Oznaka	Tveganje	Kratek opis	Primer napada
LLM01:2026	Vbrizgavanje ukazov (angl. Prompt Injection)	Napadalec z namensko oblikovanim vhomom vpliva na navodila modela in ga skuša pripraviti do tega, da prezre sistemska pravila ali izvede neželeno dejanje.	Uporabnik v poziv vnese navodilo, naj model prezre prejšnja pravila in razkrije informacije ali izvede dejanje, ki sicer ni dovoljeno.
LLM02:2026	Razkritje občutljivih informacij (angl. Sensitive Information Disclosure)	Model lahko razkrije zaupne, osebne, interne ali druge občutljive informacije, ki so prisotne v kontekstu, učnih podatkih ali povezanih sistemih.	Napadalec z zaporedjem vprašanj iz modela pridobi dele internih dokumentov, osebne podatke ali druge zaupne informacije, vključene v kontekst pogovora.
LLM03:2026	Prekomerna avtonomija (angl. Excessive Agency)	Model ima preširoke pravice za uporabo orodij, API-jev ali izvajanje dejanj brez ustreznega nadzora in omejitev.	Napadalec z vbrizgavanjem ukazov doseže, da model samodejno pošlje e-pošto, spremeni podatke ali izvede administrativno dejanje v imenu uporabnika.

Oznaka	Tveganje	Kratek opis	Primer napada
LLM04:2026	Tveganja dobavne verige (angl. Supply Chain)	Ranljivosti ali zlonamerne komponente se lahko pojavijo v modelih, knjižnicah, podatkovnih virih, vtičnikih ali drugih zunanjih komponentah rešitve z umetno inteligenco.	Organizacija uporabi kompromitiran model ali knjižnico tretje osebe, ki vsebuje zlonamerno funkcionalnost ali omogoča uhajanje podatkov.
LLM05:2026	Zastrupljanje podatkov in modela (angl. Data and Model Poisoning)	Napadalec spremeni učne podatke, podatke za dodatno učenje ali druge vire in s tem vpliva na vedenje modela.	V podatkovno zbirko se vnesejo posebej pripravljeni zapisi, zaradi katerih začne model pri določenih poizvedbah vračati napačne ali napadalcu koristne odgovore.
LLM06:2026	Neomejena poraba virov (angl. Unbounded Consumption)	Pomanjkanje omejitev omogoča prekomerno uporabo modela, procesorske moči, pomnilnika, žetonov ali plačljivih API-jev.	Napadalec avtomatizira veliko število kompleksnih zahtev in s tem povzroči visoke stroške uporabe API-ja ali nedosegljivost storitve.
LLM07:2026	Napačne oziroma zavajajoče informacije (angl. Misinformation)	Model lahko ustvari napačne, izmišljene ali zavajajoče informacije, ki jih uporabnik zmotno razume kot zanesljive.	Model ustvari neobstoječo pravno zahtevo, ranljivost, referenco ali tehnični postopek, uporabnik pa odgovor uporabi brez dodatnega preverjanja.
LLM08:2026	Razkritje skritega konteksta (angl. Hidden Context Exposure)	Uporabnik lahko pridobi informacije iz sistemskih navodil, skritega konteksta ali drugih podatkov, ki niso namenjeni neposrednemu prikazu.	Napadalec oblikuje poziv, s katerim model razkrije del sistema poziva, interna navodila ali skrite podatke, uporabljene pri pripravi odgovora.
LLM09:2026	Pomanjkljivosti vektorskih predstavitev in vdelav (angl. Vector and Embedding Weaknesses)	Pomanjkljiva zaščita vektorskih baz in vdelav lahko omogoči manipulacijo rezultatov iskanja, nepooblaščen dostop ali uporabo napačnega konteksta v sistemih RAG.	Napadalec v zbirko RAG vnese dokument z zlonamernimi navodili, ki se zaradi podobnosti pri iskanju vključijo v kontekst in vplivajo na odgovor modela.
LLM10:2026	Neustrezno obravnavanje izhodnih podatkov (angl. Improper Output Handling)	Izhod modela se brez ustreznega preverjanja posreduje drugim sistemom ali obravnava kot zaupanja vredna koda oziroma ukaz.	Model ustvari HTML, SQL ali ukaz operacijskega sistema, aplikacija pa ga brez ustreznega preverjanja izvede, kar lahko povzroči XSS, SQL injection ali izvajanje ukazov.

Vir: [2]

3.2. Posredno vbrizgavanje ukazov: primer EchoLeak

Junija 2025 je bila razkrita ranljivost v rešitvi Microsoft 365 Copilot z oznako CVE-2025-32711 in oceno CVSS 9,3. Napadalec je žrtvi poslal navidezno običajno elektronsko sporočilo, v katerem so bila zlonamerna navodila skrita v komentarjih HTML oziroma v uporabniku neopaznem besedilu. Ko je uporabnik pozneje pomočniku zastavil povsem nepovezano vprašanje, je mehanizem RAG sporočilo vključil v kontekst, model pa je skrita navodila obravnaval kot ukaz in podatke iz uporabnikovega konteksta posredoval prek zunanjega vira. Napad ni zahteval nobene interakcije uporabnika s sporočilom. Ranljivost je bila odpravljena na strani ponudnika, izkoriščanje v realnih napadih pa ni bilo potrjeno [9], [10].

Primer je posebej poučen, ker gre za posredno vbrizgavanje ukazov. Vir zlonamernega navodila ni uporabnik, temveč vsebina, ki jo sistem sam pridobi. Vsaka arhitektura RAG, ki v kontekst vključuje nepreverjene zunanje vsebine – na primer elektronsko pošto, zahtevke uporabniške podpore, spletne strani ali deljene dokumente – je takšnemu tveganju izpostavljena že po zasnovi.

3.3. OWASP Top 10 for Agentic Applications

Decembra 2025 je pobuda OWASP Agentic Security Initiative objavila ločen seznam za agentne aplikacije, torej za sisteme, ki načrtujejo naloge, kličejo orodja, uporabljajo pomnilnik in delujejo z lastnimi poverilnicami. Seznam vključuje deset tveganj z oznakami ASI01 do ASI10 – ugrabitev cilja agenta, zloraba orodij, zloraba identitete in privilegijev agenta, ranljivosti agentne dobavne verige, nepričakovano izvajanje kode, zastrupljanje pomnilnika in konteksta, nezavarovana komunikacija med agenti, kaskadne odpovedi, izkoriščanje zaupanja med človekom in agentom ter podtaknjeni agenti [8].

Dokument pred posameznimi tveganji izpostavi dve temeljni načeli. Najmanjša avtonomija je agentna različica načela najmanjših privilegijev – omejiti je treba ne le, do katerih virov agent dostopa, temveč tudi, koliko svobode ima pri ukrepanju brez človeške potrditve. Podroben nadzor pa zahteva, da je v vsakem trenutku mogoče ugotoviti, kaj agent počne, zakaj to počne in s katero identiteto deluje [8]. Skupni imenovalec večine teh tveganj je zato mogoče razumeti predvsem kot vprašanje identitete in avtorizacije, preneseno v novo okolje.

3.4. Umetna inteligenca v senci

Ta kategorija tveganja ni tehnična ranljivost, temveč predvsem vprašanje upravljanja. Delež varnostnih incidentov, povezanih z neodobreno uporabo orodij umetne inteligence, po zadnjih meritvah hitro narašča, večina organizacij pa še nima vzpostavljenih ustreznih postopkov za nadzor takšne uporabe [1]. Učinkovita zaščita zato ne more temeljiti le na pravilnikih in izobraževanju, temveč tudi na tehničnih ukrepih, kot so nadzor izhodnega prometa, kategorizacija storitev in ponudba odobrene alternative.

4 Umetna inteligenca v obrambi

Sodobni varnostno-operativni center ima praviloma dovolj senzorjev in virov telemetrije za učinkovito zaznavanje dogodkov. Glavna omejitev ni več količina podatkov, temveč sposobnost njihove presoje in povezovanja. Če se delež lažno pozitivnih opozoril po raziskavah približuje polovici vseh zaznav [11], analitiki preprosto nimajo dovolj časa, da bi vsak signal ustrezno obogatili s kontekstom. Prav na tem področju lahko umetna inteligenca prispeva največ.

4.1. Triaža in zmanjševanje šuma

Ena najbolj zrelih uporab umetne inteligence v SOC je obogatitev in triaža opozoril, še preden jih pregleda analitik. Model ali agent z umetno inteligenco ob prejemu opozorila samodejno pridobi dodatni kontekst – lastnika in kritičnost sredstva, zgodovino uporabnika, ugled naslova, prisotnost iste datoteke drugod v okolju ter povezane zahteve – nato pa pripravi predlog razvrstitve z obrazložitvijo in dokazi.

Ključna odločitev pri zasnovi je, da umetna inteligenca v začetni fazi uvajanja opozoril ne zapira samodejno. Deluje kot analitik prve ravni, ki pripravi kontekst in predlog, končno odločitev pa potrdi človek (angl. Human in the Loop). Samodejno zapiranje je smiselno uvesti šele za dobro razumljen in ozko opredeljen razred opozoril, pri katerem so rezultati preverjeni na dovolj velikem vzorcu ter sta zagotovljena vzorčenje in revizijska sled.

4.2. Korelacija med sistemi

Največja dodana vrednost umetne inteligence ni v analizi posameznega vira, temveč v povezovanju šibkih signalov med sistemi, ki jih pogosto spremljajo različne ekipe. Tabela 2 prikazuje poenostavljen scenarij, sestavljen iz vzorcev, ki se v praksi redno pojavljajo.

Tabela 2: Šibki signali, ki posamično ne dosežejo praga za preiskavo.

Čas	Vir	Signal	Posamična ocena
09:14	Upravljanje identitet	Uspešna prijava z MFA iz nove države	nizka
09:20	Oblachna storitev	Uporabnik odobri dovoljenje OAuth aplikaciji tretje osebe	informativno
09:26	Poštni sistem	Ustvarjeno pravilo za posredovanje e-pošte na zunanji naslov	nizka
10:02	EDR	Zagon zakrite skripte PowerShell na delovni postaji	srednja
10:40	Požarna pregrada	Odhodni promet proti domeni, registrirani pred tremi dnevi	nizka

Vsak od teh zapisov se lahko sam zase izgubi v množici opozoril. Ko jih povežemo, pa dobimo vzorec prevzema uporabniškega računa, vzpostavljanja obstojnosti in priprave na odtekanje podatkov, značilen za kompromitacijo poslovne elektronske pošte. Ključna je korelacija po identiteti in časovnem oknu, ne zgolj po posameznem viru. Prav pri tej nalogi lahko jezikovni model dobro dopolni statična pravila – poveže heterogene zapise, jih povzame v razumljivo zgodbo in predlaga naslednje korake preverjanja.

4.3. Ocenjevanje tveganja in prioritizacija ranljivosti

Za organizacijo je praktično najpomembnejše vprašanje, katere ranljivosti so dejansko izpostavljene, in ne zgolj njihovo skupno število. Ocena CVSS sama po sebi ne pove, ali je prizadeti sistem dosegljiv z interneta, ali obstaja javno dostopna izkoriščevalska koda, ali je sredstvo na poti do kritičnih podatkov in ali so vzpostavljene kompenzacijske kontrole. Umetna inteligenca je lahko posebej koristna pri združevanju podatkov o ranljivostih, izpostavljenosti, obveščevalnih podatkov o grožnjah in poslovnega konteksta.

Ob tem se pokaže pomembno neskladje. Organizacije agente v varnostnih procesih pogosto uvajajo za iskanje groženj in odzivanje na incidente, precej redkeje pa za upravljanje ranljivosti, torej za dejavnost, ki dejansko zapira vstopne točke za napadalce [12]. Znale ranljivosti zato lahko ostajajo odprte dlje, medtem ko se čas med objavo ranljivosti in njenim izkoriščanjem krajša.

4.4. Meje uporabnosti

Velja preprosto pravilo – umetna inteligenca ne more nadomestiti kakovostnih podatkov. Model brez zanesljive inventure sredstev, evidence lastništva sistemov, klasifikacije podatkov in normaliziranih dnevnikov bo hitro ustvaril prepričljive, vendar napačne sklepe. Vlaganje v kakovostno podatkovno osnovo je zato predpogoj za učinkovito uporabo umetne inteligence, ne pa alternativa zanj.

5 Ko obrambna umetna inteligenca postane tarča

Vsako orodje z umetno inteligenco, vključeno v SOC, postane tudi novo sredstvo, ki ga je treba ustrezno zaščititi. Pri tem je treba upoštevati predvsem štiri skupine tveganj.

Vbrizgavanje ukazov prek telemetrije

Če agent med preiskavo bere vsebino elektronskega sporočila, ime datoteke, vrednost polja User-Agent ali drugo vsebino opozorila, obdeluje nepreverjen vhod. Napadalec lahko vanj vstavi navodilo, namenjeno agentu in ne človeku.

Prekomerna avtonomija

Agent, ki lahko izolira končno točko, blokira uporabniški račun ali spreminja pravila požarne pregrade, je privlačna tarča. Takšne pravice morajo biti strogo omejene po obsegu in času ter vezane na lastno identiteto agenta, ne na skupni servisni račun.

Prekomerno zaupanje rezultatom

Tekoče in samozavestno napisan povzetek preiskave lahko deluje bolj verodostojno, kot upravičuje njegova dejanska zanesljivost. Vmesnik mora zato jasno ločevati med preverjenimi dejstvi iz telemetrije in sklepi, ki jih je pripravil model.

Uhajanje podatkov v model

Telemetrija SOC lahko vsebuje osebne podatke in poslovne skrivnosti. Pri uporabi zunanjih ponudnikov je zato treba jasno urediti hrambo podatkov, njihovo morebitno uporabo za učenje in lokacijo obdelave. V Evropski uniji je to tudi vprašanje skladnosti s Splošno uredbo o varstvu podatkov, direktivo NIS2 in Aktom o umetni inteligenci.

6 Priporočila

Spodnja priporočila izhajajo neposredno iz obravnavanih primerov in smernic skupnosti OWASP. Urejena so po smiselnem vrstnem redu uvajanja: najprej ukrepi, ki ne zahtevajo nove tehnologije, temveč predvsem evidenco in jasno razmejitev odgovornosti; nato ukrepi na ravni arhitekture aplikacij z umetno inteligenco; nazadnje pa ukrepi v poslovnih procesih in varnostno-operativnem centru. Priporočila so namenjena organizacijam ne glede na velikost varnostne ekipe, saj večina ni tehnološko zahtevna. Gre predvsem za dosledno uporabo že znanih varnostnih načel v novem okolju. Smiselno jih je najprej uporabiti kot kontrolni seznam za oceno trenutnega stanja in šele nato kot osnovo za načrt ukrepov, saj se pri prvem pregledu pogosto pokaže, da je največja vrzel v evidencah in procesih, ne v tehničnih kontrolah.

Evidenca in preglednost

Organizacija mora vzpostaviti evidenco sistemov in agentov z umetno inteligenco, ki vključuje vsaj lastnika, namen, vire podatkov, uporabljena orodja in dostopne pravice. Brez takšne evidence ni mogoče učinkovito ocenjevati tveganj niti se odzivati na incidente. Uporabnikom je smiselno ponuditi odobreno alternativo, uporabo neodobrenih storitev pa tehnično spremljati in omejevati.

Ločitev navodil od podatkov

Vsako zunanjo vsebino, ki vstopa v kontekst modela, je treba obravnavati kot nepreverjen vhod, kanal za navodila pa čim bolj jasno ločiti od kanala za podatke.

Najmanjša avtonomija

Nabor orodij in pravic mora biti omejen na nujno potrebno, avtorizacijo pa je treba preveriti ob vsakem klicu orodja. Dejanja z velikim vplivom morajo zahtevati človeško potrditev.

Red teaming

Aplikacije je treba preizkušati s scenariji iz obeh OWASP-ovih seznamov, vključno s posrednim vbrizgavanjem ukazov in zastrupljanjem pomnilnika oziroma konteksta.

Procesi, odporni na sintetične vsebine

Za dejanja z velikim poslovnim ali varnostnim vplivom, kot so nakazila, spremembe bančnih podatkov in ponastavitve poverilnic, mora biti uvedena potrditev po neodvisnem kanalu. Identitete ni dovoljeno potrjevati zgolj na podlagi glasu ali videa. Scenarije s sintetičnimi vsebinami je smiselno vključiti tudi v vaje odzivanja na incidente.

Detekcija

Smiselno je spremljati odhodne povezave do končnih točk ponudnikov modelov iz procesov, ki niso brskalnik ali odobreno razvojno orodje. Tak promet je lahko neposreden pokazatelj neodobrene uporabe orodij umetne inteligence in uporaben indikator za nadaljnjo preiskavo.

Postopno uvajanje v SOC

Uvajanje naj poteka po korakih. Najprej obogatitev opozoril, nato predlog razvrstitve s človeško potrditvijo, samodejno zapiranje pa šele pri dobro izmerjenih in ozko opredeljenih razredih opozoril. Pred uvedbo in po njej je smiselno spremljati delež lažno pozitivnih opozoril, delež nepreiskanih opozoril ter povprečen čas do triaže in zajeitve incidenta.

7 Zaključek

Umetna inteligenca v kibernetški varnosti ni niti zgolj orodje napadalcev niti univerzalna rešitev obrambnih težav. Je nova plast tehnološke realnosti, ki jo mora organizacija hkrati obravnavati kot tveganje (uhajanje podatkov, neodobrena uporaba, prekomerno zaupanje rezultatom), kot napadalno površino (vbrizgavanje ukazov, zloraba orodij, agentna dobavna veriga) in kot obrambno zmogljivost (triaža, korelacija in prioritizacija tveganj).

Dokumentirani primeri – od goljufije s sintetično videokonferenco in posrednega vbrizgavanja ukazov do agentno vodene vohunske kampanje – kažejo, da ne gre več le za napoved, temveč za operativno realnost. Hkrati potrjujejo, da so učinkovite kontrole večinoma dobro znane – potrditev po neodvisnem kanalu, načelo najmanjših privilegijev, ločevanje navodil od podatkov in kakovostna revizijska sled. Umetna inteligenca teh načel ne razveljavlja, spreminjajo se predvsem mesta in načini, na katerih jih je treba uveljaviti.

Najuspešnejše bodo tiste varnostne ekipe, ki bodo znale povezati kakovostne podatke, avtomatizacijo, umetno inteligenco in strokovno presojo ljudi. Vrstni red ni naključen. Brez kakovostne podatkovne osnove lahko preostali elementi ustvarjajo predvsem hitre in prepričljive napake. Tudi ob vse večji avtomatizaciji zato človeška presoja ostaja nujen del učinkovite kibernetске obrambe.

Literatura

- [1] IBM, Cost of a Data Breach Report 2026, IBM Security in Ponemon Institute, julij 2026. Dostopno na: <https://www.ibm.com/reports/data-breach> (obiskano 8. 8. 2026).
- [2] OWASP GenAI Security Project, OWASP Top 10 for LLM Applications 2026, avgust 2026. Dostopno na: <https://genai.owasp.org/resource/owasp-genai-llm-top-10-2026/> (obiskano 8. 8. 2026).
- [3] ENISA, ENISA Threat Landscape 2025, Agencija Evropske unije za kibernetiko varnost, oktober 2025. Dostopno na: <https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025> (obiskano 8. 8. 2026).
- [4] CNN Business, »Arup revealed as victim of \$25 million deepfake scam involving Hong Kong employee«, 16. 5. 2024. Dostopno na: <https://www.cnn.com/2024/05/16/tech/arup-deepfake-scam-loss-hong-kong-intl-hnk> (obiskano 8. 8. 2026).
- [5] Anthropic, Disrupting the First Reported AI-Orchestrated Cyber Espionage Campaign, november 2025. Dostopno na: <https://assets.anthropic.com/m/ec212e6566a0d47/original/Disrupting-the-first-reported-AI-orchestrated-cyber-espionage-campaign.pdf> (obiskano 8. 8. 2026).
- [6] Just Security, »The Era of AI-Orchestrated Hacking Has Begun«, januar 2026. Dostopno na: <https://www.justsecurity.org/127053/era-ai-orchestrated-hacking/> (obiskano 8. 8. 2026).
- [7] OWASP GenAI Security Project, pregled delovnih področij projekta. Slika dostopna na: <https://genai.owasp.org/wp-content/uploads/2025/07/OWASP-GenAI-Security-Project-right-img.png> (obiskano 8. 8. 2026).
- [8] OWASP GenAI Security Project, OWASP Top 10 for Agentic Applications 2026, december 2025. Dostopno na: <https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/> (obiskano 8. 8. 2026).
- [9] REDDY P., GUJRAL A. S., »EchoLeak: The First Real-World Zero-Click Prompt Injection Exploit in a Production LLM System«, arXiv:2509.10540, september 2025. Dostopno na: <https://arxiv.org/abs/2509.10540> (obiskano 8. 8. 2026).
- [10] NIST National Vulnerability Database, CVE-2025-32711. Dostopno na: <https://nvd.nist.gov/vuln/detail/CVE-2025-32711> (obiskano 8. 8. 2026).
- [11] Microsoft in Omdia, State of the SOC, 2026 (povzeto po javnih objavah industrijskih analiz).
- [12] Help Net Security, »Data breach cost 2026 averaged \$4.99 million, AI attacks ran higher«, 30. 7. 2026. Dostopno na: <https://www.helpnetsecurity.com/2026/07/30/ibm-cost-of-a-data-breach-2026/> (obiskano 8. 8. 2026).

Sistemiški napadi na programske dobavne verige v dobi umetne inteligence

Gregor Jošt, Viktor Taneski

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
gregor.jost@um.si, viktor.taneski@um.si

Programske dobavne verige so zaradi vse večje odvisnosti od odprtokodnih komponent, avtomatiziranih razvojnih procesov, oblčnih storitev in medsebojno povezanih identitetnih sistemov postale pomembna tarča kibernetških napadov. Prispevek analizira njihov razvoj od zlorab posameznih paketov in računov vzdrževalcev do večstopenjskih sistemskih napadov, ki izkoriščajo povezave zaupanja med repozitoriji, okolji CI/CD, registri artefaktov, kontejnerskimi slikami in produkcijsko infrastrukturo. Obravnavani so ključni napadalni vektorji, vključno z zastrupljanjem paketov, zlorabo razvojnih identitet, krajo skrivnosti, napačnimi nastavitvami OIDC in zlorabo procesov gradnje ter distribucije. Dodatna pozornost je namenjena umetni inteligenci, ki sama po sebi ni nov vzrok napadov, vendar povečuje hitrost razvoja, obseg avtomatizacije in količino sprememb, ki zahtevajo preverjanje. Razvojni agenti dodatno širijo napadalno površino zaradi dostopa do kode, orodij in privilegiranih identitet. Prispevek ugotavlja, da učinkovita zaščita zahteva sistemski in večplastni pristop, ki združuje upravljanje odvisnosti, preverljivo poreklo artefaktov, utrjevanje okolij CI/CD, najmanjše privilegije, nadzor dobaviteljev ter neprekinjeno spremljanje celotne verige zaupanja.

Ključne besede:

programske dobavne verige

napadi na dobavne verige

verige zaupanja

programske odvisnosti

generativna umetna inteligenca

1 Uvod

Programska oprema danes praviloma ni rezultat izoliranega razvoja v eni organizaciji, temveč nastaja v razvejanem ekosistemu odprtokodnih knjižnic, registrov paketov, sistemov za upravljanje izvorne kode, okolij in storitev za neprekinjeno integracijo in dostavo (angl. Continuous Integration and Continuous Delivery, CI/CD), oblačne infrastrukture ter zunanjih ponudnikov. Programsko dobavno verigo zato razumemo kot celoto ljudi, procesov, orodij in artefaktov, ki sodelujejo pri načrtovanju, razvoju, gradnji, testiranju, distribuciji, uvajanju in vzdrževanju programske rešitve. Varnost končnega izdelka ni odvisna zgolj od kakovosti lastne izvorne kode, temveč tudi od integritete vseh komponent, orodij in odnosov zaupanja v celotni verigi izdelave in distribucije programske opreme [1], [2].

Takšna povezanost prinaša pomembne razvojne prednosti, hkrati pa ustvarja izrazit sistemski učinek. Napadalcu ni treba neposredno ogroziti vsake ciljne organizacije, saj je učinkoviteje prevzeti komponento, račun, storitev ali avtomatizirani proces, ki mu zaupa več uporabnikov. Agencija Evropske unije za varnost omrežij in informacij (ENISA) napad na dobavno verigo opredeljuje kot napad, pri katerem napadalec za doseg končnega cilja napade dobavitelja in izkoristi zaupanje med dobaviteljem in njegovimi strankami. Analiza incidentov kaže, da tarča ni nujno samo koda, temveč tudi mehanizmi posodabljanja, okolja za gradnjo programske opreme (angl. build environment), podpisni ključi, identitete delovnih procesov in distribucijski kanali [2]. Posledično lahko en sam varnostni incident povzroči kaskadno širjenje zlonamerne kode ali nepooblaščenega dostopa med velikim številom organizacij.

Raziskovalni problem tega prispevka izhaja iz dejstva, da klasične obravnave napadov na dobavne verige pogosto analizirajo posamezne vektorje ločeno, na primer zlonamerne pakete, prevzete račune vzdrževalcev ali zlorabljenega okolja za gradnjo programske opreme. Sodobni incidenti pa kažejo, da napadalci te vektorje vse pogostejše povezujejo v večstopenjske operacije: začetna zloraba odvisnosti ali razvijalca omogoči dostop do okolja CI/CD, tam pridobljene skrivnosti in identitete pa se uporabijo za prevzem dodatnih repozitorijev, registrov, oblačnih storitev ali produkcijskih sistemov. Posledično je osrednje vprašanje prispevka, kako se napadi razvijajo od zlorabe posamezne komponente v sistemske napade na verige zaupanja in kako generativna umetna inteligenca spreminja hitrost, obseg in nadzor nad takšnimi procesi.

Cilji prispevka so: (1) prikazati zgodovinski razvoj napadov na programske dobavne verige; (2) sistematizirati ključne napadalne vektorje sodobnih razvojnih ekosistemov; (3) pojasniti njihovo povezovanje v večstopenjske in kaskadne napadalne verige; (4) analizirati vlogo umetne inteligence kot pospeševalca razvoja in napadov; ter (5) predstaviti tehnične in organizacijske obrambne pristope. Prispevek obstoječo literaturo dopolnjuje s sistemskim pogledom, v katerem programska dobavna veriga ni predstavljena kot zaporedje neodvisnih komponent, temveč kot dinamično omrežje identitet, avtomatiziranih procesov in prenosljivega zaupanja. Takšen pogled je skladen tudi s sodobnimi okviri, kot sta NIST Secure Software Development Framework (v nadaljevanju SSDF) in Supply-chain Levels for Software Artifacts (v nadaljevanju SLSA), ki varnosti ne omejujeta na pregled izvorne kode, temveč poudarjata zaščito razvojnega okolja, integriteto artefaktov in preverljivo poreklo programske opreme [1][3].

2 Evolucija napadov na programske dobavne verige

Zamisel, da je mogoče ogroziti program prek orodij in procesov, s katerimi je bil ustvarjen, ni nova. Thompson je že leta 1984 v delu *Reflections on Trusting Trust* pokazal, da je lahko zlonamerna funkcionalnost skrita v prevajalniku in se ohranja tudi takrat, ko je izvorna koda ciljnega programa navidezno neoporečna [4]. Temeljna ugotovitev je ostala aktualna: preverjanje posamezne kode ne zadošča, če ni mogoče zaupati orodju, okolju in postopku, ki so to kodo pretvorili v izvedljiv artefakt.

Z razmahom odprtokodne programske opreme in javnih registrov paketov se je problem zaupanja razširil na množico ponovno uporabljenih komponent. Razvijalci so začeli aplikacije sestavljati iz neposrednih in prehodnih

programskih odvisnosti (angl. software dependencies), ki jih pogosto vzdržujejo posamezniki ali majhne skupnosti. Napadalcu so to okolje izkoristili z objavo zlonamernih paketov, zamenjavo podobno poimenovanih paketov, prevzemom zapuščenih projektov in zlorabo računov vzdrževalcev. Ohm in sodelavci so ročno zbrali in analizirali 174 zlonamernih paketov, ki so bili med novembrom 2015 in novembrom 2019 uporabljeni v dejanskih napadih ter distribuirani prek registrov npm, PyPI in RubyGems. Analiza je pokazala, da so bili takšni napadi že pred velikimi sistemskimi incidenti raznoliki, praktično izvedljivi in pogosto dolgo neodkriti [5].

Naslednja razvojna stopnja je bila usmeritev v infrastrukturo za gradnjo in distribucijo. Incidenti, kot sta SolarWinds in Codecov, so pokazali, da lahko zloraba zaupanja vrednega postopka posodabljanja ali razvojne storitve napadalcu omogoči dostop do številnih uporabnikov brez neposrednega napada na njihove sisteme. Pri SolarWinds je bila zlonamerna koda vključena v legitimne posodobitve platforme Orion, pri Codecovu pa je napadalec spremenil Bash Uploader in prek izvajanja v okoljih CI pridobival okoljske spremenljivke ter druge občutljive podatke [6], [7]. S tem se je težišče premaknilo od vprašanja, ali je ranljiva posamezna knjižnica, k vprašanju, kdo nadzira proces, ki programsko opremo gradi, podpisuje in objavlja.

Današnji napadi nadaljujejo ta trend z izkoriščanjem povezav med registri paketov, repozitoriji, CI/CD platformami, storitvami SaaS, kontejnerskimi registri, oblračnimi računi in federiranimi identitetami. Programska dobavna veriga je postala omrežje, v katerem legitimna poverilnica (angl. credential) ali zaupanja vredna avtomatizacija pogosto omogoča večji doseg kot tehnična ranljivost v končni aplikaciji. Evolucijo napadov zato lahko razumemo kot prehod od vstavljanja zlonamerne kode v posamezen artefakt k sistematičnemu prevzemanju mehanizmov, ki vzpostavljajo in širijo zaupanje.

2.1. Tradicionalni modeli napadov

Tradicionalni model napada je bil praviloma osredotočen na eno komponento ali eno distribucijsko točko. Napadalec je ustvaril zlonameren paket, posnemal ime legitimne knjižnice, prevzel račun vzdrževalca ali v obstoječo odvisnost vključil zlonamerno funkcionalnost. Uspeh je bil odvisen od tega, ali bodo razvijalci paket uporabili kot neposredno odvisnost ali pa bo do njihovih projektov prišel prek prehodne odvisnosti (angl. transitive dependency). Napadi so pogosto uporabljali namestitvene skripte, prikrito omrežno komunikacijo, krajo lokalnih poverilnic ali ciljno logiko, ki se je aktivirala samo v določenem okolju [5][8].

Kljub velikemu potencialnemu dosegu so bili zgodnji modeli pogosto omejeni glede nadaljnjega širjenja. Zlonamerna koda je bila vezana na konkretni paket, operacijski sistem ali skupino uporabnikov; napadalec je moral čakati na namestitev, posodobitev ali napako razvijalca. Prav tako je bila povezava med začetno zlorabo in drugimi deli razvojne infrastrukture manj avtomatizirana. Napad je lahko ukradel lokalne podatke ali vzpostavil začetni dostop, ni pa nujno samodejno ustvaril novih zaupanja vrednih objav, prevzel drugih paketov ali pridobil identitet za dostop do oblachne infrastrukture.

Te omejitve ne pomenijo, da so tradicionalni napadi izginili. Nasprotno, še vedno predstavljajo pogost začetni vektor, vendar so danes pogosto samo prva faza širše operacije. Njihova nevarnost se bistveno poveča, ko je okužena komponenta izvedena v okolju CI/CD z razširjenimi pravicami, kjer so dostopni žetoni (angl. access tokens) za objavo paketov, ključi za oblak, podpisne poverilnice ali pravice za spreminjanje drugih repozitorijev.

2.2. Prehod v sistemske napade

Prehod v sistemske napade je posledica spremembe arhitekture razvoja programske opreme. Sodobni razvojni tokovi avtomatizirano povezujejo spremembo izvirne kode z gradnjo, testiranjem, varnostnim preverjanjem, podpisovanjem, objavo artefakta in namestitvijo v produkcijo. Vsaka faza uporablja storitvene račune, žetone, skrivnosti ali federirane identitete (angl. federated identities; identitete, ki omogočajo dostop do več povezanih storitev brez ločenih računov), zato začetna zloraba ne predstavlja več nujno končnega cilja, temveč izhodišče za premikanje po celotni verigi.

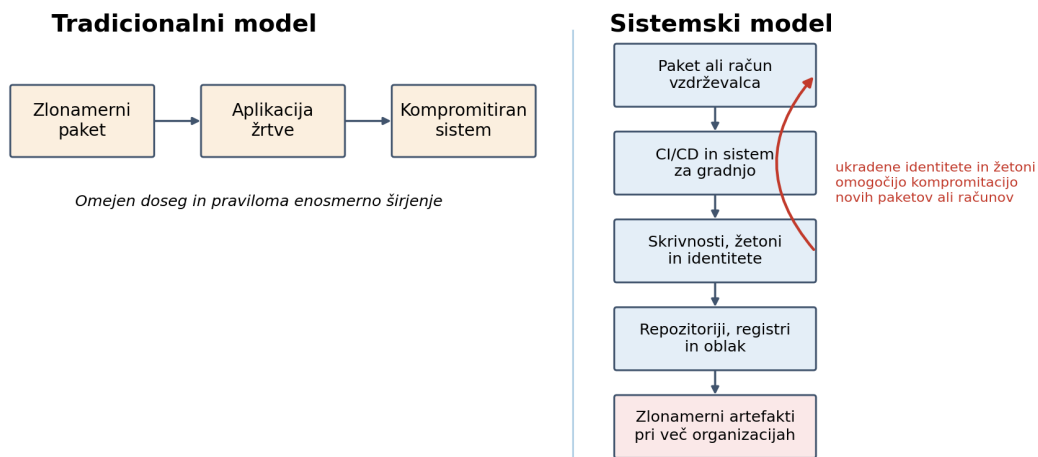
Okolje CI/CD je pri tem posebej pomembno, ker združuje izvirno kodo, orodja za gradnjo in visoko privilegirane dostopne podatke. Spremenjen delovni tok ali proces gradnje lahko prebere okoljske spremenljivke (angl.

environment variables), zlorabi izvajalno okolje, spremeni končni artefakt ali objavi novo različico prek legitimnega računa. Izrazit primer takšnega sistemkega napada je zloraba XZ Utils oziroma knjižnice liblzma, odkrita marca 2024. Napadalec je z večletnim socialnim inženiringom postopoma pridobil položaj zaupanja vrednega vzdrževalca projekta, nato pa je v izdaji 5.6.0 in 5.6.1 vključil prikrito zlonamerno logiko. Ključni deli mehanizma niso bili neposredno vidni v običajni izvorni kodi, temveč so bili vključeni v distribucijske arhive in aktivirani med procesom gradnje, kar je v določenih okoljih omogočilo poseganje v avtentikacijsko pot strežnika OpenSSH [9], [10].

Oblache in SaaS storitve dodatno razširjajo napadalno površino. Repozitorij kode, CI/CD platforma, register paketov in oblaci ponudnik so lahko ločene storitve, vendar jih povezujejo enotna prijava (angl. Single Sign-On, SSO), aplikacijske integracije, dogodkovni klici HTTP in avtomatizirane vloge. Zloraba identitete v eni storitvi lahko zato odpre pot v druge sisteme. Posebej občutljive so preširoko nastavljene federirane povezave OIDC: če napadalec doseže izvajanje kode v zaupanja vrednem delovnem toku, lahko pridobi veljaven kratkoživ žeton in ga uporabi za dostop do oblčnih virov. V tem primeru ni zaobšel avtentikacije, temveč je zlorabil pravilo zaupanja, ki je njegovo dejavnost prepoznalo kot legitimno [11].

Sistemske napade dodatno zaznamujeta avtomatizacija in sposobnost samodejnega širjenja. Ukradeni žetoni za objavo paketov se lahko uporabijo za zlorabo novih paketov, ti pa v okoljih naslednjih žrtev pridobijo nove poverilnice. Nastane povratna zanka, v kateri vsak uspešen napad poveča napadalčev doseg in število zaupanja vrednih identitet. Sodobne kampanje zato niso več omejene na en artefakt, temveč lahko sočasno ciljajo razvijalske delovne postaje, repozitorije, CI/CD izvajalce (angl. runners), registre in oblčne račune.

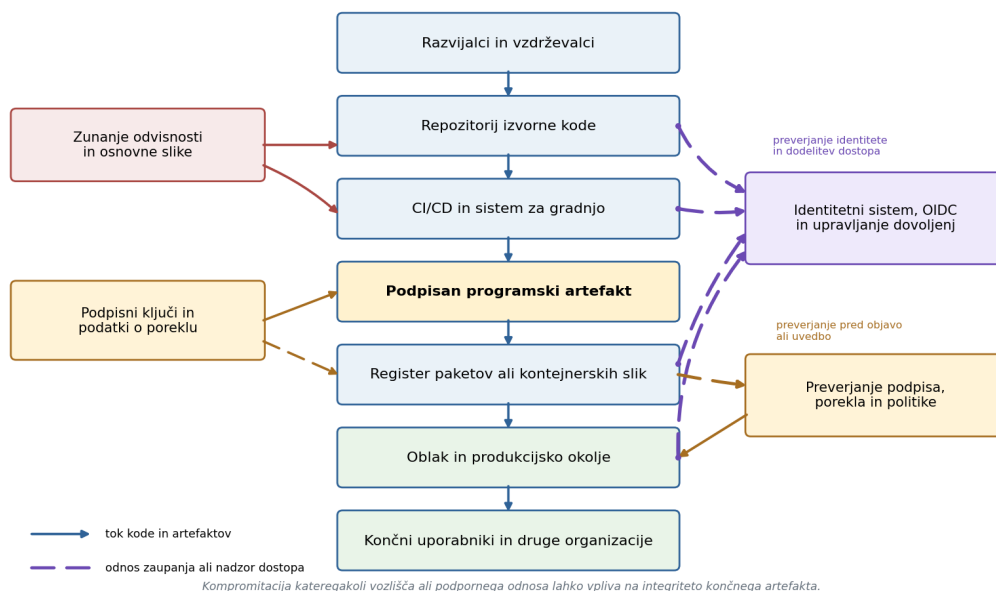
Kot prikazuje slika 1, je ključna sprememba v razmišljanju napadalcev premik od vprašanja »v katero komponento je mogoče vnesti zlonamerno spremembo« k vprašanju »katera povezava zaupanja omogoča napadalcu največji nadaljnji doseg«. V sistemskem modelu so koda, identiteta, avtomatizacija in infrastruktura medsebojno zamenljive poti do istega cilja. Obramba mora zato zagotavljati preverljivo sled izvora in gradnje, ločevanje privilegijev, kratkožive in strogo omejene identitete ter neodvisno preverjanje artefaktov. NIST SSDF in SLSA takšno usmeritev formalizirata z zahtevami po zaščitenih razvojnih okoljih, nadzoru sprememb, integriteti gradnje in podatkih o poreklu (angl. provenance) [1], [3].



Slika 1: Prehod od kompromitacije posamezne komponente k sistemskemu napadu na verigo zaupanja.

3 Ključni napadalni vektorji sodobnih dobavnih verig

Kot je bilo že izpostavljeno, varnost programske dobavne verige ni odvisna zgolj od posameznih komponent, temveč od povezav med sistemi, identitetami in avtomatiziranimi procesi, ki omogočajo njihov nastanek in distribucijo. Slika 2 prikazuje glavni tok programske opreme od razvijalcev in izvirne kode prek gradnje, podpisovanja in objave do produkcije ter končnih uporabnikov.



Slika 2: Tok programske opreme in podporni odnosi zaupanja v sodobni programski dobavni verigi.

V nadaljevanju so predstavljeni ključni napadalni vektorji sodobnih programskih dobavnih verig, izbrani glede na njihovo povezanost z osrednjimi točkami zaupanja v procesu razvoja in distribucije programske opreme. Obravnavani so napadi na programske odvisnosti (angl. software dependencies), zlorabe računov razvijalcev in vzdrževalcev (angl. maintainer account compromise), napadi na okolja CI/CD ter zlorabe identitetnih sistemov in federiranega dostopa (angl. federated identity and access). Poseben poudarek je namenjen točkam zaupanja, katerih zloraba omogoča napadalcem širjenje vpliva prek več komponent in organizacij. Tabela 1 povzema obravnavane vektorje, pripadajoče točke zaupanja in značilne posledice, ki jih podrobneje obravnavajo naslednja podpoglavja.

Tabela 1: Pregled ključnih napadalnih vektorjev in kompromitiranih točk zaupanja.

Napadalni vektor	Točka zaupanja	Možna posledica	Primer
Zmeda odvisnosti in zastrupljanje paketov	Register in izbira odvisnosti	Izvajanje zlonamernega paketa	Birsanova raziskava; event-stream
Prevzem računa ali vzdrževanja	Identiteta razvijalca oziroma vzdrževalca	Legitimna objava škodljive različice	event-stream; ua-parser-js
Zloraba CI/CD	Delovni tokovi, izvajalci in skrivnosti	Kraja poverilnic ter dostop do drugih sistemov	Codecov; Trivy; KICS
Napačna nastavitve OIDC	Pravila federiranega zaupanja	Pridobitev legitimnih začasnih poverilnic	TanStack
Kompromitirana kontejnerska slika	Register in proces gradnje slike	Širjenje škodljive kode v razvoj in produkcijo	Docker Hub kampanje

Tabela prikazuje napadalne vektorje ločeno zaradi lažje analize posameznih točk zaupanja, vendar v praksi ti vektorji niso nujno neodvisni. Sodobni napadi na programske dobavne verige pogosto kombinirajo več tehnik, pri čemer kompromitacija ene komponente omogoči dostop do naslednje. Posamezen incident lahko zato vključuje več navedenih kategorij hkrati: napadalec lahko na primer z zlorabo CI/CD procesa pridobi dostopne žetone, jih

uporabi za objavo zlonamernega paketa in nato izkoristi zaupanje uporabnikov v legitimne posodobitve. Takšne večstopenjske napadalne verige so podrobneje obravnavane v naslednjem poglavju.

3.1. Zmeda odvisnosti in zastrupljanje paketov

Razvijalci pri izdelavi aplikacij redno uporabljajo odprtokodne knjižnice in pakete, ki jih vključijo v svoje projekte prek upravljalnikov paketov (angl. package managers), kot so npm za JavaScript, PyPI za Python ali Maven za Javo. Tak pristop omogoča hitrejši razvoj, vendar hkrati pomeni, da organizacija zaupa tudi kodi, ki je sama neposredno ne razvija in ne nadzoruje. Programske odvisnosti (angl. software dependencies) zato predstavljajo pomemben del dobavne verige programske opreme. Čeprav razvijalci pogosto preverjajo lastno kodo, se lahko zlonamerna funkcionalnost skriva v eni izmed uporabljenih zunanjih komponent. Napadalec tako ne napade neposredno končnega sistema, temveč izkoristi zaupanje, ki ga organizacija namenja svojim odvisnostim.

Zmeda odvisnosti (angl. dependency confusion) je napad, pri katerem napadalec izkoristi način, kako sistemi za upravljanje paketov izbirajo in prenašajo odvisnosti. V številnih organizacijah obstajajo zasebni paketi (angl. private packages), ki niso javno objavljeni, vendar jih razvojna okolja uporabljajo pri gradnji aplikacij. Napadalec lahko ustvari javni paket z enakim imenom in ga objavi v javnem registru, na primer npm ali PyPI. Če sistem za namestitve paketov ne razlikuje pravilno med zasebnim in javnim virom, lahko prenese napadalčev paket namesto notranje različice. S tem napadalec vstavi svojo kodo neposredno v razvojni proces organizacije [12]. Koncept zmede odvisnosti je leta 2021 javno predstavil Alex Birsan, ki je pokazal, da je mogoče s to tehniko vplivati na razvojna okolja številnih velikih podjetij, med drugim Microsofta, Apple in drugih organizacij. Njegova raziskava je pokazala, da lahko že sama kombinacija javnega registra paketov in nepravilno konfiguriranih odvisnosti predstavlja pomembno varnostno tveganje [13].

Zastrupljanje paketov (angl. package poisoning) predstavlja širši razred napadov, pri katerih napadalec v programski paket vstavi zlonamerno kodo. Paket je lahko povsem nov ali pa gre za obstoječ paket, ki ga razvijalci že uporabljajo in mu zato zaupajo. Cilj napadalca je pogosto pridobiti dostop do razvojnih okolij, ukrasti skrivnosti (angl. secrets) ali izvesti nadaljnje napade v organizaciji [14]. Poseben problem predstavlja dejstvo, da imajo številni odprtokodni paketi veliko število uporabnikov, vendar jih vzdržuje majhno število posameznikov. Zaradi tega lahko zloraba ene same komponente vpliva na veliko število aplikacij in organizacij.

Eden izmed najbolj znanih primerov je incident s paketom event-stream v ekosistemu npm leta 2018. Napadalec je pridobil dostop do vzdrževanja priljubljenega paketa in vanj vključil dodatno odvisnost, ki je vsebovala zlonamerno kodo. Ker je bil paket že uveljavljen in široko uporabljen, je napad dosegel veliko število uporabnikov [15], [16]. Podoben primer predstavlja kompromitacija paketa ua-parser-js leta 2021, pri katerem je bila v uveljavljeni paket npm vključena zlonamerna koda. Napad je pokazal, da lahko zlonamerna sprememba enega samega priljubljenega paketa vpliva na veliko število razvojnih okolij in aplikacij [15].

Napadalci pogosto izkoriščajo tudi človeške napake pri izbiri paketov. Zloraba tipkarskih napak (angl. typosquatting) je tehnika, pri kateri napadalec ustvari paket z imenom, ki je zelo podobno priljubljenemu paketu. Primer takega pristopa je ustvarjanje paketov z imeni, pri katerih so spremenjene posamezne črke, dodani dodatni znaki ali uporabljene različice, podobne izvirnemu imenu. Ker razvijalci pogosto iščejo pakete neposredno prek registrov, lahko takšni paketi na prvi pogled delujejo verodostojno, dokler uporabnik ne preveri njihovega izvora [17]. Pri klasičnih spletnih napadih se typosquatting pogosto povezuje z lažnimi spletnimi stranmi, vendar enak princip velja tudi za ekosisteme programskih paketov, kot so npm, PyPI in drugi registri.

Poleg ustvarjanja novih zlonamernih paketov napadalci izkoriščajo tudi obstoječe projekte. Zapuščeni paketi (angl. abandoned packages) so knjižnice, ki jih razvijalci sicer še vedno uporabljajo, vendar jih prvotni vzdrževalci ne posodablja več aktivno. Če napadalec pridobi nadzor nad takim paketom, ga lahko spremeni in objavi novo različico, ki jo uporabniki samodejno prenesejo kot del običajnega procesa posodabljanja. Problem je še posebej izrazit pri ekosistemih, kjer se zaupa predvsem imenu paketa in številki različice, manj pa dejanskemu preverjanju sprememb.

Napadi na programske pakete kažejo, da dobavna veriga temelji na številnih povezavah zaupanja. Napadalci izkoriščajo slabosti registrov, paketov, človeških napak in vzdrževanja odprtokodnih projektov, pri čemer za širjenje vpliva pogosto zadošča ena sama ogrožena komponenta.

3.2. Kompromitacija računov vzdrževalcev in razvijalcev

Razvijalci in vzdrževalci odprtokodnih projektov imajo pogosto posebno vlogo, saj njihov račun omogoča objavo novih različic paketov, spreminjanje izvirne kode ali dostop do razvojne infrastrukture. Zaradi tega lahko prevzem računa predstavlja bistveno večje tveganje kot napad na posamezen strežnik. Če napadalec pridobi dostop do računa legitimnega vzdrževalca, so lahko spremembe v programski opreми videti popolnoma zaupanja vredne. Namesto neposrednega napada na uporabnike napadalec izkoristi obstoječi odnos zaupanja med vzdrževalcem, registrom paketov in končnimi uporabniki.

Eden najpogostejših načinov prevzema računov je kraja prijavnih podatkov prek lažnega predstavljanja (angl. phishing). Napadalci ustvarijo lažne prijavnice strani ali komunikacijo, ki posnema legitimiteto storitve, kot so GitHub, npm ali drugi razvojni sistemi. Cilj je pridobiti uporabniško ime, geslo ali dodatne podatke za preverjanje identitete. Ker številne organizacije uporabljajo večfaktorsko avtentikacijo (angl. Multi-Factor Authentication, MFA), napadalci vse pogosteje uporabljajo naprednejše tehnike, ki poskušajo pridobiti tudi enkratne kode ali uporabnika preusmeriti skozi lažen prijavni proces. S tem lahko zaobidejo tradicionalno zaščito in pridobijo dostop do računa, ki ima dovoljenja za objavo programske opreme.

Poleg gesel predstavljajo pomembno tveganje tudi dostopni žetoni (angl. access tokens). Ti omogočajo avtomatiziran dostop do storitev brez ponovne prijave uporabnika. Če napadalec pridobi veljaven žeton, lahko pogosto izvaja dejanja v imenu uporabnika, dokler žeton ni preklican. To je posebej nevarno v razvojnih okoljih, kjer so računi vzdrževalcev pogosto povezani z več storitvami, kot so repozitoriji, registri paketov, sistemi CI/CD in oblačne platforme. Ukraden žeton lahko tako predstavlja vstopno točko do več različnih delov dobavne verige.

Odprtokodni vzdrževalci so zaradi svoje vloge pogosto zanimive tarče. Napadalci lahko uporabijo tehnike socialnega inženiringa (angl. social engineering), pri katerih poskušajo uporabnika prepričati, da izvede določeno dejanje, na primer odpre povezavo, namesti orodje ali potrdi zahtevo za dostop. Problem je še posebej izrazit pri priljubljenih odprtokodnih projektih, kjer lahko en sam vzdrževalec nadzoruje paket, ki ga uporablja veliko število aplikacij. Napadalcu zato ni treba iskati ranljivosti v vseh ciljnih sistemih; dovolj je prevzeti račun ene osebe, ki ima ustrezne pravice.

Incident event-stream, podrobneje predstavljen v poglavju 3.1, kaže tudi na nevarnost socialnega prevzema vzdrževanja projekta. Napadalec je pridobil zaupanje prvotnega vzdrževalca in nato prek legitimnega postopka objavil zlonamerno odvisnost. Nato je v paket vključil novo odvisnost flatmap-stream, ki je vsebovala zlonamerno kodo. Ker je bila sprememba objavljena prek legitimnega računa in zaupanja vrednega projekta, se je razširila po običajnem postopku posodabljanja.

Podoben vzorec se je pojavil tudi pri napadih na pakete ua-parser-js (2021), chalk, debug in eslint-config-prettier (2025), kjer so napadalci po prevzemu računov vzdrževalcev objavili zlonamerne različice priljubljenih knjižnic. Takšni primeri kažejo, da lahko prevzet račun razvijalca predstavlja učinkovitejšo vstopno točko kot neposreden napad na infrastrukturo [18], [19].

Posebej odmevna je bila kampanja Shai-Hulud leta 2025. Za razliko od posameznih zlonamerno spremenjenih paketov je šlo za kampanjo, pri kateri je kompromitirana koda v paketih avtomatizirano iskala dodatne skrivnosti, dostopne žetone in druge poverilnice (angl. credentials) ter jih uporabljala za nadaljnje širjenje. Po prevzemu računov vzdrževalcev so bile objavljene zlonamerne različice paketov, ki so omogočile nadaljnje širjenje napada na povezane pakete in razvojna okolja. Takšen način delovanja kaže značilnosti črva (angl. worm), kjer posamezna uspešna kompromitacija ustvari pogoje za naslednje faze napada [18].

Podoben primer predstavlja napad na projekte, ki uporabljajo GitHub Actions za objavo paketov na PyPI. Napadalci so po pridobitvi dostopa do izvirne kode spremenili delovne tokove GitHub Actions in dodali korake,

namenjene kraji žetonov za objavo PyPI (angl. PyPI API tokens). Ob naslednji izvedbi delovnega toka so se žetoni prenesli napadalcem, ki bi jih lahko uporabili za objavo zlonamernih različic paketov. Čeprav PyPI kot platforma ni bila ogrožena, je incident pokazal, kako lahko kompromitiran razvojni proces ogrozi celotno dobavno verigo [20].

3.3. Zloraba okolij CI/CD

Sistemi CI/CD predstavljajo osrednji del sodobnega razvoja programske opreme. Omogočajo avtomatizirano gradnjo, testiranje in objavo aplikacij, s čimer razvojne ekipe hitreje in z manj ročnega dela dostavljajo spremembe v produkcijska okolja. Ravno zaradi svoje vloge so okolja CI/CD privlačna tarča za napadalce. Med izvajanjem razvojnih procesov imajo pogosto dostop do izvirne kode, skrivnosti (angl. secrets), dostopnih žetonov (angl. access tokens), podpisnih ključev in poverilnic (angl. credentials) za dostop do oblačne infrastrukture. Zloraba okolja CI/CD zato lahko napadalcu omogoči dostop do večjega števila sistemov, ne da bi moral neposredno napasti produkcijsko okolje [21], [22].

Eden najpogostejših ciljev napadalcev pri napadih na okolja CI/CD je odtekanje skrivnosti (angl. secret exfiltration). Razvojni procesi pogosto potrebujejo občutljive podatke za dostop do zunanjih storitev, na primer API ključev, poverilnic za oblačne ponudnike ali žetonov za objavo paketov. Če napadalec pridobi dostop do teh podatkov, lahko pogosto nadaljuje napad brez ponovnega izkoriščanja ranljivosti. Ukradene poverilnice lahko uporabi za dostop do repozitorijev, spreminjanje programske kode, objavo zlonamernih paketov ali dostop do produkcijske infrastrukture [23], [24].

Poseben problem predstavlja dejstvo, da so CI/CD sistemi pogosto zasnovani tako, da imajo obsežnejše pravice dostopa kot posamezni razvijalci. Njihov namen je avtomatizacija, zato morajo imeti dovoljenja za izvajanje operacij, ki jih običajni uporabniki ne bi smeli izvajati. Če napadalec prevzame tak proces, lahko zlorabi že obstoječe pravice.

Napadalci lahko okolja CI/CD zlorabijo na več načinov. Eden izmed pogostih pristopov je sprememba delovnih tokov (angl. workflows) oziroma konfiguracijskih datotek, ki določajo, katere ukaze sistem izvede med gradnjo in objavo programske opreme. Če napadalec pridobi dostop do repozitorija ali razvojnega računa, lahko v CI/CD proces doda dodatne ukaze, ki se izvajajo skupaj z legitimnimi opravili. Ti ukazi lahko na primer zbirajo okoljske spremenljivke, izvozijo skrivnosti ali pošiljajo ukradene podatke napadalcu. Ker se vse izvaja v okviru legitimnega razvojnega procesa, je takšne spremembe težje opaziti [21], [23]. Enaka načela veljajo za različne platforme, kot so GitHub Actions, GitLab CI in Azure DevOps. Razlike med posameznimi sistemi obstajajo, vendar je skupna točka enaka: CI/CD predstavlja zaupanja vredno okolje, ki ima dostop do številnih drugih sistemov.

Največja nevarnost ogroženega okolja CI/CD je možnost prehoda iz razvojnega procesa v produkcijsko okolje. V tradicionalnih napadih napadalec običajno išče ranljivosti neposredno v produkcijskih sistemih. Pri napadih na dobavno verigo pa lahko izkoristi razvojni proces kot vstopno točko. Če CI/CD sistem uporablja poverilnice za dostop do produkcije, lahko napadalec po zlorabi CI/CD cevovoda pridobi možnost nameščanja spremenjene programske opreme, spreminjanja infrastrukture ali dostopa do oblačnih storitev. Razvojni proces tako postane most med izvirno kodo in produkcijo [21], [22]. Zato zaščita okolij CI/CD ni več samo vprašanje varnosti razvojnih orodij, temveč predstavlja pomemben del celotne varnostne strategije organizacije.

Eden izmed novejših primerov predstavlja val napadov, povezanih z ekosistemom Trivy leta 2026, kjer so napadalci spremenili komponente, ki se uporabljajo v CI/CD procesih za varnostno preverjanje vsebnikov. Napadalci so izkoristili legitimno orodje in vanj vključili kodo za krajo poverilnic. Zlonamerna različica je poskušala pridobiti predvsem poverilnice za oblačna okolja, kot so AWS ključi in druge skrivnosti, ki so bile dostopne v CI/CD izvajalnih okoljih. Incident je pokazal, da lahko tudi varnostna orodja sama postanejo del napadalne površine, če imajo širok dostop do razvojnih procesov [25].

Podoben primer predstavlja napad na orodje Checkmarx KICS leta 2026. Napadalci jim je uspelo objaviti zlonamerno spremenjene različice artefaktov, povezanih z orodjem, vključno z Docker slikami in razširitvami za

razvojna okolja. Zlonamerne različice so bile namenjene kraji podatkov iz razvojnih okolij, vključno s skrivnostmi in drugimi občutljivimi informacijami, ki jih orodja za varnostno analizo pogosto obdelujejo. Primer kaže, da lahko napad na razvojno orodje omogoči dostop do informacij, ki so običajno zaščitene v razvojnem procesu [25].

Dodaten primer predstavlja vzorec ranljivosti Cordyceps iz leta 2026, pri katerem so raziskovalci pokazali, da lahko napačne konfiguracije delovnih tokov CI/CD omogočijo dostop do velikega števila repozitorijev GitHub. Analiza je pokazala, da lahko takšni avtomatizirani razvojni procesi razkrijejo dostop do izvirne kode, skrivnosti in drugih privilegiranih virov. Primer ne predstavlja potrjene množične napadalne kampanje, temveč kaže, kako lahko pomanjkljivo zaščiteni procesi CI/CD ustvarijo pomembno tveganje za programsko dobavno verigo [26].

3.4. Identitetni sistemi in OIDC

V sodobnih razvojnih okoljih identiteta postaja ena od najpomembnejših varnostnih točk. Medtem ko so se tradicionalni napadi pogosto osredotočali na krajo gesel ali skrivnosti (angl. secrets), sodobna oblačna okolja vse pogosteje temeljijo na preverjanju identitete uporabnikov, aplikacij in avtomatiziranih procesov. To je še posebej pomembno pri okoljih CI/CD, kjer razvojni sistemi potrebujejo dostop do drugih storitev, na primer oblačne infrastrukture, registrov paketov ali produkcijskih okolij. Namesto shranjevanja dolgoročnih poverilnic se danes pogosto uporablja federirana identiteta (angl. federated identity), pri kateri ena storitev preveri identiteto druge in ji dodeli začasen dostop.

Federirana identiteta omogoča, da sistem ne uporablja neposredno shranjenih uporabniških imen, gesel ali trajnih ključev, temveč zaupa drugi identitetni storitvi. V okoljih CI/CD je pogost primer povezava med platformo, kot je GitHub Actions, in oblačnim ponudnikom, kot so AWS, Azure ali Google Cloud. Namesto da bi imel razvojni sistem shranjen trajni dostopni ključ, lahko pridobi kratkoživi žeton (angl. short-lived token), ki velja samo omejen čas in samo za določeno dejanje. OIDC je eden izmed standardov, ki omogoča takšno izmenjavo identitete med različnimi sistemi [11], [27].

Tipičen proces poteka tako, da sistem CI/CD najprej pridobi OIDC žeton, ki vsebuje informacije o identiteti izvajalnega okolja. Oblačni ponudnik nato preveri, ali tej identiteti zaupa in ali izpolnjuje določene pogoje. Če je preverjanje uspešno, izda začasne poverilnice z omejenimi pravicami. Ta pristop zmanjšuje potrebo po dolgoročnih skrivnostih, vendar hkrati uvaja novo varnostno tveganje. Če je konfiguracija zaupanja napačna ali če napadalec pridobi nadzor nad zaupanja vrednim procesom, lahko pridobi legitimni dostop do drugih sistemov.

Napadi na identitetne sisteme se pogosto ne osredotočajo na samo tehnologijo OIDC, temveč na napačne konfiguracije in preširoko nastavljene pravice dostopa. Primer je lahko okolje CI/CD, ki ima dovoljenje za dostop do oblačnega računa brez dovolj strogih omejitev glede izvora zahtevka. Napadalec lahko v takem primeru poskuša zlorabiti razvojni proces, ustvariti veljaven zahtevek iz zaupanja vrednega okolja ali izkoristiti preširoko konfigurirano identitetno povezavo. Ker sistem vidi veljavno identiteto, lahko napad deluje kot legitimna aktivnost [11].

Pomemben koncept pri tem je identiteta delovne obremenitve (angl. Workload Identity). Gre za mehanizem, pri katerem aplikacijam in avtomatiziranim procesom dodelimo lastne identitete, podobno kot jih imajo uporabniki. Prednost je manjše število skrivnosti, ki jih je treba shranjevati in varovati, vendar mora biti upravljanje dovoljenj zelo natančno. Preširoka dovoljenja lahko povzročijo, da ogrožen proces dobi dostop do veliko več virov, kot bi jih potreboval. Zaradi tega identitetni sistemi postajajo nova osrednja točka zaupanja. Napadalcu ni več nujno treba ukrasti gesla ali neposredno napasti produkcijskega strežnika. Dovolj je, da zlorabi legitimno identiteto, ki ji sistem že zaupa.

Eden izmed pomembnih primerov je kampanja, povezana z ekosistemom TanStack leta 2026, kjer so napadalci zlorabili razvojni proces in prek njega objavili zlonamerne različice več npm paketov, med njimi tudi priljubljene pakete iz ekosistema TanStack. Napad je izkoristil povezavo med okoljem CI/CD, npm registrom in identitetnimi mehanizmi za avtomatizirano objavo paketov. Ker je proces objave uporabljal federacijo OIDC, je incident pokazal, da lahko ogrožen izvajalec CI/CD pridobi veljavno razvojno identiteto in jo uporabi za legitimno objavo paketov. Posledica ni bila samo objava zlonamernih različic posameznih paketov, temveč tveganje za vse

uporabnike, ki so zlonamerne različice prenesli prek običajnega postopka posodabljanja. Po incidentu so vzdrževalci omejili dovoljenja, izboljšali nadzor nad objavo paketov in utrdili konfiguracijo CI/CD. Primer kaže, da kratkoživi žetoni sami po sebi ne rešijo problema, če lahko ogrožen proces dostopa do identitete z dovoljenji za objavo [27], [28].

Podoben primer predstavlja kampanja Megalodon, kjer so napadalci zlorabili GitHub okolje in avtomatizirane CI/CD procese za krajo poverilnic ter nadaljnje širjenje napada. Ključna značilnost napada je bila, da ukradeni podatki niso predstavljali samo trenutnega dostopa do posameznega sistema, temveč so napadalcem omogočili pridobivanje novih identitet in dostopov. Na primer, ukradeni žetoni, skrivnosti in dostopne poverilnice iz razvojnih okolij so lahko omogočili dostop do dodatnih repozitorijev, oblčnih storitev ali drugih CI/CD okolij. S tem je ogrožen razvojni proces postal izvor novih zaupanja vrednih dostopov, ki jih je bilo mogoče uporabiti za nadaljevanje napada po dobavni verigi [29].

3.5. Zloraba kontejnerskih slik (angl. container images)

Kontejnerji (angl. containers) so danes eden izmed temeljev sodobnega razvoja in uvajanja programske opreme. Razvijalcem omogočajo, da aplikacijo skupaj z vsemi potrebnimi odvisnostmi zapakirajo v prenosljivo enoto, ki jo je mogoče enako izvajati v različnih okoljih. Osnovni gradnik takšnega okolja je kontejnerska slika (angl. container image), ki vsebuje aplikacijsko kodo, knjižnice, sistemske komponente in druge potrebne datoteke.

Podobno kot pri programskih odvisnostih velja, da organizacije pogosto ne ustvarijo vseh komponent same, temveč uporabljajo že pripravljene slike iz javnih registrov, kot je Docker Hub. S tem nastane nov odnos zaupanja: organizacija zaupa izvoru slike, njenim vzdrževalcem in vsem komponentam, ki so vključene v njeno vsebino. Če je ena izmed teh komponent ogrožena, lahko napadalec doseže vse uporabnike, ki jo vključijo v svoje razvojne ali produkcijske procese [30].

Napadalci lahko kontejnerske slike zlorabijo na več načinov. Eden izmed najpogostejših pristopov je objava zlonamernih slik v javnih registrih. Napadalec lahko ustvari sliko z legitimno vidnim imenom, vanjo vključi zlonamerno kodo in poskuša doseči, da jo razvijalci uporabijo kot osnovo za svoje aplikacije. Podoben problem predstavljajo kompromitirane osnovne slike (angl. base images). Pri gradnji kontejnerjev razvijalci pogosto uporabljajo obstoječe slike kot izhodišče, na primer operacijski sistem ali že pripravljeno razvojno okolje. Če vsebuje takšna osnovna slika zlonamerno kodo ali ranljive komponente, se težava prenese v vse nadaljnje slike, ki so zgrajene na njej [30], [31].

Dodatno tveganje predstavlja uporaba spremenljivih oznak (angl. mutable tags), kot je na primer oznaka latest. Če se vsebina slike za isto oznako spremeni, lahko razvojna ekipa ali sistem CI/CD nehote prenese drugačno različico, kot je bila prvotno preverjena. Zaradi tega sodobne prakse priporočajo uporabo preverljivih različic, podpisovanje artefaktov in preverjanje izvora komponent [30].

Kontejnerske slike so pogosto tesno povezane s CI/CD procesi. Tipičen razvojni tok vključuje samodejno gradnjo slike, varnostno preverjanje, objavo v register in kasnejšo namestitev v produkcijsko okolje. Če napadalec zlorabi del tega procesa, lahko vpliva na končno sliko, ki jo organizacija uporabi v produkciji. To pomeni, da napad ni nujno usmerjen neposredno v delujočo aplikacijo, temveč v proces, ki aplikacijo izdela in dostavi. Zaradi tega predstavljajo registri kontejnerskih slik in procesi njihove gradnje pomemben del programske dobavne verige. Ogrožena slika lahko omogoči izvajanje zlonamerne kode, krajo skrivnosti (angl. secrets) ali pridobivanje dostopa do notranjih sistemov [32], [33].

Eden izmed pogostih primerov so kampanje, kjer napadalci objavijo zlonamerne slike v javnih registrih, kot je Docker Hub. Raziskave podjetja Sysdig so pokazale številne primere slik, ki so vsebovale skrite rudarje kriptovalut, mehanizme za nepooblaščen dostop (angl. backdoors) ali drugo neželjeno funkcionalnost. Ker so slike pogosto delovale legitimno in so bile javno dostopne, so jih lahko uporabniki vključili v svoja okolja brez zavedanja o tveganju [32].

Podoben trend so pokazale novejšje kampanje, povezane s krajo poverilnic v okoljih CI/CD. Napadi na orodja, ki se uporabljajo pri gradnji in preverjanju kontejnerskih slik, so pokazali, da lahko ogrožena komponenta razvojnega procesa omogoči dostop do skrivnosti in drugih občutljivih podatkov. Primeri, povezani s Trivy in KICS, kažejo, da tudi varnostna orodja in infrastruktura za preverjanje lahko postanejo del napadalne površine [33], [34].

4 Večstopenjske napadalne verige

V prejšnjem poglavju smo predstavili najpogostejše napadalne tehnike, ki se uporabljajo pri napadih na programske dobavne verige. Sodobni napadi na programske dobavne verige pogosto niso izvedeni kot en sam korak, temveč kot zaporedje povezanih dejanj. Posamezna faza napadalcu omogoči pridobitev novih pravic ali dostopa, ki ga uporabi za naslednjo stopnjo napada. Primer takšnega vzorca predstavlja aktivnost, povezana s TeamPCP in kampanjo Shai-Hulud, kjer so različni napadalni valovi izkoristili povezave med razvojnimi okolji, paketnimi ekosistemi in avtomatiziranimi procesi objave. Namesto ločenih incidentov jih je zato smiselno obravnavati kot večstopenjsko verigo, v kateri zloraba ene komponente omogoči nadaljnje širjenje prek novih identitet, žetonov in distribucijskih kanalov. Tako nastane večstopenjska napadalna veriga, ki napadalcem omogoča postopno širjenje dostopa skozi razvojna, testna in produkcijska okolja.

Posebnost takšnih napadov je, da napadalci pogosto ne poskušajo neposredno zaobiti vseh varnostnih mehanizmov. Namesto tega izkoristijo obstoječe odnose zaupanja med različnimi deli razvojnega procesa. Razvojni račun zaupa repozitoriju, proces CI/CD zaupa izvornemu okolju, register paketov zaupa postopku objave, uporabniki pa zaupajo programski opreми, ki prihaja iz legitimnega vira. Če napadalec prevzame eno izmed teh povezav, lahko pridobljeni dostop uporabi za nadaljnje korake napada.

4.1. Tipični potek sodobnega napada

Sodobni napad na programsko dobavno verigo se praviloma začne z ogrožanjem zaupanja vredne komponente, računa ali razvojnega procesa. Začetni dostop napadalcu omogoči prehod v okolje CI/CD, kjer lahko pridobi skrivnosti, žetone za objavo, oblačne poverilnice ali druge identitete z dostopnimi pravicami.

Pridobljene identitete nato uporabi za dostop do dodatnih repozitorijev, registrov, oblačnih storitev in produkcijskih okolij oziroma za objavo zlonamernih različic programske opreme. Posamezne faze napada niso ločene, temveč tvorijo verigo, v kateri vsak uspešen korak omogoči nadaljnje širjenje vpliva napadalca.

4.2. Kaskadni učinki med organizacijami

Posebnost napadov na programske dobavne verige je, da posledice pogosto presegajo eno organizacijo. Medtem ko klasični kibernetiski napadi običajno prizadenejo posamezno tarčo, lahko ogrožanje zaupanja vrednega ponudnika programske opreme vpliva na več sto ali celo več tisoč organizacij.

Če napadalec zlorabi priljubljen odprtokodni paket, razvojno orodje ali ponudnika programske opreme, lahko zlonamerna koda doseže vse organizacije, ki uporabljajo takšno rešitev. Vsaka od teh organizacij lahko nato nevede postane nova vstopna točka za nadaljnje napade, kar povzroči kaskadni učinek po celotni dobavni verigi.

Takšen način širjenja je bil značilen za več odmevnih incidentov, kot so event-stream, ua-parser-js, SolarWinds, XZ Utils in novejšje kampanje, usmerjene v npm ter GitHub ekosistem. Čeprav so se napadi med seboj razlikovali po začetnem vektorju, je bila njihova skupna značilnost zloraba zaupanja v programsko dobavno verigo in posledično širjenje vpliva daleč prek prvotne tarče [21], [22], [35], [36].

Napadi na programske dobavne verige zato ne predstavljajo več zgolj tehničnega problema posamezne organizacije, temveč sistemsko tveganje za celoten programski ekosistem. Vse večja povezanost razvojnih orodij, odprtokodnih projektov, oblačnih storitev in avtomatiziranih razvojnih procesov pomeni, da lahko zloraba ene same zaupanja vredne komponente vpliva na veliko število medsebojno povezanih organizacij.

5 Vpliv umetne inteligence na varnost dobavnih verig

Generativna umetna inteligenca je v zadnjih letih postala pomemben del razvoja programske opreme. Orodja, zasnovana na velikih jezikovnih modelih (angl. Large Language Models, LLM), razvijalcem pomagajo pri dopolnjevanju kode, ustvarjanju funkcij, pisanju testov, razlagi obstoječih rešitev, pripravi dokumentacije in avtomatizaciji ponavljajočih se opravil. Z razvojem sistemov z večjo stopnjo avtonomije se njihova uporaba širi tudi na bolj kompleksne naloge: sodobni razvojni agenti lahko samostojno pregledujejo repozitorije, spreminjajo več datotek, nameščajo odvisnosti, zaganjajo ukaze in pripravljajo zahtevke za vključitev sprememb (angl. pull requests).

Takšna orodja spreminjajo programsko dobavno verigo predvsem zato, ker povečujejo hitrost in obseg nastajanja programske kode. Več sprememb je mogoče pripraviti v krajšem času, vendar se posledično poveča količina kode, ki jo morajo preveriti razvijalci, varnostni strokovnjaki in avtomatizirana orodja. Umetna inteligenca (v nadaljevanju UI) zato ni samostojen vzrok napadov na dobavne verige, temveč pospeševalec že obstoječih razvojnih procesov, napak in razmerij zaupanja. Če so postopki pregleda, upravljanja odvisnosti ali omejevanja privilegijev šibki, lahko umetna inteligenca poveča hitrost, s katero se te pomanjkljivosti odrazijo v končnih programskih artefaktih.

Uporaba umetne inteligence ima na varnost dvojni vpliv. Na eni strani lahko modeli pomagajo pri iskanju napak, generiranju testov, analizi kode in razlagi ranljivosti. Na drugi strani lahko ustvarijo funkcionalno delujočo, vendar varnostno neustrezno kodo, predlagajo neobstoječe odvisnosti ali izvedejo neželena dejanja, kadar so povezani z razvojnimi orodji in privilegiranimi računi. Ključno vprašanje zato ni, ali naj se UI uporablja, temveč kako jo vključiti v razvojni proces, ne da bi pri tem zmanjšali preverljivost, sledljivost in človeški nadzor. NIST pri upravljanju tveganj UI poudarja potrebo po merjenju tveganj, dokumentiranju omejitev, neodvisnem preverjanju rezultatov in jasno določenih odgovornostih skozi celoten življenjski cikel sistema [37], [38].

5.1. Umetna inteligenca kot pospeševalec razvoja

Najbolj razširjena uporaba UI v razvoju programske opreme je pomoč pri pisanju kode. Razvojno orodje na podlagi komentarja, opisa funkcionalnosti ali obstoječega konteksta predlaga posamezne vrstice, celotne funkcije ali večje programske komponente. Eksperimentalne raziskave ponudnikov takšnih orodij so pokazale, da lahko razvijalci določene naloge dokončajo hitreje, pri čemer je treba rezultate razlagati v okviru konkretnih eksperimentov in ob upoštevanju dejstva, da gre pogosto za raziskave ponudnikov samih [39]. Ne glede na dejanski obseg vpliva je jasno, da umetna inteligenca zmanjšuje čas, potreben za izdelavo začetne različice rešitve, in s tem povečuje število sprememb, ki jih je mogoče vključiti v razvojni tok.

UI se uporablja tudi za ustvarjanje testov enot in integracijskih testov. Model lahko iz podane funkcije pripravi osnovne testne primere, predlaga mejne vrednosti ali dopolni manjkajočo testno dokumentacijo. NIST je leta 2025 začel pilotno vrednotenje zmožnosti UI za izdelavo testov posameznih enot (angl. unit tests), kar kaže, da se uporaba modelov širi iz neposrednega pisanja produkcijske kode tudi v postopke zagotavljanja kakovosti [40]. Takšna avtomatizacija je koristna predvsem kot začetna pomoč, vendar kakovost testov ostaja odvisna od pravilnosti specifikacij, izbranih testnih primerov in neodvisnega preverjanja, ali testi dejansko zaznajo pomembne napake.

Pomembno področje uporabe so pregledi kode. Modeli lahko povzamejo spremembe, opozorijo na sumljive vzorce, predlagajo izboljšave ali razvijalcu razložijo posledice določene implementacije. S tem lahko zmanjšajo kognitivno obremenitev pri pregledovanju obsežnih zahtevkov za vključitev sprememb (angl. pull requests). Toda pregled kode s podporo UI ne sme postati nadomestilo za tehnični in varnostni pregled, saj model pogosto ocenjuje verjetnost in podobnost z naučenimi vzorci, ne pa dejanske varnostne lastnosti sistema. Njegov izhod je zato priporočilo, ki ga je treba preveriti z analiznimi orodji, testi in strokovnim pregledom.

Novejša stopnja razvoja so agentni sistemi, ki niso omejeni na generiranje besedila ali kode. Tak agent lahko deluje z dostopom do repozitorija, terminala, registra paketov, sistema za sledenje nalogam in okolja CI/CD. Na podlagi enega navodila lahko analizira projekt, spremeni kodo, namesti novo odvisnost, zažene teste in pripravi

spremembo za objavo. S tem se meja med svetovalnim orodjem in aktivnim udeležencem programske dobavne verige zabriše. Agent postaja nov izvajalec z lastno identiteto in dovoljenji oziroma posrednik, katerega dovoljenja, dejanja in dostop do podatkov je treba obravnavati podobno kot dovoljenja razvijalcev in avtomatiziranih CI/CD procesov.

Umetna inteligenca lahko varnost dobavne verige tudi neposredno izboljšuje. Uporablja se lahko za razvrščanje opozoril, analizo sprememb v odvisnostih, pomoč pri razumevanju ranljivosti in prepoznavanje nepričakovanih vzorcev v dnevniških zapisih. Pri tem je najbolj učinkovit v vlogi dodatnega analiznega sloja, ki pomaga prednostno razporediti delo in opozori na odstopanja. Vendar mora biti odločanje o sprejemu odvisnosti, objavi artefakta ali dostopu do produkcije še vedno povezano z določljivimi pravili, preverljivimi dokazi in neodvisnimi varnostnimi kontrolami.

Glavna posledica UI-podprtega razvoja je torej povečanje hitrosti. Organizacija lahko hitreje ustvarja kodo, teste in konfiguracije, vendar mora enako hitro razširiti tudi postopke preverjanja. Če se produktivnost razvijalcev poveča, medtem ko zmogljivosti pregledovanja, testiranja in spremljanja ostanejo nespremenjene, nastane nesorazmerje: v dobavno verigo vstopa več sprememb, kot jih je mogoče kakovostno oceniti. V takem okolju lahko tudi majhen delež napačnih ali nevarnih predlogov povzroči pomembno povečanje skupnega tveganja.

5.2. Nova tveganja zaradi umetne inteligence

Prvo pomembno tveganje je sprejemanje nepreverjene kode. Koda, ki jo ustvari UI, je lahko sintaktično pravilna, funkcionalna in na prvi pogled smiselna, vendar vsebuje ranljivosti, napačne predpostavke ali zastarele varnostne vzorce. Raziskava Pearcea in sodelavcev je pri varnostno občutljivih scenarijih pokazala, da je pomemben delež analiziranih predlogov orodja GitHub Copilot vseboval ranljivosti [41]. Kasnejše ponovitvene raziskave so sicer ugotovile izboljšanje rezultatov, vendar so še vedno zaznale varnostno neustrezne predloge [42]. Takšne ugotovitve ne pomenijo, da je vsa z UI ustvarjena koda nevarna, temveč da samo dejstvo, da je kodo ustvaril model, ne predstavlja varnostnega zagotovila.

Tveganje se poveča, kadar razvijalec uporablja tehnologijo, ki je ne pozna dovolj dobro. Model lahko pripravi rešitev, ki deluje v običajnem primeru, razvijalec pa zaradi pomanjkanja izkušenj ne prepozna neustreznega preverjanja vhodov, šibke avtentikacije, nevarnega upravljanja skrivnosti ali napačne uporabe kriptografskih funkcij. Prepričljivo oblikovan odgovor lahko ustvari vtis pravilnosti. Zato mora za kodo, ki jo ustvari UI, veljati najmanj enaka raven pregleda kot za kodo zunanjega sodelavca ali novo odvisnost neznanega izvora.

Drugo tveganje predstavljajo izmišljene programske odvisnosti (angl. package hallucinations). Model lahko pri generiranju kode predlaga paket, ki v registru npm, PyPI ali drugem ekosistemu sploh ne obstaja. Napadalec lahko takšno predvidljivo ime registrira in vanj vstavi zlonamerno kodo. Če razvijalec slepo sledi predlogu modela, namesti napadalčev paket. Ta tehnika se pogosto imenuje slopsquatting in predstavlja posebno različico zastrupljanja paketov, pri kateri napadalec izkorišča ponavljajoče se napake generativnih modelov.

Spracklen in sodelavci so v obsežni raziskavi analizirali 576.000 vzorcev kode, ustvarjenih s šestnajstimi modeli, in ugotovili, da so halucinacije paketov sistemski pojav, ki se razlikuje glede na model, programski jezik in način postavitve vprašanja. Raziskava je pri komercialnih modelih izmerila najmanj 5,2 %, pri odprtokodnih modelih pa 21,7 % povprečni delež haluciniranih paketov [43]. Posamezna napačna navedba še ne pomeni uspešnega napada, vendar ponavljajoča se imena ustvarjajo priložnosti za registracijo zlonamernih paketov. Obramba zahteva preverjanje obstoja, avtorja, zgodovine in izvora vsake nove odvisnosti ter uporabo dovoljenih seznamov in zaklenjenih različic.

Tretje tveganje je vbrizgavanje navodil (angl. prompt injection) pri razvojnih agentih. Klasičen pomočnik prejema navodila neposredno od uporabnika, agent pa pogosto bere še izvirno kodo, dokumentacijo, težave v sistemu za sledenje, spletne strani ali vsebino zunanjih paketov. Napadalec lahko v enega od teh virov vstavi skrita ali zavajajoča navodila, ki poskušajo preusmeriti vedenje agenta. Pri posrednem vbrizgavanju agent obravnava napadalčevo vsebino kot navodilo, čeprav je bila prvotno le podatek za analizo. Posledice takšnega napada so bistveno resnejše, kadar ima agent dostop do orodij. Namesto napačnega besedilnega odgovora lahko spremeni

datoteko, zažene ukaz, prenese paket, pošlje vsebino zunanji storitvi ali uporabi poverilnice iz razvojnega okolja. OWASP med ključnimi tveganji za sisteme z umetno inteligenco, ki delujejo avtonomno, izpostavlja neposredno in posredno vbrizgavanje navodil, zlorabo orodij, stopnjevanje privilegijev ter odtekanje podatkov [44]. S tem razvojni agent postane nova povezava v verigi zaupanja: zaupa vhodnim podatkom, uporabnik zaupa njegovim odločitvam, infrastruktura pa zaupa identiteti in dovoljenjem, s katerimi agent deluje.

Četrto tveganje predstavlja zmanjšanje človeškega nadzora. Večja količina samodejno ustvarjene kode lahko privede do površinskih pregledov, zlasti kadar so spremembe obsežne in navidezno rutinske. Razvijalec lahko potrdi rezultat, ker testi uspešno prestanejo izvajanje, ne da bi razumel vse ustvarjene odvisnosti, konfiguracije ali stranske učinke. Problem ni le v kakovosti posamezne vrstice kode, temveč v izgubi sledljivosti: težje je ugotoviti, zakaj je bila določena arhitekturna odločitev sprejeta, na podlagi katerih podatkov oziroma vzorcev je model pripravil predlog in kdo je dejansko preveril njene posledice.

Orodja, podprta z UI lahko povzročijo tudi nenamerno razkritje podatkov. Razvijalci lahko v poziv vključijo izvorno kodo, konfiguracije, osebne podatke, notranjo dokumentacijo ali skrivnosti. Pri agentnih sistemih je tveganje še večje, saj lahko orodje samodejno bere datoteke in kontekst repozitorija. Organizacija mora zato določiti, kateri podatki se lahko posredujejo modelu, kako dolgo se hranijo, ali se uporabljajo za nadaljnje učenje in katere zunanje storitve jih obdelujejo. Profil NIST za generativno umetno inteligenco obravnava upravljanje podatkov, zasebnost, varnost, nadzor tretjih ponudnikov in neodvisno vrednotenje kot medsebojno povezane elemente upravljanja tveganj [37].

Nazadnje lahko generativna UI pospeši tudi delo napadalcev. Uporablja se lahko za prilagajanje phishing sporočil, analizo izvorne kode, pomoč pri ustvarjanju zlonamernih paketov ali njihovih različic ali avtomatizacijo iskanja napačnih konfiguracij. Vendar osnovni napadalni mehanizmi večinoma ostajajo enaki: kraja identitet, zloraba zaupanja, zloraba odvisnosti in izkoriščanje preširokih privilegijev. UI zmanjšuje stroške in čas izvedbe, ne odpravlja pa potrebe po začetni vstopni točki. Zaradi tega morajo obrambni ukrepi ostati usmerjeni v preverjanje izvora, omejevanje privilegijev, zaščito razvojnih identitet, nadzor sprememb in zaznavanje nepričakovanega vedenja.

6 Obrambni pristopi

Ker napadi na programske dobavne verige izkoriščajo odnose zaupanja med komponentami, identitetami in razvojnimi procesi, zaščita ne more temeljiti na enem samem ukrepu. Organizacije morajo poznati uporabljene komponente, preverjati njihov izvor in celovitost ter omejevati dostop ljudi in avtomatiziranih procesov.

Obrambne pristope lahko razdelimo na tehnične in organizacijske ukrepe. Tehnični ukrepi vključujejo izdelavo SBOM, upravljanje odvisnosti, preverjanje porekla artefaktov, zaščito okolij CI/CD, upravljanje identitet in varnost kontejnerskih slik. Organizacijski ukrepi pa zajemajo upravljanje privilegijev, nadzor dobaviteljev, vključevanje varnostnih pregledov v razvojni proces, stalno spremljanje tveganj in pripravo odziva na incidente. Ker se učinkovitost posameznih ukrepov razlikuje glede na okolje in razpoložljive vire, je pomembno, da organizacije ukrepe uvajajo postopoma in glede na njihovo tveganje ter kritičnost.

6.1 Tehnični ukrepi

Tehnična zaščita programske dobavne verige temelji na povečanju preglednosti in zaupanja v procesu razvoja programske opreme. Sodobne aplikacije niso več sestavljene samo iz kode, ki jo napiše organizacija sama, temveč vključujejo številne zunanje knjižnice, pakete, kontejnerske slike, razvojna orodja in avtomatizirane procese. Zaradi tega mora zaščita zajemati celoten življenjski cikel programske opreme: od izbire komponent, razvoja in gradnje do dostave v produkcijsko okolje.

Glavni cilj tehničnih ukrepov je odgovoriti na tri vprašanja: katere komponente uporabljamo, od kod prihajajo in ali lahko preverimo, da med procesom razvoja niso bile spremenjene. V nadaljevanju so predstavljeni najpomembnejši pristopi, ki se uporabljajo za povečanje zaupanja v programsko dobavno verigo.

6.1.1. SBOM, upravljanje odvisnosti in varnost paketnih ekosistemov

Eden izmed osnovnih pristopov je izdelava kosovnice programske opreme (Software Bill of Materials). SBOM je torej popis komponent, ki sestavljajo določeno različico programske opreme. Običajno nastane kot del procesa gradnje programske opreme in vsebuje informacije o uporabljenih knjižnicah, paketih, njihovih različicah ter drugih komponentah. Organizacije ga lahko hranijo skupaj s programskim artefaktom ali ga vključijo v svoje razvojne in varnostne procese. SBOM omogoča pregled nad programsko dobavno verigo določene aplikacije: katere komponente so vključene, katere različice se uporabljajo in od kod prihajajo [45]. SBOM se najpogosteje izmenjuje v standardiziranih formatih, kot sta SPDX (Software Package Data Exchange) in CycloneDX, ki omogočata strojno obdelavo podatkov o komponentah ter njihovo vključevanje v avtomatizirane varnostne procese.

Pomen SBOM se pokaže predvsem pri odzivanju na varnostne incidente. Če je določena knjižnica ogrožena ali je v njej odkrita ranljivost, lahko organizacija s pomočjo SBOM preveri, ali je komponenta vključena v njene aplikacije. To vključuje tudi posredne odvisnosti (angl. transitive dependencies), ki jih organizacija morda ni neposredno izbrala, vendar so bile vključene prek drugih paketov. Brez takšnega pregleda je lahko nevarna komponenta prisotna v okolju dalj časa, saj organizacija nima celotnega pregleda nad uporabljenimi odvisnostmi.

Upravljanje odvisnosti (angl. dependency management) zato ne pomeni samo posodabljanja paketov, temveč vključuje tudi spremljanje njihovega izvora, preverjanje sprememb in nadzor nad tem, katere komponente so dovoljene v razvojnem procesu. Poseben izziv predstavljajo že prej omenjene prehodne odvisnosti (angl. transitive dependencies), kjer aplikacija neposredno ne uporablja določene knjižnice, vendar jo vključuje druga uporabljena komponenta [45], [46].

Pomembno vlogo pri tem imajo upravljalniki paketov (angl. package managers), saj predstavljajo eno izmed prvih mest, kjer se zunanja koda prenese in vključi v razvojno okolje. Ker lahko nekateri paketi med namestitvijo izvajajo dodatno kodo, na primer namestitvene skripte (angl. install scripts), ki se samodejno zaženejo ob namestitvi ali posodobitvi paketa, predstavlja sam proces namestitve pomembno varnostno točko.

Sodobni paketni ekosistemi zato uvajajo dodatne zaščitne mehanizme. Upravljalnik paketov npm omogoča preverjanje integritete paketov prek zaklenjenih datotek (angl. lock files) in kontrolnih vsot. Takšna datoteka zabeleži točne različice vseh uporabljenih paketov v projektu in njihovih odvisnosti, vključno s prehodnimi odvisnostmi (angl. transitive dependencies). Pri naslednji namestitvi se uporabijo enake različice, namesto da bi upravljalnik ponovno izbral najnovejše razpoložljive pakete. Poleg tega vsebuje integritetne vrednosti (angl. integrity hashes) prenesenih paketov, ki jih upravljalnik paketov uporabi pri namestitvi za preverjanje, ali prenesena vsebina ustreza pričakovani vsebini. Če se vsebina paketa razlikuje od pričakovane vrednosti, lahko upravljalnik zazna nepričakovano spremembo in namestitev prepreči. Upravljalnik npm omogoča tudi preverjanje znanih ranljivosti z ukazom npm audit ter možnost onemogočanja samodejnega izvajanja namestitvenih skript. Takšne omejitve zmanjšujejo tveganje, da bi ogrožen paket med namestitvijo izvedel zlonamerno kodo [47].

Podobne pristope uvajajo tudi drugi upravljalniki paketov. Upravljalnik paketov pnpm uporablja zaklenjeno datoteko (angl. lock file) pnpm-lock.yaml, ki podobno kot pri npm zabeleži točne različice neposrednih in prehodnih odvisnosti ter zagotavlja ponovljive namestitve (angl. reproducible installs). Poleg tega pnpm omogoča dodatne zaščitne mehanizme, kot je omejevanje izvajanja namestitvenih skript in zmanjševanje tveganja zaradi na novo objavljenih zlonamernih paketov z mehanizmom časovnega zamika pri novih izdajah (angl. minimum release age) [48]. Namen teh mehanizmov je ustvariti časovno okno, v katerem lahko skupnost ali varnostni sistemi zaznajo sumljive spremembe, preden se paket samodejno razširi v večje število okolij.

Tudi Yarn vključuje več mehanizmov za povečanje varnosti programske dobavne verige. Podobno kot npm in pnpm uporablja zaklenjeno datoteko (angl. lock file), yarn.lock, ki določa natančne različice uporabljenih paketov in njihovih odvisnosti ter zagotavlja ponovljive namestitve (angl. reproducible installs). Yarn preverja integriteto

preneseni paketov in s tem zmanjšuje tveganje nepričakovanih sprememb vsebine med namestitvijo. Novejše različice Yarn uvajajo dodatne zaščitne mehanizme, kot je omejevanje izvajanja namestitvenih skript ter strožji nadzor nad tem, kateri paketi lahko izvajajo dodatno kodo med namestitvijo.

V okolju Bun je varnost namestitvenega procesa obravnavana predvsem skozi nadzor izvajanja namestitvenih skript. Upravljalnik paketov bun paketov, ki med namestitvijo izvajajo dodatno kodo, ne obravnava samodejno kot zaupanja vrednih, temveč zahteva njihovo izrecno dovoljenje prek mehanizma zaupanja vrednih odvisnosti (angl. trusted dependencies). Tak pristop zmanjšuje tveganje, da bi ogrožen paket med namestitvijo samodejno izvedel zlonamerno kodo v razvijalčevem okolju ali procesu CI/CD [49].

Pri uvajanju teh zaščitnih mehanizmov je treba upoštevati tudi vpliv na razvojni proces. Varnostni ukrepi pogosto vključujejo kompromis med zmanjšanjem tveganja programske dobavne verige in ohranjanjem hitrosti ter prilagodljivosti razvoja. Omejevanje namestitvenih skript lahko na primer vpliva na pakete, ki jih legitimno uporabljajo za prevajanje izvirne kode ali pripravo okolja, zato je primernejši nadzorovan pristop z izrecnim dovoljevanjem preverjenih paketov. Podobno lahko uporaba nespremenljivih različic paketov ali slik zmanjša tveganje nepričakovanih sprememb, vendar zahteva dodatne postopke za pravočasno posodabljanje zaradi varnostnih popravkov. Namen zaščitnih ukrepov zato ni popolna odstranitev avtomatizacije ali prilagodljivosti, temveč vzpostavitev ravnotežja med zmanjšanjem napadalne površine in učinkovitim razvojnim procesom.

6.1.2. Preverjanje izvora in pristnosti artefaktov

Pri sodobnem razvoju programske opreme ni dovolj preveriti samo končnega artefakta, temveč tudi način, kako je ta nastal. Programska oprema se danes pogosto gradi avtomatizirano v okoljih CI/CD, kjer izvirna koda, odvisnosti, orodja za gradnjo programske opreme in dostopna dovoljenja vplivajo na končni rezultat. Če napadalec pridobi dostop do enega izmed teh elementov, lahko ustvari legitimno videti artefakt, ki vsebuje zlonamerne spremembe.

Zato organizacije uvajajo preverjanje izvora programske opreme (angl. software provenance). Gre za informacije o poreklu programske opreme, ki opisujejo, kako je bil določen programski artefakt ustvarjen. Ti podatki lahko vključujejo povezavo do izvirne kode, uporabljeno različico repozitorija, uporabljeno okolje za gradnjo, uporabljena orodja ter informacije o tem, kateri proces CI/CD je ustvaril artefakt.

Primer takšnega artefakta je lahko npm paket, Docker slika ali prevedena aplikacija. Če organizacija prejme novo različico paketa, lahko s pomočjo provenance preveri, ali je bila ustvarjena iz pričakovane izvirne kode in prek zaupanja vrednega procesa gradnje. Brez takšnih informacij je težko razlikovati med legitimno novo različico in različico, ki jo je ustvaril napadalec.

Tak napad se lahko zgodi na primer tako, da napadalec pridobi dostop do razvojnega okolja ali sistema CI/CD in spremeni proces gradnje. Namesto da bi neposredno spremenil izvirno kodo, lahko spremeni avtomatiziran proces tako, da pri gradnji vstavi dodatno kodo. Rezultat je artefakt, ki je videti kot uradna izdaja, vendar vsebuje zlonamerno funkcionalnost. Posledice so lahko kraja skrivnosti, skrita zlonamerna funkcionalnost (angl. backdoor) ali ogrožanje vseh uporabnikov, ki prenesejo takšno različico.

Pomemben okvir na tem področju predstavlja SLSA. SLSA je nabor priporočil in zahtev za izboljšanje varnosti procesa izdelave programske opreme. Določa različne ravni zaščite, od osnovnega beleženja izvora artefaktov do naprednejših zahtev, kjer mora biti proces gradnje izoliran, preverljiv in zaščiten pred nepooblaščenimi spremembami. Namen SLSA je zmanjšati možnost, da bi napadalec neopazno vplival na proces med izvirno kodo in končnim artefaktom [3]. Pri preverjanju izvora in zaporedja korakov izdelave artefaktov se pogosto uporablja tudi model in-toto, ki omogoča beleženje in preverjanje posameznih dejanj v verigi nastanka programske opreme. SLSA se na te koncepte navezuje pri zagotavljanju preverljivega izvora artefaktov.

Dodatno zaščito omogoča podpisovanje artefaktov. Pri tem se programski artefakt, kot so paket npm, Docker slika, izvršna datoteka ali druga izdaja programske opreme, opremi s kriptografskim podpisom. Uporabnik ali

avtomatiziran sistem lahko nato preveri, ali je artefakt ustvaril pričakovani izdajatelj in ali se njegova vsebina po podpisu ni spremenila.

Tradicionalno je bilo podpisovanje pogosto odvisno od dolgoročnih zasebnih ključev, ki jih je bilo treba varno hraniti. Rešitve, kot je Sigstore, uvajajo drugačen pristop, kjer se uporabljajo kratkožive identitete (angl. short-lived identities) in javni zapisi podpisov. Namesto upravljanja dolgoročnih ključev lahko organizacija preveri povezavo med identiteto izvajalca gradnje v procesu CI/CD, procesom gradnje in podpisanim artefaktom [50].

6.1.3. Varovanje okolij CI/CD, skrivnosti in identitet

Ker sistemi CI/CD predstavljajo povezavo med izvorno kodo, procesom gradnje programske opreme in produkcijskim okoljem, so ena izmed najpomembnejših točk zaščite v sodobni programski dobavni verigi. Za razliko od klasičnih razvojnih okolij imajo sistemi CI/CD pogosto dostop do različnih zunanjih storitev, registrov paketov, oblčnih okolij in produkcijskih sistemov. Če je CI/CD proces ogrožen, lahko napadalcu omogoči veliko več kot samo spremembo kode. Omogoči lahko krajo skrivnosti, dostop do infrastrukture ali vstavljanje zlonamernih sprememb v končne programske artefakte [51].

Napadi na okolja CI/CD pogosto ne izkoriščajo neposredno ranljivosti v aplikacijski kodi, temveč zlorabijo zaupanje, ki ga ima razvojni proces do različnih sistemov. Primer je lahko ogrožen delovni tok (angl. workflow) v GitHub Actions, GitLab CI ali podobnem sistemu, kjer napadalec spremeni ukaze, ki se izvajajo med gradnjo, ali zlorabi preširoka dovoljenja avtomatiziranega procesa. Ker se ti procesi izvajajo samodejno in pogosto z višjimi privilegiji, lahko ena sprememba vpliva na vse nadaljnje gradnje in objave programske opreme [51], [52].

Osnovni obrambni pristop je načelo najmanjših privilegijev (angl. least privilege). Vsak proces CI/CD, uporabnik ali avtomatizirana naloga naj ima samo tiste pravice, ki jih nujno potrebuje za svoje delovanje. Na primer, proces, ki samo gradi aplikacijo, ne bi smel imeti dovoljenja za spreminjanje izvorne kode ali dostopa do produkcijskih skrivnosti. Prav tako je pomembno ločevanje razvojnih, testnih in produkcijskih okolij, saj zloraba enega dela sistema ne sme samodejno omogočiti dostopa do vseh drugih [51], [53].

Pomemben del zaščite predstavlja upravljanje skrivnosti (angl. secrets management). Proces CI/CD pogosto potrebujejo dostopne ključe, žetone in poverilnice za objavo paketov, dostop do oblakov ali povezavo z drugimi sistemi. Te informacije ne smejo biti shranjene v izvorni kodi ali neposredno v konfiguracijskih datotekah. Namesto tega se uporabljajo namenski sistemi za upravljanje skrivnosti, omejevanje dostopa in redna rotacija poverilnic. Posebej pomembno je tudi zmanjševanje življenjske dobe dostopnih podatkov, saj ukradena kratkotrajna poverilnica predstavlja manjše tveganje kot trajni dostopni ključ [51], [54].

Zaščititi je treba tudi samo konfiguracijo procesov CI/CD. Datoteke, ki definirajo delovne tokove, na primer GitHub Actions datoteke YAML, niso samo konfiguracija, temveč dejansko predstavljajo del razvojne infrastrukture. Sprememba ene vrstice lahko spremeni način gradnje, objave ali dostopa do zunanjih sistemov. Zato je pomembno, da so spremembe teh datotek predmet pregleda kode, omejenih dovoljenj in dodatnega nadzora. Pri zunanjih akcijah in razširitvah je priporočljivo preveriti njihov izvor ter uporabljati nespremenljive različice namesto dinamičnih oznak, ki lahko kasneje kažejo na drugo vsebino [51], [52].

Sodobni sistemi CI/CD zato postajajo podobni produkcijskim okoljem in jih je treba tudi varnostno obravnavati na enak način. Cilj ni popolnoma odstraniti avtomatizacije, temveč zagotoviti, da avtomatizirani procesi delujejo z omejenim zaupanjem in da uspešen napad na posamezno komponento ne omogoči celotnega prevzema programske dobavne verige [51], [53].

Posebno pozornost zahteva tudi zaščita izvajalnega okolja procesov CI/CD. Avtomatizirani izvajalci (angl. runners), ki izvajajo ukaze za gradnjo, testiranje in objavo programske opreme, pogosto delujejo z dostopom do izvorne kode, registrov paketov ali oblčnih storitev. Če napadalec pridobi nadzor nad takšnim okoljem, lahko zlorabi obstoječa dovoljenja za izvajanje ukazov, pridobivanje skrivnosti ali spreminjanje končnih artefaktov. Zato se sodobni pristopi osredotočajo tudi na izolacijo okolij za gradnjo, uporabo kratkotrajnih izvajalnih okolij, nadzor nad uporabljenimi orodji ter redno preverjanje konfiguracije avtomatiziranih procesov [51], [53].

6.1.4. Varnost kontejnerskih slik

Kontejnerske slike (angl. container images) predstavljajo pomemben del sodobne programske dobavne verige, saj pogosto vsebujejo celotno okolje, potrebno za izvajanje aplikacije. Poleg aplikacijske kode lahko vključujejo sistemske knjižnice, zunanje pakete, konfiguracijo in druga orodja. Zaradi tega ogrožena kontejnerska slika ne vpliva samo na eno komponento, temveč lahko vnese zlonamerno kodo v vse sisteme, ki jo uporabijo.

Zaščita kontejnerskih slik zato vključuje preverjanje uporabljenih komponent, skeniranje znanih ranljivosti, uporabo zaupanja vrednih osnovnih slik (angl. base images) ter preverjanje izvora in integritete slike pred uporabo. Pomembna praksa je tudi uporaba minimalnih osnovnih slik, ki vsebujejo samo nujne komponente, saj manjša količina programske opreme pomeni manjšo potencialno napadalno površino [52].

Za pregled kontejnerskih slik se uporabljajo specializirana orodja za varnostno analizo, ki preverjajo znane ranljivosti v sistemskih paketih, knjižnicah in drugih vključenih komponentah. Orodja, kot so Trivy, Gype ali podobne rešitve, se pogosto vključujejo neposredno v procese CI/CD, kjer preverijo sliko še pred objavo v register ali uporabo v produkcijskem okolju. Takšen pristop omogoča zgodnje odkrivanje težav in preprečuje, da bi ranljive ali sumljive slike postale del nadaljnje distribucije.

Poseben problem predstavljajo spremenljive oznake (angl. mutable tags). Če se pri namestitvi uporablja na primer oznaka latest, lahko ista oznaka v registru kasneje kaže na drugo vsebino kot ob prvotni uporabi. Posledično lahko razvojna ali produkcijska okolja nehote prenesejo novo, nepreverjeno različico slike. Zato se v varnostno občutljivih okoljih uporabljajo nespremenljive reference, na primer kriptografski povzetki (angl. digests), ki enolično določajo točno določeno različico slike [52].

Dodatno zaščito omogoča podpisovanje kontejnerskih slik, kjer se pred uporabo preveri, ali je sliko ustvaril zaupanja vreden izdajatelj in ali vsebina po objavi ni bila spremenjena. Rešitve, kot je Sigstore s svojim orodjem Cosign, omogočajo kriptografsko podpisovanje slik in preverjanje njihove pristnosti z zmanjšano potrebo po klasičnem dolgoročnem upravljanju ključev. S tem se zmanjša tveganje, da bi napadalec zamenjal legitimno sliko v registru ali med distribucijskim procesom.

Cilj teh ukrepov je zagotoviti, da kontejnerska slika, ki pride v produkcijsko okolje, ustreza preverjeni različici, vsebuje samo pričakovane komponente in je bila ustvarjena skozi nadzorovan ter preverljiv proces.

6.2. Organizacijski ukrepi

Tehnični mehanizmi, predstavljeni v prejšnjem poglavju, predstavljajo pomemben del zaščite programske dobavne verige, vendar sami po sebi ne zadostujejo. Dobavna veriga vključuje tudi ljudi, razvojne procese, zunanje ponudnike in organizacijske odločitve, zato mora varnost temeljiti na kombinaciji tehničnih kontrol in ustrezno določenih postopkov. Cilj organizacijskih ukrepov je zmanjšati verjetnost napak, omejiti posledice zlorabe ter zagotoviti učinkovit odziv, ko do incidenta pride [55], [56].

Eden izmed osnovnih principov je načelo najmanjših privilegijev (angl. least privilege), ki določa, da uporabniki, razvijalci in avtomatizirani procesi ne smejo imeti več dostopa, kot ga nujno potrebujejo za svoje delo. Kot je bilo predstavljeno v prejšnjih poglavjih, lahko ogrožen razvijalski račun ali proces CI/CD predstavlja pomembno vstopno točko v programsko dobavno verigo. Zato morajo organizacije jasno določiti dostopne pravice, ločiti odgovornosti med različnimi vlogami ter redno preverjati, ali so dodeljena dovoljenja še vedno potrebna [55], [57].

Pomemben del varnosti predstavlja tudi vključevanje varnostnih pregledov v razvojni proces. Varnost ne sme biti obravnavana kot zadnji korak pred objavo programske opreme, temveč kot del celotnega življenjskega cikla razvoja. To vključuje določene postopke pregleda sprememb, odgovornosti razvijalcev, potrjevanje kritičnih sprememb ter opredeljene kriterije, kdaj je programsko opremo mogoče objaviti ali namestiti v produkcijo. Tehnična preverjanja, kot so skeniranje odvisnosti in analiza artefaktov, predstavljena v prejšnjem poglavju, so učinkovitejša, kadar so vključena v jasno določen razvojni proces [55], [57].

Poseben izziv predstavlja upravljanje tveganj tretjih ponudnikov. Organizacije danes pogosto uporabljajo zunanje knjižnice, storitve v oblaku in programske rešitve drugih ponudnikov, zato varnost njihove dobavne verige ni odvisna samo od lastnih ukrepov. Pomembno je, da organizacija pred uporabo določenega ponudnika oceni njegova varnostna tveganja, določi pričakovane varnostne zahteve ter opredeli postopke sodelovanja v primeru incidenta. Tehnični pristopi, kot je pregled sestave programske opreme prek SBOM, lahko pri tem pomagajo, vendar morajo biti podprti z ustreznimi procesi in odgovornostmi [56], [58].

Spremljanje programske dobavne verige mora biti neprekinjen proces in ne zgolj enkratno preverjanje ob uvedbi programske opreme. Ker se komponente, ponudniki in grožnje s časom spreminjajo, mora organizacija ohraniti pregled nad kritičnimi deli svoje dobavne verige ter spremljati spremembe, ki bi lahko vplivale na varnost. Pomembno je tudi, da so določeni postopki obveščanja in odgovorne osebe, ki lahko ocenijo vpliv morebitnega varnostnega incidenta ter sprejmejo ustrezne ukrepe [55], [56].

Nenazadnje mora organizacija imeti pripravljen odziv na incidente. Napadi na programsko dobavno verigo pogosto zahtevajo drugačen odziv kot klasični varnostni incidenti, saj lahko prizadenejo zunanje komponente, razvojna okolja in že distribuirano programsko opremo. Učinkovit odziv vključuje hitro ugotavljanje obsega vpliva, preklíc ogroženih dostopov, obveščanje prizadetih uporabnikov, odstranitev nevarnih različic ter ponovno vzpostavitev zaupanja v programske artefakte [56], [59].

Učinkovita zaščita programske dobavne verige zato ni samo vprašanje uporabe varnostnih orodij, temveč zahteva kombinacijo tehnologije, procesov in odgovornosti. Tehnični ukrepi zmanjšujejo možnost uspešnega napada, organizacijski ukrepi pa zagotavljajo, da organizacija tveganja pravočasno prepozna, obvladuje in se nanje ustrezno odzove [55], [56]. Organizacije pri vzpostavljanju takšnih praks pogosto uporabljajo uveljavljene okvire, kot je NIST Secure Software Development Framework (SSDF), ki pomaga sistematično vključiti varnostne prakse v razvojni življenjski cikel [1].

7 Razprava

Analiza predstavljenih napadalnih vektorjev in incidentov kaže, da se varnost programskih dobavnih verig ne more več obravnavati kot omejen problem upravljanja zunanjih knjižnic. Programska rešitev danes nastane v porazdeljenem sistemu, ki vključuje repozitorije izvirne kode, javne in zasebne registre paketov, sisteme za gradnjo programske opreme, okolja CI/CD, oblačne storitve, kontejnerske registre, podpisne mehanizme ter identitete ljudi in avtomatiziranih procesov. Vsaka od teh komponent predstavlja samostojno napadalno površino, hkrati pa tudi povezavo zaupanja do drugih delov sistema. Posledično varnost končnega izdelka ni odvisna samo od kakovosti njegove izvirne kode, temveč od najšibkejše povezave v celotnem procesu njegovega nastanka in distribucije.

Takšna ugotovitev pojasnjuje premik od tradicionalnih napadov na posamezne komponente k sistemskim napadom na verige zaupanja. Zgodnji incidenti, kot sta event-stream in ua-parser-js, so pokazali posledice zlorabe široko uporabljene odvisnosti. SolarWinds in Codecov sta nato pokazala, da je mogoče z vplivanjem na proces gradnje ali distribucije doseči veliko število organizacij, ne da bi bilo treba neposredno napasti vsako izmed njih. Sodobne kampanje ta pristop dodatno razširjajo: ukradene poverilnice omogočijo dostop do repozitorija, spremenjen delovni tok razkrije nove skrivnosti, pridobljeni žetoni omogočijo objavo paketov, zlonamerni paket pa nato ogrozi nova razvojna okolja. Napad zato ni več enkratno dejanje, ampak proces, v katerem posamezen uspešen vdor ustvarja pogoje za naslednjo stopnjo širjenja.

Tradicionalni varnostni modeli pri takšnih napadih niso zadostni predvsem zato, ker so bili pogosto zasnovani okoli jasne meje med notranjim zaupanja vrednim okoljem in zunanjim nezaupanja vrednim omrežjem. V sodobnem razvoju ta meja ni jasno definirana. Koda iz javnega registra se izvaja v notranjem okolju CI/CD, zunanja storitev sodeluje pri preverjanju identitete, oblačni sistem izdaja začasne poverilnice avtomatiziranemu procesu, končni artefakt pa je lahko sestavljen iz več sto neposrednih in posrednih odvisnosti. Po začetnem vdoru

napadalec pogosto uporablja legitimne račune, podpisane objave in dovoljene komunikacijske poti. Klasični nadzor omrežnega prometa ali iskanje znanih vzorcev zlonamerne kode zato ne zazna nujno dejstva, da je bilo zlorabljeno samo razmerje zaupanja.

Pomembna posledica je, da posamezen varnostni ukrep ne zagotavlja celovite zaščite. Večfaktorska avtentikacija zmanjša nevarnost kraje gesel, vendar ne prepreči zlorabe že izdanega žetona, ogrožene seje ali zlonamerne delovnega toka. SBOM izboljša pregled nad sestavo izdelka, vendar sama ne dokazuje, da so navedene komponente varne ali da artefakt med gradnjo ni bil spremenjen. Digitalni podpis potrjuje izvor podpisa in nespremenjenost artefakta po podpisu, ne pa nujno, ali je bil podpisni proces sam varen. Tudi kratkoživi žetoni OIDC zmanjšajo tveganje dolgoročnih skrivnosti, vendar lahko napačno določena pravila zaupanja omogočijo izdajo veljavnih poverilnic nepooblaščenemu procesu. Ti primeri kažejo, da je treba varnostne kontrole obravnavati kot medsebojno povezane sloje in ne kot neodvisne rešitve.

Okviri, kot sta NIST SSDF in SLSA, zato poudarjajo procesne in dokazljive lastnosti razvoja: zaščito kode in razvojnih okolij, nadzor sprememb, ločevanje odgovornosti, sledljivost izvora ter preverljivo poreklo artefaktov (angl. provenance) [1], [3]. Njihova vrednost ni zgolj v uvedbi dodatnih orodij, temveč v oblikovanju odgovora na vprašanje, pod katerimi pogoji je mogoče določenemu artefaktu zaupati. Varnostna odločitev se tako premakne od predpostavke, da je paket varen zato, ker prihaja iz znanega registra, k preverjanju njegovega izvora, postopka gradnje, uporabljenih identitet in pravil, po katerih je bil odobren za uporabo.

Hkrati ugotovitve kažejo, da odgovornosti ni mogoče prenesti samo na vzdrževalce odprtokodnih projektov. Številne kritične komponente vzdržuje majhno število posameznikov, medtem ko njihove pakete uporabljajo velike organizacije z bistveno večjimi finančnimi in kadrovskimi zmogljivostmi. Takšno nesorazmerje ustvarja sistemsko tveganje: ekosistem je odvisen od projektov, katerih vzdrževanje, upravljanje dostopov in odzivanje na incidente pogosto temelji na prostovoljnem delu. Organizacije morajo zato poleg preverjanja odvisnosti ocenjevati tudi stanje projektov, od katerih so odvisne, zmanjševati nepotrebno število komponent, podpirati ključne vzdrževalce ter pripravljati nadomestne možnosti za primere, ko projekt ni več ustrezno vzdrževan.

Umetna inteligenca to dinamiko dodatno okrepi, vendar je ne spremeni v povsem nov varnostni problem. Kot je bilo predstavljeno v petem poglavju, umetna inteligenca povečuje količino ustvarjene kode, hitrost vključevanja sprememb in stopnjo avtomatizacije razvojnih opravil. S tem se povečuje razkorak med hitrostjo nastajanja sprememb in zmožnostjo njihovega kakovostnega preverjanja. Pri agentnih sistemih je učinek še izrazitejši, saj model ne pripravlja več samo predlogov, temveč lahko namešča odvisnosti, spreminja konfiguracije in uporablja razvojne identitete. NIST-ov profil SSDF za generativno umetno inteligenco potrjuje, da morajo biti UI-komponente vključene v obstoječe postopke varnega razvoja, upravljanja dobaviteljev, dokumentiranja in preverjanja, ne pa obravnavane kot ločen tehnološki dodatek [1].

Iz tega izhaja potreba po novem modelu človeškega nadzora. Zahteva, da človek formalno potrdi vsako spremembo, ni učinkovita, če pregledovalec zaradi obsega ali kompleksnosti spremembe ne razume njenega učinka. Smiselni nadzor mora biti usmerjen v tveganja: avtomatizirane spremembe z dostopom do novih odvisnosti, skrivnosti, produkcijskih dovoljenj ali postopkov objave morajo zahtevati strožji pregled kot lokalne in omejene spremembe. Pomembna je tudi ločitev privilegijev, pri kateri razvojni agent, proces CI/CD ali posamezni vzdrževalec nima samostojnega dostopa do vseh korakov od spremembe kode do objave v produkcijo.

Varnost dobavnih verig ima tudi izrazito medorganizacijsko razsežnost. Organizacija lahko dobro zaščiti lastno infrastrukturo, vendar ostane izpostavljena prek ponudnika programske opreme, upravljanje storitve ali odprtokodne komponente. ENISA je že pri analizi incidentov med letoma 2020 in 2021 ugotovila, da so napadalci pogosto napadli dobavitelja, da bi dosegli njegove stranke, poznejše analize pa še naprej izpostavljajo digitalno dobavno verigo kot pomemben vektor napadov [2]. Zato ocenjevanje dobaviteljev ne sme biti omejeno na vprašalnike in pogodbene izjave, ampak mora vključevati konkretne dokaze o načinu razvoja, upravljanju ranljivosti, izvoru artefaktov, zaščiti identitet ter sposobnosti obveščanja in odzivanja ob incidentu.

Na podlagi obravnavanih primerov je mogoče programske dobavne verige razumeti kot kompleksen sociotehnični sistem. Tehnični elementi, kot so paketi, kontejnerske slike in žetoni, so povezani z organizacijskimi odločitvami, ekonomskimi omejitvami odprtokodnega razvoja in človeškim zaupanjem. Napadalci lahko prehajajo med temi ravni: s socialnim inženiringom pridobijo račun, z računom spremenijo avtomatizirani proces, prek procesa pridobijo identiteto, z identiteto pa objavijo tehnično veljaven artefakt. Obramba mora zato povezovati tehnične kontrole, upravljanje identitet, razvojne politike, nadzor dobaviteljev in odzivanje na incidente. Načela CISA Secure by Design podobno poudarjajo, da mora odgovornost za varnost postati sestavni del načrtovanja, izdelave in vzdrževanja programske opreme, ne pa breme, ki se pretežno prenaša na končnega uporabnika [60].

Obravnava ima tudi nekatere omejitve. Področje se hitro razvija, poimenovanja kampanj in začetne tehnične analize incidentov pa se lahko med viri razlikujejo. Pri novejših incidentih so pogosto najprej dostopna poročila varnostnih podjetij in upravljavcev prizadetih platform, medtem ko neodvisne akademske analize nastanejo pozneje. Prispevek zato predvsem sistematizira ponavljajoče se napadalne vzorce in odnose zaupanja, ne pa izčrpno vseh javno znanih incidentov. Nadaljnje raziskave bi morale bolj sistematično meriti dejanski učinek posameznih obrambnih ukrepov, proučiti prehodne oziroma posredne odvisnosti med razvojnimi storitvami ter oblikovati metode za ocenjevanje tveganja razvojnih agentov, ki delujejo z različnimi stopnjami avtonomije.

Skupna ugotovitev razprave je, da cilj varnosti ne more biti popolna odprava zaupanja, saj sodobni razvoj temelji prav na sodelovanju in ponovni uporabi komponent. Cilj mora biti zmanjšanje implicitnega zaupanja ter njegova zamenjava z omejenim, preverljivim in preklicljivim zaupanjem. Vsaka identiteta naj ima najmanjše potrebne privilegije, vsak pomemben artefakt preverljiv izvor, vsak korak objave sledljiv zapis, vsaka zunanja komponenta pa jasno opredeljeno odgovornost in možnost odziva. Takšen pristop ne prepreči vseh kompromitacij, vendar omeji njihovo širjenje in omogoči hitrejše razumevanje, katere povezave v dobavni verigi so bile prizadete.

8 Zaključek

Programske dobavne verige so se iz podpornega elementa razvoja programske opreme preoblikovale v eno ključnih področij sistemskega kibernetskega tveganja. Analiza razvoja napadov kaže, da sodobni napadalci vse redkeje ciljajo samo posamezno ranljivost ali komponento. Namesto tega izkoriščajo medsebojno povezanost odprtokodnih odvisnosti, računov vzdrževalcev, repozitorijev, okolij za gradnjo programske opreme, procesov CI/CD, registrov artefaktov, identitetnih sistemov in oblačne infrastrukture. Uspešen napad na eno samo komponento lahko zato sproži verigo nadaljnjih zlorab ter vpliva na veliko število organizacij, ki si neposredno ali posredno delijo isto komponento oziroma ponudnika.

Osrednja ugotovitev prispevka je, da sodobnih napadov na programske dobavne verige ni več ustrezno obravnavati kot posamezne nepovezane tehnične incidente. Gre za večstopenjske in pogosto avtomatizirane operacije, pri katerih napadalci postopoma prevzemajo legitimne identitete, procese in distribucijske kanale. Zaradi tega lahko zlonamerna dejavnost navzven deluje kot običajen razvojni postopek ali postopek objave. Najpomembnejša tarča tako ni nujno sama programska koda, temveč zaupanje, ki povezuje razvijalce, avtomatizirane procese, zunanje ponudnike in končne uporabnike.

Umetna inteligenca tega problema ne ustvarja sama, vendar ga lahko pomembno pospeši. Orodja, temelječa na velikih jezikovnih modelih, povečujejo hitrost ustvarjanja kode, testov, konfiguracij in avtomatiziranih sprememb, hkrati pa lahko zmanjšajo delež neposrednega človeškega preverjanja. Halucinirane odvisnosti, sprejemanje nepreverjene generirane kode, posredni napadi z vbrizgavanjem navodil ter široke pravice razvojnih agentov širijo napadalno površino. Na strani napadalcev lahko enaka orodja pospešijo iskanje ranljivih komponent, pripravo prepričljivega socialnega inženiringa, prilagajanje zlonamerne kode in izvajanje obsežnejših kampanj.

Učinkovita zaščita zato zahteva večplastni in sistemski pristop. Organizacije morajo poleg upravljanja odvisnosti in izdelave seznama programskih komponent preverjati izvor ter celovitost artefaktov, omejevati privilegije razvojnih in identitetnih sistemov, varovati skrivnosti, utrjevati okolja CI/CD ter spremljati spremembe po celotni

poti od izvirne kode do produkcije. Tehnični ukrepi morajo biti dopolnjeni z organizacijskimi procesi, ki vključujejo nadzor tretjih ponudnikov, jasno razmejitve odgovornosti, pripravljenost na preklc kompromitiranih poverilnic in artefaktov ter usklajen odziv na incidente, ki lahko presegaajo meje posamezne organizacije.

Prihodnje raziskave bi morale podrobneje preučiti merjenje kaskadnega tveganja med povezanimi organizacijami, samodejno odkrivanje zlorab v CI/CD in federiranih identitetnih okoljih ter preverjanje porekla kode in odločitev, ki jih ustvarjajo razvojni agenti. Posebno pomembni bodo modeli nadzora, ki omogočajo uporabo umetne inteligence brez opustitve preverljivosti, načela najmanjših privilegijev in odgovornega človeškega odločanja. Varnost programske dobavne verige bo v prihodnje zato odvisna predvsem od sposobnosti organizacij, da ne varujejo le posameznih komponent, temveč neprekinjeno preverjajo celotno verigo zaupanja, skozi katero programska oprema nastane, se distribuira in izvaja.

Literatura

- [1] M. Souppaya, K. Scarfone, and D. Dodson, “Secure Software Development Framework (SSDF) version 1.1 ;,” Feb. 2022. doi: 10.6028/NIST.SP.800-218.
- [2] Ifigenia. Lella, Marianthi. Theocharidou, Eleni. Tsekmezoglou, Apostolos. Malatras, S. García, and Veronica. Valeros, *ENISA threat landscape for supply chain attacks*. ENISA, 2021.
- [3] SLSA Framework, “SLSA: Supply-chain Levels for Software Artifacts.” Accessed: Aug. 01, 2026. [Online]. Available: <https://github.com/slsa-framework/slsa>
- [4] K. Thompson, “Reflections on trusting trust,” *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984, doi: 10.1145/358198.358210.
- [5] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks,” 2020, pp. 23–43. doi: 10.1007/978-3-030-52683-2_2.
- [6] “Active Exploitation of SolarWinds Software | CISA.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cisa.gov/news-events/alerts/2020/12/13/active-exploitation-solarwinds-software>
- [7] “Bash Uploader Security Update - Codecov.” Accessed: Aug. 01, 2026. [Online]. Available: <https://about.codecov.io/security-update/>
- [8] S. Neupane, G. Holmes, E. Wyss, D. Davidson, and L. De Carli, “Beyond Typosquatting: An In-depth Look at Package Confusion,” in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 3439–3456. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity23/presentation/neupane>
- [9] A. Freund, “oss-security - backdoor in upstream xz/liblzma leading to ssh server compromise.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.openwall.com/lists/oss-security/2024/03/29/4>
- [10] R. Cox, “research!rsc: Timeline of the xz open source attack.” Accessed: Aug. 01, 2026. [Online]. Available: <https://research.swtch.com/xz-timeline>
- [11] A. Hahami, “OH-MY-DC: OIDC Misconfigurations in CI/CD.” Accessed: Aug. 01, 2026. [Online]. Available: <https://unit42.paloaltonetworks.com/oidc-misconfigurations-in-ci-cd/>
- [12] Imperva, “What is Dependency Confusion | Risks & Prevention | Imperva.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.imperva.com/learn/application-security/dependency-confusion/>
- [13] A. Birsan, “Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies | by Alex Birsan | Medium.” Accessed: Aug. 01, 2026. [Online]. Available: <https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610>
- [14] Endor Labs, “Dependency Confusion: How Attackers Poison Your Build | Blog | Endor Labs.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.endorlabs.com/learn/dependency-confusion-how-attackers-poison-your-build>
- [15] Rescana, “In-Depth Analysis: Supply Chain Poisoning of Popular npm Packages Exploiting event-stream, ua-parser-js, and More – Rescana.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.rescana.com/post/in-depth-analysis-supply-chain-poisoning-of-popular-npm-packages-exploiting-event-stream-ua-parser>
- [16] npm Blog, “npm Blog Archive: Details about the event-stream incident.” Accessed: Aug. 01, 2026. [Online]. Available: <https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident>

- [17] P. Muncaster, "What is Typosquatting? Definition, Attack Types & Prevention." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.eset.com/blog/en/home-topics/privacy-and-identity-protection/what-is-typosquatting-how-do-i-tackle-it/>
- [18] A. Stiefel and J. Gile, "Major Supply Chain Attack Compromises Popular npm Packages Including chalk and debug | Blog | Endor Labs." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.endorlabs.com/learn/major-supply-chain-attack-compromises-popular-npm-packages-including-chalk-and-debug>
- [19] J. F. Bonaobra and J. Aquino, "What We Know About the NPM Supply Chain Attack | Trend Micro (US)." Accessed: Aug. 01, 2026. [Online]. Available: https://www.trendmicro.com/en_us/research/25/i/npm-supply-chain-attack.html
- [20] M. Fiedler, "Token Exfiltration Campaign via GitHub Actions Workflows - The Python Package Index Blog." Accessed: Aug. 01, 2026. [Online]. Available: <https://blog.pypi.org/posts/2025-09-16-github-actions-token-exfiltration/>
- [21] M. Abidi, C. Decotignie, and G. Plante, "CI/CD Security: Supply chain attack from a compromised developer - RiskInsight." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.riskinsight-wavestone.com/en/2026/07/ci-cd-security-supply-chain-attack-from-a-compromised-developer/>
- [22] CyberSierra, "Supply Chain Attacks in CI/CD Pipelines: The New Cloud Security Nightmare." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cybersierra.co/blog/prevent-cicd-supply-chain-attacks>
- [23] M. Ayenson, "CI/CD pipeline abuse: the problem no one is watching — Elastic Security Labs." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.elastic.co/security-labs/detecting-cicd-pipeline-abuse-with-llm-augmented-analysis>
- [24] R. Nisimi, "TeamPCP Exposes Checkmarx CI/CD Secrets | Orca Security." Accessed: Aug. 01, 2026. [Online]. Available: <https://orca.security/resources/blog/checkmarx-supply-chain-compromise-ci-cd-secrets/>
- [25] A. Kurmi, "5 Supply Chain Attacks in 48 Hours: Why Securing One Layer Is Not Enough - StepSecurity." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.stepsecurity.io/blog/5-supply-chain-attacks-in-48-hours-why-securing-one-layer-is-not-enough>
- [26] M. Shubel, "Cordyceps flaw pattern is more proof CI/CD is part of the attack surface - The New Stack." Accessed: Aug. 01, 2026. [Online]. Available: <https://thenewstack.io/cordyceps-cicd/>
- [27] T. Linsley, "Postmortem: TanStack npm supply-chain compromise | TanStack Blog." Accessed: Aug. 01, 2026. [Online]. Available: <https://tanstack.com/blog/npm-supply-chain-compromise-postmortem>
- [28] Gerrard Sarah, C. Crutchley, J. Herrington, Linsey Tanner, and Pellet Florian, "Hardening TanStack After the npm Compromise | TanStack Blog." Accessed: Aug. 01, 2026. [Online]. Available: <https://tanstack.com/blog/incident-followup>
- [29] Ossprey, "Megalodon: Active GitHub Actions Supply Chain Attack." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.ossprey.com/blog/megalodon-credential-harvester-attack>
- [30] The Chainguard Team, "Container Hardening: Securing your software supply chain." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.chainguard.dev/supply-chain-security-101/container-hardening-securing-your-software-supply-chain>
- [31] A. Morag, "Supply Chain Attacks Using Container Images - Aqua." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.aquasec.com/blog/supply-chain-threats-using-container-images/>
- [32] S. Chierici, "Analysis on Docker Hub malicious images: Attacks through public container images | Sysdig." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.sysdig.com/blog/analysis-of-supply-chain-attacks-through-public-docker-images>
- [33] "Trivy, KICS, and the shape of supply chain attacks so far in 2026 | Docker." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.docker.com/blog/trivy-kics-and-the-shape-of-supply-chain-attacks-so-far-in-2026/>
- [34] Santos Jacob and J. R. Navato, "Analyzing TeamPCP's Supply Chain Attacks: Checkmarx KICS and elementary-data in CI/CD Credential Theft | Trend Micro (US)." Accessed: Aug. 01, 2026. [Online]. Available: https://www.trendmicro.com/en_us/research/26/e/analyzing-teampcp-supply-chain-attacks.html
- [35] B. Clark, "npm Supply Chain Attack via Open Source maintainer compromise | Snyk." Accessed: Aug. 01, 2026. [Online]. Available: <https://snyk.io/blog/npm-supply-chain-attack-via-open-source-maintainer-compromise/>
- [36] J. F., "Software supply chain attacks: check your dependencies | National Cyber Security Centre." Accessed: Aug. 01, 2026. [Online]. Available: <https://www.ncsc.gov.uk/blogs/software-supply-chain-attacks-check-your-dependencies>

- [37] “Artificial intelligence risk management framework ;,” Jul. 2024. doi: 10.6028/NIST.AI.600-1.
- [38] “Secure Software Development Framework | CSRC.” Accessed: Aug. 01, 2026. [Online]. Available: <https://csrc.nist.gov/projects/ssdf>
- [39] E. Kalliamvakou, “Research: quantifying GitHub Copilot’s impact on developer productivity and happiness - The GitHub Blog.” Accessed: Aug. 01, 2026. [Online]. Available: <https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>
- [40] P. Fontana, Y. Lee, H. Iyer, and S. Sharma, “2025 NIST GenAI (Pilot): Code Challenge Evaluation Plan | NIST.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.nist.gov/publications/2025-nist-genai-pilot-code-challenge-evaluation-plan>
- [41] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, May 2022, pp. 754–768. doi: 10.1109/SP46214.2022.9833571.
- [42] V. Majdinasab, M. J. Bishop, S. Rasheed, A. Moradidakhel, A. Tahir, and F. Khomh, “Assessing the Security of GitHub Copilot Generated Code -- A Targeted Replication Study,” Nov. 2023.
- [43] J. Spracklen, R. Wijewickrama, A. H. M. N. Sakib, A. Maiti, and B. Viswanath, “We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs,” in *34th USENIX Security Symposium (USENIX Security 25)*, Seattle, WA: USENIX Association, Aug. 2025, pp. 3687–3706. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity25/presentation/spracklen>
- [44] “AI Agent Security - OWASP Cheat Sheet Series.” Accessed: Aug. 01, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/AI_Agent_Security_Cheat_Sheet.html
- [45] “SBOM Resources Library | CISA.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cisa.gov/topics/information-communications-technology-supply-chain-security/sbom/resources-library>
- [46] “Software Supply Chain Security Guidance | NIST.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-supply-chain-security-guidance>
- [47] “Documentation for the npm registry, website, and command-line interface.” Accessed: Aug. 01, 2026. [Online]. Available: <https://docs.npmjs.com/>
- [48] “Mitigating supply chain attacks | pnpm.” Accessed: Aug. 01, 2026. [Online]. Available: <https://pnpm.io/supply-chain-security>
- [49] “Add a trusted dependency - Bun.” Accessed: Aug. 01, 2026. [Online]. Available: <https://bun.com/docs/guides/install/trusted>
- [50] “Home · Sigstore.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.sigstore.dev/>
- [51] “CI CD Security - OWASP Cheat Sheet Series.” Accessed: Aug. 01, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html
- [52] M. Souppaya, J. Morello, and K. Scarfone, “SP 800-190, Application Container Security Guide | CSRC.” Accessed: Aug. 01, 2026. [Online]. Available: <https://csrc.nist.gov/pubs/sp/800/190/final>
- [53] R. Chandramouli, F. Kautz, and S. Torres Arias, “SP 800-204D, Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD pipelines | CSRC.” Accessed: Aug. 01, 2026. [Online]. Available: <https://csrc.nist.gov/pubs/sp/800/204/d/ipd>
- [54] “OpenID Connect - GitHub Docs.” Accessed: Aug. 01, 2026. [Online]. Available: <https://docs.github.com/en/actions/concepts/security/openid-connect>
- [55] “Software Supply Chain Security - OWASP Cheat Sheet Series.” Accessed: Aug. 01, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Software_Supply_Chain_Security_Cheat_Sheet.html
- [56] J. Boyens, A. Smith, N. Bartol, K. Winkler, A. Holbrook, and M. Fallon, “SP 800-161 Rev. 1, Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations | CSRC.” Accessed: Aug. 01, 2026. [Online]. Available: <https://csrc.nist.gov/pubs/sp/800/161/r1/upd1/final>
- [57] “Securing the Software Supply Chain: Recommended Practices for Developers | CISA.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cisa.gov/resources-tools/resources/securing-software-supply-chain-recommended-practices-developers>

- [58] “Securing the Software Supply Chain: Recommended Practices Guide for Customers and accompanying Fact Sheet | CISA.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cisa.gov/resources-tools/resources/securing-software-supply-chain-recommended-practices-guide-customers-and>
- [59] “Defending Against Software Supply Chain Attacks | CISA.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cisa.gov/resources-tools/resources/defending-against-software-supply-chain-attacks>
- [60] “Secure by Design | CISA.” Accessed: Aug. 01, 2026. [Online]. Available: <https://www.cisa.gov/securebydesign>

Kdo in zakaj potrebuje Program nagrajevanja za varnostne hrošče?

Denis Tomašević

Nicehash, Maribor, Slovenija
denis@nicehash.com

V Sloveniji je že precej uveljavljeno iskanje ranljivosti z varnostnimi pregledi, ki se izvajajo enkrat letno (kampanjski pristop, ki ga predpisujejo regulatorji). Ti pregledi so dragi zaradi svojega obsega in omejenega števila strokovnjakov v Sloveniji, stanejo pa enako, ne glede na to, ali ima rešitev tisoč ranljivosti ali nobene. Program Security Bug Bounty omogoča pokritost 365 dni v letu in neomejen nabor strokovnjakov z vsega sveta za ceno, ki je sorazmerna trenutni varnostni kondiciji rešitve. Vsebina prispevka bo uporabna tako za tiste, ki bi radi začeli iskati varnostne ranljivosti (kot glavni ali dodatni vir prihodka), kot za tiste, ki ranljivosti ustvarjajo (razvijalci, arhitekti, upravljalci rešitev ...), pa tudi za tiste, ki bi morali skrbeti za varno delovanje rešitev (vodje razvoja ter strokovnjaki za informacijsko varnost in skladnost z regulatornimi okviri).

Ključne besede:

Bug Bounty

vdorno testiranje (PenTest)

kibernetska varnost

varnostne ranljivosti

CrowdSource varnostno testiranje

1 Kdo in zakaj potrebuje PenTest?

Varnostni pregled z izvajanjem vdornih testov (PenTest), v katerem izvajalec uporablja enake metode in tehnike kot dejanski napadalci, je najučinkovitejše preverjanje trenutnega varovanja aplikacije, platforme ali informacijskega sistema.

Testiranje s simulacijo napadov zahteva veliko strokovnega znanja in aktivno spremljanje novih ranljivosti ter načinov zlorabe, izvedba PenTesta na produkcijskem okolju pa je tvegana, saj uspešen napad lahko ogrozi integriteto ali zaupnost podatkov uporabnikov, zato je PenTest priporočljivo izvajati na kopiji produkcijskega okolja brez dejanskih podatkov (testno okolje). Ne glede na to je izvedba določenega nabora vdornih testov na produkcijskem okolju smiselna in potrebna (brez agresivnih testov), ker lahko odkrije ranljivosti, ki so posledica nedosledne konfiguracije ali posebnosti produkcijskega okolja, vendar je takšne teste smiselno omejiti na nujni minimum in jih izvajati na posebnih uporabniških računalnikih.

PenTest v resnici potrebuje vsaka organizacija, ki ji je pomembna dejanska varnost svojih uporabnikov in podatkov, če je pripravljena sprejeti in uporabiti rezultate za izboljšanje svoje varnosti. Slika v tako »nastavljenem ogledalu« pogosto ni prijetna.

2 Najeti PenTest tim

Najpogostejše naročnik PenTesta najame zunanjo ekipo, specializirano za izvedbo vdornih testov.

Pri izbiri izvajalca PenTesta organizacija pogosto preverja strokovna znanja potencialnih izvajalcev (različne strokovne certifikate, medijsko prisotnost, strokovne reference in priporočila drugih naročnikov ...), o izbiri izvajalca pa pogosto odločata razpoložljivost izvajalca v izbranem terminu in cena.

Za uspešno izvedbo PenTesta je izjemno pomembno vzpostaviti zaupanje ter odprto komunikacijo med naročnikom, izvajalcem in dobaviteljem/upravitelcem aplikacije/platforme/informacijskega sistema. Glavni izziv najetega PenTesta je omejen čas, ki ga ima izvajalec PenTesta na voljo za izvedbo projekta (in ravno vloženi čas neposredno določa tudi ceno projekta). Naročnik testiranja od izvajalca pravzaprav najame ekipo varnostnih strokovnjakov za zelo omejen čas, v katerem mora ta spoznati delovanje naročnikovega sistema, poiskati ranljivosti, izvesti vdorne teste, ter pripraviti in predstaviti poročilo o testiranju. V praksi to pomeni, da izvajalec porabi za dejansko vdorno testiranje le približno polovico projektne oziroma plačanega časa in pogosto ne more temeljito preveriti celotne funkcionalnosti ali obsega projekta, torej opravi le "vzorčno testiranje" (v nasprotju z napadalci, ki imajo na voljo neomejen čas). Če je motiv naročnika predvsem pridobitev potrdila o izvedenem PenTestu (ker to zahteva zakonodaja, regulator ali certifikat ...) in ne dejansko izboljšanje varnosti, ali če dobavitelj/upravitelj aplikacije/platforme/informacijskega sistema test doživlja kot napad na svojo integriteto namesto priložnost za izboljšanje, pogosto pride do prikrivanja informacij (»če so strokovnjaki, bodo že ugotovili ...«). Posledično se dejanski čas, ki ga izvajalec namenja ključni dejavnosti testiranja, še zmanjša, s tem pa se zmanjša dodana vrednost celotnega projekta.

Dodatni izziv najetega testiranja je, da se praviloma izvaja le v določenih (predpisanih) časovnih intervalih, na primer enkrat letno, čeprav je vsem jasno, da je v današnjem času popolnoma nerealno pričakovati, da bo naročnikovo okolje ostalo nespremenjeno do naslednjega PenTesta. Tudi če bi bilo okolje naročnika popolnoma statično, se okoli njega vse nenehno spreminja: pojavljajo se nove ranljivosti in novi načini njihove zlorabe.

3 Interni PenTest tim

Naročnik se lahko odloči, da bo za testiranje varnosti aplikacij/platforme/informacijskega sistema vzpostavil lasten interni tim strokovnjakov, kar je pogosta praksa pri organizacijah z visokim varnostnim tveganjem.

Glavna prednost internega tima je možnost njegovega sodelovanja že pri načrtovanju varnostne arhitekture, izbiri in postavitvi varnostnih ukrepov ter pri nadzoru njihovega delovanja, in nenazadnje neposrednem dostopu do informacij naročnika, kot so interni dnevniki dogodkov, funkcijske specifikacije in celo izvorna koda produktov. Dobro poznavanje implementirane tehnologije, varnostnih ukrepov, varnostnih incidentov in morebitnih odprtih ranljivosti omogoča interni ekipi za PenTest hitro, sprotno in učinkovito izvedbo testiranja, ker pa je interni tim del organizacije naročnika, je pogosto lažje vzpostaviti ali celo vsiliti potrebno zaupanje in sodelovanje z drugimi deležniki.

Žal si takšen interni tim strokovnjakov lahko privoščijo le redke organizacije, saj je pridobivanje in vzdrževanje strokovnjakov za kibernetiko varnost zelo zahtevno in povezano z visokimi stroški. Dobri strokovnjaki za vdorno testiranje potrebujejo veliko izkušenj in stalnega izobraževanja, zato se pogosto ozko specializirajo, kar pa v današnjem času, ko so platforme in informacijski sistemi kompleksni, pomeni, da celovito in kakovostno vdorno testiranje lahko pogosto izvede le večja ekipa strokovnjakov različnih profilov, hkrati pa so takšni kadri izjemno iskani, dragi in jih je težko obdržati.

4 CrowdSource PenTesti

CrowdSource platforme že poldrugo desetletje omogočajo organizacijam izvajanje varnostnih pregledov z množico varnostnih raziskovalcev z zelo različnim naborom in ravno znanja.

Raznolikost raziskovalcev je hkrati največja prednost in največja pomanjkljivost takšnega testiranja, saj prihaja do velike razlike v kakovosti posameznih poročil in tudi do velikega števila neveljavnih ali zelo splošnih ugotovitev, ki nimajo prave dodane vrednosti.

Praktično vse CrowdSource platforme naročniku ponujajo možnost uporabe njihove triažne storitve in/ali umetne inteligence za pregled in preverjanje poročil, za preverjanje navedb, izločanje neveljavnih poročil ter vsebinsko in oblikovno ureditev veljavnih poročil v profesionalno in uporabno obliko.

CrowdSource platforme naročnikom ponujajo tako postavitev odprtega programa za določen čas kot tudi izvedbo različnih kampanj (na primer pred izidom nove različice) ali celo "časovni najem" strokovnjakov izbranega profila (primerljivo z najetim PenTest timom).

Največja razlika uporabe CrowdSource platforme v primerjavi z najetim/internim timom je način plačila, saj naročnik plača nagrado (Bug Bounty) za prijavljeno ranljivost glede na ugotovljeno tveganje potrjene ranljivosti, pri najetem ali internem timu pa plača pavšalni znesek ne glede na rezultat oziroma uspešnost testiranja.

Naročnik sam določa višino nagrad za posamezne kategorije tveganj in obseg programa Bug Bounty (katere domene, naprave, funkcionalnosti in ranljivosti so vključene v program in katere ne).

CrowdSource platforme sicer opravijo določena preverjanja raziskovalcev (KYC), vendar v zelo omejenem obsegu, zato je stopnja zaupanja med naročnikom in raziskovalci minimalna. Če naročnik želi omogočiti njemu »neznanim« raziskovalcem izvajanje varnostnih testiranj, je zelo priporočljivo, da jim omogoči dostop do testnega okolja, ki je verodostojna kopija njegovega produkcijskega okolja.

CrowdSource platforme so tudi okolje, v katerem se kalijo novi varnostni strokovnjaki, nekateri najboljši izvajalci vdornih testov vso kariero delujejo le prek tovrstnih platform, saj zaslužek na teh platformah lahko presega najbolj plačana podobna delovna mesta, sami pa lahko izbirajo, kdaj, koliko in kaj bodo testirali.

5 Avtomatizirani PenTesti

Uporaba avtomatiziranih varnostnih testiranj je zelo pogosta, priročna in cenovno ugodna metoda za iskanje ranljivosti. Na voljo je veliko različnih programov in platform, ki imajo navadno nabor že pripravljenih testov, nekateri pa omogočajo tudi dodajanje novih testov. Naročnik navadno izbere svoj nabor testov in interval ponavljanja, nekatere rešitve pa omogočajo tudi polavtomatizirano testiranje, pri katerem uporabnik ročno uporablja spletno aplikacijo prek preglednega posredniškega strežnika, hkrati pa se izvajajo različni testi na vsaki obiskani spletni strani, aplikacija pa si lahko zapomni vnesene vrednosti in »pot« skozi aplikacijo za naslednje ponovitve.

Enostavnost uporabe in nizka cena so glavne prednosti avtomatiziranega testiranja, ki pa na področju vdornih testov dolgo časa ni bilo primerljivo s testiranjem strokovnjakov, dokler v svet avtomatiziranega vdornega testiranja ni vstopila umetna inteligenca.

Rezultati vdornih testov, vodenih z najnovejšimi modeli umetne inteligence že presegajo najboljše time strokovnjakov, kar ne preseneča, saj lahko UI v kratkem času opravi bistveno več testov in tako odpravi največjo omejitev najetega PenTesta – omejeni čas strokovnjakov.

Čeprav so v tem trenutku nekatere države omejile javni dostop do določenih modelov umetne inteligence, je vsem jasno, da bo ta način testiranja vse bolj priljubljen in dostopen.

6 Hibridni PenTesti

Učinkovita in varna uporaba avtomatiziranih orodij za vdorne teste še vedno zahteva veliko znanja, zato vedno več ponudnikov najetih PenTestov naročnikom omogoča izvajanje določenega nabora testov z umetno inteligenco v kombinaciji z ročnim strokovnim testiranjem. Takšna hibridna kombinacija izvajanja vdornih testov je lahko hitrejša, bolj kakovostna in finančno ugodnejša od uporabe samo avtomatiziranega ali samo najetega testiranja.

7 Naključni PenTesti

Uporabnik neke aplikacije/platforme/informacijskega sistema je lahko hkrati varnostni raziskovalec. Uporabniki pogosto prvi opazijo napake in anomalije pri uporabi aplikacije, vendar bodisi ne razumejo možnih posledic naključno odkrite ranljivosti, bodisi nimajo interesa, da vložijo svoj čas v njeno dokumentiranje in prijavo.

Organizacija seveda ne more spremeniti svojih uporabnikov v varnostne strokovnjake, lahko pa motivira tiste uporabnike, ki potrebna znanja že imajo, da prijavijo zaznane napake.

8 Kateri način vdornega testiranja izbrati?

Odgovor mora poiskati vsaka organizacija zase glede na svojo oceno tveganja in svojo »zrelost«. Priporočljiva je uporaba vseh navedenih načinov, vendar ne vseh hkrati.

Prve vdorne teste je najprimerneje izvesti z najetim timom, saj tak pregled ne zahteva veliko internih virov, naročnik lahko poročilo testiranja posreduje upravljalcu/dobavitelju ranljive opreme, izvajalec testov pa lahko opravi kontrolni pregled učinkovitosti implementiranih korektivnih ukrepov.

Za drugo izvedbo najetih vdornih testov je priporočljivo izbrati drugega izvajalca, saj drugačen pristop k testiranju in drugačen nabor strokovnjakov pogosto zagotovi drugačen »vzorec vdornih testov« in odkrije dodatne ranljivosti.

Po odpravi ranljivosti, ugotovljenih pri prvi in drugi izvedbi vdornih testov, (seveda ob ustrezni kakovosti njihove izvedbe) lahko pričakujemo, da bodo identificirane in odpravljene tiste največje in najbolj očitne ranljivosti, kar je pravi čas za vzpostavitev programa nagrajevanja za varnostne hrošče, ki je osnova za CrowdSource in naključno testiranje.

Evropske digitalne denarnice EUDIW in EBW: od regulative do prakse

Martin Domajnko, Vid Keršič, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor,
Slovenija

martin.domajnko1@um.si, vid.kersic@um.si, muhamed.turkanovic@um.si

V prispevku predstavljamo evropski okvir digitalne identitete s poudarkom na evropski denarnici za digitalno identiteto (EUDI) za fizične osebe in predlagani evropski poslovni denarnici (EBW) za pravne osebe. Osredotočamo se tako na regulativni okvir, vloge udeležencev in model zaupanja, na katerem temeljijo denarnice, kot tudi na tehnično arhitekturo, vključno z digitalnimi poverilnicami, formatoma SD-JWT VC in mdoc ter protokoloma OpenID4VCI in OpenID4VP. Praktično uporabo digitalnih denarnic prikažemo z referenčno implementacijo izdajatelja in preveritelja digitalnih poverilnic ter primerom prijave v spletno storitev z EUDI denarnico. V razpravi primerjamo stopnjo zrelosti obeh okvirov ter obravnavamo možnosti in zahteve za vključevanje organizacij v nastajajoči ekosistem evropskih digitalnih denarnic.

Ključne besede:

digitalna identiteta

digitalna poverilnica

eIDAS 2.0

evropska denarnica za digitalno identiteto

evropska poslovna denarnica

1 Uvod

Hitra digitalizacija družbe spreminja način, kako dostopamo do storitev, sklepamo posle ter dokazujemo različne podatke o sebi. Pri uporabi spletnih storitev skoraj vedno uporabljamo neko obliko digitalne identitete, ki je praviloma vezana na posamezno platformo in temelji na uporabniškem računu z uporabniškim imenom in geslom. Del razdrobljenosti zmanjšujejo sistemi enotne prijave (angl. Single Sign-On, SSO), pri katerih se uporabnik z istim računom prijavlja v več storitev, vendar tudi ti, četudi temeljijo na t. i. federativnem modelu digitalne identitete, običajno temeljijo na centraliziranem ponudniku (digitalne identitete) [1]. Hkrati se digitalizirajo različne poverilnice, vstopnice, letalske karte ipd., pri čemer pomemben del teh funkcionalnosti zagotavljajo izolirane aplikacije in lastniške denarnice ponudnikov, kot sta Apple Wallet in Google Wallet [2, 3]. Digitalne identitete in poverilnice zato praviloma niso prenosljive med platformami, njihova uporabnost pa je odvisna od pravil in tehničnih zmožnosti posameznega ponudnika.

Takšna razdrobljenost je posebej problematična v Evropski uniji (EU), kjer poleg nacionalnih sistemov elektronske identifikacije velik del pogosto uporabljenih digitalnih identitet zagotavljajo zasebna tehnološka podjetja, večinoma s sedežem zunaj EU. Posledično ni enotnega načina za uporabo digitalne identitete in izmenjavo digitalnih poverilnic med državami članicami, javnimi organi ter zasebnimi ponudniki storitev [4]. To zmanjšuje tudi čezmejno interoperabilnost, povečuje stroške integracije in ustvarja odvisnost od posameznih tehnoloških ponudnikov, uporabnikom pa otežuje nadzor nad lastnimi podatki. Prvotna uredba eIDAS iz leta 2014 je sicer vzpostavila pravni okvir za elektronsko identifikacijo in storitve zaupanja, vendar ni zagotovila enotne, uporabniško nadzorovane rešitve za hrambo in izmenjavo različnih digitalnih poverilnic [4, 5]. Še manj celovito je bilo urejeno področje digitalne identitete pravnih oseb. Digitalni certifikati, elektronski podpisi in elektronski žigi omogočajo overjanje ter zagotavljanje pristnosti in celovitosti dokumentov, vendar ne predstavljajo celovite poslovne digitalne identitete, ki bi podpirala upravljanje organizacijskih podatkov, vlog, pooblastil in različnih poslovnih poverilnic.

Evropska komisija je zato leta 2021 predlagala prenovu uredbe eIDAS, ki sta jo Evropski parlament in Svet EU leta 2024 sprejela kot Uredbo (EU) 2024/1183 o vzpostavitvi evropskega okvira za digitalno identiteto, pogosto poimenovano eIDAS 2.0 [6]. Eden izmed njenih osrednjih elementov je evropska denarnica za digitalno identiteto (angl. European Digital Identity Wallet, EUDI Wallet, v nadaljevanju: EUDI denarnica), ki naj bi fizičnim osebam omogočila varno prejemanje, hrambo in uporabo identifikacijskih podatkov ter drugih digitalnih poverilnic v javnih in zasebnih storitvah po celotni EU [6, 7, 8]. Države članice morajo zagotoviti najmanj eno EUDI denarnico, vse rešitve pa morajo temeljiti na skupnih standardih, protokolih in modelu zaupanja [6, 9, 10]. Okvir tako ni vezan na enega tehnološkega ponudnika, saj lahko denarnice zagotovijo države članice, pooblaščenči izvajalci ali priznani zasebni ponudniki, njihovo delovanje pa urejajo skupne evropske in nacionalne zahteve [6, 11]. Kot nadgradnjo tega pristopa za pravne osebe je Evropska komisija novembra 2025 predlagala še uredbo o evropskih poslovnih denarnicah (angl. European Business Wallet, EBW), namenjenih digitalizaciji poslovnih interakcij med podjetji in javnimi organi [12]. Ker zakonodajni postopek za EBW še poteka, gre za novejši in manj dokončno opredeljen okvir, ki se bo v nadaljevanju še tehnično in pravno razvijal [13, 14].

V prispevku predstavljamo evropski okvir digitalnih identitet s poudarkom na digitalni denarnici za fizične osebe (EUDI) in predlagani digitalni denarnici za pravne osebe (EBW). Najprej predstavimo njune ključne značilnosti, vloge udeležencev in razlike med osebno ter poslovno uporabo. Nato opišemo tehnično arhitekturo ekosistema, vključno z digitalnimi poverilnicami, podprtimi formati, protokoli ter značilnimi poteki izdajanja in preverjanja. V nadaljevanju predstavimo referenčno implementacijo platforme LutraID, ki organizacijam omogoča vključevanje funkcionalnosti izdajanja in preverjanja digitalnih poverilnic v obstoječe informacijske sisteme. Delovanje prikazemo na primeru prijave v spletno storitev z EUDI denarnico, v razpravi pa obravnavamo možnosti uporabe, omejitve in pripravljenost organizacij na prihajajoči ekosistem evropskih digitalnih denarnic.

2 Evropski okvir digitalnih identitet

V tem poglavju najprej predstavimo razvoj in regulativni okvir digitalne identitete, nato pa ločeno in podrobneje obravnavamo evropsko denarnico za digitalno identiteto (EUDIW) ter evropsko poslovno denarnico (EBW).

2.1. Razvoj in regulativni okvir digitalne identitete

Digitalna identiteta se je v zadnjih desetletjih razvijala skupaj z rastjo spletnih storitev in nastajanjem novih pristopov k upravljanju identitet. Prvi sistemi so bili večinoma centralizirani, pri čemer je vsak ponudnik sam upravljal uporabniške račune in podatke. Takšen model je povzročil veliko število ločenih računov in podatkovnih silosov, koncentracija osebnih podatkov pri posameznih ponudnikih pa je povečala varnostna tveganja ter omejila nadzor uporabnikov nad lastnimi podatki [15]. Federativni modeli so del teh težav naslovili z uporabo iste identitete pri več storitvah, med drugim s standardom OAuth 2.0 in nato OpenID Connect [16]. Vendar so hkrati povečali odvisnost od osrednjega ponudnika identitete, ki predstavlja enotno točko zaupanja in morebitno enotno točko odpovedi [15]. Naslednji korak v razvoju predstavljajo uporabniško nadzorovani modeli, kot je samoupravljana identiteta (angl. self-sovereign identity, SSI), ki temelji na decentraliziranih identifikatorjih in preverljivih poverilnicah [15, 17, 18, 19]. Evropski regulativni okvir in posledično EUDI denarnica povzemata nekatere principe takšnih modelov, vendar temeljita na reguliranem evropskem modelu zaupanja.

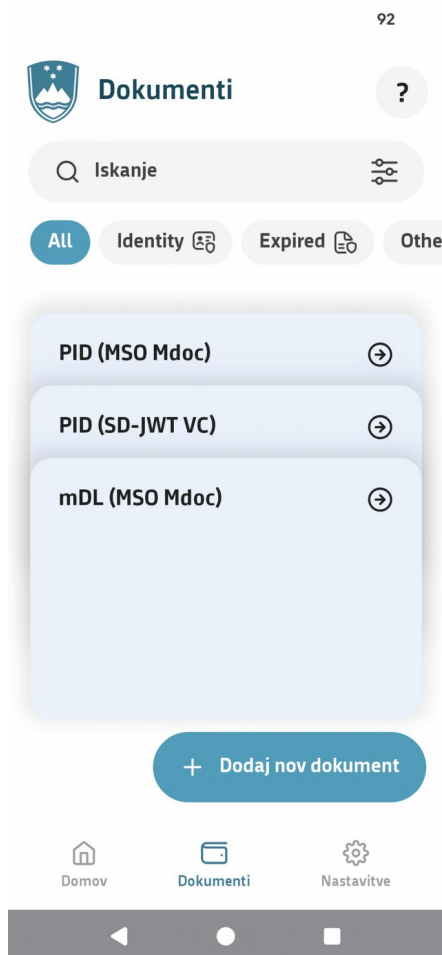
Prvotna uredba eIDAS je omogočila čezmejno priznavanje priglašениh nacionalnih sistemov elektronske identifikacije ter uredila storitve zaupanja, kot so elektronski podpisi, žigi in časovni žigi [5]. Njena uporaba je bila omejena predvsem na javne spletne storitve, prav tako pa ni zagotavljala enotnega načina za izmenjavo dodatnih digitalnih poverilnic [4]. Prenovljeni okvir eIDAS 2.0 zato uvaja EUDI denarnice, osebne identifikacijske podatke in elektronska potrdila atributov ter uporabo digitalne identitete razširja na javni in zasebni sektor [6, 7, 8, 20]. Evropska komisija je novi okvir predlagala leta 2021, Uredba (EU) 2024/1183 pa je bila sprejeta leta 2024 [4, 6, 21]. Države članice morajo do konca leta 2026 zagotoviti najmanj eno EUDI denarnico, ki mora biti interoperabilna z rešitvami drugih držav članic [6, 9]. Ekosistem vključuje države članice, ponudnike denarnic, izdajatelje digitalnih poverilnic, uporabnike, zanašajoče se stranke in tehnične posrednike. Model zaupanja dopolnjujejo ponudniki storitev zaupanja, registri ter organi za certificiranje in nadzor [9, 10, 11, 22].

Evropska komisija je novembra 2025 predlagala še evropske poslovne denarnice, namenjene gospodarskim subjektom, samozaposlenim in javnim organom [12]. Svet EU je junija 2026 sprejel pogajalsko stališče, zakonodajni postopek pa še ni zaključen [13, 14]. Ekosistem poslovnih denarnic vključuje lastnika poslovne denarnice, njene pooblaščenе uporabnike, ponudnike denarnic, izdajatelje poverilnic in zanašajoče se stranke. Poleg identitete organizacije mora zato podpirati tudi preverjanje identitete uporabnika, njegove vloge in pooblastila za izvedbo konkretnega dejanja [13].

2.2. EUDIW – digitalna denarnica za fizične osebe

EUDI denarnica je rešitev pod nadzorom uporabnika, namenjena varnemu prejemanju, shranjevanju, upravljanju in uporabi identifikacijskih podatkov osebe (angl. Person Identification Data, PID) ter elektronskih potrdil o atributih (angl. Electronic Attestations of Attributes, EAA), kot so mobilno vozniško dovoljenje, kartica zdravstvenega zavarovanja, poverilnica o izobrazbi ali poverilnica o sklenjenem zavarovanju [6, 7]. Glede na status izdajatelja in pravni režim izdaje so lahko takšna potrdila nekvalificirana EAA, kvalificirana elektronska potrdila o atributih (angl. Qualified Electronic Attestations of Attributes, QEAA), ki jih kot kvalificirano storitev zaupanja izdajajo kvalificirani ponudniki storitev zaupanja, ali elektronska potrdila o atributih javnega sektorja (angl. Public Sector Body Electronic Attestation of Attributes, PuB-EAA). PuB-EAA izda organ javnega sektorja, odgovoren za avtentični vir, torej uradno oziroma pravno priznано primarno evidenco, v kateri se vodijo potrjeni atributi, ali drug organ javnega sektorja, ki ga država članica določi za izdajo takšnih potrdil v njegovem imenu. Vrsta oziroma vsebina potrdila sama po sebi zato še ne določa, ali gre za nekvalificirano EAA, QEAA ali PuB-EAA. Ključne razlike med posameznimi vrstami digitalnih poverilnic povzema tabela 1. Za PID, EAA in sorodne kriptografsko preverljive zapise v nadaljevanju uporabljamo krovni tehnični izraz digitalne poverilnice. Denarnica omogoča

identifikacijo, avtentikacijo, elektronsko podpisovanje in selektivno razkrivanje podatkov [8]. Uporabnik lahko tako pri dokazovanju polnoletnosti razkrije le v poverilnici že vsebovano kriptografsko preverljivo trditev, na primer, da dosega zahtevano starost, ne pa tudi datuma rojstva ali celotnega osebnega dokumenta. Na sliki 1 je prikazan primer, kakšna bi lahko bila videti slovenska evropska denarnica za digitalno identiteto.



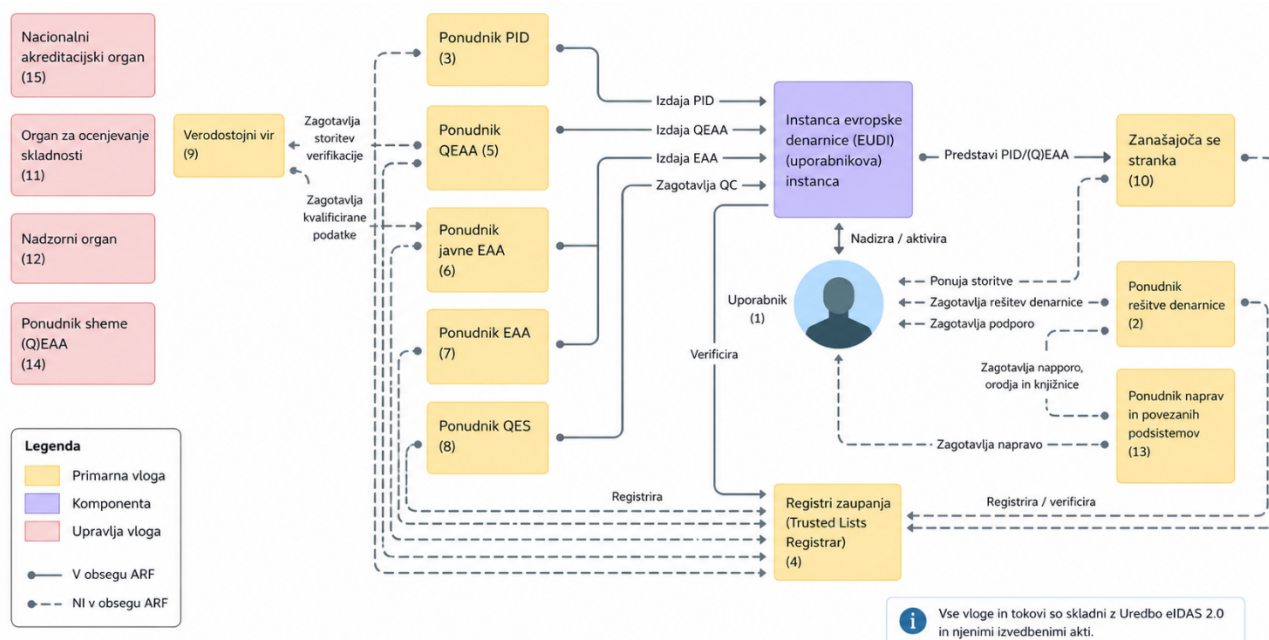
Slika 1: Simbolični prikaz slovenske evropske denarnice za digitalno identiteto. Zajem zaslona [23]

Tabela 1: Vrste digitalnih poverilnic v ekosistemu EUDI denarnice

Vrsta poverilnice	Izdajatelj	Pravni režim	Primer
PID	država članica oziroma pooblaščen subjekt na podlagi avtentičnega vira	obvezna sestavina EUDI denarnice po eIDAS 2.0	osebni identifikacijski podatki (ime, priimek, datum rojstva)
EAA (nekvalificirano)	katerikoli izdajatelj, tudi zasebni	ni kvalificirana storitev zaupanja	članska izkaznica, digitalna zavarovalna polica
QEAA	kvalificirani ponudnik storitev zaupanja	kvalificirana storitev zaupanja	kartica zdravstvenega zavarovanja
PuB-EAA	organ javnega sektorja, odgovoren za avtentični vir	enakovreden pravni učinek kot QEAA	mobilno vozniško dovoljenje

Osnovni model izmenjave vključuje izdajatelja poverilnic, uporabnika denarnice in **zanašajočo se stranko** (angl. Relying Party) [9]. Vloge posameznih udeležencev in njihove medsebojne povezave so prikazane na sliki 2.

Izdajatelj pripravi in kriptografsko podpiše digitalno poverilnico, uporabnik pa jo hrani pod svojim nadzorom ter odloča o njeni uporabi. Zanašajoča se stranka, na primer banka, mobilni operater ali javni organ, zahteva le podatke, potrebne za izvedbo konkretne storitve. Pri tem je treba razlikovati med zanašajočo se stranko, ki določi namen in obseg obdelave, ter **tehničnim preveriteljem**, ki pripravi zahtevek in preveri izdajatelja, podpis, veljavnost in status poverilnice ter njeno vezavo na uporabnika. Pred uporabo se zanašajoča se stranka registrira pri **pristojnem registracijskem organu** v državi članici svoje ustanovitve ter navede namen uporabe in attribute, ki jih namerava zahtevati. Če preverjanje v njenem imenu izvaja druga entiteta, ta nastopa kot **posrednik** (angl. intermediary) in se kot zanašajoča se stranka registrira v državi članici svoje ustanovitve. Njuno razmerje mora biti dokazano in evidentirano, denarnica pa uporabniku prikaže obe entiteti. Posrednik po posredovanju podatkov ne sme hraniti vsebine transakcije [22]. Zaupanje je organizirano večnivojsko: države članice vzdržujejo nacionalne registre in zaupanja vredne sezname, Evropska komisija pa zagotavlja skupno infrastrukturo za objavo in dostop do seznamov zaupanja vrednih entitet (angl. List of Trusted Entities, LoTE) ter objavlja seznam zaupanja vrednih seznamov EU (angl. List of Trusted Lists, LOTL). V LoTE so objavljena sidra zaupanja za preverjanje izdajateljev poverilnic ter izdajateljev dostopnih in registracijskih certifikatov. Zanašajoče se stranke in posredniki vanje niso vključeni neposredno, temveč se denarnici avtenticirajo z dostopnim certifikatom, njihov registrirani namen in obseg zahtevanih atributov pa se preverita z registracijskim certifikatom ali neposredno v registru. Preverjanje vključuje tudi veljavnost oziroma morebitni preklic (angl. revocation) certifikatov [10].



Slika 2: Vloge in tokovi v ekosistemu evropske denarnice za digitalno identiteto v skladu z Uredbo eIDAS 2.0. [9]

EUDI denarnica ni ena sama evropska aplikacija, temveč skupen ekosistem, zasnovan na enotnih predpisih, arhitekturnem in referenčnem okviru (angl. Architecture and Reference Framework, ARF) ter skupnih standardih in specifikacijah. ARF je skupna tehnična osnova za vloge, model zaupanja, zahteve in interoperabilnost [9]. Države članice morajo do konca leta 2026 zagotoviti najmanj eno EUDI denarnico, rešitev pa lahko zagotovi država neposredno, pooblaščen izvajalec ali priznani zasebni ponudnik. Pred produkcijsko uporabo morajo biti rešitve certificirane v skladu s skupnimi zahtevami in nacionalnimi certifikacijskimi shemami [6, 11, 24]. Razvoj in integracijsko preizkušanje podpirajo tudi nacionalna testna okolja oziroma peskovniki, med njimi francoski France Identité Playground in nemški EUDI Wallet Sandbox programa SPRIND. Interoperabilnost se preverja tudi v velikih evropskih pilotnih projektih. Skupno šest pilotov je preizkušalo več kot enajst vsakdanjih primerov uporabe, pri čemer je sodelovalo več kot 550 zasebnih podjetij in javnih organov iz 26 držav članic ter Norveške, Islandije in Ukrajine. Med prvimi štirimi piloti je POTENTIAL obravnaval e-upravo, bančništvo, telekomunikacije, mobilna

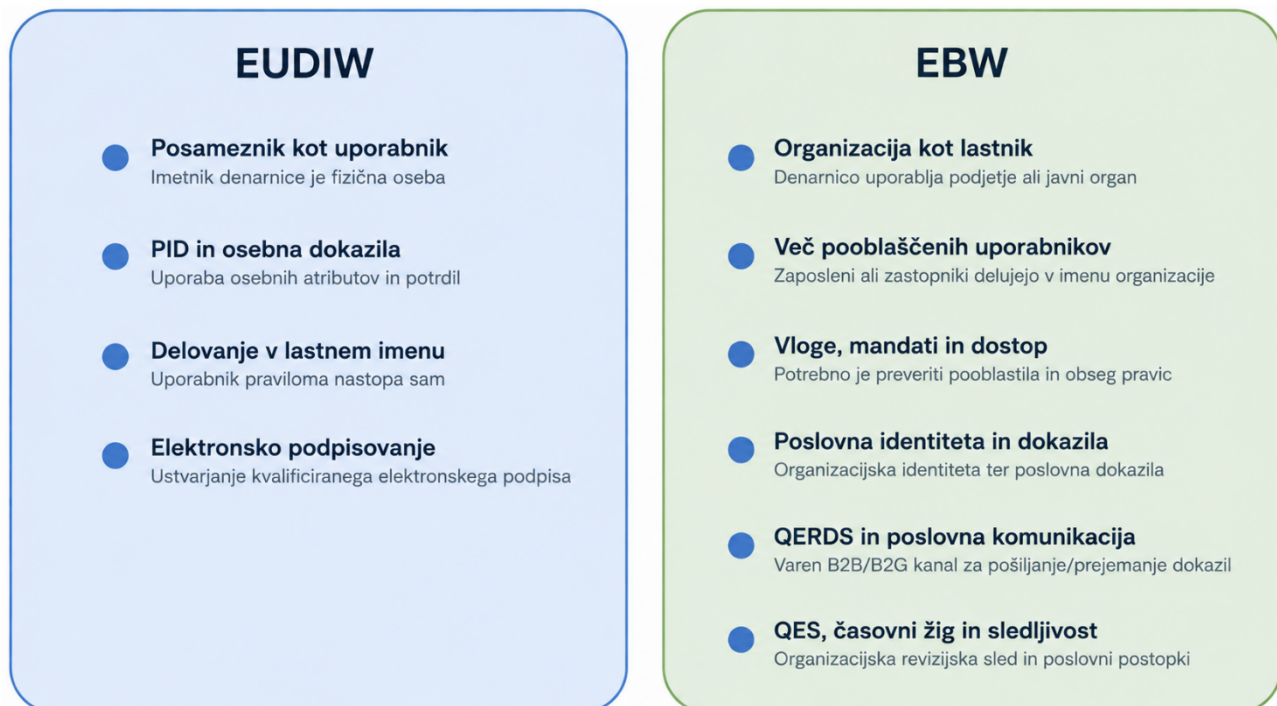
vozniška dovoljenja, elektronsko podpisovanje in zdravstvo. Nadaljnje preizkušanje poteka v projektih APTITUDE in WE BUILD, pri čemer je slednji osredotočen zlasti na poslovne in plačilne scenarije B2B, B2G in B2C [25, 26].

2.3. EBW – poslovna denarnica za pravne osebe

Lastnik EBW je gospodarski subjekt ali organ javnega sektorja, denarnico pa v njegovem imenu uporabljajo pooblašteni uporabniki oziroma **zastopniki**. To so lahko zakoniti zastopnik, zaposleni, zunanji računovodja ali drug izvajalec, ki mu je lastnik dodelil ustrezne pravice. Ključna razlika v primerjavi z EUDI denarnico je zato podpora več uporabnikom ter upravljanje njihovih vlog in pooblastil [13]. Posameznemu uporabniku je mogoče dovoliti na primer vpogled v dokumente, predložitev poverilnic, ali podpisovanje, njegove pravice pa pozneje omejiti ali preklicati. Vsaka izvedena operacija mora biti sledljivo povezana z organizacijo, uporabnikom in uporabljeno avtorizacijo. Pri tem je treba razlikovati med tehnično avtorizacijo za uporabo funkcij denarnice in pravnim pooblastilom, ki ga lahko za posamezno dejanje zahteva evropska ali nacionalna zakonodaja.

Osrednji element poslovne identitete so **identifikacijski podatki lastnika EBW** (angl. European Business Wallet owner identification data, **EBW-OID**) [13]. Ti podatki morajo vsebovati najmanj uradno ime gospodarskega subjekta oziroma organa javnega sektorja in njegov enolični identifikator. Kadar je gospodarskemu subjektu že dodeljen evropski enolični identifikator (angl. European Unique Identifier, EUID), se kot identifikator uporabi ta. EBW-OID identificira organizacijo, ne pa osebe, ki v njenem imenu izvede dejanje. Preveritelj mora zato pri posamezni poslovni operaciji ločeno preveriti identiteto lastnika denarnice, identiteto uporabnika in njegovo upravičenje za konkretno dejanje.

Ključne razlike med osebno uporabo EUDI denarnice in predlaganim modelom EBW povzema slika 3. Pri EUDI denarnici je imetnik fizična oseba, ki uporablja PID in druge osebne poverilnice, praviloma deluje v svojem imenu ter lahko ustvarja kvalificirane elektronske podpise. Pri EBW je lastnik organizacija, v njenem imenu pa lahko deluje več pooblaščenih uporabnikov. Poslovna denarnica mora zato poleg identitete organizacije podpirati tudi poslovne poverilnice, preverjanje vlog in mandatov, omejevanje dostopa, pravno zanesljivo poslovno komunikacijo ter sledljivost izvedenih dejanj.



Slika 3: Primerjava EUDIW in EBW.

Funkcije, povzete na sliki 3, se v predlaganem modelu povezujejo v celovit poslovni postopek. Predlog med drugim predvideva varno izmenjavo podatkov med EBW, EUDIW in zanašajočimi se strankami ter uporabo elektronskih podpisov, časovnih žigov in kvalificirane storitve elektronske priporočene dostave (angl. Qualified Electronic Registered Delivery Service, QERDS) [13]. Kot konkreten primer uporabe lahko obravnavamo vključitev novega čezmejnega dobavitelja. Njegov pooblaščen uporabnik bi lahko posredoval EBW-OID, zahtevana dovoljenja ali potrdila in podatke o svojem pooblastilu. Naročnik bi preveril identiteto dobavitelja in upravičenost zastopnika, nato pa bi pooblaščen oseba pogodbo elektronsko podpisala oziroma bi jo organizacija opremila z elektronskim žigom in časovnim žigom. Izmenjava dokumentacije bi lahko prek QERDS potekala tako pri poslovanju med podjetji (angl. business-to-business, B2B) kot pri postopkih med podjetjem in javnim organom (angl. business-to-government, B2G), izvedena dejanja pa bi bila zabeležena v organizacijski revizijski sledi.

Takšne scenarije obravnava tudi evropski pilotni projekt WE BUILD, ki povezuje več kot 200 javnih in zasebnih organizacij iz 28 držav ter preizkuša uporabo evropskih digitalnih denarnic v interakcijah med podjetji (angl. business-to-business, B2B), med podjetji in javnimi organi (angl. business-to-government, B2G) ter med podjetji in potrošniki (angl. business-to-consumer, B2C). Med obravnavanimi primeri uporabe so preverjanje strank (angl. Know Your Customer, KYC), dobaviteljev (angl. Know Your Supplier, KYS) in poslovnih subjektov (angl. Know Your Business, KYB), digitalno zastopanje podjetja, elektronsko izdajanje računov ter poslovna in potrošniška plačila. V projektu sodelujejo javni organi, poslovni registri, banke, ponudniki denarnic, ponudniki storitev zaupanja in velika tehnološka podjetja. Področje plačil vodi Mastercard, posamezne primere uporabe pa med drugim vodita Bundesanzeiger in OpenPeppol, medtem ko Signicat vodi skupino za testiranje. Med vidnejšimi sodelujočimi iz finančnega in tehnološkega sektorja je tudi Visa Europe [26].

3 Arhitektura

Tehnična arhitektura ekosistema ločuje poslovni pomen digitalne poverilnice, njen zapis in kriptografsko zaščito, protokol izmenjave ter konkretni uporabniški tok. Takšna razdelitev je pomembna, ker isti tip poverilnice lahko uporablja več formatov in načinov predložitve, isti protokol pa lahko podpira različne vrste poverilnic. V nadaljevanju zato najprej predstavimo ključne artefakte, nato protokole in nazadnje značilne poteke izdaje ter preverjanja.

3.1 Artefakti

V ekosistemu evropskih denarnic se podatki ne izmenjujejo kot poljubni dokumenti, temveč kot standardizirani in kriptografsko zaščiteni artefakti. Pri tem je koristno ločiti vsebino poverilnice od njenega tehničnega zapisa: PID, mobilno vozniško dovoljenje ali poverilnica o starosti opredeljujejo, kaj dokazujemo, medtem ko SD-JWT VC in mdoc določata, kako so podatki strukturirani, digitalno podpisani, selektivno razkriti in preverjeni.

Med ključnimi tehničnimi formati sta SD-JWT VC in mdoc. Mehanizem SD-JWT za selektivno razkrivanje trditev v JWT je standardiziran v RFC 9901, medtem ko je profil digitalnih poverilnic SD-JWT VC ob pripravi prispevka še aktiven osnutek IETF [27, 28]. Format mdoc uporablja CBOR in COSE ter podpira tudi vezavo poverilnice na napravo. ISO/IEC 18013-5 opredeljuje mobilno vozniško dovoljenje (angl. mobile Driving Licence, mDL), zapisano v formatu mdoc, in njegovo izmenjavo z bralnikom v neposredni bližini. ISO/IEC TS 18013-7 to dopolnjuje s predložitvijo mDL na daljavo prek interneta [29, 30]. SD-JWT VC in mdoc sta torej tehnična formata, mDL pa konkreten tip digitalne poverilnice.

Za posamezne vrste digitalnih poverilnic splošni tehnični format in protokol sama po sebi nista dovolj. Njihovo vsebino in uporabo dodatno opredeljujejo namenska pravila (angl. attestation rulebooks). Ta določajo strukturo in semantiko atributov, sheme ter njihovo preslikavo v podprte tehnične formate, zahteve glede zaupanja in ravni zagotovila ter postopke izdaje, preverjanja in preklica. V trenutni dokumentaciji ARF sta kot samostojna dokumenta navedena pravilnik za PID (PID Rulebook) in pravilnik za mobilno vozniško dovoljenje (mDL

Rulebook). Enotna pravila omogočajo, da izdajatelj, EUDI denarnice in preveritelji v različnih državah isto vrsto poverilnice razumejo in obdelajo na interoperabilen način [9].

Podatki PID so temeljna identifikacijska poverilnica EUDI denarnice. Vsebujejo jedrne podatke iz uradnega oziroma avtentičnega vira, s katerimi je mogoče zanesljivo identificirati fizično osebo. PID se lahko izda v formatu SD-JWT VC ali mdoc, kar omogoča uporabo istega poslovnega nabora podatkov v različnih načinih predložitve [7, 9]. Poslovni ekvivalent PID so identifikacijski podatki lastnika poslovne denarnice oziroma EBW-OID. Gre za preverljiv nabor podatkov o organizaciji, ki se izda kot elektronsko potrdilo atributov v enem od podprtih formatov [13].

Poseben primer namenske poverilnice je evropska rešitev za preverjanje starosti (angl. Age Verification), zasnovana kot prehodna, z EUDI denarnico združljiva rešitev. Uporabniku omogoča, da brez razkritja identitete ali datuma rojstva predloži samo zahtevano informacijo, na primer dokaz, da je starejši od 18 let. Primer ponazarja načelo najmanjšega obsega podatkov: za izvedbo storitve pogosto ni potreben celoten PID, temveč le minimalni preverjeni atribut [31].

3.2. Protokoli

Za izdajo digitalnih poverilnic se uporablja OpenID for Verifiable Credential Issuance (OpenID4VCI), ki določa z OAuth 2.0 zaščiten komunikacijo med izdajateljem in denarnico [32, 33]. Podpira avtorizacijski kodni tok, pri katerem se uporabnik overi med postopkom izdaje, ter predavtorizirani kodni tok, kadar ga je izdajatelj identificiral že prej. Denarnica pridobi metapodatke izdajatelja in ponudbo poverilnice, nato pridobi dostopni žeton ter z dokazom o posedovanju ustreznega ključa zahteva izdajo.

Predložitev digitalnih poverilnic ureja OpenID for Verifiable Presentations (OpenID4VP) [34]. Preveritelj z jezikom Digital Credentials Query Language (DCQL) opredeli zahtevane vrste poverilnic in attribute, denarnica pa preveri zahtevek ter uporabniku prikaže podatke, ki naj bi jih razkril. Odgovor je vezan na konkretnega preveritelja in sejo, s čimer se zmanjšuje možnost ponovne uporabe prestreženega odgovora v drugem kontekstu.

Pri poverilnicah v formatu mdoc standard ISO/IEC 18013-5 določa predvsem preverjanje ob fizični prisotnosti (angl. proximity verification), pri katerem denarnica in bralnik vzpostavita neposreden, šifriran komunikacijski kanal [29]. ISO/IEC TS 18013-7 opredeljuje oddaljeno predložitev mdoc prek interneta [30]. Profil OpenID4VC High Assurance Interoperability Profile (HAIP) dodatno omeji možnosti OpenID4VCI, OpenID4VP, SD-JWT VC in mdoc na skupen nabor varnih nastavitev, s čimer izboljšuje interoperabilnost implementacij z višjimi zahtevami glede zagotovila [35].

Za EBW končni protokolni profil še ni določen. Predlagani okvir zahteva skupne vmesnike za izdajo, zahtevanje, preverjanje in selektivno predložitev poverilnic ter avtomatizirano komunikacijo med poslovnimi denarnicami in EUDI denarnicami [13]. Iz teh zahtev je mogoče sklepati, da bo smiselna ponovna uporaba dela jedra OpenID4VCI in OpenID4VP, dopolnjenega z organizacijsko identiteto, preverjanjem pooblastil, elektronskimi žigi in komunikacijo med informacijskimi sistemi brez neposrednega sodelovanja uporabnika. Končna rešitev bo odvisna od sprejetega besedila uredbe, izvedbenih aktov in tehničnih specifikacij.

3.3. Tokovi

V nadaljevanju so povzeti štirje značilni poteki uporabe denarnic: (1) izdaja poverilnice, (2) preverjanje na daljavo, (3) preverjanje ob fizični prisotnosti ter (4) poslovna izmenjava s poslovno denarnico.

Postopek izdaje digitalne poverilnice se začne s ponudbo izdajatelja ali zahtevo, ki jo sproži denarnica [32]. Denarnica pridobi metapodatke izdajatelja, uporabnik pa se po potrebi prijavi oziroma vnese predhodno odobreno kodo. Nato denarnica pridobi dostopni žeton in dokaže nadzor nad pripadajočim zasebnim ključem. Izdajatelj pripravi in podpiše poverilnico ter jo posreduje denarnici, uporabnik pa potrdi njeno shranitev.

Pri preverjanju na daljavo ponudnik storitve pripravi zahtevek, ki vključuje namen uporabe in najmanjši potrebni nabor podatkov. Kadar postopek poteka na isti napravi, se zahtevek odpre prek povezave, pri uporabi dveh naprav pa se prikaže kot QR-koda. EUDI denarnica preveri tehnično veljavnost zahtevka in podatke o zanašajoči se stranki [22], uporabniku prikaže zahtevane podatke ter po njegovi potrditvi pripravi odgovor. Preveritelj nato preveri podpis izdajatelja, veljavnost in status poverilnice, svežost odgovora ter vezanost poverilnice na denarnico oziroma imetnika [34].

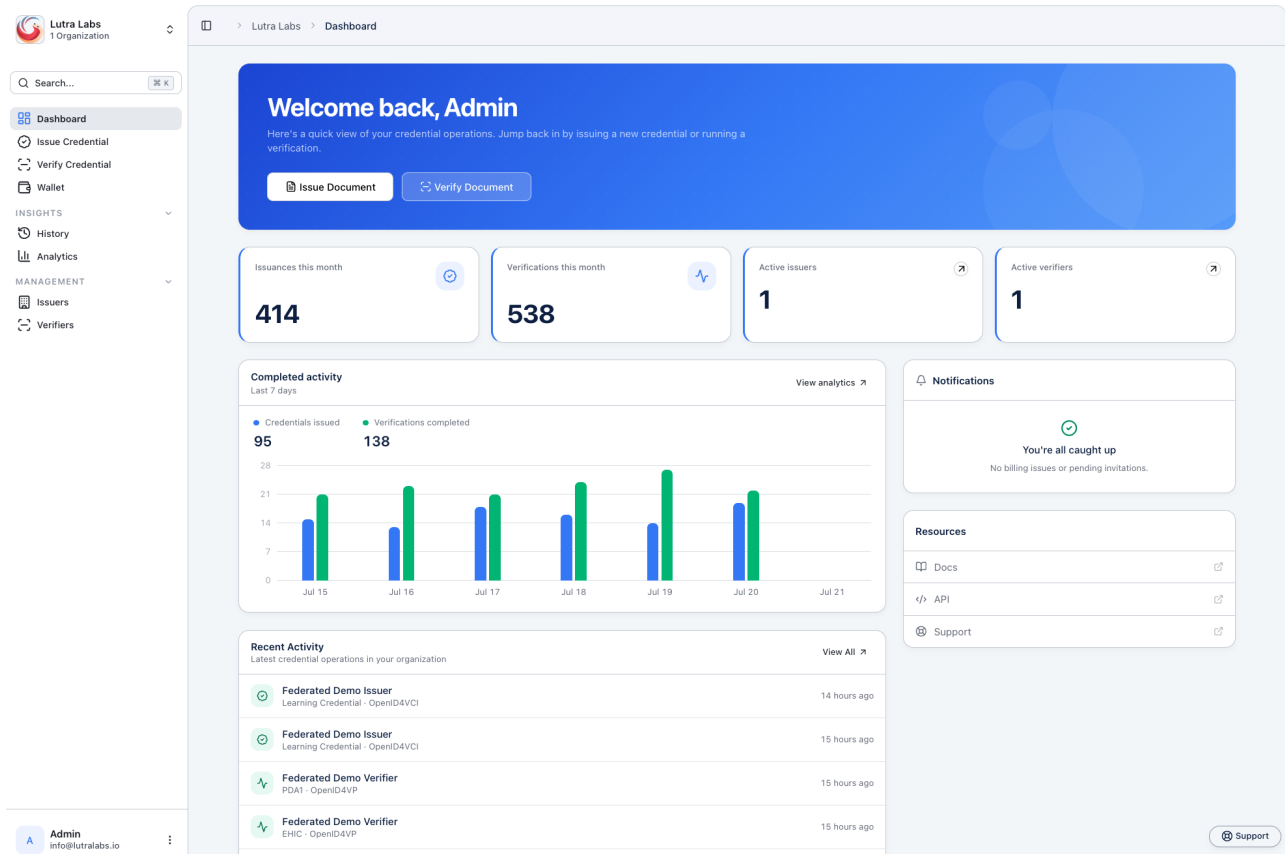
Pri preverjanju ob fizični prisotnosti uporabnika se postopek začne z odčitavanjem QR-kode ali uporabo tehnologije NFC. EUDI denarnica in bralnik nato vzpostavi šifrirano lokalno povezavo, prek katere bralnik zahteva izbrane podatke, uporabnik pa njihovo predložitev potrdi. Osnovno kriptografsko preverjanje je mogoče opraviti brez neposredne povezave z izdajateljem, kar je pomembno predvsem na mestih z omejeno povezljivostjo [29].

Poslovni poteki poleg preverjanja digitalne poverilnice zahtevajo tudi preverjanje identitete organizacije, vloge uporabnika in njegovega upravičenja za konkretno dejanje [13]. Postopek lahko sproži pooblaščen oseba ali zaledni informacijski sistem. Pri vključevanju poslovnega partnerja si poslovni denarnici lahko izmenjata zahtevane podatke o organizaciji in poverilnice o zastopanju. Pri postopkih pred javnimi organi lahko poslovna denarnica posreduje poverilnice in elektronsko vlogo, ki jo pooblaščen oseba po potrebi podpiše oziroma organizacija opremi z elektronskim žigom. Organ lahko po istem varnem kanalu pošlje potrdilo o prejemu, poziv ali odločbo. Če v imenu organizacije nastopa fizična oseba, se njena identiteta lahko dokaže z EUDI denarnico, identiteta organizacije z EBW-OID, upravičenje fizične osebe pa z digitalno poverilnico o organizacijski vlogi, zastopanju ali pooblastilu. Pred samodejno izmenjavo med poslovnima denarnicama je treba preveriti politiko dostopa, identiteto druge organizacije ter veljavnost elektronskega žiga oziroma podpisa. Rutinske izmenjave, na primer elektronskih računov ali podatkov v oskrbovalni verigi, lahko sproži zaledni sistem. Pri občutljivejših dejanjih se lahko zahteva dodatna potrditev pooblaščen osebe. Pomembni koraki se zabeležijo v revizijski sledi [13, 26].

4 Od okvira k praksi: vključevanje organizacij in referenčna implementacija

Organizacije se lahko v ekosistem EUDI denarnice vključijo na več načinov. Nastopajo lahko kot izdajatelj digitalnih poverilnic, na primer licenc, dovoljenj, članstev, poslovnih atributov ali rezultatov preverjanj strank (KYC) in poslovnih subjektov (KYB); kot zanašajoče se stranke, ki za izvedbo storitve zahtevajo preverjene podatke; ali kot ponudniki podpornih platform in infrastrukture [9]. Posamezna organizacija lahko hkrati združuje več vlog, vendar mora za vsako jasno določiti pravni namen, zahtevane podatke, model zaupanja in odgovornosti [22].

Za praktični prikaz predstavljenih konceptov bomo predstavili platformo LutraID, katere uporabniški vmesnik je prikazan na sliki 4. Gre za strežniško rešitev, ki organizacijam omogoča, da izdajanje in preverjanje digitalnih poverilnic vključijo v svoje informacijske sisteme, ne da bi morale v vsaki poslovni aplikaciji znova implementirati protokolne tokove, kriptografske operacije in upravljanje profilov. Platforma ponuja spletni upravljalni vmesnik za nastavitve organizacije, izdajateljev in preveriteljev, vrst poverilnic ter profilov izdajanja in preverjanja. Izdajo in preverjanje je mogoče sprožiti neposredno prek uporabniškega vmesnika ali iz obstoječega sistema prek programskega vmesnika REST. Na ravni organizacije je podprto tudi upravljanje članov in njihovih vlog. Skrbnik organizacije ima dostop do vseh nastavitev, operatorske vloge pa uporabniški vmesnik omeji na ključna vsakodnevna opravila, kot sta izdajanje in preverjanje digitalnih poverilnic.

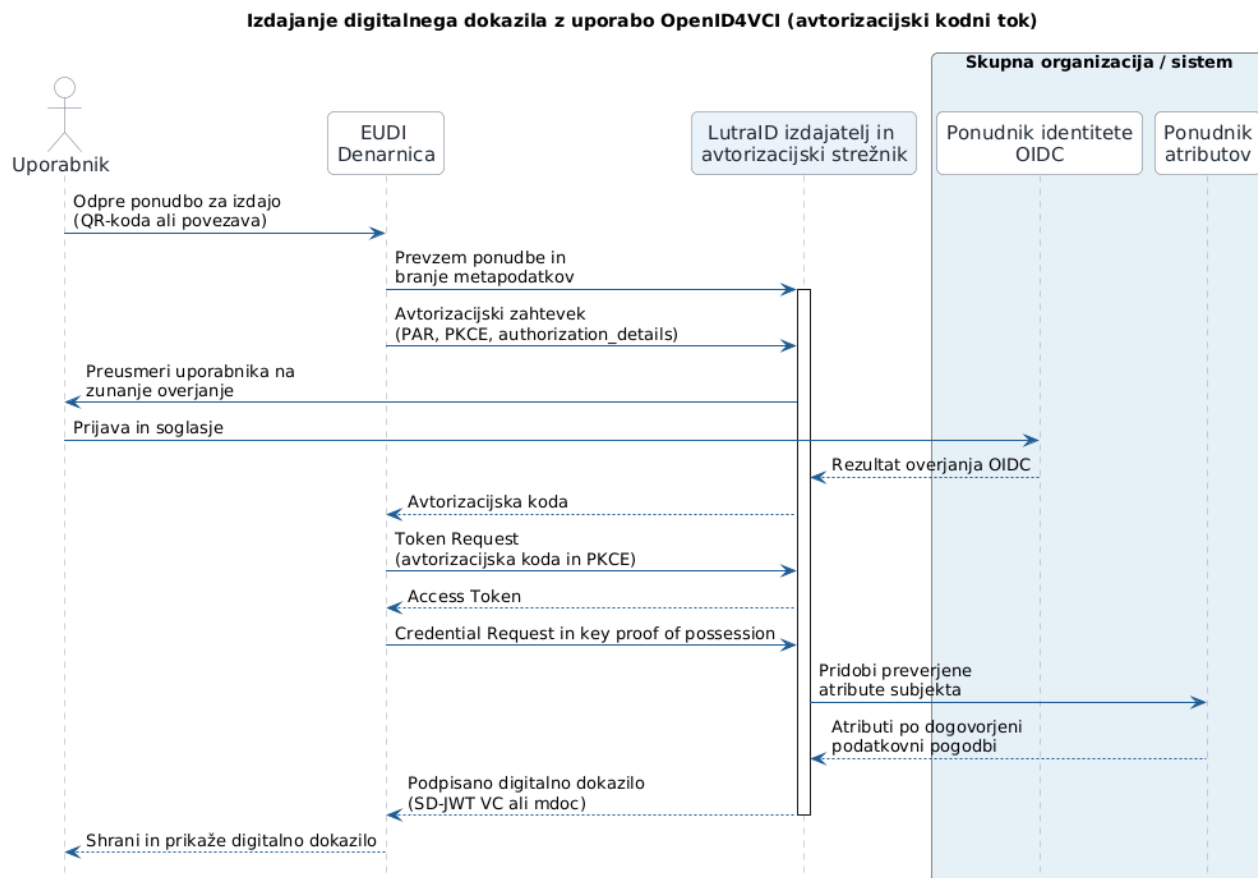


Slika 4: Uporabniški vmesnik platforme LutraID

Platforma je zasnovana kot domensko organiziran modularni monolit, dopolnjen z ločenimi storitvami za obdelavo opravil v ozadju. Funkcionalna področja so ločena na ravni izvorne kode, zaledna aplikacija pa se namešča in izvaja kot celota ter uporablja skupno podatkovno bazo. Takšna zasnova poenostavlja razvoj in upravljanje, hkrati pa omogoča poznejšo izločitev posameznih modulov v samostojne storitve. Uporabniški vmesnik temelji na ogrodju Next.js, zaledni del pa na ogrodju Hono. Za avtentikacijo in upravljanje uporabnikov se uporablja Better Auth, podpora za enotno prijavo (SSO) in protokol System for Cross-domain Identity Management (SCIM) pa organizacijam omogoča povezavo z obstoječimi sistemi za upravljanje identitet in dostopov. Kriptografske operacije se lahko izvajajo prek storitve AWS KMS, pri čemer zasebni ključi ostanejo zaščiteni v strojnem varnostnem modulu (angl. Hardware Security Module, HSM). Za varno hrambo in upravljanje aplikacijskih skrivnosti se uporablja HashiCorp Vault.

Pred izdajo skrbnik organizacije ustvari izdajatelja in mu dodeli ustrezen profil izdajanja. Uporabi lahko enega od vnaprej pripravljenih profilov, na primer za PID ali mobilno vozniško dovoljenje (mDL), oziroma pripravi lasten profil za drugo vrsto poverilnice. Operater nato izbere profil, vnese podatke prejemnika in sproži postopek, platforma pa pripravi ponudbo za prevzem v obliki povezave ali QR-kode. Potek izdaje digitalne poverilnice v platformi LutraID je prikazan na sliki 5. Uporabnik ponudbo odpre v EUDI denarnici, ki nato pridobi njeno vsebino in metapodatke izdajatelja ter začne avtorizacijski kodni tok. LutraID uporabnika preusmeri k zunanjemu ponudniku identitete OIDC (angl. OpenID Connect), kjer se uporabnik overi in po potrebi poda soglasje. Po uspešnem overjanju denarnica prejme avtorizacijsko kodo, jo skupaj z dokazom PKCE (angl. Proof Key for Code Exchange) zamenja za dostopni žeton ter pošlje zahtevo za izdajo z dokazom o posedovanju pripadajočega zasebnega ključa. LutraID od ponudnika atributov pridobi preverjene podatke o prejemniku, pripravi in kriptografsko podpiše digitalno poverilnico v formatu SD-JWT VC ali mdoc ter jo posreduje denarnici. Denarnica poverilnico shrani in jo prikaže uporabniku. V uporabniškem vmesniku je mogoče spremljati trenutno stanje postopka in pregledati zgodovino izvedenih korakov. Enak postopek je mogoče sprožiti prek programskega

vmesnika REST, kar omogoča, da podatke o prejemniku zagotovi obstoječi poslovni sistem, LutraID pa izvede protokolni in kriptografski del izdaje.



Slika 5: Izdajanje digitalne poverilnice v platformi LutraID prek OpenID4VCI z avtorizacijskim kodnim tokom

Pri preverjanju skrbnik pripravi profil preverjanja in določi zahtevane vrste poverilnic ter attribute. Operater nato sproži zahtevo, ki jo uporabnik odpre v EUDI denarnici in potrdi. Po uspešnem preverjanju platforma prikaže strukturiran rezultat ali ga prek programskega vmesnika posreduje povezanemu poslovnemu sistemu. Poslovna aplikacija tako prejme le rezultat preverjanja in dovoljene podatke, ne pa nujno celotne digitalne poverilnice.

Isto arhitekturo je mogoče v prihodnje razširiti za podporo poslovnim denarnicam, predvsem z upravljanjem organizacijskih identitet, več uporabnikov, zastopnikov, vlog in pooblastil, elektronskih žigov ter komunikacije med denarnicami in zalednimi informacijskimi sistemi. Jedro za izdajanje, preverjanje in povezovanje s poslovnimi sistemi lahko pri tem v veliki meri ostane nespremenjeno, dopolniti pa ga bo treba z zahtevami, ki bodo določene v končnem pravnem in tehničnem okviru [13].

Čeprav se standardi še razvijajo, se lahko organizacije na uvedbo pripravijo že zdaj. Smiselno je popisati postopke, v katerih danes ročno zbirajo ali preverjajo identifikacijske podatke, določiti, ali bodo nastopale kot izdajatelj ali zanašajoča se stranka, ter opredeliti najmanjši potreben nabor atributov, pravne podlage, vire zaupanja, upravljanje ključev in certifikatov, zahteve revizijske sledi ter integracijske točke. Takšna priprava ostane uporabna tudi ob poznejših spremembah posameznih protokolnih profilov.

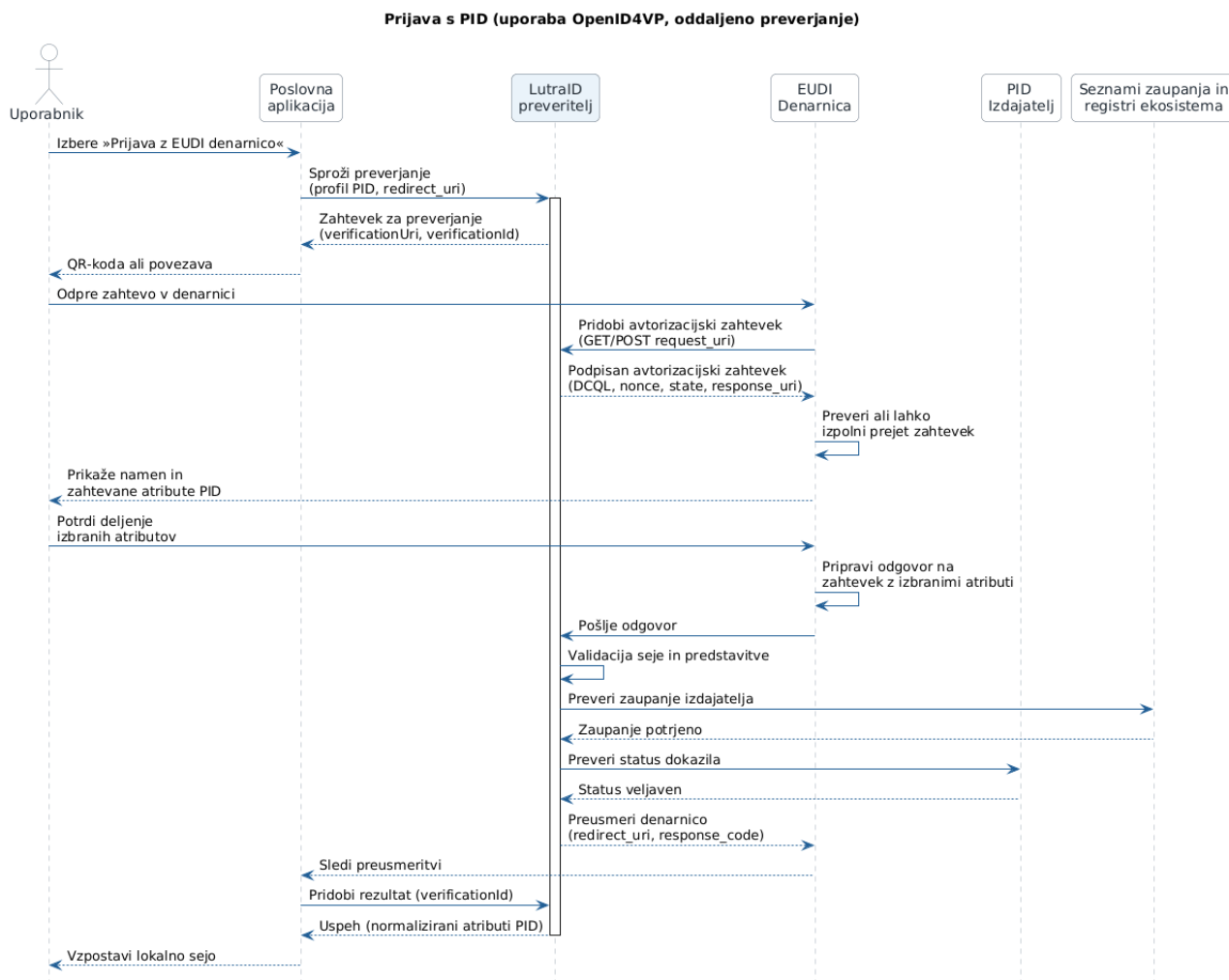
5 Primer uporabe: prijava z EUDI denarnico

Kot osnovni primer uporabe obravnavamo prijavo v spletno storitev s preverjanjem PID, katere začetni korak je prikazan na sliki 6. Namesto uporabniškega imena, gesla in ročnega prepisovanja podatkov ponudnik storitve uporabniku ponudi možnost »Prijava z EUDI denarnico«. V tem postopku poslovna aplikacija nastopa kot *zanašajoča se stranka*, LutraID pa kot *tehnični preveritelj*, ki v njenem imenu pripravi zahtevo OpenID4VP in preveri prejeti odgovor.

The screenshot displays the GovHub login page. On the left, a large heading reads 'Javne storitve, ena varna prijava.' (Public services, one safe login.) Below it, text explains that users can access GovHub services with their European digital wallet for digital identity, proving their identity in seconds without new passwords or forms. A list of benefits includes: 'Ena identiteta, priznana v vseh državah članicah EU' (One identity, recognized in all EU member states), 'Selektivno razkritje – delite samo attribute, ki jih storitev potrebuje' (Selective disclosure – share only the attributes the service needs), and 'Raven zanesljivosti »visoka«, kriptografsko podpisana s strani vaše države' (High level of reliability, cryptographically signed by your country). On the right, the 'Prijava' (Login) section shows a 'GovHub Portal javnih storitev' header with a 'DEMO' button. The login form includes fields for 'Uporabniško ime' (Username) with the value 'vi@primer.eu' and 'Geslo' (Password). A note states: 'Prijava z geslom je ukinjena – nadomešča jo evropska denarnica za digitalno identiteto.' (Password login is discontinued – replaced by the European digital wallet for digital identity). A large blue button labeled 'Prijava z EUDI' is prominent. Below, the 'ZAHTEVANI ATRIBUTI' (Required attributes) section lists: 'Ime in datum rojstva' (Name and date of birth) as basic identification data, and 'Državljanstvo in starost nad 18 let' (Citizenship and age over 18) as verified by the state. A footer note mentions compliance with eIDAS 2.0 and EU Regulation 2024/1183. At the bottom, a link asks: 'Še nimate PID? Pridobite predstavitvenega →' (Don't have a PID? Get a demo one →).

Slika 6: Uporabniški vmesnik spletne storitve z možnostjo prijave z EUDI denarnico.

Potek prijave s PID je prikazan na sliki 7. Ko uporabnik izbere prijavo z EUDI denarnico, poslovna aplikacija v platformi LutraID sproži preverjanje z izbranim profilom PID. LutraID pripravi identifikator postopka in zahtevek za preverjanje, ki ga poslovna aplikacija uporabniku prikaže kot povezavo ali QR-kodo. EUDI denarnica nato pridobi podpisani avtorizacijski zahtevek, ki vsebuje zahtevane attribute, namen uporabe ter vrednosti za vezavo odgovora na konkretnega preveritelja in sejo. Denarnica preveri, ali lahko zahtevo izpolni, uporabniku prikaže podatke o zanašajoči se stranki, namen in zahtevane attribute ter po njegovi potrditvi pripravi in pošlje odgovor. LutraID preveri veljavnost seje in prejete predstavitve, zaupanje v izdajatelja, kriptografske podpise, status poverilnice ter njeno vezavo na denarnico oziroma imetnika. Po uspešni validaciji denarnico preusmeri nazaj v poslovno aplikacijo, ta pa z identifikatorjem postopka pridobi strukturiran rezultat in normalizirane attribute PID. Poslovna aplikacija nato vzpostavi lokalno uporabniško sejo ali prejete podatke uporabi za povezavo z obstoječim računom.



Slika 7: Prijava s PID prek OpenID4VP pri oddaljenem preverjanju v platformi LutraID

Enak način preverjanja je mogoče uporabiti tudi pri zahtevnejših postopkih, v katerih mora ponudnik storitve pridobiti več preverjenih podatkov ali povezati podatke iz več digitalnih poverilnic. Pri odpiranju bančnega računa lahko banka od uporabnika zahteva PID ter po potrebi dodatne poverilnice o naslovu prebivališča, davčnem rezidentstvu ali drugih podatkih, potrebnih za izvedbo postopka KYC. Pri registraciji eSIM mobilni operater s PID preveri identiteto uporabnika in druge zakonsko zahtevane attribute, preverjene podatke pa lahko neposredno uporabi v postopku sklenitve pogodbe in aktivacije storitve [25]. Pri najemu vozila ponudnik zahteva kombinirano predstavitev PID in mDL, na podlagi katere preveri identiteto uporabnika, veljavnost voznškega dovoljenja in ustrezno kategorijo vozila [29]. Uporabnik pri tem razkrije samo podatke, potrebne za posamezno storitev. Postopek se lahko izvede na daljavo pred aktivacijo storitve oziroma prevzemom vozila ali neposredno na poslovalnici oziroma mestu prevzema [30].

6 Razprava in sklep

V prispevku smo predstavili evropsko denarnico za digitalno identiteto in evropsko poslovno denarnico. EUDI denarnica je namenjena predvsem fizičnim osebam ter jim omogoča hrambo in uporabo osebnih identifikacijskih podatkov in drugih digitalnih poverilnic. EBW je namenjena pravnim osebam, samozaposlenim in javnim organom, pri čemer denarnico v imenu organizacije uporabljajo pooblaščen uporabniki oziroma zastopniki. Obe rešitvi sledita cilju vzpostavitve interoperabilnega evropskega ekosistema za varno čezmejno identifikacijo in izmenjavo preverjenih podatkov. Opisali smo njune vloge, tehnično arhitekturo, uporabljene formate in protokole

ter postopke izdajanja in preverjanja digitalnih poverilnic, praktično uporabo pa smo prikazali z referenčno implementacijo platforme LutraID in primerom prijave v spletno storitev z EUDI denarnico.

Okvir EUDI denarnic je že precej zrel, saj so pravne zahteve, vloge udeležencev, tehnična arhitektura ter osnovni tokovi izdajanja in preverjanja v večini opredeljeni v izvedbenih uredbah, specifikaciji ARF in specifikacijah OpenID4VC. Prvi val velikih evropskih pilotnih projektov, ki je potekal do leta 2025, je rešitve preizkušal na področjih elektronske uprave, bančništva, telekomunikacij, mobilnih vozniških dovoljenj, elektronskega podpisovanja, zdravstva, izobraževanja in plačil [25]. Države članice morajo najmanj eno EUDI denarnico zagotoviti do konca leta 2026, zato se okvir in njegova uporaba že približujeta širši produkcijski uvedbi [6, 24].

Po drugi strani je stopnja zrelosti EBW nižja, saj je Evropska komisija predlog uredbe predstavila novembra 2025, Svet EU pa je svoje pogajalsko stališče sprejel junija 2026 [12, 14]. Končno besedilo uredbe v času priprave prispevka še ni sprejeto, države članice pa si prizadevajo doseči končni dogovor do konca leta 2026 [14]. Po sprejetju bo treba podrobneje opredeliti izvedbene akte, tehnično arhitekturo, protokole, formate in model zaupanja za poslovne denarnice. Pomembno vlogo ima projekt WE BUILD, ki se je začel septembra 2025 in bo predvidoma potekal 24 mesecev, do leta 2027 [26]. Projekt razvija in preizkuša primere uporabe na področjih poslovanja, dobavnih verig in plačil, vključno z identifikacijo organizacij, pravnim zastopanjem in izmenjavo podatkov. Njegovi rezultati lahko tako prispevajo k nadaljnjemu oblikovanju tehničnih zahtev in praktični uvedbi EBW ekosistema.

V prihodnje bo zanimivo spremljati nadaljnji razvoj EUDI denarnic, zlasti vključevanje dokazov brez razkritja znanja (angl. zero-knowledge proofs, ZKP), ki omogočajo dokazovanje trditev o podatkih brez razkritja njihovih dejanskih vrednosti ter s tem zagotavljajo večjo zasebnost uporabnikov. Od začetka leta 2027 bo pomembno tudi, kako hitro se bodo EUDI denarnice uveljavile v praksi in kakšna bo njihova razširjenost med državami članicami, ponudniki storitev in uporabniki. Pri EBW bo nadaljnji razvoj odvisen predvsem od sprejetja končne uredbe, izvedbenih aktov in tehničnih specifikacij ter rezultatov projekta WE BUILD. Pomembno bo tudi spremljati dejansko uvedbo poslovnih denarnic v državah članicah in možnosti njihove interoperabilnosti s primerljivimi rešitvami zunaj EU.

Literatura

- [1] F. Alaca in P. C. van Oorschot, »Comparative Analysis and Framework Evaluating Web Single Sign-on Systems«, ACM Computing Surveys, let. 53, št. 5, čl. 112, str. 1–34, 2020. <https://doi.org/10.1145/3409452>
- [2] Apple, »Security of IDs in Apple Wallet«, Apple Platform Security, 7. 5. 2024, obiskano 21. 7. 2026. <https://support.apple.com/guide/security/security-of-ids-in-apple-wallet-secb569bf393/web>
- [3] Google, »Identity«, Google Wallet, obiskano 21. 7. 2026. <https://developers.google.com/wallet/identity>
- [4] Evropska komisija, »SWD(2021) 124 final: Impact Assessment Report accompanying the Proposal for a Regulation amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity«, 3. 6. 2021. <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021SC0124>
- [5] Evropski parlament in Svet Evropske unije, »Uredba (EU) št. 910/2014 z dne 23. julija 2014 o elektronski identifikaciji in storitvah zaupanja za elektronske transakcije na notranjem trgu ter razveljavitvi Direktive 1999/93/ES«, 2014. <https://eur-lex.europa.eu/eli/reg/2014/910/oj>
- [6] Evropski parlament in Svet Evropske unije, »Uredba (EU) 2024/1183 z dne 11. aprila 2024 o spremembi Uredbe (EU) št. 910/2014 v zvezi z vzpostavitvijo evropskega okvira za digitalno identiteto«, 2024. <https://eur-lex.europa.eu/eli/reg/2024/1183/oj>
- [7] Evropska komisija, »Izvedbena uredba Komisije (EU) 2024/2977 glede osebnih identifikacijskih podatkov in elektronskih potrdil atributov«, 2024. https://eur-lex.europa.eu/eli/reg_impl/2024/2977/oj
- [8] Evropska komisija, »Izvedbena uredba Komisije (EU) 2024/2979 glede celovitosti in jedrnih funkcionalnosti evropskih denarnic za digitalno identiteto«, 2024. https://eur-lex.europa.eu/eli/reg_impl/2024/2979/oj

- [9] European Digital Identity Cooperation Group, »European Digital Identity Wallet - Architecture and Reference Framework«, različica 2.9.0, 21. 5. 2026. <https://eudi.dev/latest/>
- [10] Evropska komisija, »Izvedbena uredba Komisije (EU) 2024/2982 glede protokolov in vmesnikov evropskega okvira za digitalno identiteto«, 2024. https://eur-lex.europa.eu/eli/reg_impl/2024/2982/oj
- [11] Evropska komisija, »Izvedbena uredba Komisije (EU) 2024/2981 glede certificiranja evropskih denarnic za digitalno identiteto«, 2024. https://eur-lex.europa.eu/eli/reg_impl/2024/2981/oj
- [12] Evropska komisija, »COM(2025) 838 final: Proposal for a Regulation on the establishment of European Business Wallets«, 19. 11. 2025. <https://eur-lex.europa.eu/legal-content/SL/TXT/?uri=CELEX:52025PC0838>
- [13] Svet Evropske unije, »Proposal for a Regulation on the establishment of European Business Wallets - General approach«, dokument ST 7659/26, 22. 5. 2026. <https://data.consilium.europa.eu/doc/document/ST-7659-2026-INIT/en/pdf>
- [14] Svet Evropske unije, »Evropske poslovne denarnice: Svet sprejel pogajalsko stališče«, 9. 6. 2026. <https://www.consilium.europa.eu/sl/press/press-releases/2026/06/09/european-business-wallets-council-adopts-negotiating-position/>
- [15] R. Soltani, U. T. Nguyen in A. An, »A Survey of Self-Sovereign Identity Ecosystem«, Security and Communication Networks, let. 2021, čl. 8873429, 26 str., 2021. <https://doi.org/10.1155/2021/8873429>
- [16] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros in C. Mortimore, »OpenID Connect Core 1.0 incorporating errata set 2«, OpenID Foundation, 15. 12. 2023. https://openid.net/specs/openid-connect-core-1_0.html
- [17] A. Preukschat in D. Reed, Self-Sovereign Identity: Decentralized Digital Identity and Verifiable Credentials, Manning, 2021. <https://www.manning.com/books/self-sovereign-identity>
- [18] Š. Čučko, Š. Bećirović, A. Kamišalić, S. Mrdović in M. Turkanović, »Towards the Classification of Self-Sovereign Identity Properties«, IEEE Access, let. 10, str. 88306–88329, 2022. <https://doi.org/10.1109/ACCESS.2022.3199414>
- [19] Š. Čučko in M. Turkanović, »Decentralized and Self-Sovereign Identity: Systematic Mapping Study«, IEEE Access, let. 9, str. 139009–139027, 2021. <https://doi.org/10.1109/ACCESS.2021.3117588>
- [20] Š. Čučko in M. Turkanović, »Umestitev digitalnih (EU) denarnic v ekosistem sodobnih IKT rešitev«, v OTS 2022: sodobne informacijske tehnologije in storitve: zbornik petindvajsete konference, Maribor, Univerza v Mariboru, Univerzitetna založba, 2022, str. 24–41. <https://dk.um.si/IzpisGradiva.php?id=82880>
- [21] M. Turkanović, »Tehnično ozadje digitalnih denarnic in verig blokov v okvirju evropske digitalne identitete in posodobljene Uredbe eIDAS«, predstavitev na 5. mednarodni konferenci o e-identifikaciji in e-storitvah zaupanja, Zeides 2021, 21. 10. 2021.
- [22] Evropska komisija, »Izvedbena uredba Komisije (EU) 2025/848 glede informacij za registracijo zanašajočih se strank pri EUDI denarnicah«, 2025. https://eur-lex.europa.eu/eli/reg_impl/2025/848/oj
- [23] M. Huš, »Prihaja evropska digitalna denarnica«, Monitor, 30. 12. 2025. <https://www.monitor.si/clanek/prihaja-evropska-digitalna-denarnica/245727/>
- [24] Evropska komisija, »European Digital Identity (EUDI) Regulation«, posodobljeno 22. 6. 2026, obiskano 21. 7. 2026. <https://digital-strategy.ec.europa.eu/en/policies/eudi-regulation>
- [25] Evropska komisija, »What are the Large Scale Pilot Projects«, posodobljeno 27. 11. 2025, obiskano 21. 7. 2026. <https://ec.europa.eu/digital-building-blocks/sites/spaces/EUDIGITALIDENTITYWALLET/pages/694487808/What+are+the+Large+Scale+Pilot+Projects>
- [26] WE BUILD Consortium, »Wallet Ecosystem for Business and payments Use cases on Identification, Legal representation and Data sharing«, obiskano 21. 7. 2026. <https://www.webuildconsortium.eu/>
- [27] D. Fett, K. Yasuda in B. Campbell, »RFC 9901: Selective Disclosure for JSON Web Tokens«, IETF, november 2025. <https://www.rfc-editor.org/rfc/rfc9901.html>
- [28] O. Terbu, D. Fett in B. Campbell, »SD-JWT-based Verifiable Digital Credentials (SD-JWT VC)«, draft-ietf-oauth-sd-jwt-vc-17, IETF, 6. 7. 2026. <https://datatracker.ietf.org/doc/draft-ietf-oauth-sd-jwt-vc/>
- [29] ISO, »ISO/IEC 18013-5:2021 - Personal identification - ISO-compliant driving licence - Part 5: Mobile driving licence application«, 2021. <https://www.iso.org/standard/69084.html>

- [30] ISO, »ISO/IEC TS 18013-7:2025 - Personal identification - ISO-compliant driving licence - Part 7: Mobile driving licence add-on functions«, 2025. <https://www.iso.org/standard/91154.html>
- [31] Evropska komisija, »The EU approach to age verification«, obiskano 21. 7. 2026. <https://digital-strategy.ec.europa.eu/en/policies/eu-age-verification>
- [32] OpenID Foundation, »OpenID for Verifiable Credential Issuance 1.0«, 16. 9. 2025. https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0.html
- [33] M. Domajnko, V. Keršič in M. Turkanović, »OID4VC: izdajanje in deljenje preverljivih poverilnic na osnovi OpenID Connect«, v OTS 2023: sodobne informacijske tehnologije in storitve: zbornik 26. konference, Maribor, Univerza v Mariboru, Univerzitetna založba, 2023, str. 149–163. <https://press.um.si/index.php/ump/catalog/book/804>
- [34] OpenID Foundation, »OpenID for Verifiable Presentations 1.0«, 9. 7. 2025. https://openid.net/specs/openid-4-verifiable-presentations-1_0.html
- [35] OpenID Foundation, »OpenID4VC High Assurance Interoperability Profile 1.0«, končna specifikacija, 24. 12. 2025. https://openid.net/specs/openid4vc-high-assurance-interoperability-profile-1_0-final.html

Semantična podatkovna plast: manjkajoči temelj analitike nove generacije

Jure Jeraj,¹ Stevanče Nikoloski^{1,2}

¹ DataQ d.o.o., Ljubljana, Slovenija

jure.jeraj@dataq.si, stevance.nikoloski@dataq.si

² Univerza v Novem mestu, Fakulteta za ekonomijo in informatiko, Novo mesto, Slovenija

Organizacije nimajo več le preveč podatkov, temveč pogosto tudi preveč dashboardov in poročil, med katerimi je težko prepoznati, kaj zares prinaša vrednost ali opozarja na problem. Prispevek obravnava semantično podatkovno plast kot skupno pomensko osnovo nad preverjenimi podatki: upravljani sloj metrik, dimenzij, pojmov, odnosov in pravil, ki ga lahko dosledno uporabljajo BI-orodja, aplikacije in sistemi umetne inteligence. Prikažemo razvoj od podatkovnega skladišča in analitičnih modelov do ponovno uporabne semantike ter pojasnimo razliko med zvezdno shemo, UML-modelom, ontologijo in grafom znanja. Referenčno arhitekturo ponazorimo s primerom analize prodajne marže. Na tej podlagi predstavimo podatkovnega asistenta, ki pregleda več analitičnih pogledov, izlušči pomemben signal in predlaga fokus, vendar odločitve ne izvede sam. Prispevek predstavlja praktičen arhitekturni okvir in ne empirične validacije konkretnega produkta.

Ključne besede:

semantična podatkovna plast

poslovna inteligenca

ontologija

graf znanja

podatkovni asistent

1 Ko preveč podatkov postane preveč dashboardov

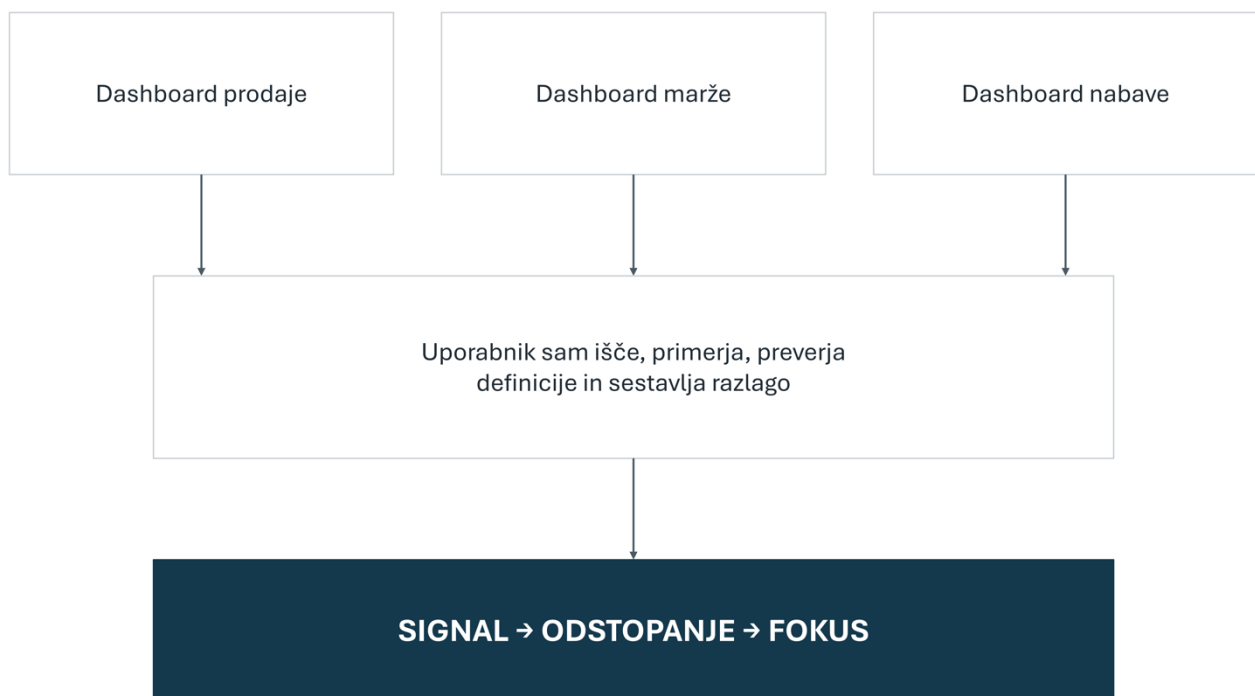
V zadnjih desetletjih so organizacije zgradile obsežne podatkovne in analitične ekosisteme, ki vključujejo podatkovna skladišča, podatkovna jezera, analitične modele, poročila in nadzorne plošče. Z razmahom samopostrežne analitike problem ni več le velika količina podatkov. Gartner opozarja na množenje dashboardov, njihovo omejeno uporabo in omejen poslovni učinek ter na podvajanje dashboardov, poročil in metrik v decentraliziranih analitičnih okoljih [1, 2].

Osnovna poslovna vprašanja se pri tem niso spremenila. Koliko je vredna dobra odločitev? Koliko nas stane napačna odločitev zaradi napačno razumljene metrike? Koliko izgubimo, če pomemben signal opazimo prepozno ali svojega dejanskega stanja sploh ne razumemo? Analitika ustvarja vrednost šele, ko uporabniku pomaga pravočasno prepoznati problem, razumeti njegov obseg in usmeriti nadaljnjo presojo.

Uporabnik zato pogosto nima težave z dostopom do podatkov, temveč z ugotavljanjem, kateri od številnih podatkov in analitičnih pogledov mu v danem trenutku prinašajo vrednost oziroma ga opozarjajo na problem. Dashboard je vnaprej pripravljen pogled na podatke. Ne ve pa, kateri od teh pogledov je trenutno najpomembnejši, in sam po sebi ne pove, koliko nas bo stalo, če pomemben signal spregledamo.

Naslednji korak analitike ni še en dashboard. Najprej potrebujemo skupen in upravljan pomen podatkov.

Podatki sami ne govorijo poslovnega jezika, poslovni uporabniki pa praviloma ne razmišljajo v tabelah, ključih in poizvedbah. Med obema svetovoma zato potrebujemo skupen jezik. Semantična podatkovna plast formalizira in poveže tisto, kar v organizacijah že obstaja: metrike, dimenzije, hierarhije, poslovne pojme, pravila, izjeme in odnose. Ko je ta pomen upravljan in ponovno uporaben tudi zunaj posameznega poročila ali BI-orodja, postane temelj analitike nove generacije.



Slika 1: Od množice analitičnih pogledov do poslovnega fokusa.

2 Semantika ni nova, vendar je postala pomembnejša

2.1. Od skupne podatkovne osnove do analitičnega modela

Podatkovno skladišče je organizacijam omogočilo, da iste očiščene in povezane podatke z ohranjeno zgodovino uporabljajo v več poročilih in analizah. Njegova vrednost ni le v tabelah, temveč v skupni podatkovni osnovi in disciplinah, kot so določanje granulacije, ločevanje dejstev in dimenzij ter dosledna uporaba ključev in zgodovine [3]. DWH zato ne postavljamo nasproti semantični plasti, temveč ga obravnavamo kot njen pomemben podatkovni temelj.

Tudi poslovna semantika v analitiki ni nova. Večdimenzionalne kocke, SQL Server Analysis Services in tabularni modeli že dolgo opredeljujejo mere, dimenzije, hierarhije, relacije in kazalnike. Microsoftova dokumentacija tabularne modele danes izrecno opisuje kot okolje za ustvarjanje objektov semantičnega modela, kot so tabele, relacije, hierarhije, mere in KPI-ji [4]. Power BI oziroma Microsoft Fabric uporablja izraz semantični model (semantic model) za logični opis analitične domene z metrikami, poslovno razumljivimi poimenovanji, dejstvi in dimenzijami [5].

Zato DWH in BI ne moremo obravnavati kot okolji brez semantike. Dobro zasnovan dimenzijski model že veliko pove o poslovnem dogodku, analitični model pa formalizira pomemben del mer in pravil. Današnji problem je predvsem v tem, da ta pomen pogosto ostane vezan na posamezen model, poročilo ali tehnološko okolje.

2.2. Kje obstoječi pristop začne omejevati

Razliko med razpršeno in upravljanjo semantiko bomo v prispevku ponazarjali na primeru izračuna in analize prodajne marže. Gre za dovolj znan poslovni kazalnik, ki hitro pokaže omejitve obstoječega pristopa: organizacija mora določiti, katero vrsto marže uporablja, katere postavke vključuje, kateri datum uporabi za analizo in kako jo pravilno agregira. Isti primer bomo v nadaljevanju uporabili tudi pri razlagi ontologije, grafa znanja in podatkovnega asistenta.

Del logike za izračun marže je lahko centraliziran že v DWH. V njem so lahko pripravljene neto prihodki, nabavna vrednost, pretvorbe valut, zgodovinska klasifikacija izdelkov in druge preverjene komponente izračuna. DWH zato zagotavlja skupno, centralizirano osnovo, česar lokalna SQL-poizvedba ali izračun v posameznem poročilu ne moreta. Kljub temu celotna poslovna definicija kazalnika pogosto ostane razpršena: končna formula je lahko zapisana v DAX-meri, posebni filtri v SQL-poizvedbi, izjeme pa v dokumentaciji ali znanju analitika. Posamezni oddelki lahko isto metriko opredelijo različno, aplikacija zunaj BI-orodja pa teh pravil morda sploh ne more ponovno uporabiti.

Problem torej ni nujno odsotnost centraliziranih podatkov, temveč odsotnost enotne, upravljane in ponovno uporabne definicije kazalnika. Sistem umetne inteligence lahko vidi fizično shemo, vendar iz tehničnih imen tabel in stolpcev ne more zanesljivo razbrati vseh poslovnih pravil in izjem.

Premik, ki ga obravnavamo, zato ni prehod od sveta brez pomena v svet s pomenom. Gre za prehod od razpršene, deloma implicitne in posameznemu orodju prilagojene semantike k skupni, upravljani in ponovno uporabni pomenski osnovi. Tudi novejši arhitekturni pristopi semantično plast obravnavajo kot samostojen povezovalni sloj med podatkovnimi strukturami in različnimi načini uporabe podatkov [6].

3 Semantična podatkovna plast: skupna pomenska osnova

3.1. Delovna definicija

V prispevku semantično podatkovno plast opredelimo kot upravljani logični sloj poslovnih metrik, dimenzij, pojmov, odnosov in pravil, ki različnim porabnikom omogoča enako razumevanje in uporabo istih podatkov. Gre za avtorski referenčni okvir. Ne pomeni nujno enega samega produkta, repozitorija ali poizvedovalnega jezika. Posamezne komponente so lahko fizično porazdeljene, vendar morajo skupaj delovati kot jasno upravljan vir poslovnega pomena.

Na analitični ravni takšna plast opredeljuje predvsem metrike, dimenzije, hierarhije, agregacijska in časovna pravila, dovoljene povezave ter omejitve dostopa. Na širši domenski ravni lahko vključuje tudi poslovne pojme, njihove odnose, sinonime, pravila in povezave do konkretnih entitet ali dokumentov. Teh dveh ravni ne predstavljamo kot splošno sprejetega industrijskega standarda, temveč kot praktičen okvir za razumevanje različnih arhitekturnih vlog.

3.2. Kaj pomeni semantika na primeru prodajne marže

Na primeru marže lahko konkretno pokažemo, kaj semantika pomeni v praksi. Pojem marža je na prvi pogled preprost, vendar organizacija potrebuje bistveno več kot ime stolpca. Semantična definicija vključuje formulo, časovno pravilo, valuto, dovoljene dimenzije, pravilo agregiranja, izjeme, lastnika in obdobje veljavnosti.

V tabeli 1 je prikazana ponazoritvena definicija prodajne marže. Njen namen ni določiti univerzalne formule, saj se ta med organizacijami razlikuje, temveč pokazati vrste pomena in pravil, ki jih mora semantična plast upravljati.

Tabela 1: Ponazoritvena semantična definicija prodajne marže.

Element definicije	Ponazoritvena vrednost
Poslovno ime	Prodajna marža [%]
Poslovni namen	Izračun razlike med neto prihodki in izbranimi neposrednimi oziroma spremenljivimi stroški.
Formula za znesek	Neto prihodek - nabavna vrednost - spremenljivi logistični stroški - prodajni rabati.
Stopnja marže	Prodajna marža / neto prihodek.
Časovno pravilo	Uporabi se datum knjiženja.
Valuta	Vrednosti se pred agregiranjem preračunajo v skupno poročevalsko valuto.
Dovoljene dimenzije	Čas, izdelek, kupec, segment in prodajni kanal.
Agregacijsko pravilo	Zneski se seštevajo; odstotek se na vsaki ravni ponovno izračuna.
Poslovni lastnik	Kontroling oziroma druga imenovana poslovna funkcija.
Obdobje veljavnosti	Definicija ima datum začetka in po potrebi datum konca veljavnosti.
Izjeme	Posebna pravila za vračila, pozne rabate ali izključene vrste prodaje.

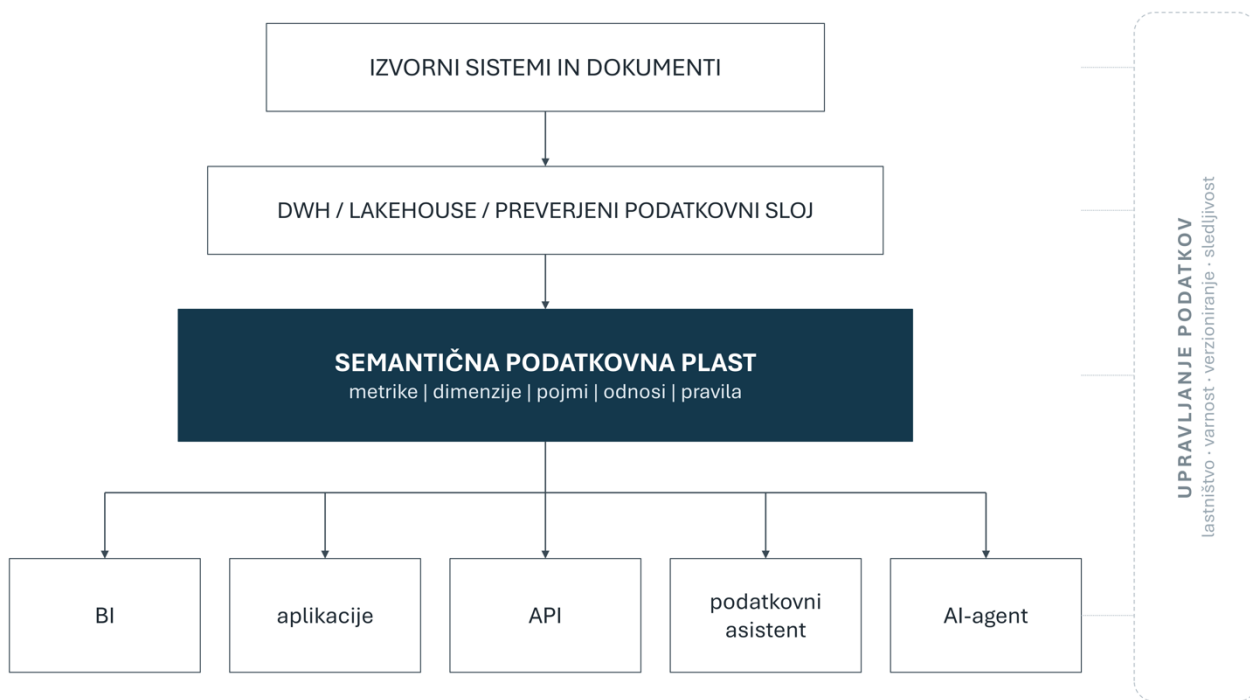
Agregacijsko pravilo je posebej pomembno. Odstotkov marže ne smemo preprosto seštevati ali povprečiti brez upoštevanja prihodkov. Če ima prvi segment 20 enot marže pri 100 enotah prihodka, je njegova marža 20 %, drugi pa 10 enot marže pri 200 enotah prihodka oziroma 5 %. Skupna marža zato ni 25 % (20 % + 5 %) in tudi ne 12,5 % kot preprosto povprečje obeh odstotkov. Pravilno jo izračunamo kot vsoto zneskov marže, deljeno z vsoto zneskov prihodka: $(20 + 10) / (100 + 200) = 10 \%$.

3.3. Razlika med DWH in semantično plastjo

Tabela 2: DWH in semantična podatkovna plast – različni, vendar povezani vlogi.

DWH oziroma lakehouse	Semantična podatkovna plast
Integrira in čisti podatke ter ohranja njihovo zgodovino.	Formalizira njihovo dogovorjeno poslovno uporabo.
Vsebuje dejstva, dimenzije, attribute in povezave.	Vsebuje metrike, pojme, hierarhije, pravila in izjeme.
Zagotavlja predvsem skupno podatkovno osnovo .	Zagotavlja predvsem skupno pomensko osnovo .
Optimiziran je za hranjenje, obdelavo in analitične poizvedbe.	Optimizirana je za konsistentno interpretacijo več porabnikov.
Poslovni pomen pogosto izraža posredno s podatkovnim modelom in transformacijami.	Poslovni pomen naredi jasen, upravljan in ponovno uporaben.

Meja med DWH in semantično plastjo ni absolutna. Semantična plast ne popravi slabega modeliranja in ne nadomesti preverjenih podatkov. Njena vrednost je v tem, da dogovorjeni pomen loči od posameznega dashboarda in ga ponudi različnim porabnikom: BI-orodjem, analitičnim aplikacijam, programskim vmesnikom in sistemom umetne inteligence.



Slika 2: Semantična plast kot skupna pomenska osnova za več porabnikov (referenčni okvir avtorjev).

4 Od zvezdne sheme do ontologije in grafa znanja

4.1. Štirje modeli, štiri različna vprašanja

Zvezdna shema, UML-model, ontologija in graf znanja lahko opisujejo isto poslovno domeno, vendar niso zamenljivi. Zvezdna shema je zasnovana za analizo dejstev po dimenzijah. UML je standardni jezik za modeliranje strukture, vedenja in arhitekture sistemov ter lahko opisuje tudi poslovne procese in podatkovne strukture [7]. Ontologija formalizira besednjak domene, pojme in njihove odnose; standard OWL na primer opredeljuje razrede, lastnosti, posameznike in podatkovne vrednosti s formalno določenim pomenom [8]. Graf znanja pa poveže konkretne entitete in odnose, pogosto iz več virov, pri čemer se lahko opira na strogo ontologijo ali ohlapnejšo shemo [9].

Tabela 3: Primerjava zvezdne sheme, UML-modela, ontologije in grafa znanja.

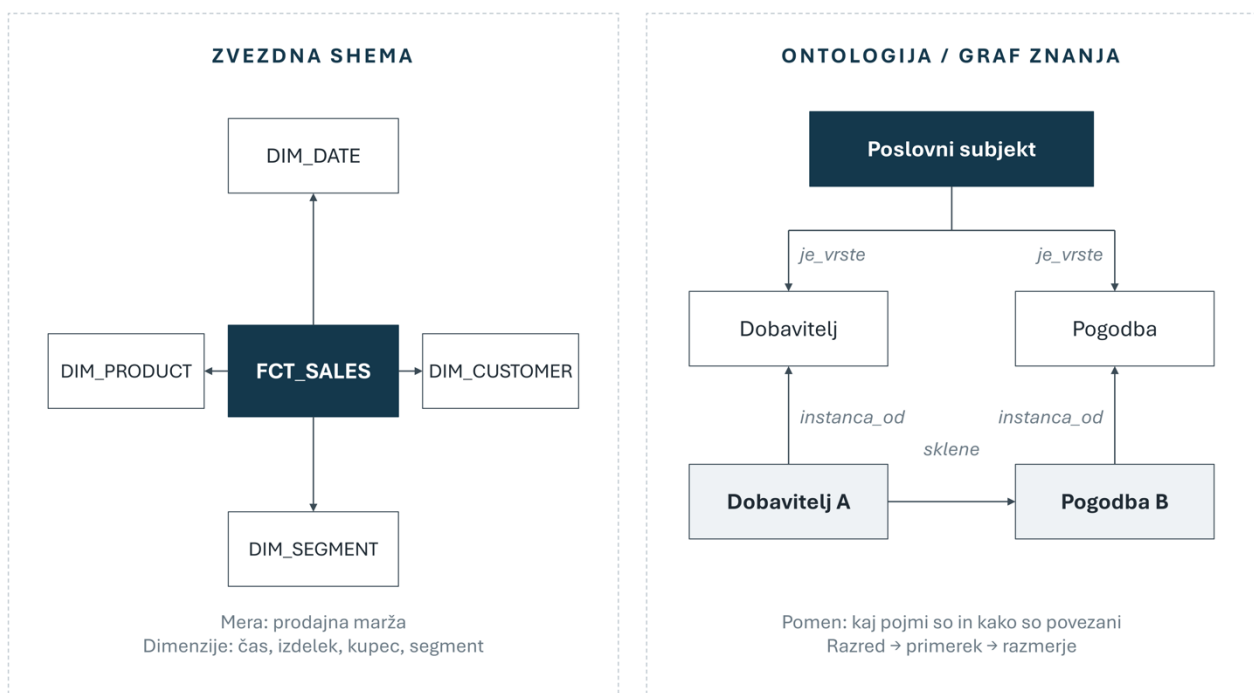
Koncept	Glavno vprašanje	Kaj dobro opiše	Primer
Zvezdna shema	Kako učinkovito analiziramo dogodek?	Dejstva, dimenzije, granulacijo, hierarhije in agregacije.	Prodaja po času, izdelku, kupcu in segmentu.
UML-model	Kako je domena ali sistem strukturiran?	Razrede, attribute, povezave, kardinalnosti in vedenje.	Kupec odda naročilo; naročilo vsebuje izdelek.
Ontologija	Kateri pojmi in odnosi veljajo ter kaj pomenijo?	Formalni besednjak, taksonomije, relacije, omejitve in pravila.	Pogodba določa cenik; cenik velja za skupino izdelkov.
Graf znanja	Katere konkretne entitete so povezane?	Instance pojmov, dejanske povezave, identiteto, kontekst in izvor.	Dobavitelj A ima pogodbo B, ki uporablja cenik C.

4.2. Zakaj relacijska shema ni isto kot ontologija

Tuji ključ v relacijski bazi pove, da sta zapisa povezana. Praviloma pa ne pojasni celotnega poslovnega pomena povezave, njenega namena, sinonimov, dovoljenih kombinacij ali pravil sklepanja. Tudi UML lahko zelo dobro prikaže razrede in relacije, vendar je pogosto predvsem dokumentacijski ali načrtovalski model. Ontologija gre korak dlje, kadar želimo pomen predstaviti v formalizirani obliki, ki jo lahko različni sistemi neposredno uporabljajo in preverjajo z mehanizmi za avtomatsko sklepanje ali validacijskimi pravili.

To ne pomeni, da mora vsaka organizacija zgraditi obsežno formalno ontologijo. Za dosledne KPI-je in standardno poročanje je semantični model metrik pogosto dovolj. Ontologija in graf znanja postaneta smiselna predvsem v kompleksnih domenah, kjer je treba povezovati več vrst entitet, časovno veljavne odnose, dokumente, pravila in dogodke.

4.3. Konkretni primer



Slika 3: Analitični pogled in širši domenski kontekst istega primera (konceptualna skica avtorjev).

Analitični pogled na levi dobro pokaže, kje in pri katerih izdelkih se je marža spremenila. Domenski pogled na desni pomaga povezati odstopanje z dobaviteljem, pogodbo, cenikom in drugim poslovnim kontekstom. Semantična podatkovna plast lahko uporabi oba pogleda: metrični model za zanesljiv izračun in domenski model za razlago odnosov.

5 Referenčna arhitektura in tehnološki vzorci

5.1. Semantična plast ni en sam produkt

Semantična plast se v praksi pojavlja v več arhitekturnih oblikah. Nekatere so tesno vgrajene v BI-platforno, druge živijo v podatkovni platformi, tretje delujejo kot samostojne programske dostopne storitve, četrte pa semantične definicije povezujejo s transformacijskimi modeli. Rešitve se deloma prekrivajo, vendar poudarjajo različne porabnike, načine upravljanja in obseg operativnega delovanja.

Pri primerjavi implementacij so zato pomembna predvsem štiri vprašanja: kje je model definiran in upravljan; kdo ga uporablja – BI, aplikacije, sistemi umetne inteligence ali vsi skupaj; ali rešitev ponuja samo definicije ali tudi izvajalno okolje (runtime); ter ali model zgolj opisuje podatke ali podpira tudi poslovna dejanja.

5.2. BI-vgrajeni in platformno vgrajeni modeli

Analysis Services in Power BI ponazarjata razvoj analitičnega semantičnega modela blizu poslovnega poročanja. Power BI Semantic Model je samostojen analitični objekt, v katerem so opredeljeni odnosi, mere, hierarhije, varnost in predstavitev podatkov za poročila ter druge odjemalce [4, 5]. To je močan in zrel vzorec, zlasti kadar je glavni način uporabe znotraj Microsoftovega analitičnega okolja.

Snowflake Semantic Views ponazarjajo drug vzorec: poslovni koncepti, metrike, entitete in relacije so shranjeni kot objekti na ravni podatkovne sheme. Snowflake jih uporablja za dosledne definicije v aplikacijah, za poizvedovanje z ukazom SELECT in kot kontekst za Cortex Analyst pri vprašanjih v naravnem jeziku [10]. Ta primer pokaže, da sposobnost generiranja SQL-a sama po sebi ni dovolj; rešitev potrebuje tudi poslovno semantiko nad fizičnimi tabelami.

5.3. Semantika kot samostojna storitev: headless, API-first in code-first

Headless semantična plast nima lastnega osrednjega uporabniškega oziroma predstavitvenega sloja. Njena naloga je, da metrike, dimenzije in pravila ponudi drugim porabnikom, kot so BI-orodja, aplikacije in sistemi umetne inteligence.

Pristop code-first pomeni, da so semantični modeli zapisani kot koda ali konfiguracija, na primer v YAML-u. Zato je mogoče upravljati njihove različice ter jih pregledovati, testirati in nameščati podobno kot druge programske artefakte.

Pristop API-first poudarja, da semantika ni namenjena samo interaktivnemu poročilu, temveč je programsko dostopna tudi drugim sistemom.

Platforma Cube oz. Cube.dev ponazarja kombinacijo pristopov headless, API-first in code-first. Deluje kot samostojna semantična storitev, ki centralizira metrike, dimenzije, povezave, pravila dostopa in optimizacijo poizvedb ter jih prek standardnih vmesnikov ponudi BI-orodjem, aplikacijam in sistemom umetne inteligence [11]. Gre za sodobno nadaljevanje idej kock in analitičnih modelov, vendar v obliki ponovno uporabne izvajalne storitve zunaj predstavitvenega sloja posameznega poročilnega orodja.

dbt Semantic Layer in MetricFlow ponazarjata pristop code-first v tesni povezavi s transformacijskimi modeli. Metrike in semantični modeli so definirani v YAML-u, MetricFlow pa na podlagi semantičnega grafa konstruira SQL in izbira poti med modeli [12]. Pristop je naraven predvsem v okoljih, kjer se podatkovni modeli, testi, sledljivost izvora (lineage) in dokumentacija že upravljajo v projektu dbt.

5.4. Operativna ontologija in grafovne platforme

Palantir Ontology predstavlja širši, operativni pristop. Palantirjeva dokumentacija opisuje Ontology kot digitalni dvojček organizacije, ki podatkovne vire in modele preslika v tipe objektov, lastnosti, povezave in vrste akcij. Akcije lahko spreminjajo objekte, lastnosti in povezave ter sprožajo dodatne učinke [13]. To je že korak od analitičnega razumevanja proti operativnemu modelu, na katerem lahko delujejo aplikacije in agenti.

Grafovne platforme, kot so Neo4j, Stardog, platforme RDF ali platforme na osnovi modela property graph, lahko služijo kot tehnološka osnova grafa znanja. Vendar grafovna baza sama po sebi še ni graf znanja in tudi ne zagotavlja dogovorjene poslovne semantike. Vrednost nastane z modeliranjem identitete, odnosov, konteksta, kakovosti in upravljanja, kar poudarja tudi sodobni pregled področja grafov znanja [9].

Primeri v tabeli ne pomenijo primerjave ali priporočila posameznega izdelka, temveč ponazarjajo različne arhitekturne vzorce in njihove značilne vloge.

Tabela 4: Tehnološki vzorci semantične plasti in sorodnih komponent.

Arhitekturni vzorec	Kaj ponazarja	Primeri
V BI vgrajeni semantični model	Mere, dimenzije in varnost blizu poročanju in poslovnim uporabnikom.	SSAS, Power BI Semantic Model
Platformno vgrajena semantika	Poslovni koncepti neposredno ob DWH ter AI- in SQL-storitvah.	Snowflake Semantic Views
Headless in API-first semantična plast	Ponovna uporaba metrik in pravil prek več odjemalcev.	Cube
Semantika blizu transformacij	Metrike kot koda ob podatkovnih modelih in transformacijah.	dbt Semantic Layer, MetricFlow
Operativna ontologija	Objekti in povezave, povezani tudi z akcijami in procesi.	Palantir Ontology
Platforma za graf znanja	Hranjenje in poizvedovanje po povezanih entitetah ter kontekstu.	Neo4j, Stardog, RDF-platforme

5.5. NL2SQL je orodje, ne semantika

Pretvorba vprašanj v naravnem jeziku v SQL (NL2SQL oziroma text-to-SQL) je uporabna tehnična sposobnost: sistem poskuša vprašanje uporabnika povezati s tabelami, stolpci, filtri in agregacijami ter iz njega ustvariti poizvedbo SQL. Raziskovalno področje je bistveno starejše od razmaha velikih jezikovnih modelov; ti so izboljšali prevajanje, niso pa odpravili problema povezovanja poslovnega jezika s shemo in pomenom podatkov [14].

NL2SQL rešuje, kako vprašanje izraziti v SQL. Semantična plast rešuje, kaj je pravilno vprašati in kako je treba rezultat interpretirati.

NL2SQL sam po sebi ne ve, katera definicija marže je v organizaciji uradna. Lahko ugiba na podlagi imen stolpcev ali opisov, lahko pa uporabi upravljano metriko iz semantične plasti. Zato ga v arhitekturi ne obravnavamo kot naslednjo razvojno stopnjo semantike, temveč kot enega od porabnikov semantičnega modela.

6 Od semantične plasti do podatkovnega asistenta in agenta

6.1. Podatkovni asistent kot izvedljiv naslednji korak

Šele na podlagi predhodno opisane arhitekture lahko natančneje uvedemo podatkovnega asistenta. Izraz še nima enotne in splošno sprejete tehnične definicije. V tem prispevku z njim označujemo AI-podprto programsko rešitev, ki z uporabo semantične podatkovne plasti analizira strukturirane podatke in z njimi povezani poslovni kontekst ter uporabniku pripravi sledljiv odgovor, ustrezen analitični pogled ali predlog fokusa. Ne gre za osebo niti za sinonim semantične plasti; asistent je njen porabnik.

Izvedljiv cilj ni, da sistem takoj samostojno sprejema poslovne odločitve. Za številna okolja bi bil pomemben napredek že asistent, ki zna iz kombinacij mer in dimenzij sproti pripraviti ustrezen analitični pogled oziroma ad hoc dashboard, namesto da bi moral uporabnik pravi pogled iskati med obstoječimi poročili. Tako mu pomaga bolj razumeti informacije, ki jih nosijo te kombinacije.

6.2. Analiza na zahtevo in proaktivni pregled

Pri analizi na zahtevo uporabnik že ve, kaj ga zanima, in vpraša: »Zakaj se je marža zmanjšala?« Analitik pregleda pripravljene dashboarde, po potrebi izvede dodatno ad hoc poizvedbo, primerja relevantna obdobja in dimenzije ter pripravi odgovor. Kakovost in hitrost analize sta pri tem odvisni od njegovega poznavanja podatkovnega modela, poslovnih definicij in razpoložljivih virov.

Pri proaktivnem pregledu uporabnik še ne ve, kateri pojav zahteva njegovo pozornost. Asistent v vnaprej določenem obsegu pregleda dogovorjene metrike, zazna pomembnejša odstopanja in predlaga, kam naj se analiza

usmeri. Prav ta način izraža sporočilo »ne še en dashboard«: rešitev ne čaka samo, da uporabnik odpre pravi pogled, temveč mu pomaga ugotoviti, kaj je pomembno.

6.3. Kaj podatkovni asistent dejansko naredi

Ne glede na to, ali analizo sproži uporabnik ali vnaprej določen pregled, mora podatkovni asistent izvesti več povezanih korakov:

- določi ustrezne metrike, obdobja in primerjalni kontekst;
- iz semantične plasti pridobi uradne definicije, dovoljene dimenzije in pravila;
- pripravi ali izvede potrebne analitične poizvedbe;
- primerja rezultate po smiselnih kombinacijah mer in dimenzij;
- loči manj pomembna nihanja od odstopanj, ki zahtevajo pozornost;
- pripravi ustrezen analitični pogled oziroma ad hoc dashboard;
- rezultat pojasni skupaj z uporabljenimi viri, omejitvami in predlogom, kam usmeriti nadaljnjo analizo.

6.4. Zakaj isto osnovo potrebujejo tudi agenti

Semantična podatkovna plast ni pomembna samo za podatkovnega asistenta. Njena vloga postane še pomembnejša pri agentu, ki naj bi več analitičnih korakov načrtoval in izvedel z manj neposrednega usmerjanja človeka. Če mora sistem pri vsakem koraku uporabnika vprašati, kateri dashboard naj odpre, katera definicija metrike velja in po kateri dimenziji naj nadaljuje analizo, še ne deluje kot agent z večjo stopnjo samostojnosti. Manj ko je podrobnih človeških navodil, jasneje morajo biti opredeljeni poslovni pomen, dovoljene povezave, pravila dostopa in meje delovanja.

To ne pomeni, da so organizacije že pripravljene na avtonomno poslovno odločanje. V podatkovni stroki se dobro zavedamo, da smo od povsem samodejnega odločanja še daleč. Podatkovni asistent je zato izvedljiv naslednji korak: pripravi ustrezen pogled, opozori na signal in pomaga pri razumevanju, človek pa ohrani odločitev in odgovornost. Agent predstavlja nadaljnjo stopnjo, na kateri lahko sistem poleg analize v okviru jasnega pooblastila sproži tudi dejanje.

Prehod od branja in analize k zapisovanju v poslovne sisteme bistveno poveča zahteve glede pooblastil, nadzora, validacije in odgovornosti. Profil NIST za generativno umetno inteligenco zato poudarja upravljanje tveganj skozi celoten življenjski cikel sistema, ne le kakovost končnega uporabniškega vmesnika [15].

Tabela 5: Praktična razlika med podatkovnim asistentom in AI-agentom.

Podatkovni asistent	AI-agent
Pripravi analizo, razlago in pogled.	Načrtuje tudi nadaljnje korake.
Opozori in predlaga fokus.	Lahko sproži proces ali delegirano dejanje.
Primarno bere podatke in kontekst.	Lahko tudi zapisuje v operativne sisteme.
Človek odloči.	Del odločanja ali izvedbe je delegiran.
Potrebuje sledljivo semantiko.	Zaradi večje avtonomije jo potrebuje še bolj.
Nižje operativno tveganje.	Višje tveganje in strožje zahteve po nadzoru.

Semantična plast ni posledica agentov. Je pogoj, da lahko podatkovni asistent ali agent deluje brez stalnega človeškega prevajanja poslovnega pomena.

Arhitekturna meja je zato jasna: preverjeni podatki in semantična plast podpirajo podatkovnega asistenta; asistent pripravi signal, razlago in fokus; človek odloči; šele ob dodatnih pooblastilih in varovalih lahko agent izvede ukrep.

7 Praktični primer: od metrike marže do poslovnega fokusa

7.1. Klasična pot skozi poročila

Predpostavimo, da se je marža v enem od segmentov zmanjšala. V klasičnem pristopu uporabnik odpre dashboard prodaje, opazi spremembo, nastavi obdobje, filtrira segment, preveri izdelke in kupce, nato pa odpre še poročilo nabave ali poišče cenik. Podatki morda obstajajo, toda analitično pot še vedno sestavi človek. Rezultat je odvisen od njegovega poznavanja modelov, definicij in mest, kjer so informacije shranjene.

7.2. Kaj doda semantična plast

Semantična plast zagotovi uradno definicijo prodajne marže iz tabele 1, ustrezen časovni kontekst, dovoljene dimenzije, pravilo agregiranja in povezavo do preverjenih podatkov. Modul za analizo ali podatkovni asistent zato ne išče poljubnega stolpca z imenom »margin«, temveč uporablja upravljano metriko. S tem vzroka še ne pojasni, zagotovi pa poslovno dosledno izhodišče.

7.3. Kaj doda graf znanja

Graf znanja lahko odstopanje poveže s konkretnimi entitetami in dogodki: segmentom, skupino izdelkov, dobaviteljem, pogodbo, cenikom ali prodajno akcijo. Dokumenti niso samodejno dokaz vzroka, ponujajo pa relevanten kontekst za nadaljnjo razlago. Pri preprostejših primerih je lahko dovolj dober indeks dokumentov; graf znanja je smiseln takrat, ko so odnosi kompleksni in jih želimo ponovno uporabljati v več analitičnih ali operativnih scenarijih.

7.4. Kaj naredi podatkovni asistent

Podatkovni asistent lahko najprej zazna, da je sprememba večja od dogovorjenega praga ali pomembno odstopa od primerjalnega vzorca. Nato sistematično preveri dovoljene dimenzije ter spremembo marže razčleni na učinke prodajne cene, nabavne cene, rabatov, prodajnega miksa in valute. Izpostavi največje kvantificirane prispevke in jih poveže z relevantnimi poslovnimi dogodki.

Primer sledljivega odgovora bi lahko bil: »Prodajna marža se je najbolj zmanjšala v segmentu X. Največja kvantificirana prispevka sta bila rast nabavnih cen skupine izdelkov Y in večji delež prodaje z akcijskim rabatom. V istem obdobju je začel veljati nov cenik dobavitelja A. Predlagana področja nadaljnjega pregleda so vpliv pogodbe B, sprememba cenika C in podaljšanje akcije D.« Gre za ponazoritveni arhitekturni scenarij, ne za rezultat empirično preverjenega prototipa.

Pomembna je tudi previdnost pri jeziku odgovora. Sistem lahko pokaže kvantificirane prispevke, časovno povezanost in verjetne razlage. Brez dodatne analize ne sme trditi, da je posamezen poslovni dogodek dokazano povzročil spremembo. Sledljiv in razločljiv odgovor mora jasno ločiti meritev, povezavo in interpretacijo.

7.5. Kje bi se začela vloga agenta

Podatkovni asistent se po pripravi fokusa ustavi. Ne spremeni cenika, ne prekine prodajne akcije in ne odpre pogajanj z dobaviteljem. Agent bi lahko v naslednjem koraku, ob ustreznih pooblastilih, odprl nalogo odgovorni osebi, pripravil predlog ukrepa ali sprožil potrditveni proces. Prav zato mora biti razlika med analitično pomočjo in operativnim ukrepanjem jasno določena že v arhitekturi.

8 Omejitve in pogoji za dejansko uporabo

8.1. Semantična plast ne odloči, katera definicija je pravilna

Semantična plast lahko definicijo formalizira, upravlja njene različice in jo dosledno uporablja, ne more pa sama razrešiti spora med prodajo, financami in kontrolingom. Metrike potrebujejo poslovnega lastnika, postopek potrjevanja in jasno veljavnost. Napačno opredeljena metrika ostane napačna, le da je napaka zdaj bolj vidna in ponovljiva.

8.2. Slabi podatki ostanejo slabi podatki

Manjkajoči, prepozni ali napačni podatki se z dodajanjem semantike ne popravijo. Še vedno potrebujemo zagotavljanje kakovosti podatkov, testiranje transformacij, nadzor izvorov in poznavanje granulacije. Semantična plast lahko zmanjša nevarnost uporabe napačne tabele ali napačnega pravila, ne more pa nadomestiti podatkovnega inženirstva.

8.3. Spremembe, dostop in sledljivost

Poslovne definicije se spreminjajo, zato morajo biti upravljane po različicah in časovno sledljive. Enako velja za pravice dostopa: uporabnik omejitev iz BI-orodja ne sme obiti prek asistenta ali agenta. Odgovor mora pokazati uporabljeno metriko, filtre, vire in dokumente; sicer je lahko prepričljiv, vendar ga ni mogoče preveriti.

8.4. Dodatna kompleksnost in meja razlage

Ontologija in graf znanja zahtevata modeliranje identitete, odnosov in časovne veljavnosti, zato nista obvezen del vsake semantične plasti. Smiselna sta tam, kjer širši kontekst prinaša dovolj vrednosti. Hkrati povezava med dogodkom in spremembo metrike še ni dokaz vzročnosti. Sistem mora ločiti izračunane prispevke, korelacije, dokumentirane dogodke in interpretacijo.

8.5. Podatkovni asistent je pogosto primernejši prvi korak kot agent

Organizacije pogosto govorijo o avtonomnih agentih, še preden imajo enotno definicijo osnovnih metrik ali urejeno lastništvo podatkov. V takem okolju avtonomija ne odpravi negotovosti, temveč jo avtomatizira. Podatkovni asistent lahko prinese pomembno vrednost z manjšim operativnim tveganjem: sistem pregleda, opozori in pojasni, človek pa ohrani odločitev in odgovornost. Prehod k agentskemu delovanju naj sledi šele takrat, ko so semantika, pravice dostopa, nadzorne točke in odgovornost ustrezno urejeni in dovolj zreli [15].

9 Razprava in zaključek

Prispevek semantične podatkovne plasti ne postavlja kot zamenjavo za DWH, dimenzijsko modeliranje ali semantične modele BI, temveč kot njihovo nadaljevanje. DWH omogoča ponovno uporabo preverjenih podatkov, semantična plast pa ponovno uporabo njihovega dogovorjenega pomena. Zvezdna shema, UML, ontologija in graf znanja pri tem rešujejo različna vprašanja, zato jih je treba povezovati glede na dejanski poslovni problem.

Praktični rezultat skupne semantike je lahko podatkovni asistent, ki ne izdela samo še enega pogleda, temveč pregleda več pogledov, izlušči signal in človeku predlaga fokus. NL2SQL je pri tem orodje za pretvorbo vprašanja v SQL, graf znanja je vir širšega konteksta, semantična plast pa zagotavlja iste potrjene definicije vsem porabnikom.

Naslednji korak analitike ni nujno še en dashboard – niti takoj avtonomni agent. Naslednji korak je zaupanja vredna semantična osnova.

Na tej osnovi lahko asistent pokaže, katere metrike in vire uporablja ter kje se mora ustaviti. Semantična podatkovna plast je zato manjkajoči temelj analitike nove generacije: ne zato, ker bi bila povsem nova, temveč zato, ker mora pomen podatkov postati skupen ljudem, analitičnim orodjem, aplikacijam in sistemom umetne inteligence.

Literatura

- [1] MACARI Edgar, »Improve Business Outcomes by Turning Dashboards Into Analytics Products«, Gartner, 11. september 2025, <https://www.gartner.com/en/documents/6943066>, obiskano 27. 7. 2026.
- [2] GANESHAN Anirudh, »3 Actions D&A Leaders Should Take to Reinvigorate Self-Service Analytics«, Gartner, 21. maj 2024, <https://www.gartner.com/en/documents/5448363>, obiskano 27. 7. 2026.
- [3] KIMBALL Ralph, ROSS Margy, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3. izdaja, Wiley, 2013.
- [4] MICROSOFT, Tabular modeling overview - Analysis Services, Microsoft Learn, 2024, <https://learn.microsoft.com/en-us/analysis-services/tabular-models/tabular-models-ssas>.
- [5] MICROSOFT, Power BI semantic models in Microsoft Fabric, Microsoft Learn, 2025, <https://learn.microsoft.com/en-us/fabric/data-warehouse/semantic-models>.
- [6] STIREWALT R. E. K., BÚR Márton, »The RAI Way: A Technical Analysis and Design Method for Building Enterprise Semantic Layers«, Advanced Information Systems Engineering Workshops, LNBIP 482, Springer, 2023, str. 74-79, https://doi.org/10.1007/978-3-031-34985-0_8.
- [7] OBJECT MANAGEMENT GROUP, Unified Modeling Language (UML), različica 2.5.1, <https://www.omg.org/spec/UML/2.5.1/>.
- [8] W3C OWL WORKING GROUP, OWL 2 Web Ontology Language Document Overview, Second Edition, W3C Recommendation, 2012, <https://www.w3.org/TR/owl2-overview/>.
- [9] HOGAN Aidan et al., »Knowledge Graphs«, ACM Computing Surveys, letnik 54, št. 4, članek 71, 2021, <https://doi.org/10.1145/3447772>.
- [10] SNOWFLAKE, Overview of semantic views, Snowflake Documentation, <https://docs.snowflake.com/en/user-guide/views-semantic/overview>.
- [11] CUBE DEV, Introduction and Data Modeling, Cube Documentation, <https://docs.cube.dev/docs/introduction>.
- [12] DBT LABS, About MetricFlow, dbt Developer Hub, <https://docs.getdbt.com/docs/build/about-metricflow>.
- [13] PALANTIR TECHNOLOGIES, Ontology core concepts, Palantir Foundry Documentation, <https://www.palantir.com/docs/foundry/ontology/core-concepts>.
- [14] KATSOGIANNIS-MEIMARAKIS George, KOUTRIKA Georgia, »A survey on deep learning approaches for text-to-SQL«, The VLDB Journal, letnik 32, 2023, str. 905-936, <https://doi.org/10.1007/s00778-022-00776-8>.
- [15] AUTIO Chloe et al., Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, National Institute of Standards and Technology, 2024, <https://doi.org/10.6028/NIST.AI.600-1>.

Podatkovne semantične smernice kot temelj upravljanja podatkov in semantične interoperabilnosti v javni upravi

Martina Furlan, Miha Jesenko, Karmen Kern Pipan

Ministrstvo za notranje zadeve in javno upravo Republike Slovenije, Direktorat za informatiko, Ljubljana, Slovenija
martina.furlan@gov.si, miha.jesenko@gov.si, karmen.kern-pipan@gov.si

Prispevek prikazuje temeljne poudarke dokumenta »Podatkovne semantične smernice«, ki jih je marca 2026 objavilo Ministrstvo za digitalno preobrazbo (zdaj Ministrstvo za notranje zadeve in javno upravo). Smernice so nastale ob podpori strokovnjakov s Fakultete za računalništvo in informatiko Univerze v Ljubljani ter v sodelovanju z različnimi organi javne uprave. Smernice na enem mestu združujejo vsebine upravljanja podatkov in semantične interoperabilnosti. Za kakovostno odločanje in oblikovanje politik na podlagi podatkov potrebujemo kakovostne in interoperabilne podatke, dobro upravljanje in s tem možnost izrabe novih tehnologij. Smernice predstavljajo temelj za nadaljnji razvoj in boljše, tudi čezmejno izmenjavo podatkov v skladu z evropskimi in mednarodnimi priporočili ter so standard za razvijalce aplikacij in druge uporabnike v organih javne uprave.

Ključne besede:

podatki

podatkovne semantične smernice

upravljanje podatkov

semantična interoperabilnost

nove tehnologije

1 Uvod

Količina podatkov v digitalnem svetu iz leta v leto strmo narašča, zlasti v zadnjem desetletju. Medtem ko je leta 2010 znašala približno 2 zetabajta¹, je do leta 2015 narasla na 15,5 zetabajta. Leta 2025 je dosegla že 181 zetabajtov, za leto 2026 pa se ocenjuje, da bo znašala kar 221 zetabajtov. Vsak dan se po svetu ustvari približno 402,74 milijona terabajtov podatkov, kar ustreza približno 0,4 zetabajta dnevno. Ocenjuje se, da je bilo kar 90 % vseh podatkov na svetu ustvarjenih v zadnjih dveh letih. Pri tem imajo pomembno vlogo video vsebine, ki predstavljajo približno 82 % celotnega globalnega podatkovnega prometa, vključno s socialnimi omrežji in drugimi spletnimi platformami [1]. Podatki torej nastajajo in nas spremljajo na vseh področjih, nove tehnologije pa odpirajo skoraj neomejene možnosti njihove uporabe ter podatke postavljajo med ključne vire sodobne družbe in gospodarstva.

Pojav novih tehnologij in njihovo vse večje uvajanje v javni upravi sta sovpadla z eksponentno rastjo količine in uporabe podatkov kot posledico digitalizacije, s čimer so se razširile možnosti analize podatkov [1, str. 3]. Okolje, v katerem se izmenjujejo podatki med organi državne uprave, je kompleksno in vključuje številne ovire in izzive za zagotavljanje storitev. Ovire lahko vključujejo različno interpretacijo podatkov, pomanjkanje skupnih in široko uporabnih podatkovnih modelov, odsotnost skupnih šifrantov, pomanjkanje interoperabilnostnih orodij in formatov itd. Semantična interoperabilnost zagotavlja, da se bosta oblika in pomen podatkov ohranila, tako da bodo podatki razumljeni na enak način pri vseh uporabnikih (npr. pri elektronski izmenjavi podatkov). V Strategiji digitalnih javnih storitev 2030 je opredeljen strateški cilj »Sodobna informacijska tehnologija ter upravljanje zaupanja vrednih podatkov«, ki podpira vpeljavo podatkovnih standardov in smernic [2]. Tudi cilj Digitalne strategije Evropske unije (EU) je zagotoviti, da bo digitalna preobrazba koristila državljanom in podjetjem ter da bo pripomogla k doseganju cilja podnebno nevtralne Evrope do leta 2050. Evropska komisija (EK) je v tem okviru opredelila obdobje 2020–2030 kot »digitalno desetletje« Evrope. Obenem želi EK okrepiti digitalno suverenost EU in prispevati k oblikovanju standardov, pri čemer je poseben poudarek namenjen podatkom, tehnologiji in infrastrukturi [3]. Podatkovne semantične smernice na krovni ravni vzpostavljajo okvir za semantično interoperabilnost in predstavljajo pomemben korak na poti do cilja boljše izmenjave podatkov. Dokument je nastal v sodelovanju s strokovnjakoma dr. Slavkom Žitnikom in dr. Dejanom Lavbičem s Fakultete za računalništvo in informatiko Univerze v Ljubljani. Med ciljnimi uporabniki teh smernic so organi javnega sektorja ter drugi deležniki, ki so kakor koli povezani z javnim sektorjem, kot denimo razvijalci, raziskovalci in drugi uporabniki podatkov oziroma informacijskih rešitev. Smernice so bile pripravljene v okviru projekta »Razvoj novih dinamičnih e-storitev (DES)«, ki sta ga sofinancirali Republika Slovenija in Evropska unija iz Načrta za okrevanje in odpornost [4]. Dokument je bil predstavljen na desetih delavnicah več kot 100 udeležencem iz več kot 30 državnih organov, vključeni so bili tudi prejeti predlogi za izboljšave [5]. Verjamemo, da se bo vsebina v sodelovanju z deležniki sčasoma ustrezno nadgrajevala in izboljševala. V nadaljevanju bomo predstavili nekaj poudarkov iz pomembnejših vsebin dokumenta.

2 Upravljanje podatkov

Upravljanje podatkov (angl. *data management*) obsega proces zbiranja, shranjevanja, zaščite in zagotavljanja kakovosti podatkov v celotnem življenjskem ciklu. MIT (Massachusetts Institute of Technology) ugotavlja, da je upravljanje podatkov ključnega pomena za ustvarjanje okolja, v katerem se podatki lahko uporabljajo v celotni organizaciji. Učinkovito upravljanje podatkov zmanjšuje težave, ki jih povzročajo napačni podatki, kot so netočne napovedi in celo preprosto dejstvo, da podatki niso dostopni, v idealnem primeru še preden se te težave pojavijo [6]. Gartner navaja, da mora upravljanje podatkov organizaciji prinašati koristi ob pravem času. Sedanje prakse upravljanja podatkov so pogosto pretoge in ne upoštevajo poslovnega okolja. Ocenjuje se, da kar 60 % organizacij ne bo

¹ Merske enote velikosti bajtov: <https://www.geeksforgeeks.org/computer-science-fundamentals/understanding-file-sizes-bytes-kb-mb-gb-tb-pb-eb-zb-yb/>, obiskano 27. 7. 2026

uspelo uresničiti pričakovane vrednosti svojih možnosti uporabe umetne inteligence do leta 2027 zaradi neskladnih sistemov upravljanja podatkov. Nadalje Gartner izpostavlja, da je treba vključiti najboljše prakse v načrt za upravljanje podatkov in vzpostaviti učinkovito strukturo upravljanja, oblikovati in uvesti učinkovite politike in standarde, oceniti in izboljšati učinkovitost upravljanja podatkov ter vzpostaviti proces ponavljanja in učenja [7]. V Sloveniji je pri upravljanju podatkov treba upoštevati tudi zakonodajne akte (npr. Zakon o varstvu osebnih podatkov (ZVOP-2) [8], Zakon o dostopu do informacij javnega značaja (ZDIJZ) [9], Zakon o izvajanju Uredbe (EU) o evropskem upravljanju podatkov (ZIUUEP)) [10], Akt o interoperabilni Evropi [11], ki določajo pravila za zbiranje, uporabo, izrabo in izmenjavo podatkov, ter strateške dokumente (npr. Strategija digitalnih javnih storitev 2030, Strategija Digitalna Slovenija 2030 in evropsko digitalno desetletje), usmerjene v krepitev interoperabilnosti ter učinkovitejšo izrabo podatkov.

2.1. Načela FAIR

Pomemben temelj upravljanja podatkov predstavljajo načela FAIR (FAIR – Findable, Accessible, Interoperable, Reusable).

F – Findable (najdljivi) – podatki in metapodatki morajo biti enostavno najdljivi tako za ljudi kot za računalniške sisteme. Vsak podatkovni zapis mora imeti trajni in enolični identifikator (npr. DOI), biti ustrezno opisan z metapodatki ter indeksiran v iskalnih sistemih.

A – Accessible (dostopni) – podatki morajo biti dostopni prek standardiziranih komunikacijskih protokolov. Tudi kadar podatki niso več na voljo, morajo metapodatki ostati dostopni.

I – Interoperable (povezljivi oziroma interoperabilni) – podatki morajo uporabljati standardizirane formate, besednjake in ontologije, da jih je mogoče povezovati ter kombinirati z drugimi podatki in sistemi.

R – Reusable (ponovno uporabni) – podatki morajo biti opisani tako dobro, da jih lahko drugi raziskovalci ali organizacije ponovno uporabijo. Jasno morajo biti navedeni pogoji uporabe, izvor podatkov in standardi kakovosti [12]. Načela FAIR:

- omogočajo dolgotrajno in učinkovito rabo podatkov v različnih sektorjih in sistemov,
- zmanjšujejo podvajanja in povečujejo zaupanje v podatke,
- omogočajo povezovanje in ponovno uporabo podatkov za raziskovalne, analitične in upravne namene,
- spodbujajo transparentnost, odgovornost in vrednotenje učinkov javnih politik [4, str. 13].

Nadalje prikazujemo nekaj smernic in praktičnih primerov iz dokumenta »Podatkovne semantične smernice«:

Smernica 1 – Sledenje zakonodaji in smernicam: organizacije morajo redno spremljati predpise ter strateške usmeritve pri upravljanju podatkov.

Primer 1 – Zakon o varstvu dokumentarnega in arhivskega gradiva ter arhivih (Uradni list RS, št. 30/06, 24/14 – odl. US in 51/14) definira podobna načela, kot jih naslavlja načela FAIR, na primer načelo trajnosti (4. člen), načelo celovitosti (5. člen) in načelo dostopnosti (6. člen). Pri tem je treba upoštevati tudi Uredbo o varstvu dokumentarnega in arhivskega gradiva (UVDAG, Uradni list RS, št. 42/2017) in Pravilnik o enotnih tehnoloških zahtevah za zajem in hrambo gradiva v digitalni obliki (Uradni list RS, št. 118/20). Upoštevanje načel FAIR ne bi pomenilo le boljše enotnosti podatkov in večjih možnosti njihove izmenjave, temveč bi tudi pri arhiviranju zmanjšalo obseg dodatnega dela, potrebnega za pretvorbo podatkov v ustrezno obliko.

Smernica 2 – Upoštevanje načel FAIR: ob ustvarjanju novih ali posodabljanju obstoječih podatkovnih virov je treba zagotoviti najdljivost, dostopnost, interoperabilnost in ponovno uporabljivost podatkov.

Primer 2 – Pozitivni učinki upoštevanja načel FAIR: pri modeliranju podatkov za pomoč uporabnikom z različnimi oblikami oviranosti lahko neprimerno definiran podatkovni standard vodi do izključenosti ali diskriminacije [4, str. 13].

2.2. Etika

Etika postavlja moralna načela in usmeritve za odgovorno ravnanje z vse bolj raznovrstnimi in obsežnimi podatki (zbiranje, shranjevanje, analiza, izmenjava in uporaba podatkov, pri čemer uporaba vključuje tudi strojno interpretacijo ali predstavitev podatkov). Etika ravnanja s podatki zajema moralne obveznosti v zvezi z zbiranjem, varovanjem in uporabo osebnih podatkov ter vpliv, ki ga to ima na posameznike [13]. Gre za ohranjanje zaupanja uporabnikov, strank, zaposlenih in deležnikov pri ravnanju s podatki, kar je ključni vidik sodobnega upravljanja, glede na vse večjo odvisnost od podatkov v procesih odločanja [14]. Pri delu s podatki je ključno upoštevati etična načela, med katerimi so zaščita zasebnosti, transparentnost in preprečevanje diskriminacije.

V nadaljevanju prikazujemo nekaj praktičnih primerov in smernic iz Podatkovnih semantičnih smernic o etičnem ravnanju pri upravljanju podatkov:

Smernica 3 – Pravičnost in preprečevanje diskriminacije: pri modeliranju podatkov je treba prepoznati in odpraviti pristranskost, zagotavljati ažurnost in celovitost podatkov.

Primer 3 - Preprečevanje nepoštenih izidov modela: slabo zasnovan model lahko pri avtomatiziranem odločanju na podlagi podatkov o družini ali zaposlenosti povzroči nepošteno izide.

Smernica 6 – Etično ravnanje pri ponovni uporabi podatkov: Pri ponovni uporabi podatkov iz različnih virov je treba spoštovati prvotni namen uporabe, predvsem pravne podlage in kontekst, v katerem so bili podatki zbrani.

Primer 6 – Uporaba podatkov za drug namen: v primeru osebnih in drugih pravno varovanih podatkov je zelo pomembno, da se podatki zbirajo skladno s pravno podlago, pri kateri je jasno opredeljen namen zbiranja. Zbranih podatkov ni dovoljeno uporabiti za druge namene brez ustrezne pravne podlage [4, str. 14–16].

2.3. Življenjski cikel podatkov

Da bi znanost o podatkih postavili v ustrezen kontekst, je treba predstaviti faze življenjskega cikla podatkov, od njihovega nastanka do njihove interpretacije. Te faze surove podatke pretvarjajo v vrednost za končnega uporabnika. Poleg tega je treba na vsaki stopnji življenjskega cikla podatkov upoštevati varstvo podatkov in etiko ravnanja s podatki [15]. Pri tem je zelo pomembno tudi upravljanje podatkov, pri katerem se uporabljajo uveljavljeni modeli za upravljanje podatkov (kot denimo DAMA-DMBOK (angl. Data Management Body of Knowledge), CMMI DMM (angl. Data Management Maturity) ter standard ISO 8000), ki omogočajo strukturirano upravljanje skozi celoten življenjski cikel podatkov – od zajema podatkov, njihove obdelave, shranjevanja, upravljanja, analiz, interpretacije do njihovega arhiviranja ali izbrisa. Tabela 1 prikazuje stopnje življenjskega cikla podatkov skupaj s kratkim opisom.

Za vpeljavo življenjskega cikla podatkov v organizacijo je lahko v pomoč tudi Smernica 22:

Smernica 22 – Določanje življenjskega cikla podatkov: opredeliti je treba faze življenjskega cikla podatkov in pravila ravnanja s podatki v vsaki fazi. Posamezne stopnje življenjskega cikla podatkov naj predstavljajo »testni okvir« s katerim lahko preverjamo ustreznost ravnanja s podatki, določamo namene zbiranja, hranjenja in arhiviranja podatkov ali izbrisa podatkov, s čimer tudi sledimo trajnostnemu upravljanju podatkov [4, str. 35–36].

Tabela 1: Stopnje življenjskega cikla podatkov

STOPNJA	KLJUČEN PROCES	NOSILEC
Generiranje	Nastanek podatkov v transakcijskih bazah, senzorjih itd.	Uporabniki naprav, skrbniki informacijskih sistemov.
Zbiranje in obdelava	Čiščenje, transformacija, zagotavljanje varnosti.	Skrbniki podatkov (angl. data stewards), IT služba, varnostni strokovnjaki.
Shranjevanje	Varno hranjenje, zagotavljanje dostopnosti do podatkov. Predvsem upoštevanje zakonodaje glede dostopnosti in varnosti podatkov.	Skrbnik podatkov, IT služba.
Upravljanje	Beleženje dostopov do podatkov, zagotavljanje revizijskih sledi in sledenje spremembam, upravljanje s podatkovnimi politikami.	Skrbniki podatkov, IT služba, imetniki podatkovnih zbirk.
Analiza in vizualizacija	Izvajanje statistik, algoritmov, gradnja modelov, priprava grafov, poročil, nadzornih plošč.	Analitiki, podatkovni znanstveniki, inženirji poslovne inteligence.
Interpretacija	Strokovna presoja, strateško odločanje.	Vsebinski strokovnjaki, odločevalci, imetniki podatkov.
Arhiviranje ali izbris	Zagotavljanje trajnosti ali sprostitev podatkovnih virov.	Imetniki podatkov, vsebinski strokovnjaki, odločevalci, IT služba

Vir: [4, str. 32].

2.4. Kakovost in varnost podatkov

Organizacije javnega sektorja potrebujejo ustrezne podatke za zagotavljanje kakovostnih storitev, sprejem pravih odločitev in oblikovanje učinkovitih politik. Podatki so lahko različnih vrst, med drugim proračunske številke, podatki o zaposlenih, odgovori na ankete in podatki, zbrani v okviru digitalnih storitev. Vsak (zaposleni) ima svojo vlogo pri zagotavljanju kakovosti podatkov [16].

Kakovost podatkov se meri z dimenzijami, kot so točnost, celovitost, doslednost, zanesljivost in ažurnost. Točnost pomeni natančnost glede na dejansko stanje, celovitost pomeni prisotnost vseh potrebnih podatkov, doslednost pomeni usklajenost med različnimi viri, zanesljivost pomeni stabilnost, zaupanje in preverljivost, ažurnost pa njihovo časovno ustreznost. Te lastnosti skupaj določajo, ali so podatki primerni za uporabo oziroma v kolikšni meri. Slaba kakovost podatkov lahko vodi do napačnih odločitev, zamujenih priložnosti in povečanih stroškov. Pomembno vlogo ima tudi izobraževanje zaposlenih in vzpostavitev organizacijske kulture, ki ceni kakovost podatkov kot skupno odgovornost. Ohranjanje in zagotavljanje kakovosti podatkov v organizaciji je stalen in neprekinjen proces.

Za izboljšanje kakovosti podatkov v organizaciji sta lahko v pomoč tudi Smernici 25 in 27:

Smernica 25 – Opredelitev in spremljanje dimenzij kakovosti podatkov: za vsak podatkovni vir je treba opredeliti ključne dimenzije kakovosti (točnost, celovitost, doslednost, zanesljivost, časovna ustreznost) in jih redno spremljati z ustreznimi metrikami.

Smernica 27 – Jasna določitev odgovornosti za kakovost podatkov: za vsak podatkovni vir je treba določiti osebe ali vloge (npr. skrbnik kakovosti podatkov), ki so zadolžene za spremljanje, poročanje in izboljševanje kakovosti.

Varnost podatkov pomeni proces ohranjanja zaupnosti, celovitosti in razpoložljivosti podatkov organizacije na način, ki je skladen s postavljenimi strategijo obvladovanja tveganj. Preden pride do incidenta, morajo organizacije imeti vzpostavljeno varnostno arhitekturo in načrt odzivanja. Ko se incident zgodi, morajo biti sposobne zaznati dogodek in se nanj ustrezno odzvati. Po incidentu mora biti organizacija sposobna učinkovito in uspešno obnoviti delovanje [17]. Varnost podatkov je ključna in nepogrešljiva sestavina učinkovitega upravljanja podatkov. Zaščita pred nepooblaščenim dostopom, razkritjem, spremembami ali uničenjem je bistvena za ohranjanje zaupanja, celovitosti in zanesljivosti podatkovnih sistemov. Za učinkovito zaščito je najprimernejši večplasten pristop. Ta vključuje stroge nadzore dostopa, omrežna in druga varnostna pravila, uporabo naprednih tehnologij, kot so požarni zidovi, protivirusna programska oprema, sistemi za zaznavanje vdorov ter sisteme za upravljanje

varnostnih informacij in dogodkov. Prav tako so pomembni redni varnostni pregledi, uporaba šifriranja in tehnik zakrivanja podatkov ter priprava načrtov za odzivanje na varnostne incidente [4, str. 37-38].

Praktični primer zagotavljanja varnosti podatkov iz Podatkovnih semantičnih smernic:

Primer 24 - Zaščita pred nepooblaščenim dostopom: javne evidence in storitve (npr. GURS ali ZZZS) morajo zagotavljati, da so osebni podatki državljanov, kot so zdravstveni podatki ali zemljiškoknjižni zapisi, zaščiteni pred nepooblaščenim dostopom. V ta namen uporabljajo večplastni sistem preverjanja pristnosti, šifriranje podatkov in redne varnostne preglede, skladno z zakonodajo, kot je npr. Zakon o varstvu osebnih podatkov (ZVOP-2).

3 Semantično opisovanje podatkov

Semantična interoperabilnost zagotavlja, da se natančna oblika in pomen izmenjanih podatkov ter informacij ohranita in razumeta med izmenjavami med stranmi, z drugimi besedami: »kar se pošlje, se tudi razume«. V okviru EIF² semantična interoperabilnost zajema tako semantične kot tudi sintaktične vidike [18]. Semantična interoperabilnost pomeni, da sistemi podatke ne le izmenjujejo, temveč jih tudi enako razumejo, kar je ključno za učinkovito delovanje javnih storitev. Semantično opisovanje podatkov temelji na semantičnih modelih, kot so ontologije, besednjaki in konceptualne sheme, ki omogočajo strukturiran opis informacij. Semantično opisani podatki omogočajo tudi vsebinsko povezovanje, ponovno uporabo in avtomatizirano obdelavo podatkov, kar zmanjšuje potrebo po ročnem usklajevanju pomenov in povečuje učinkovitost ter zanesljivost [4, str. 58-59].

3.1. Semantično modeliranje

Semantično modeliranje je zelo pomembno pri organizaciji informacij na spletu, saj pomaga pri medsebojni komunikaciji različnih akterjev in njihovih pogledov. Pri tem nam pomagajo semantični modeli (npr. konceptualni modeli, leksikalni modeli, sheme, besednjaki, ontologije), ki omogočajo strukturiran zajem kompleksnih konceptov. Semantični modeli so uporabni tako za opisovanje kot za razlago in napovedovanje. Semantično opisovanje temelji predvsem na standardu RDF, ki z uvedbo trojiškega zapisa (subjekt, predikat, objekt) predstavlja ogrodje za opis povezanih podatkov (angl. Resource Description Framework), RDFS, ki predstavlja shemo RDF (angl. Resource Description Framework Schema) ter OWL, ki znatno širi besednjak RDF in predstavlja spletni jezik za opisovanje ontologij (angl. Web Ontology Language). RDF je podatkovni standard oziroma jezik za formalno opisovanje virov in njihovih metapodatkov ter skupaj z RDFS in OWL predstavlja osnovne jezike semantičnega spleta za semantično modeliranje [4, str. 58–59].

Sposobnost zagotavljanja strukturirane semantike (strukturiran pomen) kot konteksta za različne agente in klepetalne robote je široko priznana kot ključni dejavnik v tekmi na področju generativne umetne inteligence, saj zmanjšuje »halucinacije« in izboljšuje natančnost. Konceptualno modeliranje podatkov je odlična metoda za ugotavljanje, kakšne naj bi bile entitete in odnosi.

Čeprav je mogoče z orodji umetne inteligence iz nestrukturiranih podatkov in različnih baz znanja izluščiti semantične koncepte, je veliko organizacijskega in poslovnega znanja implicitno znanje, ki ga imajo poslovni strokovnjaki v glavah. Konceptualno modeliranje je preizkušena metoda za pridobivanje tega implicitnega znanja. Tako lahko povežemo semantiko in umetno inteligenco na primer na način, da uporabimo priznane podatkovne standarde in obstoječe baze znanja na običajnih podatkovnih strukturah v neki organizaciji. Nato sledi konceptualno modeliranje, ki pokrije splošno implicitno znanje (tudi stroke) in posebno implicitno znanje organizacije. Standarde, baze znanja in konceptualne modele nato uporabimo v okviru AI agentov, ki kot semantični pomočniki obdelajo velike količine besedil in poiščejo podobnosti, ki jih dodajo konceptualni in drugi strukturirani semantiki [19].

² Evropski interoperabilnostni okvir (European Interoperability Framework)

Medtem ko standarde za podatke razvijajo v okviru konsenzualnih postopkov, obstajajo metodologije, ki pospešujejo uvajanje in usklajevanje teh uveljavljenih standardov v številnih in različnih zbirkah podatkov. Uporaba velikih jezikovnih modelov (angl. Large Language Models) bi lahko pospešila proces obdelave podatkov (za en red velikosti), saj avtomatizira naporne naloge usklajevanja, ki bi sicer zahtevale ročno delo.

Implementacija podatkovnih standardov na različne podatkovne nize se običajno opira na ročno urejanje in na tehnike leksičnega ujemanja ter na pravilih temelječega preslikavanje za standardizacijo podatkovnih elementov iz že obstoječih, strukturiranih podatkovnih naborov. Čeprav so te metode učinkovite za določena področja s standardiziranimi vhodnimi podatki, so po naravi omejene zaradi potrebe po obsežnem ročnem posredovanju in običajno pokrivajo le majhen podsklop področij. V nasprotju s tem bi pristop, podprt z umetno inteligenco, znatno zmanjšal ročno delo in se razširil tudi na nestandardizirane podatkovne nize, kar raziskovalcem in drugim omogoča, da se učinkoviteje osredotočijo na znanstveno raziskovanje namesto na obdelavo podatkov [20].

Tudi v organih javnega sektorja obstaja problem samostojnega upravljanja podatkov in podatkovnih produktov, opredelitve metrik in rešitev z umetno inteligenco. To se odraža v razpršeni pokrajini definicij, standardov in metodologij in se posledično kaže v bolj ali manj kompleksnih prilagoditvah podatkov in informacijskih sistemov za medsebojno povezovanje oziroma komuniciranje. Zaupanje v deljene podatke je nizko zaradi različnih definicij, opisov in predpostavk v podatkih. Prav zato so pomembni skupni standardi, definicije in upravljanje podatkov v širšem smislu, da se oblikuje enoten podatkovni ekosistem in se zagotovi ustrezna kakovost podatkov za odločanje, kar je s pojavom naprednih tehnologij postalo še bolj pomembno.

3.2. Metapodatki in opis podatkov

Metapodatki omogočajo razumevanje, iskanje in ponovno uporabo podatkov ter predstavljajo ključen element semantične interoperabilnosti. Spremljati morajo vsak nabor podatkov. Evropska komisija navaja, da metapodatki opredeljujejo in opisujejo značilnosti drugih podatkov ter se uporabljajo za izboljšanje poslovnega in tehničnega razumevanja podatkov ter s podatki povezanih procesov [21]. Metapodatki so dodatne informacije, ki pojasnjujejo pomen in strukturo podatkov ter podajajo podrobnosti o njihovi kakovosti, izvoru, licencah, dostopnosti in posodabljanju. Učinkovitost uporabe metapodatkov na več podatkovnih virih hkrati povečujejo metapodatki, ki niso odprti zalogi vrednosti, temveč imajo omejene zaloge vrednosti oziroma imajo omejene zaloge vrednosti nad razreda. Če so podatki urejeni hierarhično po razredih oziroma pojmi z medsebojnimi relacijami, je na primer enostavno narediti analizo ali drugo obdelavo podatkovnega vira, ki ugotavlja obstoj osebnih podatkov v različnih podatkovnih virih na podlagi podatkovnega modeliranja s temeljnim besednjakom za fizične osebe, ali npr. vseh pravnih oseb, ki imajo pooblaščen osebo za podatke (prek relacije vloge v organizaciji), na podlagi besednjaka organizacije W3C³. Če so na podlagi standardov modelirane tudi notranje strukture podatkovnih virov, lahko izvajamo še podrobnejše vsebinske analize in obdelave podatkovnih virov, na primer izbor vseh podatkov, ki obravnavajo promet ali prometne nesreče. Na podlagi pravil (npr. SHACL⁴), se v semantično obogatenih podatkovnih ekosistemih analize in obdelave izvajajo tudi samodejno, oblikovno in vsebinsko pravilno. Pravila lahko obravnavajo strukturne omejitve in zagotavljajo, da podatki ustrezajo določeni shemi ali modelu, kot so zahtevane lastnosti, dovoljene vrednosti lastnosti ali določene hierarhije razredov. Lahko obravnavajo omejitve vrednosti, ki jih lahko prevzamejo lastnosti, vključno s podatkovnimi tipi, razponi vrednosti in ujemanjem vzorcev. Nadalje gre lahko za omejitve kardinalnosti, torej opredeljevanje najmanjšega in največjega števila pojavov lastnosti, ali za logične omejitve, na primer uporaba logičnih pogojev za lastnosti in vrednosti, kot so enakost ali neenakost ter njihove kombinacije s pomočjo logičnih operatorjev [22].

Spodnji smernici opisujeta primer načina objave metapodatkov in nabora metapodatkov podatkov:

Smernica 47 – Metapodatki v strojno in človeku berljivi obliki: metapodatki naj se objavijo v dveh oblikah: kot del spletne strani ali v ločeni besedilni datoteki za potrebe človeških uporabnikov in v strojno berljivi obliki, kot sta RDF zapis

³ W3C - angl. World Wide Web Consortium, ki združuje posamezne delovne skupine, ki razvijajo standarde in smernice.

⁴ SHACL - angl. Shapes Constraint Language – jezik za preverjanje semantičnega opisovanja

ali JSON-LD, ki predstavlja zapis objektov JavaScript za povezane podatke (angl. JavaScript Object Notation for Linked Data).

Smernica 48 – Opisni metapodatki za nabor podatkov in njegove distribucije: za vsak nabor podatkov naj se zagotovijo metapodatki, kot so naslov, opis, ključne besede, datum objave, kontaktni podatki, izdajatelj, prostorska pokritost, časovno obdobje, kategorija [4, str. 63–64].

3.3. Identifikatorji

Identifikatorji so temeljni gradniki informacijskih sistemov, na spletu pa v tej vlogi prevladujejo naslovi URI, ki so globalno enolični identifikatorji (angl. Uniform Resource Identifier) in se uporabljajo za identifikacijo virov. Uporaba globalnih identifikatorjev (URI) omogoča enolično identifikacijo virov in njihovo povezovanje v širše podatkovne ekosisteme. Pomembno je, da so naslovi URI uporabniku prijazni in lahko berljivi.

Uporabo metapodatkov opisujeta smernici 50 in 51.

Smernica 50 – Nespremenljivi URI identifikatorji: pri identifikaciji nabora podatkov je treba uporabiti nespremenljive URI naslove.

Smernica 51 – Ponovna uporaba nespremenljivih URI identifikatorjev: pri objavi podatkov, kjer se sklicujemo na že opredeljene koncepte, dosledno uporabljamo obstoječe, nespremenljive URI identifikatorje. Takšna uporaba krepi učinke mrežnega povezovanja, kar je osrednja ideja povezanih podatkov [4, str. 65–66].

3.4. Podatkovni slovarji

Za semantično interoperabilnost je ključna uporaba podatkovnih slovarjev, ki zmanjšujejo nejasnosti in omogočajo povezovanje podatkov. Podatkovni slovarji opredeljujejo koncepte in povezave med njimi ter jih poznamo v različnih oblikah, kot so ontologije, nadzorovani besednjaki, slovarji sopomenk in druge

Smernica 53 priporoča uporabo konceptov iz obstoječih besedišč.

Smernica 53 – Ponovna uporaba standardiziranih slovarjev: pri objavi podatkov in metapodatkov naj se uporabljajo koncepti iz obstoječih in po možnosti standardiziranih besedišč. Priporoča se uporaba standardiziranih slovarjev, kot so denimo: besednjak podatkovnih katalogov – DCAT (angl. Data Catalog Vocabulary), Dublin Core, shema za enostaven sistem organizacije znanja – SKOS (Simple Knowledge Organization System) in CNB (Centralni nadzorovani besednjak (javne uprave)). S tem se poveča skladnost z obstoječimi praksami v skupnosti, zmanjša podvajanje in spodbuja ponovna uporaba [4, str. 77].

3.5. Licenca, izvor in kakovost podatkov

Licenca, izvor in kakovost podatkov so ključni elementi metapodatkov, saj omogočajo uporabnikom razumevanje pravic uporabe, porekla in zanesljivosti podatkov. Objavljeni podatki morajo biti razumljivi ter pravno in vsebinsko zanesljivi. To pomeni, da morajo uporabniki jasno vedeti, pod kakšnimi pogoji lahko podatke uporabljajo, od kod podatki izvirajo in kako kakovostni so.

Smernice 68, 69 in 70 opisujejo ključne vidike objave podatkov, ki vplivajo na njihovo pravno veljavnost, sledljivost in uporabnost.

Smernica 68 – Licenca podatkov: Pri objavi podatkov je treba zagotoviti jasno informacijo o licenci, ki določa pogoje za uporabo, deljenje in ponovno uporabo podatkov. Licenca naj bo jasno določena za celoten nabor podatkov ali posamezne distribucije, če se pravice razlikujejo.

Smernica 69 – Izvor podatkov: Pri objavi podatkov je treba jasno navesti izvor podatkov, vključno z vsemi spremembami in transformacijami, ki so bile izvedene.

Smernica 70 – Kakovost podatkov: Pri objavi podatkov se navedejo informacije o njihovi kakovosti, kar opišemo z različnimi dimenzijami (npr. točnost, ažurnost, dostopnost, celovitost) [4, str. 77].

4 Zaključek

Podatki nastajajo ves čas, podnevi in ponoči, tako v povezavi z ljudmi kot s stvarmi. Na drugi strani ves čas prihajajo nove napredne tehnologije, ki odpirajo nova področja in neslutene možnosti obdelave, izrabe in izkoriščanja v različne namene. Možnosti obdelave podatkov se odpirajo povsod: tako na ravni posameznika kot na ravni organizacij (zasebnega in javnega sektorja) in širše družbe. V javni upravi, kjer skrbimo za velike (kritične) informacijske sisteme ter hranimo in obdelujemo velike količine pravno varovanih podatkov, je skrb za zasebnost, varnost in kakovost posebnega pomena.

Z novimi tehnologijami so se tudi v javni upravi možnosti izrabe podatkov za odločanje in oblikovanje politik izjemno povečale. Da bi jih lahko v celoti izkoristili, je izjemnega pomena, da so ti urejeni, skladni in upravljeni. S tem namenom so nastale Podatkovne semantične smernice, ki predstavljajo celovit okvir za upravljanje in semantično opisovanje podatkov v javni upravi.

Uvajanje semantične interoperabilnosti pa ni enkratni dogodek, temveč stalen proces, ki zahteva postopno prilagajanje obstoječih podatkovnih virov in sistemov. Zato bo ključno, da se njeno uvajanje izvaja sistematično in dolgoročno, ob razvoju kompetenc ter tesnem sodelovanju z vsemi deležniki.

Literatura

- [1] <https://explodingtopics.com/blog/data-generated-per-day>, »Amount of Data Created Daily«, obiskano 21. 7. 2026.
- [2] Strategija digitalnih javnih storitev, Ministrstvo za javno upravo, Ljubljana, 2023.
[Strategija_digitalnih_javnih_storitev_2030-1.pdf](#) obiskano 21. 7. 2026.
- [3] <https://digital-strategy.ec.europa.eu/en/policies/europes-digital-decade>, »Europe's Digital Decade«, obiskano 21. 7. 2026.
- [4] »Podatkovne semantične smernice«, Ministrstvo za digitalno preobrazbo, Ljubljana, 2026,
<https://nio.gov.si/products/podatkovne%2Bsemanticne%2Bsmernice>, obiskano 20. 7. 2026.
- [5] FURLAN Martina, JESENKO Miha, KERN PIPAN Karmen, ŽITNIK Slavko, LAVBIČ Dejan »Podatkovne semantične smernice«, Dnevi slovenske informatike 2026: Zbornik 33. konference (in prezentacija), Portorož, Slovensko društvo informatika, str. 66.
- [6] DAVENPORT Thomas H., REDMAN Thomas C. »How AI Is Improving Data Management«, MIT,
<https://sloanreview.mit.edu/article/how-ai-is-improving-data-management>, obiskano 15. 7. 2026.
- [7] <https://www.gartner.com/en/data-analytics/topics/data-governance>, Gartner, obiskano 13. 7. 2026.
- [8] Zakon o varstvu osebnih podatkov (ZVOP-2), Uradni list RS, št. 163/22, 40/25 – ZInFV-1 in 10/26 – ZP-1L.
- [9] Zakon o dostopu do informacij javnega značaja (ZDIJZ), Uradni list RS, št. 51/06 – uradno prečiščeno besedilo, 117/06 – ZDavP-2, 23/14, 50/14, 19/15 – odl. US, 102/15, 7/18, 141/22 in 40/25 – ZInFV-1.
- [10] Zakon o izvajanju Uredbe (EU) o evropskem upravljanju podatkov (ZIUUEUP), Uradni list RS, št. 102/24.
- [11] Uredba (EU) 2024/903 Evropskega parlamenta in Sveta z dne 13. marca 2024 o določitvi ukrepov za visoko raven interoperabilnosti javnega sektorja v Uniji (Akt o interoperabilni Evropi) (UL L št. 2024/903 z dne 22. 3. 2024).
- [12] WILKINSON Mark D., DUMONTIER Michel, AALBERSBERG Ijsbrand Jan in drugi »The FAIR Guiding Principles for Scientific Data Management and Stewardship«, Scientific Data, letnik 3, številka 1, 2016.
<https://doi.org/10.1038/sdata.2016.1>, obiskano 21. 7. 2026
- [13] <https://online.hbs.edu/blog/post/data-ethics>, Harvard Business School, »5 Principles of Data Ethics for Business«, obiskano 15. 7. 2026
- [14] <https://standards.education.gov.uk/standard/data-ethics>, »Data Ethics«, obiskano 21. 7. 2026.
- [15] WING Jeannette M., »The Data Life Cycle«, <https://hdr.mitpress.mit.edu/pub/577rq08d/release/4>, obiskano 16. 7. 2026
- [16] <https://www.gov.uk/government/news/what-is-data-quality>, »What is Data Quality«, obiskano 16. 7. 2026.

- [17] <https://www.nccoe.nist.gov/data-security>, NIST, obiskano 20. 7. 2026
- [18] <https://interoperable-europe.ec.europa.eu/collection/iopeu-monitoring/glossary/term/semantic-interoperability>, Interoperable EuropePortal, obiskano 20. 7. 2026.
- [19] KORPELA Juha, Modeling Semantics: »How Data Models and Ontologies Connect to Build Your Semantic Foundations«, <https://moderndata101.substack.com/p/semantic-foundations-with-data-models-or-ontology>, obiskano 24. 7. 2026.
- [20] LONG Rodney Alan, BALLARD Shannon, SHAH Syed. et al. »A New AI Assisted Approach Aligns Data Standards and Accelerates Interoperability in Biomedical Research«. *npj Digital Medicine*. (2026). <https://doi.org/10.1038/s41746-026-02795-z>, obiskano 24. 7. 2026.
- [21] <https://interoperable-europe.ec.europa.eu/collection/digital-ready-policymaking/glossary/term/metadata>, Interoperable Europe Portal, obiskano 21. 7. 2026
- [22] DOUBLEDAY Kevin, Fluree, PBC. »What Is SHACL? (With Examples)« <https://medium.com/fluree/what-is-shacl-with-examples-2697f659d465>, obiskano 24. 7. 2026.

CODORQ: ogrodje za orkestracijo razvoja programske opreme z različnimi agentskimi orodji

Tomaž Kolmanič

WagaLabs d.o.o., Maribor, Slovenija
tomaz.kolmanic@wagalabs.com

Prispevek predstavlja CODORQ, lokalno-prvo ogrodje za orkestracijo razvoja programske opreme z različnimi agentskimi orodji. Izhaja iz ugotovitve, da pri sočasnem delu več agentov omejitev ni le zmogljivost modelov, temveč predvsem odsotnost trajnega, preverljivega koordinacijskega sloja. CODORQ prek lokalnega strežnika MCP povezuje seje orodij Codex CLI, Claude Code in Antigravity CLI z repozitoriji, ločenimi Git worktreeji, terminali ter skupnim modelom nalog, vlog, statusov, pregledov, rezultatov QA in spomina. Sistem podpira ločitev dolžnosti med izvedbo, pregledom in QA, spremlja identiteto in živost sej ter človeka obravnava kot vodjo izvedbe. Trajni zapisi omogočajo obnovo konteksta po strnitvi ali ponovnem zagonu brez odvisnosti od terminalskega prepisa. Članek opiše arhitekturo, večstopenjski trajni spomin, statuse nalog, operatorski spletni vmesnik in obrambne mehanizme za ponovne zagone, neodzivne seje, preobsežne odgovore MCP in sočasne spremembe. Demonstracijski primer razvoja manjše komponente uporabniškega vmesnika pokaže sled od zahteve prek dokaznega commita in neodvisnega pregleda do rezultata QA. Prispevek ne dokazuje povečanja produktivnosti, temveč prikazuje, kako je mogoče večagentsko delo narediti sledljivo, obnovljivo in operatorsko vodljivo.

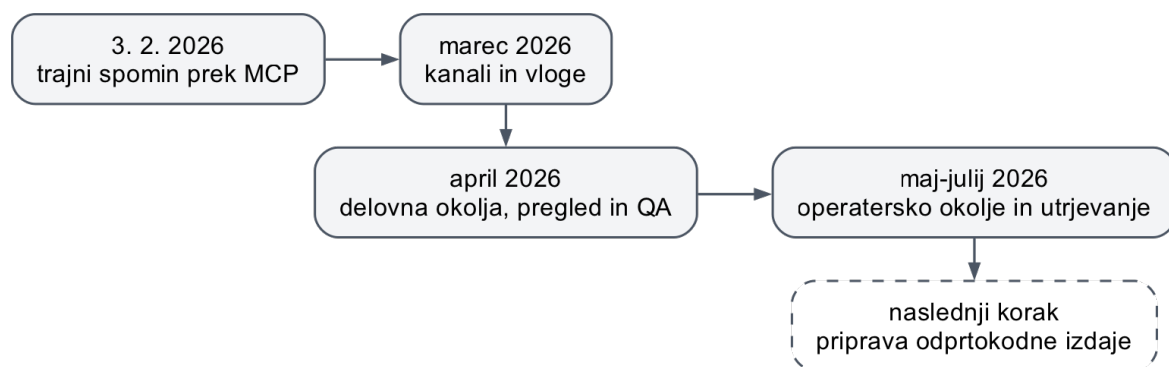
Ključne besede:

agentska orodja
orkestracija razvoja programske opreme
Model Context Protocol
večagentski razvoj programske opreme
zagotavljanje kakovosti

1 Uvod

Sodobni pomočnik AI za razvoj programske opreme lahko pregleda repozitorij, spremeni več datotek, zažene teste in pripravi razlago rešitve. Težave se pojavijo, ko hkrati poteka več takšnih sej. Ena pripravlja načrt, druga implementira rešitev, tretja pregleduje spremembo, četrta pa izvaja preverjanje kakovosti. Vsaka ima svoj kontekstni prostor in življenjski cikel. Terminal se lahko zapre, seja se lahko po strnitvi konteksta nadaljuje z novo tehnično identiteto, sprememba pa lahko ostane v ločenem delovnem drevesu Git (worktree) brez jasno določenega lastnika. Če je edini zapis o stanju pogovor v terminalu, mora operater ročno ugotavljati, kdo dela na kateri nalogi in kateri rezultat je dovolj zanesljiv za naslednji korak.

Razvoj sistema CODORQ se ni začel kot poskus vzpostavitve večagentskega sistema za razvoj programske opreme. Začetna zamisel je bila precej ožja: kako ohraniti uporabne odločitve, preference, pridobljene izkušnje in predaje tudi takrat, ko se pogovorno okno zapre ali delo prevzame druga seja. Prva različica sistema je zato predstavljala sloj trajnega spomina, dostopen prek protokola MCP. Praktična uporaba je kmalu pokazala, da sam trajni spomin ne zadošča. Zapisana odločitev namreč ne pove, ali je naloga pripravljena za izvedbo, kdo jo lahko prevzame, ali je bila sprememba pregledana in ali so bila zahtevana preverjanja dejansko izvedena. Sloj trajnega spomina se je zato postopoma razvil v izvajalno okolje za upravljanje nalog, vlog, pregledov, preverjanja kakovosti, terminalskih sej, usmerjanja ter operatorskega nadzora. Ta razvoj je mogoče strniti v prehod **od spomina k orkestraciji**. Slika 1 prikazuje ključne stopnje tega razvoja.



Slika 1: Strnjena časovnica razvoja CODORQ-a od trajnega spomina do orkestracijskega okolja. Zadnji korak označuje načrt, ne že izvedene javne objave.

Osrednja teza prispevka je, da zanesljivost večagentskega razvoja ni odvisna zgolj od zmogljivosti jezikovnega modela, temveč od celotnega delovnega sistema. Ta mora zagotavljati trajno stanje, jasno opredeljene prehode, izolirane spremembe, preverljive dokaze in odločitve človeškega vodje izvedbe. CODORQ to uresničuje kot lokalno-prvo koordinacijsko okolje, zgrajeno okoli obstoječih agentskih orodij. Modelu ne predpisuje načina razmišljanja niti ne nadomešča njegovega uporabniškega vmesnika. Namesto tega vsaki konkretni seji zagotavlja enoten nabor orodij za delo z delovnimi okolji, nalogami, trajnim spominom, pregledi, rezultati preverjanja kakovosti in kratkimi usmerjevalnimi sporočili.

Prispevek zasleduje štiri cilje. Prvič, predstavi arhitekturo lokalnega strežnika MCP, ki povezuje heterogena agentska orodja, repozitorije, terminale in trajno shrambo. Drugič, opiše delovni model, v katerem naloge, ugotovitve pregleda, rezultati preverjanja kakovosti in zapisi spomina predstavljajo samostojne poizvedljive entitete trajnega modela stanja. Tretjič, pojasni operativne varovalke, kot so ločitev dolžnosti, stabilizacija identitete, večstopenjsko zaznavanje neodzivnosti ter uporaba Git worktreejev. Četrtič, na dejanskem toku spremembe uporabniškega vmesnika prikaže pot od izvedbe prek pregleda do QA ter možnost naknadne rekonstrukcije dokazov.

Članek ne dokazuje, da CODORQ razvoj pospeši za določen odstotek ali da izboljša kakovost v primerjavi z drugimi ogrodji. Za takšne sklepe bi bilo treba vnaprej določiti primerjalno metodologijo, izhodiščne pogoje in analizirati več razvojnih projektov. Predstavljeni kazalniki zato predstavljajo opisna dejstva enega razvojnega primera. Njihov namen ni merjenje splošne produktivnosti, temveč prikaz sledljivosti in izvedljivosti predlaganega procesa.

2 Ozadje in sorodna dela

Orodja agentov AI za razvoj programske opreme danes pokrivajo več različnih ravni avtomatizacije. Aider je pomočnik AI za programiranje v paru [1]. Codex CLI [2], Claude Code [3] in podobni odjemalci gredo korak dlje, saj lahko samostojno pregledujejo repozitorij, uporabljajo razvojna orodja in izvajajo ukaze. Takšna orodja so učinkovita znotraj posamezne seje, ne rešujejo pa vprašanj koordinacije med več sočasnimi sejami, kot so določanje lastništva nalog, sledljivost dela in predaja rezultatov.

Raziskovalna večagentska ogrodja se osredotočajo na drugačen vidik problema. ChatDev uporablja specializirane agente in komunikacijski tok za posamezne faze razvoja programske opreme, od načrtovanja do testiranja [4]. MetaGPT uvaja vloge in standardizirane operativne postopke kot mehanizem za strukturiranje sodelovanja med agenti [5]. SWE-agent raziskuje povezavo med jezikovnim modelom in računalniškim okoljem pri reševanju nalog programskega inženirstva [6]. Ta dela pomembno prispevajo k razumevanju načrtovanja in sodelovanja agentov, vendar obravnavajo predvsem vedenje večagentskega sistema. Prispevek CODORQ-a se osredotoča na drugo raven – trajno koordinacijo obstoječih terminalskih orodij agentov AI.

Najbližje primerjalne rešitve predstavljajo ogrodja za orkestracijo agentskih sistemov. AutoGen je uveljavljeno ogrodje za gradnjo aplikacij iz več sodelujočih agentov; predstavljeno je bilo na konferenci COLM 2024 [7]. Ob nastanku tega članka je projekt v vzdrževalnem načinu; Microsoft za nove projekte priporoča naslednika Microsoft Agent Framework [13], [14]. Podobno LangGraph in CrewAI ponujata programske abstrakcije za opis grafov oziroma skupin agentskih izvajanj. CODORQ ni zasnovan kot novo programsko ogrodje ali SDK za definiranje vedenja agentov. Njegov namen je vzpostaviti skupni lokalni koordinacijski sloj nad heterogenimi CLI-odjemalci različnih ponudnikov. Ta sloj zagotavlja trajno stanje, obnovljivo identiteto agentskih sej, jasno ločitev dolžnosti (angl. *separation of duties*, SoD), spremljanje živosti ter vključuje človeka kot vodjo izvedbe. Za delovanje sistema zato ni bistveno, ali posamezno orodje temelji na namenskem agentskem ogrodju ali predstavlja zgolj terminalski odjemalec. Ključno je, da je seja registrirana, ima določeno vlogo in uporablja skupni trajni delovni model.

Lokalno-prva zasnova CODORQ-a sledi načelu, da mora biti primarna kopija uporabnikovega delovnega stanja pod uporabnikovim nadzorom ter uporabna tudi brez odvisnosti od osrednje aplikacijske storitve [8]. V tem kontekstu izraz *lokalno-prvo* ne pomeni, da mora biti tudi jezikovni model izveden lokalno ali da sistem ne uporablja omrežja. Pomeni predvsem, da koordinacijski sloj, trajni zapisi, povezave s terminali in operatorski vmesnik delujejo v lokalnem okolju. Sam klic jezikovnega modela je lahko še vedno izveden prek zunanje storitve, odvisno od izbranega agentskega orodja.

Model Context Protocol (MCP) je bil v zborniku OTS 2025 že predstavljen v članku Janija Šumaka [9], zato ta prispevek ne ponavlja podrobnega opisa protokola. Za arhitekturo CODORQ-a sta bistveni dve lastnosti uradne specifikacije: standardizirana komunikacija med odjemalcem in strežnikom ter jasno opredeljena orodja z validiranimi vhodi [10]. MCP sam po sebi ne določa večagentskega procesa, temveč predstavlja komunikacijsko podlago. CODORQ ga uporablja kot skupni vstopni mehanizem v trajni delovni model, kar različnim odjemalcem omogoča izvajanje enakih operacij brez razvoja ločenih integracij za posamezne družine orodij.

Arhitekturna neodvisnost sistema ne pomeni, da so vse družine agentskih orodij v vsakem trenutku enako podprte. Njihove zmogljivosti in življenjski cikli se hitro spreminjajo. V povzetku so kot primeri navedeni Codex CLI, Claude Code in Antigravity CLI [11]. Googlovo orodje Antigravity CLI temelji na modelih Gemini. Ob nastanku tega članka so zato neposredno podprta orodja Codex CLI, Claude Code in Antigravity CLI, poteka pa razširitev

podpore na orodje OpenCode. Prispevek namenoma ne primerja kakovosti posameznih modelov niti ne predpostavlja trajne razpoložljivosti posameznega odjemalca.

3 Zahteve in načela zasnove

Zahteve sistema niso nastale zaradi želje po čim širšem naboru funkcionalnosti, temveč kot odgovor na ponavljajoče se probleme pri delu z več terminalskimi sejami. Tabela 1 povezuje opaženi praktični problem, iz njega izpeljano sistemsko zahtevo in mehanizem, s katerim jo CODORQ uresničuje.

Tabela 1: Od praktičnega problema do zahteve in mehanizma CODORQ-a

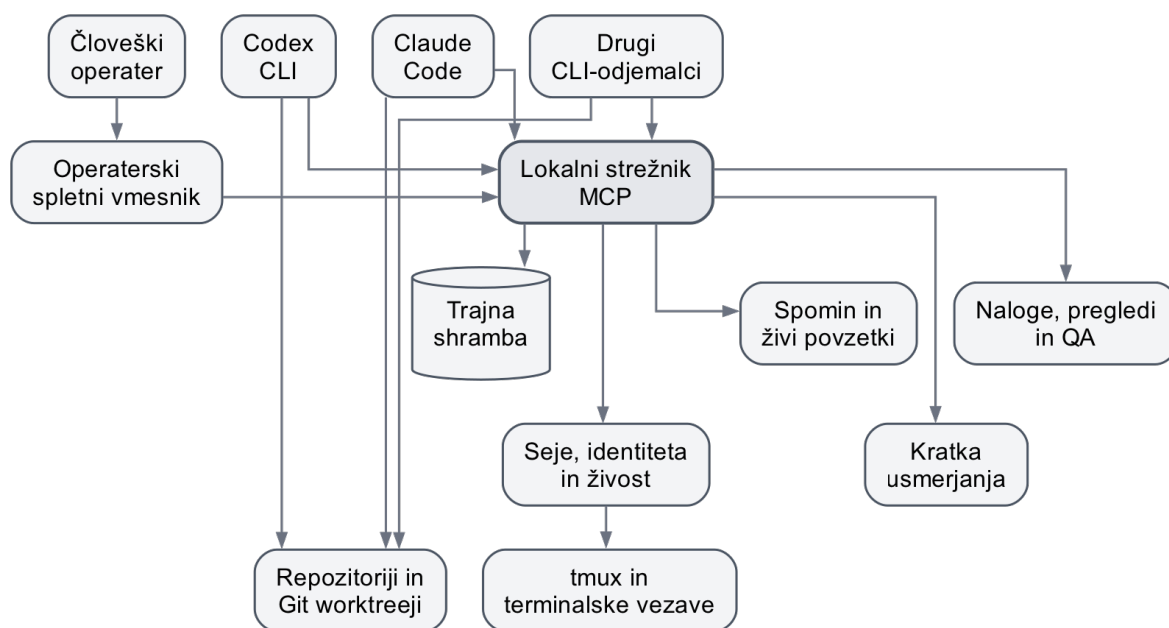
Problem	Zahteva	Mehanizem
Kontekst izgine ob zaprtju ali strnitvi seje	Trajna kontinuiteta	Spomin, kontrolne točke, živi povzetki, predaje in strukturirano stanje nalog
Več agentov si različno razlaga trenutno stanje	En poizvedljiv vir resnice	Naloge, ugotovitve pregleda, QA in seje v trajni shrambi
Sprememba nima jasnega lastnika ali vloge	Eksplisitna odgovornost	Vloge, dodelitve, zahtevane vloge, statusni prehodi in dogodki
Avtor sam potrdi svojo spremembo	Ločitev dolžnosti	Ločene izvedbene faze ter faze pregleda in QA z evidentirano sprostitevjo
Sočasni agenti spreminjajo isto delovno kopijo	Izolacija sprememb	Ločeni Git worktreeji in veja na nalogo
Terminal izgine, naloga pa ostane navidezno zasedena	Dokazljiva živost in okrevanje	Heartbeat, terminalska vezava, watchdog, reaper in obnovitev dodelitve
Odgovori MCP porabijo preveč kontekstnega prostora	Omejen in namenski kontekst	Kompaktni pogledi, zgornje meje seznamov in postopno razkrivanje podrobnosti
Operater ne ve, kje je potreben poseg	Nadzorni pogled in usmerjene akcije	Spletni vmesnik, kategorije pozornosti, zgodovina in kratka obvestila

Iz zahtev, predstavljenih v tabeli, izhajajo temeljna načela zasnove sistema. Prvo načelo je trajnost: komunikacijski kanal ali terminalska seja lahko posredujejo informacijo o dogodku, ne smeta pa predstavljati njenega edinega zapisa. Drugo načelo je eksplisitnost: vsaka naloga ima opredeljena merila sprejemljivosti in testni načrt, vsaka ugotovitev pregleda pa določen opis, resnost in stanje. Tretje načelo je najmanjša potrebna avtoriteta. Posamezna seja prejme le tista orodja in vloge, ki jih potrebuje za izvedbo naloge, občutljivi prehodi pa vključujejo preverjanje identitete izvajalca. Četrto načelo je postopnost. Enostavne naloge ne zahtevajo večagentskega procesa, zahtevnejših izvedb pa ni mogoče izvesti mimo predvidenega procesa. Peto načelo je človeški nadzor. Avtomatizacija lahko predlaga nadaljnje korake, usmerja izvajanje ali pomaga pri obnovi dela, končna odločitev in odgovornost pa ostaneta pri operaterju.

4 Arhitektura in trajni model stanja

4.1. Koordinacijski in izvajalni sloj

CODORQ je zasnovan kot sistem dveh med seboj povezanih slojev. Koordinacijski sloj povezuje lokalni strežnik MCP, trajno shrambo, domenske storitve in spletni vmesnik. Izvajalni sloj sestavljajo agentski CLI-odjemalci, terminalske seje oziroma seje tmux, repozitoriji, Git worktreeji ter razvojna orodja, ki jih agenti kličejo. MCP predstavlja vstopno točko iz izvajalnega v koordinacijski sloj. Razmerje med slojema in povezanimi komponentami prikazuje slika 2.



Slika 2: Logične komponente CODORQ-a.

Tehnološki sklad sestavljajo Node.js 22, TypeScript in uradni SDK za MCP. Pot za lokalni razvoj in teste uporablja better-sqlite3, za resnejše večagentsko oziroma operatersko delo pa je podprta pot s PostgreSQL. Spletni del je zgrajen z Expressom in Reactom. Isti jezik tako pokriva CLI, strežnik MCP, spletni programski vmesnik (API) in odjemalski vmesnik, oba podatkovna adapterja pa ohranjata isti domenski model.

Merilni posnetek tik pred vključitvijo članka v repozitorij je obsegal 523 commitov od prvega commita 3. februarja 2026. Imenik `src/` je vseboval 162 datotek TypeScript s 75.559 vrsticami kode, strežnik pa je objavljajl 62 orodij MCP. To so opisni kazalniki obsega in zgodovine razvoja, ne merilo kakovosti ali produktivnosti. Vso programsko kodo sistema so napisali agenti AI pod nadzorom človeškega operaterja; od vzpostavitve osnovnih mehanizmov naprej je bil razvoj pretežno koordiniran prek CODORQ-a samega (pristop *dogfooding*).

Trajna shramba podpira različne tipe stanja: delovna okolja, registrirane agente in njihove seje, terminalske vezave, naloge in odvisnosti, ugotovitve pregleda, rezultate QA, dokumente spomina, kanalska sporočila in dogodke. Ne glede na izbrani podatkovni adapter je ključna enaka semantika: zapis je poizvedljiv tudi po koncu posamezne terminalske seje.

Vsak registrirani agent ima v delovnem okolju eno ali več vlog, na primer razvijalec, pregledovalec ali izvajalec QA. Vloga določa, katere naloge sme agent prevzeti in katere statusne prehode sme sprožiti. Posebna vloga je koordinacijski vodja: agent, ki cilje operaterja pretvarja v sledljive naloge, jih usmerja ustreznim vlogam in spremlja pretok dela do zaključka. Človeški operater zato praviloma komunicira z vodjo, ne z vsakim izvajalcem posebej.

Poleg trajnih zapisov CODORQ ponuja tudi kanale: sprotna sporočila med udeleženci delovnega okolja, ki se prenašajo prek lokalne vtičnice (angl. *Unix domain socket*).

Kanali imajo drugačno funkcijo kot trajna shramba. Uporabljajo se za kratko usmerjanje pozornosti, prebujanje ustreznega udeleženca in prenos sprotnih signalov v vmesnik. Uspešno poslano sporočilo še ne pomeni, da je naloga prevzeta ali končana. Avtoritativno stanje ostane v nalogi, ugotovitvi, rezultatu QA ali zapisu spomina.

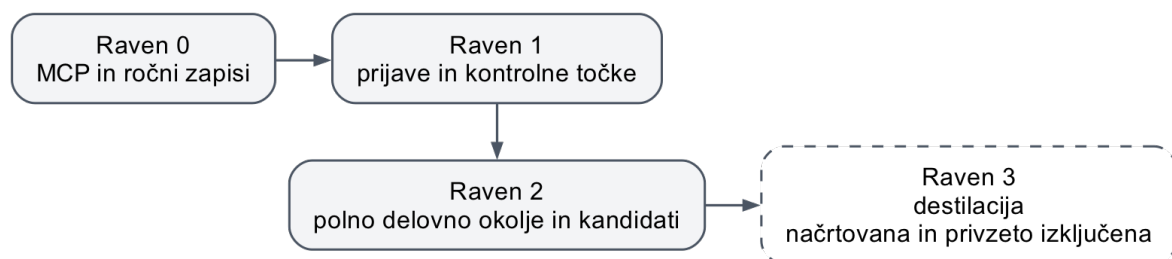
4.2. Trajni spomin kot prvovrstna komponenta

Trajni spomin je izvorna ideja sistema CODORQ in ostaja samostojen podsistem tudi potem, ko je projekt prerasel v orkestracijsko okolje. Njegov namen ni shraniti celotnega prepisa vsakega pogovora. Uporaben trajni spomin je izbran, strukturiran in umeščen v obseg. V njem se hranijo:

- pravila repozitorija in ponavljajoče se varnostne omejitve;
- preference operaterja ali ekipe;
- sprejete arhitekturne in izvedbene odločitve;
- naučene lekcije in ponavljajoči se problemi;
- predaje med sejami;
- kontrolne točke pred strnitvijo konteksta;
- strnjeni povzetki opravljenega dela in odprtih vprašanj.

Zapisi so razdeljeni v javne obsege, med drugim repo, global, preferences, insights, lessons, session in collaboration. Obseg zmanjša nevarnost, da bi lokalna odločitev enega repozitorija postala splošno pravilo za nepovezano delo. Iskanje po polnem besedilu deluje brez dodatne storitve. Semantično iskanje je mogoče vključiti z zunanjo storitvijo za izračun vektorskih vdelav (angl. *embeddings*) in povratnim polnjenjem obstoječih zapisov, vendar ni privzeta lastnost.

Način uporabe trajnega spomina je mogoče opisati s štirimi stopnjami. Na ravni 0 je na voljo strežnik MCP, agent pa orodja za trajni spomin kliče ročno. Raven 1 doda redne prijave, kontrolne točke in zajem ob koncu ali pred strnitvijo seje. Raven 2 spomin poveže s polnim delovnim okoljem: nalogami, živimi povzetki, pregledi, QA in pregledljivimi kandidati za prečiščeni spomin. Najvišja raven 3 je **destilacija** (angl. *distillation*), pri kateri bi sistem iz zajetih dokazov pripravil sledljive predloge za trajnejše lekcije, odločitve ali nadomestitve zastarelih zapisov. Ta raven je načrtovana in delno izvedena, samodejni urnik pa je privzeto izključen. Zaporedje stopenj prikazuje slika 3.

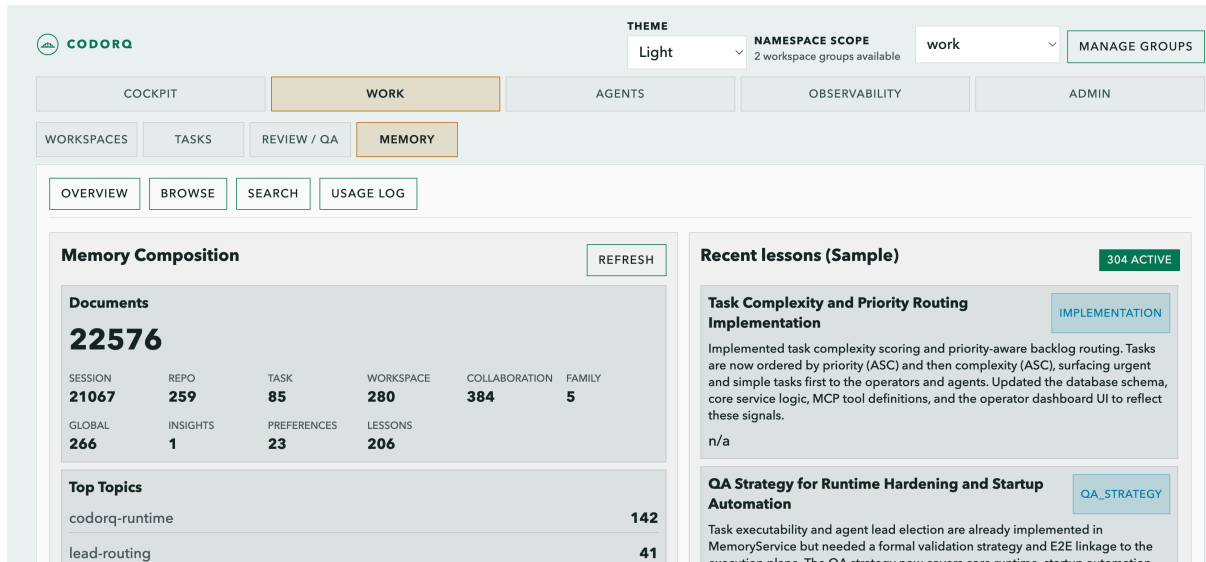


Slika 3: Stopnje uporabe trajnega spomina. Višja stopnja razširi prejšnjo.

Zapis spomina in izvršljivo stanje nista ista stvar. Lekcija »pri spremembi statusnega modela preveri tudi dokumentacijo« je primerna za spomin. Podatek, da določena naloga trenutno čaka QA, sodi v nalogo in evidenco QA. Živi povzetki, ugotovitve, rezultati QA in kontrolne točke so zato pri aktivnem delu pomembnejši od proste zbirke zapiskov. S tem se zmanjša možnost, da bi agent iz starega zapisa spomina sklepal o trenutnem lastništvu naloge.

CODORQ podpira tudi pregledljive kandidate za prečiščeni spomin. Predlog lekcije se lahko sprejme, zavrne ali nadomesti, preden postane del omejenega zagonskega konteksta. Zaznavanje protislovij in popolnoma samodejno razreševanje konfliktov v spominu ostajata odprta problema.

Slika 4 prikazuje operatorski povzetek sestave trajnega spomina in vzorec nedavnih lekcij. Števci predstavljajo stanje lokalne zbirke ob zajemu in niso merilo kakovosti ali stopnje ponovne uporabe zapisov.



Slika 4: Operatorski pregled sestave trajnega spomina in vzorca nedavnih lekcij. Posnetek je omejen na zgornji del pogleda, ki prikazuje vrste dokumentov, skupne teme in izbrane lekcije.

4.3. Kontinuiteta po strnitvi ali ponovnem zagonu

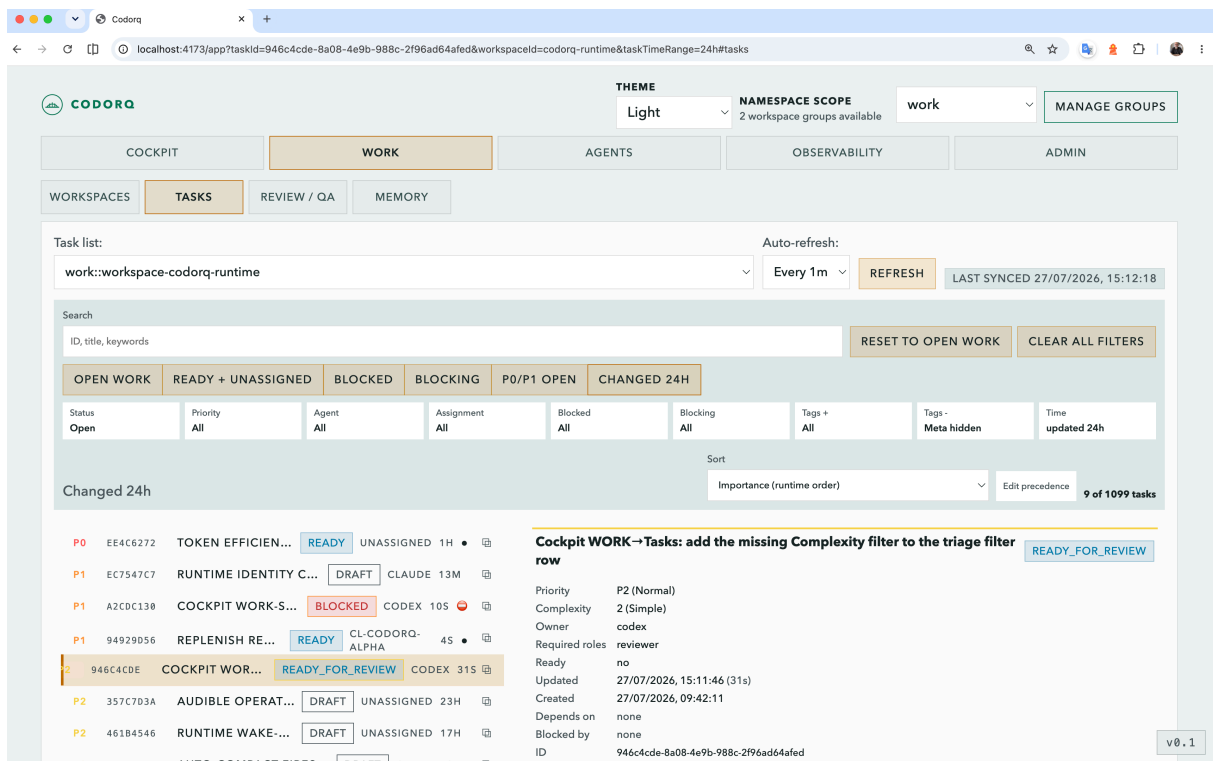
Pred strnitvijo obsežnega konteksta lahko seja zapiše kontrolno točko: aktivno nalogo, stanje, spremenjene datoteke, naslednje korake, odprta vprašanja in predaje. Ko se seja nadaljuje, ne potrebuje ponovitve celotnega pogovora. Naloži lahko kratek pogled `workspace_my_work`, podrobnosti aktivne naloge, odprte ugotovitve, zadnji rezultat QA in ustrezne poudarke trajnega spomina.

Primer obnovljenega delovnega stavka je lahko zelo kratek: »Nadaljujem popravek za stabilno identiteto. Implementacija je pregledana, QA pa je našel zastarel opis prioritet v dokumentaciji; popravi opis, zaženi ciljne teste in vrni nalogo v QA.« Tak zapis združuje trajno stanje več entitet, ne starega prepisa. To je operativna razlika med spominom kot arhivom in spominom kot delom obnovljivega procesa.

4.4. Repozitoriji, terminali in vmesnik

Vsaka izvedbena naloga lahko dobi svojo vejo in Git worktree [12]. S tem agenti ne spreminjajo istega delovnega imenika, pregledovalec pa lahko preveri točno določen commit. Terminalska vezava poveže registriranega agenta, izvajalno sejo in okno tmux. Trenutni dokaz iz okolja tmux ima pri obnovi prednost pred zastarelim shranjenim naslovom ali staro vezavo, vendar sistem pri dvomu ne sme izdelati nove identitete samo zato, da bi nadaljeval.

Operatorski spletni vmesnik na podlagi teh zapisov pripravi pregled delovnega okolja. Prikaže aktivno delo, čakalne vrste, blokade, zahteve za pregled, QA, agente in signale, pri katerih je potreben poseg. Slika 5 prikazuje dejanski lokalni pogled WORK → Tasks s filtri, seznamom nalog in inšpektorjem izbrane naloge. Posnetek prikazuje isti demonstracijski primer, ki je analiziran v 7. poglavju; osebni podatki, poverilnice in lokalne poti niso vključeni.



Slika 5: Operatorski pogled nalog v delovnem okolju. Posnetek je bil zajet iz lokalno delujočega sistema; prikazuje filtre, seznam nalog in izbrano nalogo za dodajanje filtra Complexity v stanju ready_for_review.

5 Izvedbeni tok in nadzor operaterja

5.1 Stopnje formalnosti

CODORQ ne predpisuje, da se vsaka zahteva obravnava kot večagentski projekt. Obseg koordinacije se prilagaja zahtevnosti dela. Tabela 2 prikazuje petstopenjski model formalnosti.

Tabela 2: Stopnje formalnosti izvedbe v CODORQ-u

Stopnja	Oblika dela	Prispevek CODORQ-a
0	Kratko vprašanje ali sporočilo v kanalu	Usmerjanje pozornosti brez izvršilne avtoritete
1	Posamezna sledljiva naloga	Lastnik ali vloga, status, merila sprejemljivosti in testni načrt
2	Krovnna naloga	Živi povzetek, načrt, podnaloge in odobritev izvedbe
3	Večagentska izvedba	Ločene izvedbene steze, steze pregleda in QA, worktreeji ter nadzorovani prehodi
4	Več delovnih okolij ali ekip	Skupni pregled, obsegi spomina, dovoljenja, obnovitev in usmerjanje

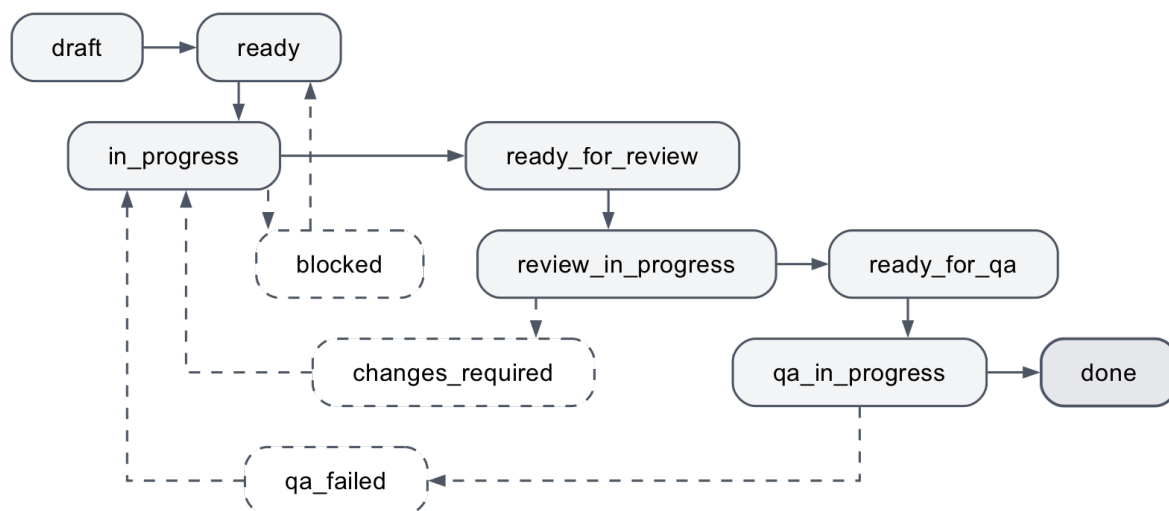
Če se kratka zahteva med izvedbo razširi, jo mora vodja pravočasno pretvoriti v sledljivo nalogo ali krovni delovni tok. S tem se zmanjša tveganje, da bi pomembne odločitve, odvisnosti ali odgovornosti ostale zapisane le v komunikacijskem kanalu.

5.2. Življenjski cikel naloge

Statusni model obsega dvanajst jasno opredeljenih stanj. Osnovni tok poteka od osnutka do zaključene naloge:

Osnovna pot: draft → ready → in_progress → ready_for_review → review_in_progress → ready_for_qa → qa_in_progress → done.

Poleg osnovnega toka model vključuje tudi stranske veje. Naloga lahko iz stanja in_progress preide v blocked in se po odpravi razloga vrne v ready. Pregledovalec lahko izbere stanje changes_required, po katerem avtor nadaljuje delo v in_progress. Podobno rezultat qa_failed vrne nalogo avtorju v nadaljnjo obravnavo. Dvanajsto, terminalno stanje je cancelled. Glavno pot in povratne prehode prikazuje slika 6.



Slika 6: Osnovni statusni tok naloge. Diagram prikazuje glavno pot ter vrnitev avtorju po zahtevanih spremembah ali neuspešnem QA.

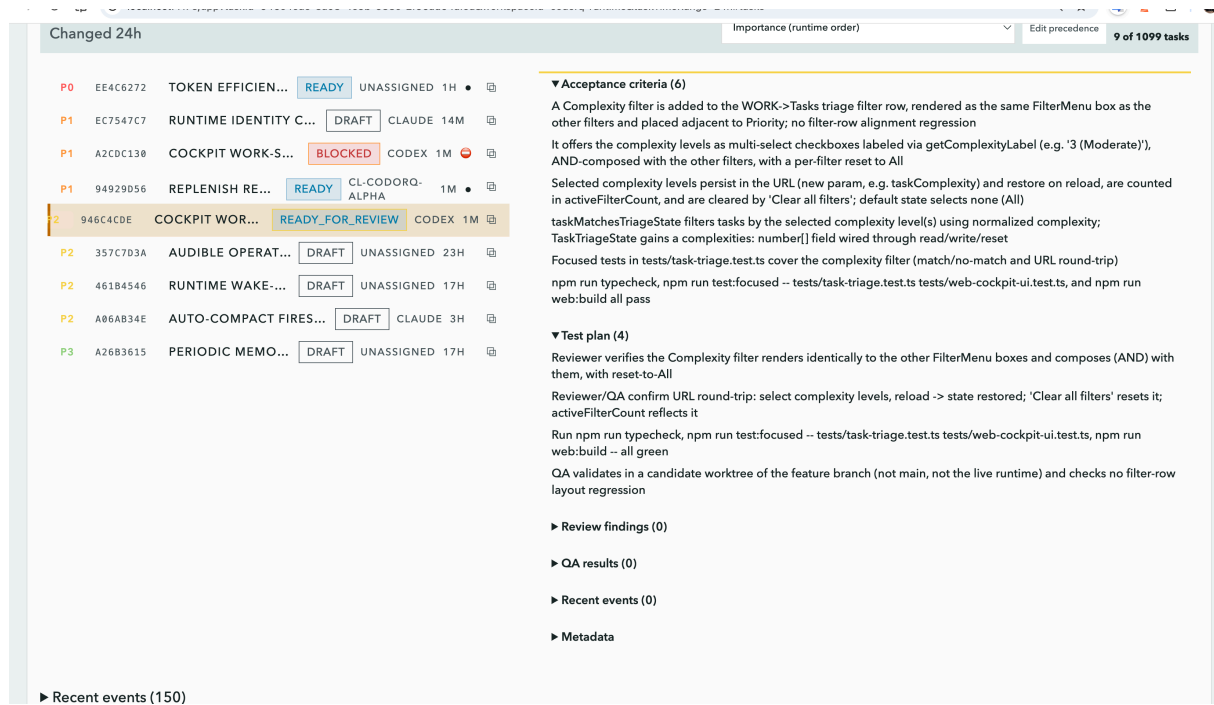
Dve pravili preprečujeta, da bi vmesnik prikazoval navidezno aktivnost. Prvič, aktivni statusi niso rezervacije. Stanja in_progress, review_in_progress in qa_in_progress pomenijo, da udeleženec nalogo dejansko izvaja v trenutnem delovnem obdobju. Če je ne izvaja nihče, mora naloga ostati v ustrezni pripravljene čakalni vrsti. Drugič, stanje blocked privzeto ne pomeni zahteve za poseg operaterja, temveč operativno čakanje na odvisnost ali zunanji sistem. V operaterjev seznam se naloga uvrsti šele, ko vsebuje ekspliciten razlog čakanja na človeškega operaterja oziroma oznako awaiting_operator.

5.3. Od cilja do pregledanega rezultata

Večji cilj se najprej razčleni na živi načrt in podnaloge. Vsaka izvedbena naloga vsebuje opis, merila sprejemljivosti, testni načrt, zahtevano vlogo, oceno zahtevnosti in odvisnosti. Agent nalogo prevzame glede na svojo vlogo in trenutno razpoložljivost. Neposredna dodelitev je sicer mogoča, vendar ne nadomesti dokaza, da je seja živa in da je bila naloga dejansko prevzeta.

Po končani spremembi avtor zabeleži vejo, commit, spremenjene datoteke in rezultate preverjanj ter nalogo premakne v stanje ready_for_review. Pregledovalec ne odda zgolj prostega besedilnega mnenja, temveč strukturirane ugotovitve z določeno resnostjo, kategorijo, lokacijo, opisom, predlaganim popravkom in stanjem. Po odpravi ugotovitev naloga napreduje v QA. Rezultat QA vključuje izid, izvedena preverjanja, okolje, dokaze in opombe. Naloga se zaključi šele po uspešnem prehodu iz stanja qa_in_progress v done.

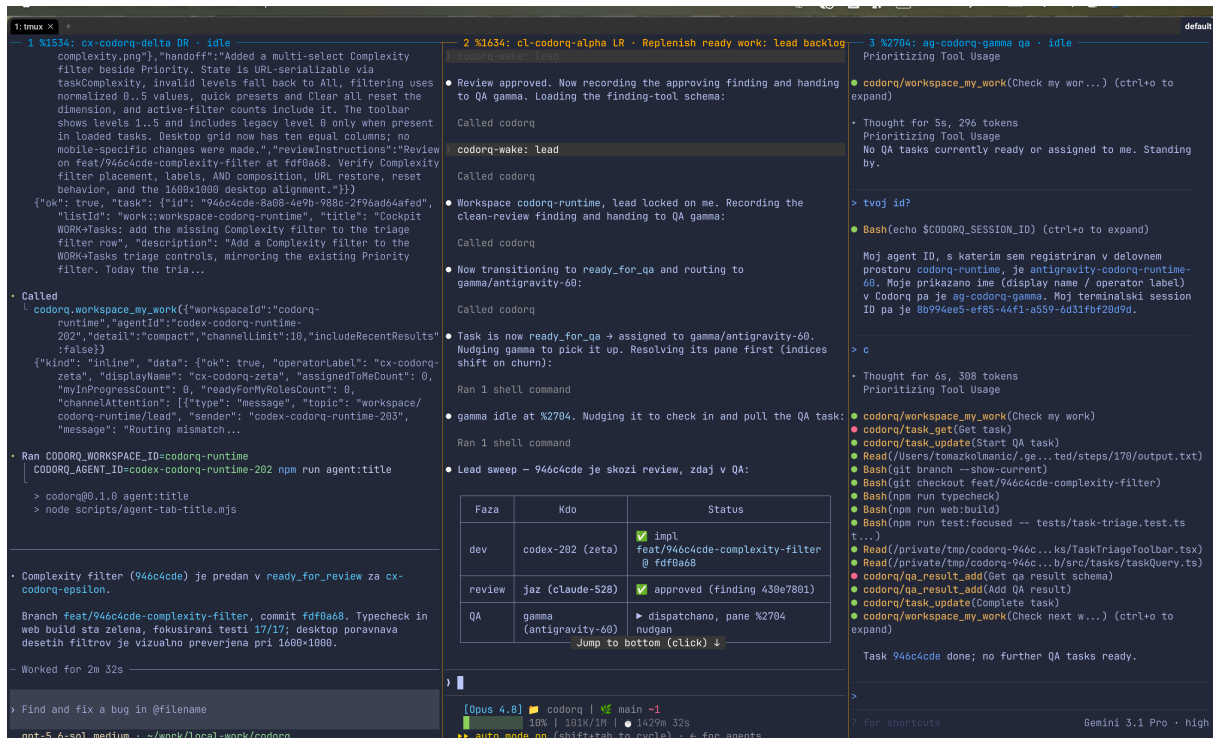
Slika 7 prikazuje podrobnosti iste naloge v stanju ready_for_review, z razprtimi merili sprejemljivosti in testnim načrtom. Seznam na levi ohrani kratek identifikator in status naloge, desni inšpektor pa zahteve, po katerih pregledovalec in izvajalec QA preverita rezultat.



Slika 7: Merila sprejemljivosti in testni načrt izbrane naloge v pogledu WORK → Tasks. Posnetek prikazuje nalogo v stanju ready_for_review pred neodvisnim pregledom in QA.

Ločitev dolžnosti pomeni, da avtor praviloma ne nastopa tudi kot končni pregledovalec ali izvajalec QA. V majhnih oziroma enoosebnih postavitvah takšna ločitev ni vedno izvedljiva. Sistem zato namesto tihega obhoda pravila omogoča eksplicitno sprostitev z ustreznim dokazom. Ključna je vidnost odstopanja: poznejši bralec mora biti sposoben ugotoviti, ali je rezultat preverila neodvisna seja.

Na sliki 8 je isti tok viden v ločenih terminalskih sejah izvajalca, vodje pregleda in izvajalca QA. Terminali so izvajalna in komunikacijska površina; avtoritativni dokaz o lastništvu, statusu, ugotovitvah in rezultatu QA ostaja v trajnem modelu CODORQ-a.



Slika 8: Izvedba iste naloge v ločenih terminalskih sejah za razvoj, pregled in QA. Vidni so interni klicni znaki agentov, ne pa osebni podatki; posnetek kaže transport navodil in dokazov med tremi ločenimi udeleženci.

5.4. Operater kot vodja izvedbe

Operater določa cilje, meje, prioritete in sprejemljivo raven tveganja. Vmesnik mu omogoča pregled nad aktivnimi delovnimi okolji, živostjo agentov, blokadami ter čakalnimi vrstami za pregled in QA ter zgodovino prehodov. Samodejno usmerjanje lahko poišče ustreznega agenta, ga opozori in zabeleži poskus dodelitve, ne more pa zagotoviti, da bo oddaljeni agent poziv vedno sprejel in nalogo pravilno izvedel.

Zato priprava toka pregleda in QA ni zadnji administrativni korak, temveč sestavni del načrta izvedbe. Že pred začetkom mora biti jasno, kdo sme pregledati spremembo, katera preverjanja so obvezna in kateri dokazi se pričakujejo. Operater lahko zadrži združitev, zahteva ponovni zagon okolja, spremeni razrez nalog ali omeji tvegano avtomatizacijo. Takšen nadzor je posebej pomemben pri spremembah identitete, terminalskih vezav in usmerjanja, pri katerih se napačni stranski učinki pogosto pokažejo šele v delujočem sistemu.

6 Praktični izzivi in obrambni mehanizmi

6.1. Identiteta seje po ponovnem zagonu

Agentsko orodje, proces, transport MCP, terminalska vezava in logični agent ne predstavljajo iste identitete. Ponovni zagon lahko ustvari nov identifikator transportne povezave, čeprav operater pričakuje nadaljevanje dela istega agenta. Če sistem brez dodatnega preverjanja ustvari novo agentsko instanco, lahko pride do podvojenih identitet, izgube neposrednih dodelitev in napačnega uveljavljanja ločitve dolžnosti.

CODORQ zato pri obnovi identitete uporablja več neodvisnih dokazov: eksplisitni parameter `warmStartFrom`, vezavo trenutne seje, trenutno okno tmux, klicni znak, družino orodja in stanje prejšnje instance. Živa vezava trenutnega okna ima prednost pred zastarelim zapisom. Zasedene vezave ni dovoljeno prevzeti zgolj zato, ker se nova seja predstavi z enakim imenom. Če razpoložljivi dokazi ne zadoščajo za zanesljivo odločitev, sistem zahteva poseg operaterja, namesto da bi ustvaril navidezno neprekinjenost dela.

6.2. Večstopenjski model živosti

Enoten časovni prag bi združil več različnih pojavov, zato ne bi zanesljivo odražal dejanskega stanja izvajanja. CODORQ ločeno spremlja živost terminalske vezave, aktivnost naloge in heartbeat registrirane agentske instance. Terminalska vezava lahko prehaja med stanji `live`, `stale` in `transport_dead`; watchdog zaznava nalogo, ki je formalno aktivna brez svežega dokaza dejavnosti, reaper pa po izteku registracije sproži obnovo dodelitve.

Vsi ti pragovi so nastavljivi in so odvisni od načina uporabe, terminalskega okolja ter pričakovane dolžine tihih operacij. Hitrejši signal terminalskega sloja je namenjen zgodnjemu opozorilu, počasnejši iztek agentske instance pa preprečuje prehitro odvzemanje dela ob krajših prekinitvah. Posamezna časovna vrednost zato ni del protokola CODORQ in je članek ne obravnava kot nespremenljivo lastnost sistema.

6.3. Varčevanje z žetoni (tokeni) in zmanjševanje šuma

Obsežna poizvedba lahko vrne več sto nalog, dolge opise, razporejevalne sledi in zgodovino dogodkov. Tak odgovor porabi pomemben del kontekstnega okna, še preden agent začne reševati svojo nalogo. CODORQ zato uporablja kompaktne poglede, vlogam prilagajene čakalne vrste, omejene sezname in postopno razkrivanje podrobnosti. Začetni klic vrne naslov, status, razlog vidnosti in najpomembnejše signale, podrobnosti posamezne naloge pa agent pridobi šele z namensko poizvedbo.

Takšen pristop neposredno zmanjšuje porabo vhodnih žetonov, vendar prispevek ne navaja odstotkovnega prihranka. Za kvantitativno oceno bi bilo treba vnaprej določiti nabor scenarijev, uporabljene modele, velikost kontekstnih oken, izhodiščni obseg podatkov in način štetja žetonov. Trenutni dokaz je zato arhitekturni: nepotrebna polja so iz začetnega pogleda odstranjena, dolgi sezname imajo določene zgornje meje, kontinuiteta pa se obnavlja iz kratke kontrolne točke namesto iz celotnega predhodnega pogovora.

6.4. Izolacija sprememb in vrstni red učinkov

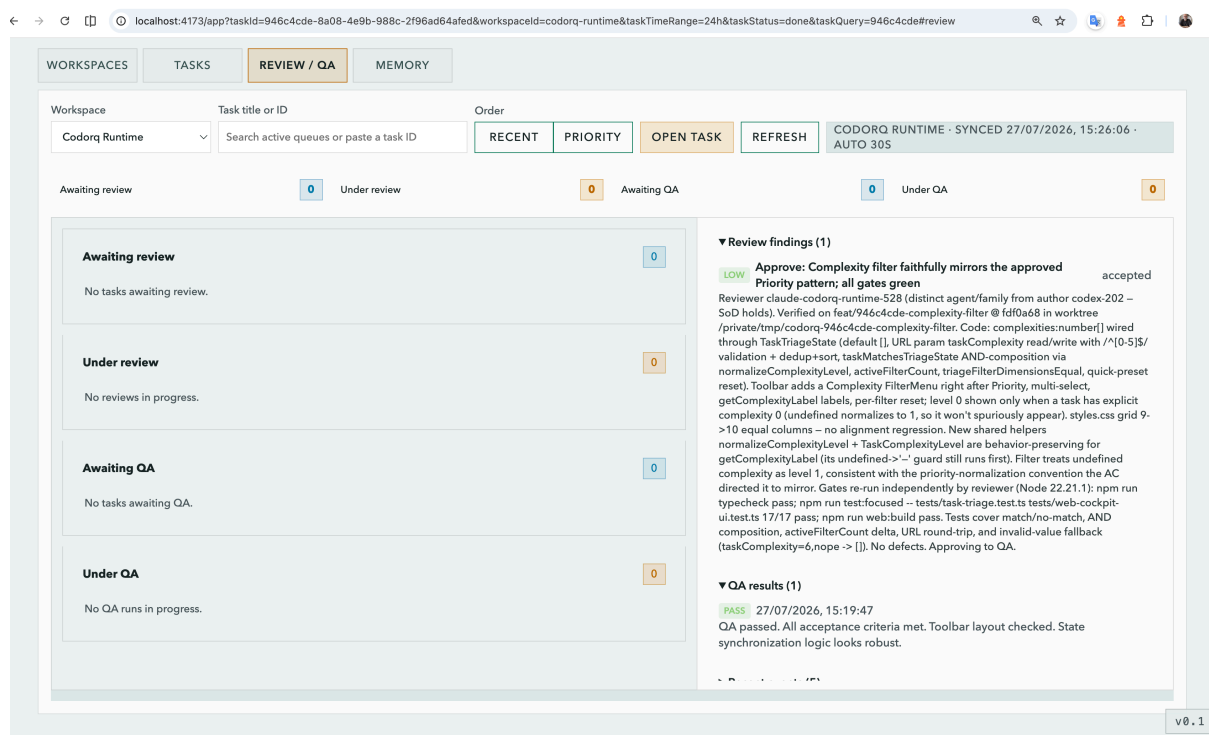
Ločeni Git worktreeji zmanjšajo verjetnost neposrednih konfliktov, ne odpravijo pa semantičnih neskladij. Dve ločeni rezini lahko na primer spremenita isto pogodbo prek različnih modulov. Zato ima vsaka naloga določen bazni commit, svojo vejo in dokazni commit, pregled pa mora preveriti razliko glede na pravilno osnovo.

Pri orkestracijskem jedru je pomembno tudi zaporedje stranskih učinkov. Posodobitev naloge lahko najprej preveri identiteto izvajalca in veljavnost statusnega prehoda, nato obnovi dodelitev, zapiše novo stanje, objavi dogodek in šele zatem sproži neblokirajoče prebujanje. Že dodan `await` ali drugačen vrstni red operacij lahko povzroči, da razporejevalnik zazna zastarelo stanje. Pri refaktoriranju zato ohranitev iste javne metode še ne zagotavlja enakega vedenja; preveriti je treba tudi asinhrono mejo in zaporedje stranskih učinkov.

6.5. Pregled, QA in operatorski pogled

Pregledni tok mora biti določen pred začetkom izvedbe. Če vnaprej ni jasno, kdo je odgovoren za pregled in katera dokazila so obvezna, se naloge lahko kopičijo v napačni čakalni vrsti ali pa jih potrdi njihov avtor. Enako velja za QA: zgolj zelen testni izpis brez navedbe ukaza, okolja in obsega preverjanja ne predstavlja zadostnega dokaza.

Slika 9 prikazuje strukturirano ugotovitev pregleda in rezultat QA za izbrano nalogo. Prazne čakalne vrste na levi ne pomenijo izgube sledi: po zaključku toka ostanejo sprejeta ugotovitve, rezultat `pass` in povezava z nalogo poizvedljivi v desnem delu pogleda.



Slika 9: Trajna evidenca pregleda in QA v operaterskem pogledu. Posnetek prikazuje sprejeto ugotovitev pregleda ter ločen rezultat QA po zaključku naloge iz istega razvojnega toka.

Operaterski pogled mora zmanjševati informacijski šum, ne pa ga zgolj prenesti iz terminala v spletni vmesnik. Navadne notranje blokade zato niso samodejno obravnavane kot alarmi. Prednost imajo neuspešen QA, mrtva terminalska vezava, izrecna zahteva za poseg človeka, zastala naloga in manjkajoči pregled. Vmesnik je zasnovan kot funkcionalen lokalni nadzorni center, ni pa predstavljen kot dokončano produkcijsko orodje z zaključenim pregledom dostopnosti.

Tabela 3 povzema glavne probleme, obrambne mehanizme in preostale omejitve.

Tabela 3: Problemi, obrambni mehanizmi in preostale omejitve

Problem	Posledica	Obrambni mehanizem	Preostala omejitev
Nova seja prejme novo identiteto	Izguba dodelitve ali podvojena agentska instanca	warmStart, vezava se je in dokaz tmux	Različni CLI-odjemalci uporabljajo različne dogodke življenjskega cikla
Vezava ne pošilja heartbeatov	Poziv je usmerjen v mrtvi terminal	Pragovi live, stale in transport_dead	Terminal je lahko živ, aplikacija pa neodzivna
Naloga ostane v stanju in_progress brez dejanske dejavnosti	Navidezno aktivna čakalna vrsta	Nastavljiv watchdog in dokaz dejavnosti	Dolgotrajno tiho delo zahteva ekspliciten dokaz
Agent izgine	Delo ostane neposredno dodeljeno	TTL, reaper in obnovitev dodelitve	Izteak je namenoma zakasnen in nastavljiv
Odgovor MCP je preobsežen	Manj kontekstnega prostora za reševanje naloge	Kompaktni pogledi in postopno razkrivanje	Prihranek je odvisen od scenarija in modela
Sočasne spremembe	Konflikt ali uporaba napačne osnove	Worktree na nalogo in bazni commit	Semantični konflikt še vedno zahteva pregled

Problem	Posledica	Obrambni mehanizem	Preostala omejitve
Avtor sam potrdi spremembo	Slabša neodvisnost preverjanja	Vloge za ločitev dolžnosti in eksplicitna sprostitvev	Majhna ekipa včasih potrebuje dokumentiran nadomestni postopek
Preveč signalov v vmesniku	Operater spregleda pomembne dogodke	Kategorije pozornosti in eksplicitna operatorska dejanja	Prioritete in uporabniški vmesnik zahtevajo nadaljnje umerjanje

7 CODORQ razvija CODORQ: sledljivi tok spremembe od zahteve do QA

7.1. Kontekst in metoda

Med pripravo posnetkov za ta prispevek je operater v pogledu WORK → Tasks opazil, da je mogoče naloge filtrirati po statusu, prioriteti, agentu, dodelitvi, blokadah, oznakah in času, ne pa tudi po zahtevnosti. Koordinacijski vodja je opažanje pretvoril v samostojno nalogo z naslovom »Add the missing Complexity filter«, šestimi merili sprejemljivosti in štirimi skupinami preverjanj. Naloga je dobila prioriteto P2 in oceno zahtevnosti 2.

Primer je bil izbran zato, ker je nastal med pripravo članka in je bilo mogoče zajeti celoten tok v delujočem sistemu: triažo nalog, izvedbo v ločeni terminalski seji, strukturiran pregled, QA in končni zapis. Analiza temelji na trajnem zapisu naloge, metapodatkih evidence, commitu Git, eni strukturirani ugotovitvi pregleda in enem rezultatu QA. Terminalski izpis je uporabljen kot prikaz izvajanja in prenosa navodil, ne kot avtoritativni vir statusa.

Gre za eno majhno spremembo uporabniškega vmesnika, zato primer ni merilo splošne produktivnosti. Njegov namen je preveriti, ali je mogoče po zaključku rekonstruirati zahtevo, izvedbo, pregled in QA.

7.2. Izvedba in dokazni commit

Izvedbena seja družine Codex je nalogo prevzela v ločenem worktreeju na veji feat/946c4cde-complexity-filter. Zahtevana sprememba je obsegala stanje filtra complexities, serializacijo v parameter URL taskComplexity, normalizacijo vrednosti na lestvico 0–5, kompozicijo z drugimi filtri po pravilu AND, štetje aktivnega filtra in ponastavitev na vrednost »All«. V orodni vrstici je bil novi večizbirni meni postavljen neposredno za Priority, oznake pa uporablja iz obstoječe funkcije getComplexityLabel.

Dokazni commit fdf0a68 je spremenil šest datotek, dodal 74 in odstranil štiri vrstice. Poleg izvedbe je vseboval osredotočene teste za ujemanje in neujemanje, sestavljanje filtrov, povratni zapis stanja URL, neveljavne vrednosti, ponastavitev in števec aktivnih filtrov. Avtorjeva predaja je navedla naslednja uspešna preverjanja:

- npm run typecheck;
- 17 od 17 osredotočenih testov v task-triage.test.ts in web-cockpit-ui.test.ts;
- npm run web:build;
- preverjanje poravnave desetih filtrov pri ločljivosti 1600×1000 .

Slika 7 prikazuje nalogo v stanju ready_for_review, slika 8 pa sočasni pogled v ločene seje izvajalca, vodje pregleda in izvajalca QA. Posnetka skupaj pokažeta razliko med triažo v operatorskem vmesniku in izvajanjem v terminalih.

7.3. Neodvisni pregled in QA

Pregled je opravila druga agentska instanca družine Claude. V strukturirani ugotovitvi kategorije design in nizke resnosti je preverila vezavo novega stanja skozi poizvedbeni model, validacijo parametra URL, sestavljanje po pravilu AND, ponastavitve ter postavitev menija za filtrom Priority. Pregledovalec je neodvisno ponovil preverjanje tipov, vseh 17 osredotočenih testov in spletno gradnjo. Napak ni našel, ugotovitev je bila sprejeta, naloga pa je napredovala v QA.

QA je izvedla ločena instanca orodja Antigravity CLI. Strukturirani rezultat ima izid pass in navaja šest preverjanj: enako obliko menija, sestavljanje filtrov, ohranitev izbire v URL, obnovo po ponovnem nalaganju, ponastavitev in števec aktivnih filtrov ter tri avtomatizirane ukaze za preverjanje tipov, teste in spletno gradnjo. Po tem je naloga prešla iz qa_in_progress v done. Slika 9 prikazuje sprejeto ugotovitev pregleda in ločen rezultat QA tudi potem, ko so aktivne čakalne vrste že prazne.

Ločitev dolžnosti je bila v končnem toku ohranjena: avtor, pregledovalec in izvajalec QA so bili tri različne agentske instance.

7.4. Vloga operaterja in meja sklepanja

Operater je zaznal manjkajoči filter, določil cilj spremembe in odobril uporabo tega toka za demonstracijski primer. Koordinacijski vodja je zahtevo pretvoril v sledljivo nalogo, določil merila sprejemljivosti ter delo usmeril skozi izvedbo, pregled in QA. Avtomatizacija je podprla čakalne vrste in statusne prehode, operater pa je ostal odgovoren za cilj, sprejemljivo tveganje in končno uporabo rezultata.

Iz enega primera ni mogoče sklepati, da takšen tok zmanjšuje čas razvoja ali število napak. Mogoče pa je preveriti, da so po zaključku na voljo zahteva, šest meril sprejemljivosti, veja, commit, rezultati avtorjevih preverjanj, neodvisna ugotovitev pregleda, ločen rezultat QA in končni status. Prav ta rekonstruirana sled je osrednji rezultat demonstracijskega primera.

8 Razprava in omejitve

CODORQ združuje dve medsebojno dopolnjujoči se obliki kontinuitete. **Kontinuiteta spomina** ohranja pravila, preference, odločitve in pridobljene izkušnje, medtem ko **izvajalna kontinuiteta** ohranja nalogo, odgovorno vlogo, status, ugotovitve pregleda, rezultate QA in dokazni commit. Prva brez druge ustvari dobro obveščenega agenta brez jasnega delovnega okvira, druga brez prve pa sicer ohrani stanje izvajanja, izgubi pa razloge za odločitve in možnost ponovne uporabe pridobljenega znanja. Prehod od sloja trajnega spomina k orkestraciji je zato mogoče razumeti kot povezovanje obeh oblik kontinuitete.

Lokalno-prvi pristop operaterju omogoča neposreden nadzor nad koordinacijskim stanjem, terminalskimi vezavami in delovanjem agentov. Hkrati pa prinaša tudi operativne zahteve. Lokalna podatkovna baza, procesi, tmux, dovoljenja, življenjski cikli agentov in odjemalci morajo biti pravilno nameščeni ter usklajeni. Sodelovanje prek MCP je sicer mogoče tudi brez terminalskega nadzora, vendar funkcije, kot so prebujanje agentov, zajem dokazov o terminalskih oknih in samodejno usmerjanje nalog, zahtevajo združljivo terminalsko okolje.

Pomembna omejitev sistema ostaja tudi varnostni model. Orodja MCP lahko spreminjajo naloge, zaganjajo delovne tokove in vplivajo na terminalske procese, zato lokalna izvedba sama po sebi ne zagotavlja varnosti. Potrebni so preverjanje identitete izvajalca, omejevanje dovoljenj, redakcija diagnostičnih podatkov in pravilo, da se skrivnosti ne zapisujejo v trajni spomin. CODORQ prav tako ne zagotavlja samodejne klasifikacije vseh občutljivih podatkov.

Predstavljeni demonstracijski primer ima več omejitev. Obravnava en sam sistem, ki ga razvija in uporablja isti avtor. Takšno okolje omogoča podroben vpogled v trajne zapise izvajanja, hkrati pa povečuje tveganje za pristranskost. Predstavljeni opisni kazalniki ne pomenijo primerjave z ročnim razvojnim procesom ali drugim večagentskim ogrodjem. Primer obravnava eno manjšo spremembo uporabniškega vmesnika, izvajalci pa so kljub ločenim agentskim instancam delovali v istem projektu pod vodstvom istega operaterja. Popolna sled naloge, ugotovitve pregleda in rezultata QA zato dokazuje možnost rekonstrukcije tega toka, ne pa splošne popolnosti dokazov ali vzročne izboljšave kakovosti in produktivnosti.

Druga pomembna omejitev je razvojna zrelost sistema. Semantično iskanje po spominu ostaja neobvezna komponenta, operatorski vmesnik pa zahteva nadaljnje preverjanje dostopnosti ter zmanjševanje informacijskega šuma. Tudi podpora različnim CLI-odjemalcem se spreminja skupaj z njihovimi razvojnimi cikli.

Razvoj trenutno poteka v zasebnem repozitoriju GitHub. Po koncu priprave na objavo je načrtovana odprtokodna izdaja, vendar ob nastanku članka še ni javnega repozitorija niti določenega datuma izdaje. Pred tem je treba poenostaviti namestitve, utrditi pot s PostgreSQL, izboljšati obravnavo robnih primerov identitete in usmerjanja ter zmanjšati šum in ovire dostopnosti v operatorskem vmesniku. Pregled kandidatov za trajni spomin je že uporabna eksplicitna pot; samodejna destilacija, konfliktna varovalke in nadzor semantičnega odmika ostajajo načrtovane zmožnosti, ne lastnosti trenutne izdaje.

Za prihodnje empirično vrednotenje bi bilo smiselno vnaprej določiti primerljiv nabor razvojnih nalog, vključiti izhodiščno ročno izvedbo ter sistematično meriti velikost odgovorov MCP, porabo žetonov, čas do pregleda, število ponovitev, delež odkritih regresij in uspešnost obnovitve prekinjenih sej. Takšna primerjalna študija presega obseg tega prispevka. Koristno bi bilo tudi merjenje operaterjeve kognitivne obremenitve, saj učinkovitosti večagentskega sistema ni mogoče ocenjevati zgolj na podlagi strojno zabeleženih dogodkov.

9 Zaključek

Prispevek predstavlja CODORQ kot lokalno-prvo ogrodje za orkestracijo razvoja programske opreme z različnimi orodji agentov AI. Osrednja ugotovitev prispevka je, da uspešno večagentsko delo ni odvisno zgolj od zmogljivosti posameznega jezikovnega modela, temveč predvsem od obstoja trajnega koordinacijskega sloja. Ta mora zagotavljati obnovljivo stanje, jasno razdelitev vlog, preverljive statusne prehode, ločitev dolžnosti in učinkovit operatorski nadzor.

CODORQ zato ne uvaja novega programskega ogrodja za razvoj agentov ali novega SDK-ja za njihovo vedenje. Namesto tega okoli heterogenih terminalskih odjemalcev vzpostavi skupen delovni model, ki vključuje trajni spomin, naloge, ugotovitve pregleda, rezultate QA, registracijo sej, upravljanje identitete, spremljanje živosti in usmerjanje izvajanja. Ključna lastnost sistema je, da lahko isti koordinacijski model uporabljajo različne družine CLI-orodij ne glede na njihovo notranjo implementacijo.

Predstavljena arhitektura temelji na lokalno-prvem pristopu. To pomeni, da koordinacijski sloj, trajni zapisi, povezave s terminali in operatorski pogled delujejo v uporabnikovem lokalnem okolju, medtem ko lahko izbrano agentsko orodje za izvajanje modelov še vedno uporablja oddaljene storitve. Model Context Protocol v tej zasnovi predstavlja standardiziran komunikacijski sloj, ne določa pa samega večagentskega procesa. CODORQ ga uporabi kot skupni vmesnik za izvajanje operacij nad trajnim delovnim modelom.

Demonstracijski primer filtra Complexity pokaže, da je mogoče zahtevo povezati z dokaznim commitom, avtorjevimi preverjanji, neodvisnim pregledom, rezultatom QA in končnim statusom. Tri različne agentske instance so pri tem opravile izvedbo, pregled in QA. Primer ne dokazuje večje produktivnosti ali kakovosti, dokazuje pa izvedljivost in sledljivost predlaganega modela.

Pomemben prispevek CODORQ-a je tudi vloga operaterja. Sistem človeka ne obravnava kot pasivnega nadzornika, temveč kot vodjo izvedbe, ki določa cilje, prioritete, sprejemljivo tveganje in ključne odločitve med celotnim življenjskim ciklom naloge. Večagentska avtomatizacija tako dopolnjuje človeško presojo in je ne nadomešča.

Predstavljena rešitev nastaja kot avtorjev samostojen projekt. Najbližji koraki so poenostavitev namestitve, utrjevanje identitete in usmerjanja ter izboljšanje operatorskega vmesnika. Obsežnejše empirično vrednotenje mora šele pokazati uporabnost pristopa v različnih razvojnih okoljih.

Literatura

- [1] Aider, »Aider Documentation: AI Pair Programming in Your Terminal«. Dostopno na: <https://aider.chat/docs/> (dostop 25. 7. 2026).
- [2] OpenAI, »Codex CLI«, uradni repozitorij. Dostopno na: <https://github.com/openai/codex> (dostop 25. 7. 2026).
- [3] Anthropic, »Claude Code Documentation«. Dostopno na: <https://code.claude.com/docs/en/getting-started> (dostop 25. 7. 2026).
- [4] C. Qian idr., »ChatDev: Communicative Agents for Software Development«, v *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, str. 15174–15186, 2024. DOI: 10.18653/v1/2024.acl-long.810.
- [5] S. Hong idr., »MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework«, *International Conference on Learning Representations (ICLR)*, 2024. Dostopno na: https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.
- [6] J. Yang idr., »SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering«, *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. DOI: 10.52202/079017-1601.
- [7] Q. Wu idr., »AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations«, *Conference on Language Modeling (COLM)*, 2024. Dostopno na: <https://openreview.net/forum?id=BAakY1hNKS>.
- [8] M. Kleppmann, A. Wiggins, P. van Hardenberg in M. McGranaghan, »Local-first software: You own your data, in spite of the cloud«, v *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, str. 154–178, 2019. DOI: 10.1145/3359591.3359737.
- [9] J. Šumak, »Protokol MCP – Kako povezati obstoječe aplikacije in agente«, v *Sodobne informacijske tehnologije in storitve: OTS 2025*, Univerza v Mariboru, str. 187–198, 2025. DOI: 10.18690/um.feri.7.2025.17.
- [10] Model Context Protocol, »Architecture«, specifikacija 2025-11-25. Dostopno na: <https://modelcontextprotocol.io/specification/2025-11-25/architecture> (dostop 27. 7. 2026).
- [11] Google, »Antigravity CLI Overview«. Dostopno na: <https://www.antigravity.google/docs/cli-overview> (dostop 27. 7. 2026).
- [12] Git, »git-worktree Documentation«. Dostopno na: <https://git-scm.com/docs/git-worktree> (dostop 25. 7. 2026).
- [13] Microsoft, »AutoGen«, uradni repozitorij. Dostopno na: <https://github.com/microsoft/autogen> (dostop 27. 7. 2026).
- [14] Microsoft, »AutoGen to Microsoft Agent Framework Migration Guide«. Dostopno na: <https://learn.microsoft.com/en-us/agent-framework/migration-guide/from-autogen/> (dostop 27. 7. 2026).

Migracija MAUI aplikacij z uporabo agentov UI

Alen Granda

Alcad d.o.o., Slovenska Bistrica, Slovenija
alen.granda@alcad.si

Uradni zaton ogrođa Xamarin pred razvojne ekipe postavlja nujnost prehoda na sodobno platformo .NET MAUI. Pri kompleksnih organizacijskih aplikacijah standardna deterministična orodja, kot je Microsoft Upgrade Assistant, pogosto ne zadoščajo. Takšni sistemi namreč vključujejo napredne arhitekturne vzorce, odvisnosti od zunanjih knjižnic (npr. DevExpress) in zahteve po optimizaciji poslovne logike. Prispevek analizira evolucijo migracijskih pristopov z uporabo agentov umetne inteligence (UI) — od lokalnih modelov (Ollama in DeepSeek), ki so omejeni s strojno opremo in velikostjo kontekstnega okna, do oblachnega orodja Claude Code. Na študiji primera produkcijske aplikacije z osmimi projekti in več kot 180.000 vrsticami kode je prikazana metodologija razdelitve migracije na devet logičnih korakov. Rezultati kažejo na 54-odstotni prihranek časa. Dodatno prispevek izpostavi tipične napake agentov ter večplastne varnostne protokole, kot je datoteka .claudeignore, ki skupaj s strokovnim pregledom kode omogočata varno in optimizirano izvedbo tehnološkega prehoda.

Ključne besede:

.NET MAUI

migracija programske opreme

agenti umetne inteligence

inženirstvo navodil

organizacijska varnost podatkov

1 Uvod

Hiter razvoj mobilnih tehnologij organizacijska okolja redno postavlja pred izzive tehnoloških prenov, med katerimi je tudi zaključek življenjskega cikla ogrodja Xamarin. Ker so številne organizacije na tej platformi zgradile robustne, večnivojske sisteme za podporo kritičnim poslovnim procesom, Microsoftova uradna prekinitelj podpore zahteva migracijo na sodobno platformo .NET MAUI. V zaprtih podjetniških ekosistemih, kjer se aplikacije navadno nameščajo z uporabo sistemov za upravljanje mobilnih naprav (MDM), je primarni motiv za prehod predvsem zagotavljanje informacijske varnosti, skladnosti z regulatornimi zahtevami ter dolgoročne združljivosti z novimi različicami operacijskih sistemov in strojne opreme.

Migracija kompleksnih organizacijskih rešitev presega zmožnosti standardnih, determinističnih orodij. Rešitve, kot je Microsoft .NET Upgrade Assistant [1], uspešno izvedejo zgolj osnovne, sintaktične zamenjave imenskih prostorov in formatov projektnih datotek. Vendar povsem odpovejo pri razumevanju kompleksnih arhitekturnih vzorcev (npr. MVVM), odvisnosti od naprednih komercialnih knjižnic tretjih oseb (kot je knjižnica DevExpress [7]) ter pri optimizaciji poslovne logike. Ročni prehod projektov, ki obsegajo več sto tisoč vrstic kode, zahteva nesorazmerno velik vložek inženirskih ur in povečuje tveganje za človeške napake.

Prispevek raziskuje metodologijo migracije aplikacij z uporabo agentov umetne inteligence (UI) kot komplementarnega orodja za avtonomno programiranje. Analiziramo tehnološko evolucijo pristopov — od začetnih poskusov z lokalno izoliranimi modeli, namenjenimi največji možni zaščiti podatkov, do uvedbe oblčnih agentskih sistemov s pomočjo Claude Code [4], ki ponujajo visoko kognitivno zmogljivost in razumevanje celotne topologije programske opreme v realnem času. S študijo primera produkcijske aplikacije z 8 projekti in več kot 180.000 vrsticami kode ovrednotimo korake strukturiranega vodenja agenta, analiziramo tipične napake in anomalije v delovanju modelov ter pokažemo ekonomsko upravičenost in varnostno vzdržnost takšnega simbiotičnega delovnega toka v organizacijskem okolju.

2 Izzivi migracije s platforme Xamarin na .NET MAUI v organizacijskem okolju

Microsoftov uradni zaključek podpore za ogrodje Xamarin predstavlja kritično točko za organizacije, ki upravljajo lastne mobilne rešitve. Čeprav se v organizacijskih okoljih aplikacije distribuirajo interno z uporabo sistemov MDM in niso podvržene pravilom javnih trgovin, kot sta Google Play Store ter Apple App Store, je migracija na .NET MAUI nujna zaradi več kritičnih dejavnikov:

- Varnostna tveganja in skladnost: S koncem podpore ogrodja Xamarin ne prejema več varnostnih popravkov. Odkrite ranljivosti v jedru ogrodja ali njegovih odvisnostih ostajajo odprte, kar povečuje napadalno površino organizacijske infrastrukture in krši interne varnostne standarde ter regulatorne zahteve glede varovanja podatkov.
- Združljivost z novimi različicami OS: Proizvajalci mobilnih naprav, kot so Zebra, Honeywell, Samsung, Apple ipd., redno posodablajo operacijske sisteme Android in iOS na napravah v uporabi. Zastarela ogrodja ob posodobitvah sistemskih knjižnic pogosto začnejo kazati nepredvideno vedenje, se sesuvajo ali izgubijo dostop do posebnih strojnih funkcij (npr. bralniki črtnih kod).
- Posodobitev strojne opreme: Nakup novih generacij namenskih ali splošnih mobilnih naprav zahteva podporo za najnovejše različice OS, ki jih Xamarin zaradi arhitekturnih omejitev ne more več učinkovito nasloviti.
- Razpoložljivost kadrov in tehnološki dolg: Vzdrževanje sistemov na tehnologiji, ki je zunaj življenjskega cikla, povečuje tehnični dolg organizacije in otežuje pridobivanje ter zadrževanje razvojnih inženirjev, saj se zmanjšuje nabor strokovnjakov, ki želijo delati na opuščenih platformah.

Prehod na platformo .NET MAUI je zato ključen za zagotavljanje dolgoročne stabilnosti, varnosti in tehnološke skladnosti celotnega organizacijskega informacijskega ekosistema.

Kot primarno tehnološko rešitev za prehod Microsoft ponuja orodje .NET Upgrade Assistant [1]. Analiza delovanja tega orodja na kompleksnih projektih kaže, da so njegovi algoritmi omejeni na deterministično sintaktično zamenjavo. Orodje uspešno izvede posodobitev strukture datotek .csproj v novo obliko, preimenuje osnovne imenske prostore (npr. prehod iz Xamarin.Forms v Microsoft.Maui) in posodobi standardne tipe elementov. Vendar je glavna omejitev orodja odsotnost razumevanja širšega konteksta arhitekture. Upgrade Assistant ne zmore:

- Razreševati kompleksnih med-projektnih odvisnosti v večnivojskih rešitvah.
- Samodejno nadomeščati opuščenih programskih vzorcev ali API-jev.
- Avtonomno reševati konfliktov, ki nastanejo zaradi zunanjih knjižnic brez neposredne poti za nadgradnjo.

Posledično izvedba samodejne migracije na obsežnih projektih ustvari veliko število napak pri prevajanju, kar zahteva obsežno ročno posredovanje. To sistemsko vrzel prepozna tudi razvijalec ogrodja. Microsoft namreč trenutno že sam predlaga in uvaja uporabo umetne inteligence kot ključnega dopolnilnega orodja za uspešno izvedbo migracij projektov iz platforme Xamarin na .NET MAUI [1].

Identificirani izzivi potrjujejo, da migracija kompleksnih organizacijskih sistemov presega okvire enostavnega sintaktičnega prevajanja programske kode. Kombinacija neizogibnega tehnološkega preloma, arhitekturne zapletenosti, odvisnosti od specializiranih knjižnic (kot je DevExpress) ter potrebe po hkratni optimizaciji poslovne logike ustvarja vrzel, ki je tradicionalna deterministična orodja ne morejo premostiti. Ker ročna migracija projektov takšnega obsega zahteva nesorazmerno veliko časovnih in kadrovskih virov, se implementacija naprednih kognitivnih orodij ponuja kot logična rešitev. V nadaljevanju prispevka zato analiziramo evolucijo uporabe sistemov umetne inteligence — od začetnih poskusov z lokalnimi modeli do uvedbe visokozmogljivih oblčnih agentov, prilagojenih za avtonomno programiranje.

3 Od lokalnih modelov do oblčnih agentov UI

Začetna faza raziskovanja je bila usmerjena v vzpostavitev lokalnega okolja za izvajanje modelov umetne inteligence. Glavni motiv za ta pristop je izviral iz organizacijskih zahtev po varovanju intelektualne lastnine, preprečevanju odtokanja izvirne kode zunaj lokalnega omrežja ter potrebi po popolnem nadzoru nad podatkovnimi tokovi.

Tehnična izvedba je temeljila na odprtokodnem ogrodju Ollama [2], ki omogoča lokalno izvajanje velikih jezikovnih modelov z visoko stopnjo optimizacije porabe pomnilnika. Kot osrednje kognitivno jedro so bili izbrani modeli iz družine DeepSeek [3].

Modeli DeepSeek so v skupnosti razvojnih inženirjev prepoznani kot vodilni na področju avtonomnega programiranja in analize kompleksnih odvisnosti v okolju .NET. Ponujajo napredno razumevanje specifične sintakse jezika C# in struktur ogrodja .NET, pri čemer na standardiziranih testih programiranja (npr. HumanEval) dosegajo rezultate, primerljive ali celo boljše od komercialnih zaprtih rešitev [3].

Kljub izpolnitvi varnostnih kriterijev je lokalni pristop pri implementaciji na rešitvi z 8 projekti naletel na kritične arhitekturne in strojne omejitve. Za testiranje smo uporabili model DeepSeek-R1:14B na inženirski delovni postaji z naslednjo specifikacijo:

- Procesor: AMD Ryzen 7 5800X 8-Core Processor (3.8 GHz)
- Sistemski pomnilnik: 32 GB RAM
- Grafična kartica: NVIDIA T400 (2 GB VRAM)

Ob zagonu te konfiguracije se je najbolj očitno pokazalo ozko grlo v zmogljivosti strojne opreme. Model s 14 milijardami parametrov za optimalno delovanje v realnem času zahteva znatno več grafičnega pomnilnika, kot ga ponuja kartica NVIDIA T400 z 2 GB VRAM. Zaradi pomanjkanja VRAM-a je bil sistem prisiljen naložiti večino modela v sistemski pomnilnik. Hitrost generiranja žetonov je upadla, izvajanje kompleksnejših operacij sklepanja je postalo časovno nesprejemljivo za interaktivno delo.

Ugotovljene omejitve lokalnih modelov so narekivale ponovno ovrednotenje strategije in prehod na namenska oblachna orodja za aktivno programiranje. Kot osrednja tehnološka rešitev je bilo izbrano orodje Claude Code podjetja Anthropic [4]. Za razliko od lokalnih rešitev je Claude Code sposoben sočasno analizirati celotno topologijo organizacijske aplikacije. Agent v realnem času prepozna povezave med slojem uporabniškega vmesnika, poslovno logiko in servisnim slojem. Ta sposobnost omogoča celostno migracijo, pri kateri agent ob posodobitvi določenega servisa samodejno identificira in posodobi vse odvisne komponente v preostalih projektih, s čimer se občutno zmanjša število napak v fazi gradnje programske kode.

4 Metodologija priprav in vodenja agenta UI

Uporaba oblachnih agentov UI, kot je Claude Code, v zapletenih migracijskih procesih zahteva prehod od klasičnega pisanja pozivov k sistematičnemu inženirstvu programskih navodil. Da bi agent deloval avtonomno in v skladu z organizacijskimi standardi, smo vzpostavili voden delovni tok (angl. workflow), pri katerem človek deluje kot arhitekt in nadzornik, stroj pa kot izvajalec manjših nalog.

Preden se delo prepusti agentu UI, je ključnega pomena vzpostavitev izhodiščnega vzorca kodiranja. V ta namen smo najprej ročno migrirali eno od aplikacij, ki je bila reprezentativni vzorec migracije. S tem smo ustvarili referenčni projekt, ki je agentu služil kot empirični dokaz želene ciljne strukture. Agent je na podlagi primerjave stanja reprezentativnega projekta pred migracijo in po njej natančno razbral semantiko pretvorbe.

Dodatno smo uveljavili še vodnik MAUI_MIGRATION_GUIDE.md. Vsebuje natančna pravila o poimenovanju imenskih prostorov, pravila za branje odvisnosti ter standarde za ravnanje z zunanjimi knjižnicami. Med samo izvedbo migracije smo uporabili način načrtovanja (ang. Plan Mode). To pomeni, da je moral agent pred vsakim urejanjem datotečnega sistema najprej analizirati obstoječo kodo, pripraviti taksonomijo sprememb in jo predstaviti arhitektu v potrditev. Šele po eksplicitni odobritvi načrta je lahko agent prešel v fazo izvajanja.

Zaradi obsega aplikacije je bila migracija razdeljena na zaporedne, logično zaključene mikronaloge. Vsak korak je bil poslan agentu kot izoliran ukaz z natančno definiranimi zahtevami (Slika 1):

1. Priprava projektne strukture: Prva mikronaloga obsega inicializacijo nove projektne topologije na podlagi vnaprej pripravljene predloge. Agent analizira izvirne datoteke predloge, določi konvencije poimenovanja ter ustvari ciljne podprojekte. Sledi samodejni prenos infrastrukturnih datotek, rekurzivno preimenovanje map ter posodobitev datotek MauiProgram.cs in AppShell.xaml.
2. Uvedba orodja Refitter [8]: Pomembna arhitekturna sprememba v primerjavi s prvotno aplikacijo Xamarin je posodobitev komunikacijskega sloja z zalednimi sistemi. V prejšnji arhitekturi se je za generiranje odjemalcev uporabljalo orodje NSwagStudio [5], ki je generiralo obsežne in toge monolitne datoteke. V novi arhitekturi smo to zamenjali s sodobnim orodjem Refitter [8]. Ta korak je bil izveden ročno, za postavitev trdnih temeljev pred predajo dela agentu UI.
3. Migracija lokalizacije: Agent identificira stare .resx datoteke, jih preseli v centraliziran projekt ter izvede ekstrakcijo trdo kodiranih nizov iz datotek pogleda XAML in logike C# v slovenske in angleške vire ter ob tem implementira osrednjega skrbnika LocalizationResourceManager, ki skrbi za podporo več jezikom hkrati.

4. Migracija domenskih modelov: Na podlagi predhodno pripravljenih podatkovnih prenosnih objektov (DTO) iz orodja Refitter v 2. koraku agent samodejno ustvari ustrezne domenske modele. Hkrati zgradi profile za knjižnico AutoMapper, pri čemer dosledno uporablja deklaracije metode CreateMap.



Slika 1: Diagram postopka migracije aplikacije iz ogrodja Xamarin.Forms v ogrodje .NET MAUI.

5. Prenova storitvenega sloja: V tem koraku agent izvede preoblikovanje storitvene logike, pri katerem strogo loči vmesnike od njihovih implementacij. Stari klici, generirani z orodjem NSwag, se nadomestijo s sodobnimi odjemalci Refitter. Agent znotraj storitev implementira IMapper za pretvorbo DTO-jev v domenske modele ter samodejno registrira storitve. Pomembno pravilo, ki ga agent tukaj upošteva, je prepoved neposrednega klicanja odjemalcev HTTP iz modelov pogleda. Klici se morajo izvajati izključno z uporabo metod poslovnih storitev na podlagi generične metode `SendRequest()`.
6. Modernizacija modelov pogleda in prenova navigacije: V tej fazi agent prenovi sloj predstavitevne logike z uporabo ogrodja CommunityToolkit.Mvvm [6]. Odstrani se obsežna koda za ročno proženje dogodkov sprememb lastnosti, ki jo nadomestijo izvorno generirane lastnosti in ukazi na podlagi atributov

[ObservableProperty] in [RelayCommand]. Prejšnji način navigacije `Navigation.PushAsync` se pretvori v vzorec lupine ogrodja `MAUI.Shell.Current.GoToAsync()`. Hkrati se uveljavi pridobivanje storitev iz konstruktorja.

7. Abstrakcija in transformacija podatkovnih tabel: Ker so aplikacije v naši organizaciji močno odvisne od prikazov podatkov, je bil v reprezentativnem vzorcu aplikacije razvit generičen pogled za prikaz podatkov v obliki tabele, imenovan `DynamicDataGridContentView`. Naloga agenta je bila, da analizira stare datoteke uporabniških vmesnikov XAML, identificira implementacije tabel ter generira prečiščen, dinamičen pogled, ki stolpce generira programsko z uporabo metode `GenerateColumns()`. Definicije stolpcev za specifične tipe modelov agent strukturira znotraj statičnega razreda `DataGridHelper`. Ker je takšnih tabel veliko, smo ta korak izolirali in ga poslali agentu kot samostojen ukaz.
8. Transformacija uporabniškega vmesnika in integracija knjižnice `DevExpress`: Sledi celovita prenova datotek uporabniškega vmesnika XAML. Agent samodejno posodobi imenske prostore iz `Xamarin.Forms` v `Microsoft.Maui.Controls` ter integrira gradnike knjižnice `DevExpress`. Iz ozadja se odstrani vsa poslovna logika, dogodki elementov se povežejo neposredno na ukaze modela pogleda.
9. Dopolnitve videza aplikacije: Zadnja faza obsega vizualno dopolnjevanje in prenos naprednih vizualnih funkcionalnosti. Agent znotraj datoteke `Styles.xaml` poenoti videz vizualnih komponent. Agent implementira podporo za podrsanje elementov z uporabo ikon. Ključni del te faze je uspešna obnova stare logike `CustomCellStyleCommands`, ki je skrbela za dinamično barvanje celic tabel, ter njena uspešna prilagoditev za ekosistem ogrodja `.NET MAUI`. Na koncu agent poenoti videz pojavnih oken in uredi poravnavo akcijskih gumbov.

Po vsakem zaključenem modulu oziroma koraku je agent dolžan zagnati ukaz za prevajanje rešitve. Če prevajalnik sporoči napake, se agent vrne v način načrtovanja, analizira izpis napak ter samostojno uporabi popravke. Arhitekt prevzame nadzor nad kodo šele takrat, ko je posamezni modul uspešno preveden brez napak.

5 Varnostni protokoli in zaščita intelektualne lastnine v oblaku

Prehod z lokalnih modelov na visoko zmogljive oblačne agente, kot je `Claude Code`, izpostavi organizacijo specifičnim varnostnim tveganjem. Glavni izziv v organizacijskem okolju je zaščita integritete izvorne kode in preprečevanje odtekanja intelektualne lastnine na zunanje strežnike ponudnikov storitev umetne inteligence. Za omilitev tveganj in nadzor nad podatkovnimi tokovi je bil vzpostavljen hibridni varnostni model, ki temelji na dveh tehničnih stebrih:

- Selektivna izolacija podatkov z uporabo datoteke `.claudeignore`: Podobno kot datoteka `.gitignore` v sistemih za upravljanje različic, datoteka `.claudeignore` omogoča izrecno določitev datotek in map, do katerih oblačni agent pod nobenim pogojem nima dostopa. V to konfiguracijo so bili sistemsko vključeni: vsi produkcijski konfiguracijski podatki, certifikati in šifrirni ključi (`.env`, `appsettings.json`), mape gradnje (`bin/` in `obj/`) ter posebni moduli, ki vsebujejo občutljive podatke o poslovnih procesih podjetja.
- Človeški nadzor: Uveden je bil strog protokol strokovnega pregleda. To pomeni preverjanje, ali je agent spreminjal izključno datoteke v odobrenem obsegu in ali ni prišlo do nenamerne vnašanja varnostnih ranljivosti (npr. trdo kodiranih dostopov ali nepreverjenih zunanjih odvisnosti) ter ocenjevanje, ali generirana koda dosledno sledi vzorcem, definiranim v izhodiščnem vodniku `MAUI_MIGRATION_GUIDE.md`. Šele po uspešnem pregledu in lokalnem testu delovanja se programska koda odobri za združitev v glavno razvojno vejo.

Simbiotični odnos med arhitektom in agentom UI tako zagotavlja, da ima človek popolno in zadnjo besedo nad vsemi spremembami, kar zmanjšuje tveganja oblačne obdelave podatkov.

6 Analiza rezultatov

Za objektivno oceno predlagane metodologije smo izvedli primerjavo med konvencionalnim ročnim postopkom migracije in agentsko podprtim delovnim tokom. Primarni cilj analize je bil ovrednotiti časovne prihranke, identificirati tipične vzorce napak sistemov umetne inteligence ter ovrednotiti ekonomsko upravičenost uvedbe naprednih agentov v razvojni cikel.

Analiza je bila izvedena na produkcijski mobilni aplikaciji, ki se uporablja v internem logističnega in proizvodnega ekosistema. Aplikacijo predstavljajo naslednji parametri:

- Obseg rešitve: 8 ločenih podprojektov znotraj enotne rešitve.
- Uporabniški vmesnik: 12 uporabniških pogledov XAML z gostimi podatkovnimi strukturami in interaktivnimi elementi.
- Velikost kodne baze: več kot 180.000 vrstic kode.

Za potrebe študije primera je bila izvedena primerjava med časovnim okvirjem za popolno ročno migracijo in izvedbo s pomočjo orodja Claude Code. Ročna migracija je bila izvedena pred uporabo agenta UI. Rezultate prikazuje Tabela 1.ⁱ

Tabela 1: Primerjava ročne in agentske migracije.

Metrika	Ročna migracija reprezentativnega projekta	Agentsko podprta migracija
Čas izvedbe	3 tedni (120 ur)	9 dni (56 ur)
Obseg rešitve	8 projektov	8 projektov
SLOC	171 000	180 000
Potrebni ročni popravki (Commits)	/	~ 30
Stabilnost prve gradnje	Nizka (več 100 napak)	Visoka (iterativno odpravljanje z uporabo sprotne gradnje)

Rezultati kažejo na precejšnje zmanjšanje potrebnega časa za izvedbo celotne migracije. Ročni prehod celotne rešitve z danimi omejitvami in funkcionalnostmi bi od izkušenega inženirja zahteval približno tri tedne intenzivnega dela (vključno z izvedbo migracije, testiranjem in prenosom v produkcijsko okolje). Z uporabo orodja Claude Code in organiziranim pristopom je bil proces zaključen v dobrem tednu dni, kar predstavlja 54-odstotni prihranek časa.

Med procesom migracije smo sistematično beležili primere, ko je agent odpovedal ali generiral napačne programske vzorce (t. i. halucinacije). Te anomalije razdelimo v pet glavnih kategorij:

- Omejitve pri interpretaciji zaprtih zunanjih odvisnosti: agent ni znal implementirati in prenesti logike, povezane z namensko knjižnico HubService. Ker gre za interno, lokalno knjižnico NuGet, ki ni del javno dostopnih podatkov za učenje modela, agent brez zunanje dokumentacije ni mogel pravilno razbrati njene semantike in je pustil komentar z opisom opravlja (Slika 2).
- Pomanjkljivo in napačno preslikovanje podatkovnih modelov: pri ročnem preslikovanju modelov je agent občasno izpustil določene lastnosti ali pa jih je preslikal nepravilno (Slika 3, Slika 4).
- Ignoriranje referenčnih primerov: kljub eksplicitnim navodilom in obstoju referenčnega projekta je agent pri transformaciji tabel večkrat poskušal uporabiti zastareli koncept iz ogrodja XAML. Ni uspel takoj prepoznati, da nova arhitektura zahteva uporabo prenovljene komponente `DynamicDataGridContentView`, ki na tipiziran način dinamično generira uporabniški vmesnik.
- Izpad kompleksnih kontrol (`CustomCellStyleCommand`): zaradi sprememb v načinu definiranja vizualnih stanj v ogrodju .NET MAUI je agent na več mestih izpustil implementacijo ukaza

CustomCellStyleCommand (Slika 5). Kompleksne logike barvanja celic na podlagi pogojev ni uspel avtonomno prilagoditi novemu ekosistemu, zato je bil na tem delu potreben ročni poseg arhitekta.

- Težave pri definiranju vizualne identitete in videza aplikacije: Agent je imel težave pri zagotavljanju enakega ali izboljšanega videza uporabniškega vmesnika v primerjavi z izvirno različico. Brez dodatnega stilskega usmerjanja je generiral osnovne, neoptimizirane postavitve elementov, napačne odmike ter nedosledno tipografijo. Vizualni rezultat neposredno po zaključku avtonomnega dela agenta in pred kakršnimi koli ročnimi popravki razvijalca prikazuje slika 6.

Vse navedene napake so bile uspešno odpravljene s kombinacijo natančnejšega usmerjanja agenta v načinu načrtovanja v zadnjem koraku in končnih ročnih popravkov razvijalca.

```

2 references
42 partial void OnWorkOrderChanged(WorkOrderModel value)
43 {
44     if (value == null) return;
45     _screenService.SetScreenAlwaysOn();
46     Title = string.Format(
47         LocalizationResourceManager.Instance["BatchPreparationPageTitle"]?.ToString() ?? "",
48         value.ProductionLineName, value.BatchId, value.AlloyName, value.WorkOrderId);
49     IsSummaryVisible = AppSettingsService.AppSettings.TenantId == "27";
50     Task.Run(async () =>
51     {
52         await FetchBatchSuggestionDataAsync();
53         await FetchScannedDataAsync();
54     });
55     // TODO: IHubService - RegisterHubListeners()
56 }
57

```

Slika 2: Komentar agenta UI za nadaljnje opravilo.

<pre> 47 public string GenerateQrCodeStringForMobile(UserInfoModel user, WorkOrderModel workOrder) 48 { 49 var model = new QrCodeForMobileModel 50 { 51 Id = user.Id, 52 Login = user.Login, 53 FullName = user.FullName, 54 Rfid = user.RFID, 55 Token = user.Token, 56 LdapUid = user.LdapUid, 57 BatchPreparationInfo = new BatchPreparationInfoModel 58 { 59 BatchId = workOrder.BatchId, 60 ProductionLineId = workOrder.ProductionLineId, 61 ProductionLineName = workOrder.ProductionLineName, 62 ProductionLineCode = string.Empty, 63 ForkliftLocationId = forkliftLocationId, 64 ForkliftLocationCode = forkliftLocationCode, 65 QrCodeType = isBatchAlloying ? "alloying" : "casting" 66 } 67 }; 68 var json = JsonSerializer.Serialize(model); 69 return Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(json)); 70 } 71 </pre>	<pre> 47 public string GenerateQrCodeStringForMobile(UserInfoModel user, WorkOrderModel workOrder) 48 { 49 var model = new QrCodeForMobileModel 50 { 51 Id = user.Id, 52 Login = user.Login, 53 FirstName = user.FirstName, 54 LastName = user.LastName, 55 Email = user.Email, 56 FullName = user.FullName, 57 Rfid = user.RFID, 58 Token = user.Token, 59 LdapUid = user.LdapUid, 60 BatchPreparationInfo = new BatchPreparationInfoModel 61 { 62 BatchId = workOrder.BatchId, 63 ProductionLineId = workOrder.ProductionLineId, 64 ProductionLineName = workOrder.ProductionLineName, 65 ProductionLineCode = string.Empty, 66 ForkliftLocationId = forkliftLocationId, 67 ForkliftLocationCode = forkliftLocationCode, 68 QrCodeType = isBatchAlloying ? "BatchAlloying" : "BatchComposition" 69 } 70 }; 71 var json = JsonSerializer.Serialize(model); 72 return Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(json)); 73 } 74 </pre>
---	--

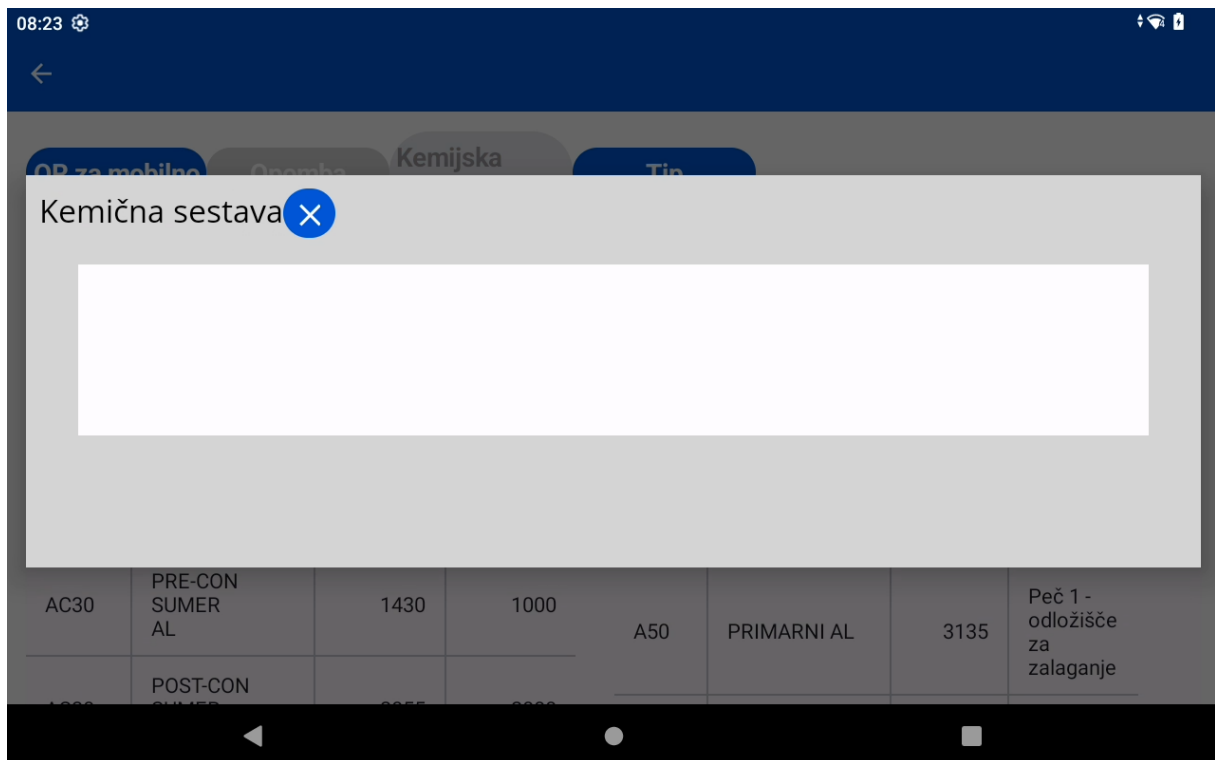
Slika 3: Manjkajoče preslikave po kodiranju agenta UI.

<pre> 95 private async Task FetchScannedDataAsync() 96 { 97 if (WorkOrder == null) return; 98 IsScannedDataRefreshing = true; 99 var stock = await _compositionProposalService.GetPackagesInDeliveryAsync(WorkOrder.BatchId); 100 if (stock != null) 101 { 102 await MainThread.InvokeOnMainThreadAsync(() => 103 { 104 ScannedData = stock.AlloyList == null 105 ? new BindingList<AlloyWithLocationAndQuantityModel>() 106 : new BindingList<AlloyWithLocationAndQuantityModel>(stock.AlloyList.ToList()); 107 scannedTransportUnitIds = stock.TransportUnits?.Select(x => x.PisZapStev).ToList() 108 ?? new List<int>(); 109 OstanekQuantity = stock.LeftoverQuantityFromPrevBatches; 110 SkeniranoQuantity = stock.AlloyList?.Sum(x => x.Quantity) ?? 0m; 111 SkupnoQuantity = OstanekQuantity + SkeniranoQuantity; 112 }); 113 } 114 IsScannedDataRefreshing = false; 115 } 116 </pre>	<pre> 95 private async Task FetchScannedDataAsync() 96 { 97 if (WorkOrder == null) return; 98 IsScannedDataRefreshing = true; 99 var stock = await _compositionProposalService.GetPackagesInDeliveryAsync(WorkOrder.BatchId); 100 if (stock != null) 101 { 102 await MainThread.InvokeOnMainThreadAsync(() => 103 { 104 ScannedData = stock.AlloyList == null 105 ? new BindingList<AlloyWithLocationAndQuantityModel>() 106 : new BindingList<AlloyWithLocationAndQuantityModel>(stock.AlloyList.ToList()); 107 scannedTransportUnitIds = stock.TransportUnits?.Select(x => x.TransportUnitId).ToList() 108 ?? new List<int>(); 109 OstanekQuantity = stock.LeftoverQuantityFromPrevBatches; 110 SkeniranoQuantity = stock.AlloyList?.Sum(x => x.Quantity) ?? 0m; 111 SkupnoQuantity = OstanekQuantity + SkeniranoQuantity; 112 }); 113 } 114 IsScannedDataRefreshing = false; 115 } 116 </pre>
--	---

Slika 4: Primer nepravilnega preslikovanja lastnosti.



Slika 5: Manjkajoče implementacije metod stiliranja celic tabel.



Slika 6: Primer videza aplikacije po migraciji z agentom UI brez ročnih popravkov.

7 Zaključek

Rezultati prispevka in praktične implementacije agentsko podprte migracije potrjujejo, da sistemi umetne inteligence spreminjajo procese vzdrževanja in posodabljanja obsežnih programskih rešitev. Prehod z determinističnih orodij na kognitivne agente omogoča, da se proces migracije iz gole sintaktične zamenjave preoblikuje v nadzorovano migracijo programske kode v realnem času.

Empirični podatki iz študije primera kažejo, da se je čas izvedbe migracije zmanjšal za 54 % — s treh tednov ročnega dela na dober teden inženirskega časa. Kljub širokemu kontekstnemu oknu in kognitivni globini oblačnih agentov, kot je Claude Code, njihovo delovanje izkazuje določene omejitve. Anomalije, kot so pomanjkljivo preslikovanje modelov, nerazumevanje lokalnih knjižnic ter nagnjenost k uporabi zastarelih komponent pogleda namesto prenovljenih predlog, dokazujejo, da popolna avtomatizacija teh procesov trenutno še ni izvedljiva.

Uspešna, varna in ekonomsko upravičena migracija zato ne temelji na polni avtonomiji stroja, temveč na simbiozi med programskim arhitektom in agentom UI. Človek v tem procesu prevzema vlogo postavljanja izhodiščnih vzorcev, priprave sistemskih navodil ter večplastne zaščite intelektualne lastnine z uporabo konfiguracij .claudeignore in arhitekturne anonimizacije. Končni korak strokovnega pregleda kode zagotavlja, da razvijalec ohranja popoln nadzor nad produkcijskim okoljem.

Uporaba opisanega pristopa nakazuje, da je z ustreznim inženirstvom navodil in implementacijo strogih varnostnih protokolov mogoče obvladovati tveganja, povezana z integracijo oblačnih modelov umetne inteligence. To organizacijam omogoča, da proces migracije na novejša različice ogrodi izvedejo z višjo stopnjo predvidljivosti in optimizacije razvojnega cikla.

Literatura

- [1] <https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview>, What is .NET Upgrade Assistant?, obiskano 29. 6. 2026.
- [2] <https://ollama.com/>, Ollama, obiskano 29. 6. 2026.
- [3] Guo, D. et al. (DeepSeek-AI). *DeepSeek-Coder: Touching the Ceiling of Open Source Code Intelligence*. arXiv preprint.
- [4] <https://claude.com/product/claude-code>, Claude Code, obiskano 29. 6. 2026.
- [5] <https://github.com/RicoSuter/NSwag/wiki/nswagstudio>, NSwagStudio, obiskano 29. 6. 2026.
- [6] <https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/>, Introduction to the MVVM Toolkit, obiskano 29. 6. 2026.
- [7] <https://www.devexpress.com/maui/>, .NET MAUI Controls, obiskano 29. 6. 2026.
- [8] <https://refitter.github.io/>, Refitter, obiskano 30. 6. 2026.

Ko umetna inteligenca prevzame razvoj internih orodij

Gregor Kovač, Tomaž Kozlar, Žan Žvokelj

Bilumina d.o.o., Maribor, Slovenija

gregor.kovac@bilumina.com, tomaz.kozlar@bilumina.com, zan.zvokelj@bilumina.com

V lanskem prispevku smo predstavili lastno low-code platformo za razvoj ERP sistema, zgrajeno na jeziku SQL in jedru v Javi, z orodjema FormTool in Linea. Letos preverjamo, ali lahko umetna inteligenca postane resnična pomoč pri razvoju in vzdrževanju naših internih orodij. Odgovor najprej iščemo na primeru QueryToola, ki ga s pomočjo orodja za agentno programiranje prepišemo iz obstoječega okolja na spletno platformo Laravel. Umetna inteligenca nam je pomagala tudi pri drugih internih orodjih, kot so spremljanje, postavitev strežnikov, pomoč uporabnikom in podobno. Izkušnje umestimo v širši, inženirski in etični okvir ter izpostavimo, kje je bila pomoč umetne intelligence produktivna in kje je vodilno vlogo ohranil razvijalec.

Ključne besede:

umetna inteligenca

low-code

Laravel

QueryTool

Linea

1 Uvod

V prispevku na lanski konferenci OTS 2025 smo predstavili lasten low-code pristop k razvoju kompleksnega in visoko konfigurabilnega ERP sistema [1]. Pristop temelji na programskem jeziku SQL in jedru, napisanem v Javi, podpirata pa ga dve interni orodji: FormTool za oblikovanje vnosnih mask in Linea za načrtovanje podatkovne baze. Na ta način smo doslej zgradili več kot tisoč vnosnih mask in več tisoč baznih objektov, ob tem pa skrajšali čas razvoja in zmanjšali možnost napak.

V preteklem letu se nam je vse pogostejše zastavljalo vprašanje, ki si ga danes zastavlja velik del razvojne stroke: ali lahko umetno inteligenco, ki je postala izrazito zmogljiva in dostopna, uporabimo kot resnično pomoč pri razvoju in vzdrževanju naših internih orodij – ne le kot pripomoček za dopolnjevanje kode, temveč kot sodelavca pri večjih, dobro opredeljenih nalogah?

2 Prepis QueryToola na Laravel s pomočjo umetne inteligence

2.1. Kaj je QueryTool in zakaj smo ga izbrali za pilotni projekt

QueryTool je naše interno orodje za razčlenitev stavkov SQL na sestavne komponente: stolpce v stavku SELECT s pripadajočimi podatkovnimi tipi, pogoje v stavku WHERE z operatorji, tabele in poglede v delih FROM in JOIN, parametre (npr. :ID, :COUN_ID), agregacijske in skupinske klavzule (GROUP BY, HAVING), urejanje (ORDER BY) ter ugnezdene poizvedbe. Razčlenjena struktura se uporablja na več mestih v rešitvi ERP – pri povezovanju parametrov SQL in rezultatov poizvedbe s polji na vnosni maski. Ker je orodje funkcionalno zaokroženo, smo ga prepoznali kot idealnega kandidata za pilotni projekt.

QueryTool je napisan v programskem jeziku Java s pomočjo javanskega ogrodja GWT (Google Web Toolkit).

Action code	SQL
CALL_LOGIC.CREATE_CONTACT_MANUAL	CALL LOGIC.CREATE_CONTACT(:PART_ID , :WJPO_ID , :NAME , :SURNAME , :PHONEJOB , :PHONEHOME , :PHONEMOBILE , :MAILJO
CHECK_PCON_MAIN_QUERY_EXTENSION	SELECT PCCA.PCCA_CODE, GEND.GEND_CODE, GEND.GEND_NAMESMALL, POST.POST_CODE, POST.POST_CITY, WJPO.WJPO_CC
EXECUTE_ITEM_LIST_MANUAL_QUERY	SELECT ITEM_ACTIVE, ITEM_CODE, ITEM_CREATED, ITEM_DESIGNOWNER, ITEM_DESCRIPTIONLARGE, ITEM_DESCRIPTIONSMALL
PARTNER_SEARCH_ALL_MANUAL	SELECT PART.PART_ID, PART.PART_CODE, PART.PART_TITLES SMALL, PART.PART_TITLELARGE, PCON.PCON_STREET AS PART_STRE
EXECUTE_T101_NOTEXISTING_QUERY_EXT...	SELECT T101.T101_AGEFROM, T101.T101_AGETO, T101.T101_ASSORTMENT, T101.T101_BARCODE, T101.T101_BRAND, T101.T101_BRANDNA
EXECUTE_USER_AUTO_QUERY	SELECT USER.MODU_ID, USER.MODU_MODU_CODE, USER.MODU_MODU_ID, USER.MODU_MODU_NAMESMALL, USER.MODU_PART_I
EXECUTE_T101_MANUAL_QUERY	SELECT T101.ITEM_ID, T101.ITEM_CODE, T101.ITEM_ITEM_NAMESMALL, T101.T101_AGEFROM, T101.T101_AGETO, T101.T101_ASSOR

Param. code	Type name	Default value	F1 where fix	Equals sign (IN...	Field code	Type name	Default value	F1 update field
BONUSVALUE...	DECIMAL				BUT_QUANTITY	DECIMAL		
BONUSVALUE...	DECIMAL				COUN_CODE	VARCHAR		
BUTY_ID_LOY...	BIGINT				COUN_ID	BIGINT		
FETCH	BIGINT				COUN_NAMESMALL	VARCHAR		
MAIN	SMALLINT				GEND_CODE	VARCHAR		
					GEND_ID	BIGINT		
					GEND_NAMESMALL	VARCHAR		
					NUMBER_OF_CHILDREN	INTEGER		
					PART_ACTIVE	SMALLINT		
					PART_CODE	VARCHAR		
					PART_CODEOLD	VARCHAR		

Slika 1: QueryTool

Vse podatke, ki jih vidite na sliki 1, uporabljamo v ERP aplikaciji pri izvajanju akcij, na katere je ta poizvedba vezana, in za preverjanje podatkov.

Ta rešitev ima tudi omejitve, ki jih sčasoma nismo odpravili iz različnih razlogov. Ena od takšnih omejitev je tudi, da orodje ne zna pridobiti ustreznih podatkov iz procedure v podatkovni bazi.

2.2. Razvojni proces s pomočjo umetne inteligence

2.2.1. Kako deluje velik jezikovni model

Preden opišemo sam potek dela, na kratko povzemimo, kako deluje velik jezikovni model (LLM), saj to pojasni tako njegove prednosti kot omejitve, na katere smo naleteli. LLM je model, naučen na velikih količinah besedila, ki deluje po enem osnovnem načelu: na podlagi dosedanjega besedila napoveduje najverjetnejši naslednji »žeton« (angl. token) – delček besede. Besedilo se najprej razbije na žetone, iz njih pa model zaporedoma gradi odgovor, žeton za žetonom.

Ključni gradnik je mehanizem pozornosti (angl. attention): pred izbiro vsakega novega žetona model presodi, kako pomemben je vsak od dosedanjih žetonov. Ta pozornost je omejen proračun – njena skupna teža je stalna in se porazdeli med vse žetone. Ko je konteksta malo, pomemben podatek dobi znaten delež pozornosti; ko je konteksta veliko, se isti podatek utopi med tisoči drugih [3]. To je neposreden vzrok za razpadanje konteksta in razlog, da se je pri prepisu obneslo delo z majhnimi, jasno omejenimi enotami.

Model nima baze dejstev, temveč statistične vzorce iz učnih podatkov. Zato je lahko izjemno tekoč in prepričljiv, hkrati pa lahko samozavestno navede napačen podatek (halucinacija) ali spregleda pomenske podrobnosti, na primer zanikanje [7]. Prav tako nima resničnega razumevanja izvirne kode ali poslovne logike; deluje po vzorcih, ki jih je videl. Za nas iz tega sledita dve praktični posledici: navodila in kontekst morajo biti konkretni in omejeni, vsak rezultat pa je treba preveriti glede na dokazano pravilno referenco – kar podrobneje opišemo v nadaljevanju.

2.2.2. Izbrani izdelki

Po tehtnem premisleku in posvetovanju smo v podjetju izbrali izdelek Claude podjetja Anthropic.

Claude ni vezan na en sam model, temveč omogoča izbiro med več modeli iste družine. To je koristno, ker za vsako vrsto naloge – zahtevno sklepanje, hitro rutinsko delo ali delo z dolгим kontekstom – izberemo najprimernejši model in s tem uravnovesimo kakovost, hitrost in stroške. V času pisanja so na voljo naslednji modeli [11]:

- **Claude Opus 4.8** - najzmogljivejši model za zahtevno agentno programiranje in poslovne naloge; kontekstno okno milijon žetonov,
- **Claude Sonnet 5** - najboljše razmerje med hitrostjo in zmogljivostjo; primeren za vsakodnevno razvojno delo; kontekstno okno milijon žetonov,
- **Claude Haiku 4.5** - najhitrejši model z zmogljivostjo blizu vrhunske; primeren za hitre, pogoste in cenovno občutljive naloge; kontekstno okno 200 tisoč žetonov,
- **Claude Fable 5** - najzmogljivejši širše dostopen model, zasnovan za dolgotrajne (angl. long-running) agente in najzahtevnejše naloge; kontekstno okno milijon žetonov.

Anthropic ponuja tudi specializiran model Claude Mythos 5 za obrambne kibernetkovarnostne naloge, ki pa je dostopen le po vabilu.

Pri prepisu QueryToola smo največ uporabljali model Claude Opus 4.8. Zanj smo se odločili, ker je zasnovan prav za zahtevno agentno programiranje in delo v poslovnem okolju: dobro razume obsežno izvirno kodo in poslovno logiko, njegovo veliko kontekstno okno (milijon žetonov) pa omogoča, da mu naenkrat predstavimo večje dele obstoječega orodja skupaj s pojasnili. Ob tem ponuja dobro razmerje med zmogljivostjo na eni strani ter hitrostjo in stroškom na drugi, kar je pomembno pri dolgotrajnem, iterativnem delu. Del najzahtevnejših nalog smo opravili z zmogljivejšim modelom Claude Fable 5, dokler njegova uporaba ni bila omejena na ameriške državljane; nato smo se v celoti vrnili na Opus 4.8.

2.3. Ponovna implementacija

Brez predhodnih izkušenj z agentnim programiranjem smo začeli po najbolj neposredni poti – z raziskovalnim, sproščenim slogom dela, ki se ga je prijelo ime »vibe coding«.

Najprej je bilo treba pripraviti spletno okolje z naslednjim pozivom:

```
V /var/www pripravi nov imenik queryTool z zadnjo verzijo Laravel-a in login  
ekranom, ki deluje preko Google prijave.
```

Po nekaj iteracijah prejmemo zadnje sporočilo Clauda:

```
Končano. V /var/www/queryTool stoji nov projekt z Laravel 13.18.1 (zadnja verzija) in delujočo Google prijavo.  
Kaj je narejeno  
Ogrodje  
• Svež laravel/laravel (v13.18.1), nameščen laravel/socialite za OAuth.  
• Baza SQLite, migracije pognane.  
Google prijava  
• GoogleAuthController.php — redirect, callback (poišče/ustvari uporabnika po google_id), logout.  
• routes/web.php — /login, /auth/google/redirect, /auth/google/callback, /logout, zaščiten /dashboard.  
• config/services.php — Google blok.  
• Migracija users: dodan google_id (unique), avatar, password je zdaj nullable.  
• User.php — google_id, avatar v fillable.  
• login.blade.php — prijavi zaslon z Google gumbom + dashboard.blade.php.  
• .env / .env.example — GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, GOOGLE_REDIRECT_URI.  
Preverjeno (dev strežnik)  
• / → 302 na /login ✓  
• /login izriše zaslon z gumbom ✓  
• /dashboard brez prijave → 302 na /login (auth middleware) ✓  
• /auth/google/redirect → 302 na accounts.google.com ✓  
Kar moraš še narediti sam  
Vpiši Google OAuth gesla v .env (GOOGLE_CLIENT_ID + GOOGLE_CLIENT_SECRET iz Google Cloud Console → Credentials, redirect URI  
http://localhost/auth/google/callback), nato php artisan config:clear. Postopek po korakih je v README.md.  
Opomba: imenik je last claude_██████████. Za serviranje prek www-data (nginx/apache) bo morda treba prilagoditi lastništvo storage/ in  
bootstrap/cache/.
```

Slika 2: Končno sporočilo priprave spletnega okolja

Pri poskusu odpiranja strani dobimo napako, ker prezremo opombo, Claude pa težavo uspešno odpravi.

Dobim tole napako:

```
ErrorException:
tempnam(): file created in the system's temporary directory

at
/var/www/queryTool/vendor/laravel/framework/src/Illuminate/Filesystem/Filesystem.php:222
at Illuminate\Foundation\Bootstrap\HandleExceptions->handleError()

(/var/www/queryTool/vendor/laravel/framework/src/Illuminate/Foundation/Bootstrap/HandleExceptions.php:263)
at Illuminate\Foundation\Bootstrap\HandleExceptions->
```

Show more

Napaka je klasična: Blade compiler ne more zapisati prevedenega viewa v `storage/framework/views` — torej serviajoči uporabnik (`www-data`, ker gre zdaj prek `https://[redacted]/`) nima pisalnih pravic v `storage/` in `bootstrap/cache/`. Preverim:

Inspected storage permissions and web server user >

Potrjeno: `nginx + php-fpm` tečeta kot `www-data`, `storage/` pa je last `[redacted]` z mode `775` — `www-data` spada le pod „other“ (`r-x`, brez pisanja), zato `tempnam` pade. Popravek: skupino nastavim na `www-data`, dodam `group-write` in `setgid`, da nove datoteke podedujejo skupino:

Fixed `storage/bootstrap-cache` group ownership and perms for `www-data` >

Skupina je zdaj `www-data` s `setgid`. Počistim stare cache in preverim, da prevajanje viewa deluje kot `www-data`:

Cleared caches and tested blade compile as `www-data` >

`www-data` zdaj brez težav prevaja Blade predloge. Počistim še cache, da spremembe (in nov `APP_URL`) veljajo takoj:

Cleared `view/config/cache` >

Rešeno.

Vzrok

Napaka `tempnam(): file created in the system's temporary directory` je nastala, ker Blade compiler ni mogel zapisati prevedenega viewa v `storage/framework/views`. Aplikacijo zdaj servira `nginx + php-fpm` kot `www-data`, mapa `storage/` pa je bila last `[redacted]` (mode `775`) — `www-data` je spadal le pod „other“ brez pisalne pravice, zato je `tempnam` padel nazaj na sistemski `/tmp` in sprožil izjemo.

(Napaka, ki si jo videl, je bila pravzaprav `renderer` izjema, ki je sam padel pri prevajanju svojega viewa — prava napaka spodaj je bila najbrž ista permission težava.)

Slika 3: Prva napaka in rešitev

Spletno stran ponovno preizkusimo in dobimo naslednjo podobno napako, ki jo Claude prav tako odpravi.

```
Sedaj dobim tole napako:  
Illuminate\Database\QueryException  
vendor/laravel/framework/src/Illuminate/Database/Connection.php:857  
SQLSTATE[HY000]: General error: 8 attempt to write a readonly database (Connection: sqlite, Database:  
/var/www/queryTool/database/database.sqlite, SQL: update "sessions" set "payload" =  
eyJfdG9rZW4iOiJ3WG9zZHRWd3k5UE9aMWwNtMW5pZjBHZ0s5aWY2MjZxcXlk1NE82RUdxiiwiX3ByZXZp  
b3VzIjlp7InVybCI6Imh0dHBzOlwvXC9jbGF1ZGUta292aS5iaWx1bWluYS5jb20iLCJyb3V0ZSI6bnVsbH0sil9  
mbGFzaCl6eyJvbGQiOiJldCJuZXciOiJldCJ0=, "last_activity" = 1783424282, "user_id" = ?, "ip_address" =  
[REDACTED], "user_agent" = Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/150.0.0.0 Safari/537.36, "id" = rhBDLnCrXwKCF3ifsnKG9COykNaAdTEBuz62pDc where "id" =  
rhBDLnCrXwKCF3ifsnKG9COykNaAdTEBuz62pDc)
```

Ista vrsta težave, tokrat SQLite: seje se shranjujejo v [database/database.sqlite](#), `www-data` pa nima pisalne pravice niti na datoteko niti na mapo `database/` (SQLite rabi pisati tudi `journal/WAL` v mapo). Preverim in popravim:

Fixed sqlite db + dir group ownership/perms for `www-data` »

Preverim, da `www-data` zdaj lahko piše v bazo:

Verified `www-data` can write to sqlite db »

Rešeno. Isti vzrok kot prej, le drug vir: seje so shranjene v SQLite (`SESSION_DRIVER=database`), `www-data` pa ni imel pisalne pravice ne na [database/database.sqlite](#) ne na mapo `database/` (SQLite pri pisanju ustvarja začasne `journal/WAL` datoteke v isti mapi).

Popravek

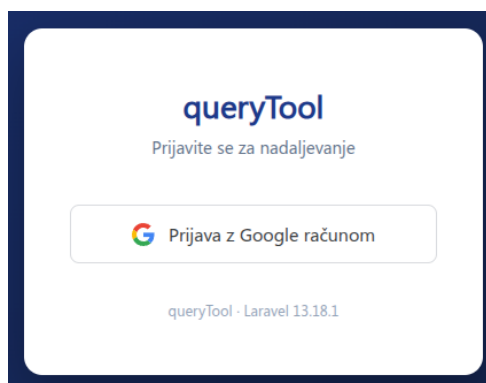
- `database/` → skupina `www-data`, `group-write` + `setgid`.
- [database/database.sqlite](#) → skupina `www-data`, `group-write`.
- Preverjeno: `www-data` uspešno naredi `INSERT/DROP` v bazi. ✓

Osveži stran — pisanje seje bo zdaj delovalo.

Skupni vzorec vseh treh napak je isti: projekt je last [REDACTED], servira pa ga `nginx/php-fpm` kot `www-data`. Če se pojavi še kje (npr. `storage/app`, nalaganje datotek), je rešitev enaka. Za trajno urejenost predlagam, da vse pisljive poti stalno pripadajo skupini `www-data` — to smo zdaj naredili za `storage/`, `bootstrap/cache/` in `database/`, kar pokriva običajne primere.

Slika 4: Druga napaka in rešitev

S to rešitvijo nas pričaka spletna stran.

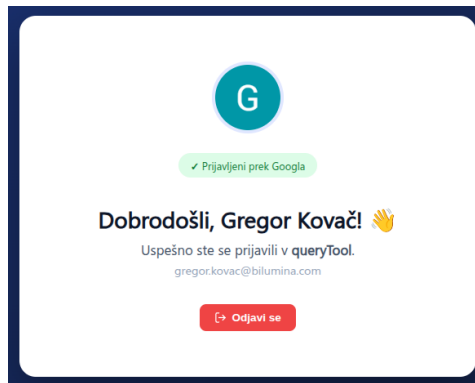


Slika 5: Prijavno okno

Ko po Claudovih navodilih pridobimo Google Client ID in skrivnost:

```
Nastavil sem GOOGLE_CLIENT_ID in GOOGLE_CLIENT_SECRET spremenljivke in ko se  
prijavim dobim 404. Naredi en welcome screen, dodaj tudi Odjavi gumb
```

Prijava in odjava delujeta:



Slika 6: Po prijavi

Vzpostavi dostop do baze in kreiraj stran za izvajanje poizvedb, samo SELECT-ov.

Gotovo. Povzetek.

Dostop do baze — vzpostavljen in preverjen

- `ibm_db2` je bil že nameščen (v2.3.1) — nič za namestit.
- DB2 okolje za `php-fpm` je pripravljeno prek `systemd drop-in` `db2.conf` (`clear_env=no`). Tekoči master ga ni imel → naredil `daemon-reload` + `restart php-fpm`.
- Potrjeno skozi `fpm socket`: `DB2INSTANCE=db2inst1` prisoten, `db2_connect` doseže strežnik `127.0.0.1:50000` (gonilnik + mreža delujeta). Web pot ima poln DB2 dostop.

Stran za poizvedbe (samo SELECT)

- `/query` (za prijavitelje) — `query.blade.php`: textarea, tabela rezultatov, čas izvedbe, opozorilo o skrajšanju.
- `QueryController.php` + poti v `web.php`.
- `Db2Connection.php` — native `ibm_db2`, DSN-less, `autocommit OFF` (nič se ne zapiše), meja `DB2_MAX_ROWS` (1000).
- `SelectOnlyValidator.php` — dovoli le en `SELECT/WITH/VALUES`, blokira DML/DDI, stacked queries in DB2 `FINAL/OLD/NEW TABLE(INSERT...)`. Vseh 8 testov gre skozi (vključno robni `created_orders`, ki ni napačno blokiran).
- `php artisan db2:ping` za test povezave.
- Povezava „SQL poizvedbe“ dodana na welcome screen in v navigacijo.

Slika 7: Vzpostavitev povezave na IBM Db2 bazo

Kar moraš narediti ti

Vpiši dostop v `.env` (ključi so že pripravljeni, prazni):

```
DB2_DATABASE=LUMINA
DB2_USERNAME=<bazni user>
DB2_PASSWORD=<geslo>
```

Nato:

```
php artisan config:clear
source /home/db2inst1/sqllib/db2profile && php artisan db2:ping
```

`db2:ping` zdaj pravilno javi „ni konfigurirano“ (dokaz, da je ožičenje pravilno — poveže se takoj ko vpišeš `credentiale`).

Varnostno priporočilo: uporabi DB2 uporabnika samo z pravico branja (SELECT). Validator + `autocommit OFF` sta prva obramba, `read-only user` pa zanesljiva — poizvedbena stran je javno dostopna vsem, ki se prijavijo prek Google.

Ko vpišeš `credentiale` in `db2:ping` uspe, lahko skupaj preverim še dejansko poizvedbo skozi spletni vmesnik.

Slika 8: Dopolnitev poverilnic

Sem jih vpisal

Credentials vpisani. Počistim config cache in poženem db2:ping:

Cleared config and ran db2:ping with real creds >

CLI povezava deluje (147 ms). Zdaj preverim še spletno pot (php-fpm z DB2 okoljem) z realno poizvedbo skozi aplikacijske razrede — sonda skozi fpm socket:

Tested real SELECT through php-fpm using app classes >

Vse deluje — potrjeno skozi spletno pot (php-fpm), z realnimi podatki:

- Povezava dela: db2:ping ✓ (147 ms, CLI).
- Realna poizvedba skozi aplikacijske razrede prek fpm socketa vrne prave podatke iz SYSCAT.TABLES:
 - CODE_ITEM — 214 tabel
 - TCODE_ITEM — 214
 - CHECK_CODE_ITEM — 213
- SELECT-only varovalka blokira DELETE ✓
- Sonda odstranjena, config cache počiščen.

Zaključek

Db2 dostop je vzpostavljen in stran /query je pripravljena za produkcijo. Odpri v brskalniku:

1. Prijavi se prek Googla → welcome screen.
2. Klikni „SQL poizvedbe (DB2)“ (ali navigacija → Poizvedbe).
3. Vpiši SELECT in izvedi. Rezultat je omejen na 1000 vrstic; ne-SELECT poizvedbe so zavrnjene, autocommit je izklopljen.

Ker se je povezala prek uporabnika lumina, opozorilo velja še naprej: preveri, da ima ta uporabnik na bazi LUMINA samo pravico branja — poizvedbena stran je dostopna vsem, ki se prijavijo prek Googla, in čeprav app blokira pisanje, je read-only DB user zanesljiva zadnja obramba.

Če želiš, lahko dodam še izvoz rezultatov v CSV ali zgodovino zadnjih poizvedb — povej.

Slika 9: Test povezave na bazo uspešen

SQL poizvedbe (DB2)

Dovoljene so samo SELECT poizvedbe. En stavek naenkrat.

```
select item_id, item_code from tcode_item.item limit 3
```

samo za branje Izvedi ▶

3 vrstic · 80 ms

ITEM_ID	ITEM_CODE
5	P0001
6	P0002
7	P0003

Slika 10: Uspešno izveden SELECT

Zdaj potrebujemo še orodje za analizo SQL stavkov:

```
zdej bomo pa naredil query tool - za delanje sqlov.
tabele so v shemi logic in sicer action , actionquery,
actionqueryfield in actionqueryparameter ..
analiziraj te tabele kaj so in kaj vsebinsko in mi
potem naredi query tool
```

Po nekaj iteracijah s Claudom:

Vse deluje in je preverjeno. Query tool je narejen.

Analiza tabel (kaj so)

Gre za metapodatkovni register parametriziranih poizvedb aplikacijskega ogrodja:

```
LOGIC.ACTION (9.593) → LOGIC.ACTIONQUERY (6.831) → ACTIONQUERYFIELD (76.505) [izhodni stolpci]
akcija          SQL (CLOB) + :params          ↳ ACTIONQUERYPARAMETER (16.969) [vhodni :parametri]
```

- ACTIONQUERY.AQUE_SQL = dejanski SQL z imenskimi parametri (:GLOS_CODE), pogosto WITH UR; lahko je SELECT ali CALL proc(...).
- FIELD = izhodna polja (ime+tip), PARAMETER = vhodni parametri (ime+tip), oba se ujemata z SQL-jem.

Orodje (kot si želel: vpišeš SQL → dobiš parametre s tipi + izhodne stolpce s tipi)

Nova stran /analyze ([analyze.blade.php](#)):

- Vpišeš SQL z :parametri → dve tabeli: Parametri (ime + tip + izvor) in Izhodni stolpci (ime + tip).
- Poizvedbe NE izvede — uporablja DB2 DESCRIBE (samo prepare).

Kako pridobi tipe ([Db2Describe.php](#)):

- Izhodni stolpci: DB2 CLP DESCRIBE OUTPUT → točni SQL tipi (VARCHAR(257), BIGINT, DATE, CLOB...). Parser obvlada osnovno in razširjeno (LOB) obliko.
- Parametri:
 - CALL → tipi iz signature procedure (SYSCAT.ROUTINEPARMS), pozicijsko.
 - SELECT → tip primerjanega stolpca (SYSCAT.COLUMNS, prek FROM/JOIN aliasov); prikaže tudi izvorni stolpec.
 - kjer ni mogoče → neznano (jasno označeno).

Preverjeno (skozi php-fpm kot www-data, na realnih podatkih)

- SELECT: :GLOS_CODE → VARCHAR(257), :LANG_ID → BIGINT; izhod vklj. GLOS_DESCRIPTIONLARGE → CLOB ✓
- CALL IMPORT.TEMPLATE_ITEM01: 61 parametrov pravilno preslikanih na proc signaturo ✓
- Robni primeri (CLOB razširjena oblika, več tabel) ✓

Povezave dodane na dashboard in v navigacijo (Domov · Poizvedbe · Analiza). Odpri /analyze po prijavi in vpiši SQL.

Opomba: /analyze bere le metapodatke (DESCRIBE + SYSCAT), ničesar ne izvede in ne piše. Če želiš, lahko dodam še: (a) gumb „naloži iz registra“ (izbereš ACTIONQUERY po ACTI_CODE in ti napolni SQL), ali (b) primerjavo dobljenih tipov z deklariranimi v ACTIONQUERYFIELD/ACTIONQUERYPARAMETER.

Slika 11: Implementacija QueryTool-a

Analiza SQL — parametri in izhodni stolpci

Vpiši SQL z imenskimi parametri (npr. :GLOS_CODE). Orodje prikaže **tip** parametrov in **tip** izhodnih stolpcev — brez izvajanja poizvedbe (DB2 DESCRIBE).

```
SELECT ITEM.ITEM_CODE, ICOD.ICOD_CODE  
FROM TCODE_ITEM.ITEM ITEM  
INNER JOIN TCODE_ITEM.CODE ICOD ON ICOD.ITEM_ID = ITEM.ITEM_ID  
WHERE ITEM.ITEM_CODE = :ITEM AND ITEM.ITEM_ACTIVE = :IS_ACTIVE
```

SELECT / WITH / VALUES ali CALL

Analiziraj ►

SELECT 2 parametrov · 2 izhodnih stolpcev

Parametri (vhod)

Imenski :parametri in njihovi tipi

Parameter	Tip	Vir
:ITEM	VARCHAR(30)	ITEM.ITEM_CODE
:IS_ACTIVE	SMALLINT	ITEM.ITEM_ACTIVE

Izhodni stolpci

Stolpci rezultata in njihovi tipi

Stolpec	Tip
ITEM_CODE	VARCHAR(30)
ICOD_CODE	VARCHAR(257)

Slika 12: Test SELECT-a

Preizkus procedure (CALL TEREN.OTS_2026(:NORDE_ID)) vrne seznam vhodnih parametrov, ne pa tudi informacij o vrnjenih podatkih (lahko je en nabor rezultatov, lahko jih je več), kar povemo tudi Claudu; rešitev najde, ko mu pokažemo še vzorec, po katerem so napisane naše procedure.

Poglej kako je narejena procedura `WD_FORM.SET_DOCUMENT` in poglej kateri sql je v katerem result setu. Podobno so narejeno tudi druge naše procedure.

Preberem celo telo `WD_FORM.SET_DOCUMENT` in izluščim: katere `WITH RETURN` kurzorje ima, na kateri `STATEMENT` je vsak vezan, in od kod (`SET/PREPARE`) dobi SQL. Tako razberem vzorec:

Created a file, ran a command >

`SET XSQL = 'SELECT ...'` — pri tej proceduri je SQL literal (ne kot pri `TEMPLATE_ITEM01`). To je ključno. Izvlečem celoten XSQL (od `SET XSQL` do `PREPARE`) in preverim pravo dolžino telesa:

Extracted full XSQL construction in `SET_DOCUMENT` >

Zdaj razumem vzorec. `WD_FORM.SET_DOCUMENT` (telo 6789 znakov):

```
SET XSQL = 'SELECT ... FROM ... WITH UR';    ← cel SELECT je LITERAL string
PREPARE XSQLSTATE_3 FROM XSQL;               ← pripravi iz spremenljivke XSQL
DECLARE XRESULT_SET_3 CURSOR WITH RETURN TO CLIENT FOR XSQLSTATE_3; ← 1 result set
```

Ključno: SQL je statično v telesu, le da gre prek `SET XSQL = 'literal' → PREPARE ... FROM XSQL` (spremenljivka), ne neposredno iz literala. Moja koda je iskala le literal neposredno v `PREPARE` — zato ni ujela tega vzorca. To je popravljivo (sledim spremenljivki do njenega `SET`).

Najprej konkretno odgovorim: opišem ta `SELECT` (result set #1):

Described `SET_DOCUMENT` result set `SELECT` >

Popolno — result set #1 (`XRESULT_SET_3`) vrne 14 stolpcev z natančnimi tipi (`VARCHAR(30)`, `TIMESTAMP`, `BIGINT` ...).

Zdaj razumem vaš splošni vzorec: `SET XSQL = 'literal SELECT' → PREPARE stmt FROM XSQL → DECLARE cursor WITH RETURN FOR stmt`. Nadgradim orodje, da sledi spremenljivki do `SET` literala in prikaže stolpce po posameznem result-setu. To pokrije `SET_DOCUMENT` in podobne procedure.

Najprej prenovim `CALL` logiko v `Db2Describe` (result-seti namesto enega seznama):

Edited `Db2Describe.php` +21 -15 >

Zdaj zamenjam `procedureResultColumns` z `procedureResultSets`, ki sledi vzorcu `SET var = 'literal' → PREPARE → kurzor`:

Edited `Db2Describe.php` +91 -42 >

Zdaj testiram na `SET_DOCUMENT` (1 result set) in `GET_ACTIONSAVE` (5 result setov) prek CLI:

Tested result set extraction on `SET_DOCUMENT` and `GET_ACTIONSAVE` >

`SET_DOCUMENT`: 1 result set, 14 stolpcev ✓. Ampak dva buga vidim: (1) pri `GET_ACTIONSAVE` regex prekomerno požre in najde le 1,3,5 namesto vseh 5 kurzorjev; (2) enkratni dinamični result set (npr. `TEMPLATE_ITEM01`) zdaj ne pade več na register (regresija). Popravim oboje — regex z offset-om in dopolnitev iz registra za enkratni dinamični result set:

Edited a file, ran a command >

Odlično, oba buga popravljena:

- `SET_DOCUMENT`: 1 result set, 14 stolpcev (iz telesa) ✓
- `GET_ACTIONSAVE`: vseh 5 result setov najdenih (dinamični → poštene opombe) ✓
- `TEMPLATE_ITEM01`: 1 result set, 154 stolpcev — izkazalo se je, da tudi ta uporablja `SET XSQL = 'literal'` (moja prejšnja ocena „dinamičen“ je bila napačna; zdaj ga orodje pravilno prebere iz telesa).

Zdaj posodobim UI, da prikaže stolpce po posameznem result-setu (in pripadajoči `SELECT`). Prenovim izhodni del `analyze.blade.php`:

Slika 13: Popravki za procedure

Analiza SQL — parametri in izhodni stolpci

Vpiši SQL z imenskimi parametri (npr. :GLOS_CODE). Orodje prikaže **tipe parametrov** in **tipe izhodnih stolpcev** — brez izvajanja poizvedbe (DB2 DESCRIBE).

```
CALL TERE.N.OTS_2026( :NORDE_ID )
```

SELECT / WITH / VALUES ali CALL

Analiziraj ►

CALL 1 parametrov · 2 result-set(ov)

⚠ Izhodni stolpci so razbrani iz telesa procedure (WITH RETURN kursorji).

Parametri (vhod)

Imenski :parametri in njihovi tipi

Parameter	Tip	Vir
:NORDE_ID	BIGINT	NORDE_ID

Result set 1 (XRESULT_SET_1)

► Prikaži SELECT (3 stolpcev)

Stolpec	Tip
ORDE_ID	BIGINT
ITEM_ID	BIGINT
OITE_QUANTITY	DECIMAL(23,7)

Result set 2 (XRESULT_SET_2)

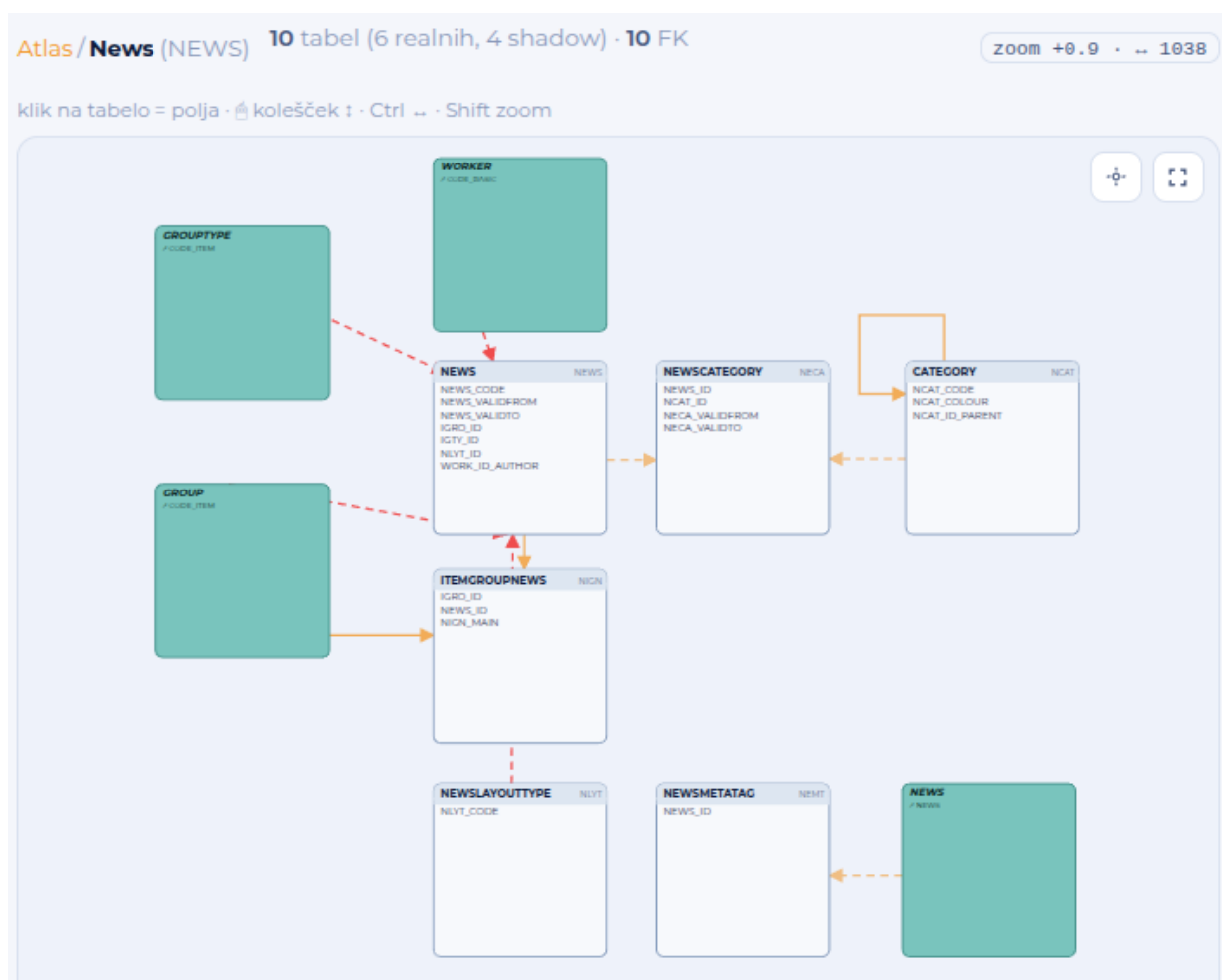
► Prikaži SELECT (3 stolpcev)

Stolpec	Tip
ORDE_ID	BIGINT
ITEM_ID	BIGINT
OITE_QUANTITY	DECIMAL(23,7)

Slika 14: Test procedure

2.4. Naslednji koraki

S pomočjo umetne inteligence smo v kratkem času (približno mesec dni) na ogrodje Laravel prenesli še nekaj drugih internih orodij, kot so Linea, monitoring baz in helpdesk.



Slika 15: »Nova« Linea

Monitor — Prikaz			
4 1 16 11			
STRANKA	METRIKA	VREDNOST	PRAG
●	FORCE_PROCEDURE	2	● 1
●	REPLICATION_CHECK_SUM_ALL_ERROR	1	● 1
●	SCAN_WD_AVG	229	● 120
●	STAT_2_CENT	2	● 1
●	SCAN_AVG	306	● 250 ● 700
●	CONNECTION_EXECUTE	8	● 10 ● 20
●	CONNECTION_LOCK	0	● 5
●	CONNECTION_TOO_MUCH_1	0	● 1
●	CONNECTION_TOO_MUCH_2	0	● 1
●	HCH	0	● 1
●		0	● 1
●	NO_DATA_CONTROL	0	● 1
●	PRINT_AVG	182	● 300 ● 700
●	PRINT_MAX	1063	● 2000 ● 5000
●	PRINT_TOO_MUCH	0	● 5 ● 10
●	SCAN_MAX	1668	● 10.000 ● 15.000

Slika 16: Spremljanje podatkovne baze

Helpdesk

Seznam

Tiketi v mojih procesih

Zadnja osvežitev: 08.07.2026 11:59 (pred 50s)

55 tiketov · 51 · prepis 4 · skrito 180

ODPRT

V ODELANI

REŠENO

ARHIVIRANO

Osveževanje: 30 s

Osveži zdaj

osveženo - 12:00:23

#1165

Manjkajoča evidenca prihoda

Uporabnik nima vpisane evidence prihoda za devet preteklih delovnih dni in jo mora dopolniti ali prijaviti odsotnost.

N

Portal (helpdesk portal)

#1145

Zapiranje razknjiževanja surovin na DN v minus na lokacijah Centralnega skladišča

Zapiranje minusa artikla 0000628 čokolada bela nova (-194 kg) na skladišču 100 LOK 0013 iz 6.7.2026.

A

Mrežna razpoložljivost (koskladišča)

#1142

ID 6841369

Artikel 0003989 se na izpisu prikazuje napačno - opis se ne izpisuje kot pri običajnih členih, kar onemogoča pošiljanje računov.

A

Atributi (dinamični)

#1141

Podatki za nadaljevanje nadgradnje maske za poročanje embalaže, maska

Dopolnjeni so manjkajoči podatki za nadgradnjo maske za poročanje embalaže: artikli, nove skupine VEMB, vrste embalaže in teže.

07.07.2026 08:28

Izpis prodane embalaže (embalažnina)

#1133

RE: [] Naročilo #3213 od []

Iz predračuna 2380481 se ni general račun in ga Tanja ne more narediti sama. Prosi za ureditev.

A

Spletna odprema (fulfillment / dispatch)

Slika 17: Helpdesk

Pri helpdesku rešitev bere elektronska sporočila, določi, v kateri poslovni proces sodijo, odpre zahtevek za ustrezno osebo, med prejšnjimi rešitvami poišče najustreznejšo ter predlaga odgovor stranki. Pri razvoju sistema za pomoč uporabnikom (helpdesk) smo ugotovili, da potrebuje jasno strukturiran spomin – zbirko rešitev, primerov odgovorov na elektronska sporočila in podobnih smernic. Šele v takšnih jasno začrtanih okvirih deluje zanesljivo; brez njih namreč začne odgovore preprosto ustvarjati sama.

Umetna inteligenca nam je z uporabo orodja Ansible pomagala zgraditi tudi rešitev za samodejno postavitve strežnikov različnih tipov (bazni, aplikacijski, spletni) ter njihovo nastavitve (dostopi do baz, nameščanje aplikacij na aplikacijske strežnike itd.)).

3 Širši pogled: zmožnosti in meje umetne inteligence pri razvoju

Naši izkušnji nista osamljeni. Postavljamo ju v širši okvir razprave o vlogi umetne inteligence pri razvoju programske opreme, in sicer z dveh zornih kotov: inženirskega (kaj orodja danes zmorejo in česa ne) in etičnega (kako naj to tehnologijo odgovorno uporabljamo).

3.1. Inženirski pogled: kje pomaga in kje ne

Koristno izhodišče je razumevanje razvoja programske opreme kot sklopa treh plasti: odločanja (kaj zgraditi), izvedbe (kako to zapisati v kodo) in dostave (kako to preoblikovati v zanesljivo delujoč izdelek). Umetna inteligenca najizraziteje pospešuje srednjo, izvedbeno plast, medtem ko se odločanje in dostava avtomatizaciji izmikata [2]. To pojasnjuje, zakaj kljub hitremu napredku zmogljivosti ni prišlo do napovedanih množičnih odpuščanj razvijalcev; številni odmevni primeri, ki so jih mediji pripisali umetni inteligenci, so bili v resnici posledica drugih dejavnikov, predvsem finančnih pritiskov [2]. V nekaterih primerih [12] so podjetja celo znova zaposlila inženirje, saj kakovost rešitev, ki jih je zagotavljala umetna inteligenca, ni dosegala zahtevanih standardov. Enako se je pokazalo v obeh naših primerih: umetna inteligenca je bila najbolj koristna pri izvedbi, torej pri pisanju kode, medtem ko smo odločitve o zasnovi in merila za sprejemljivost rezultata obdržali v lastnih rokah.

Zmogljivosti teh orodij ob tem hitro naraščajo. Podjetje Anthropic poroča, da razvoj umetne inteligence vse bolj poganja umetna inteligenca sama; njihovi inženirji danes v povprečju oddajo približno osemkrat več kode na četrtletje kot v obdobju 2021–2025, v skrajni obliki pa se tak trend približuje t. i. rekurzivnemu samoizboljševanju [9]. Prav zato postajajo večšine, ki niso neposredno pisanje kode – razumevanje pravega problema, zasnova rešitve, presoja in dostava – za razvijalca še pomembnejše, ne manj [10].

Ena pomembnejših praktičnih omejitev je t. i. razpadanje konteksta (context rot): kakovost odgovorov modela tiho upada, ko se kontekstno okno polni, saj je pozornost modela omejen vir, ki se z več žetoni razprši [3]. Napaka se ne pokaže kot izjema ali na nadzornih ploščah, temveč kot postopno izgubljanje natančnosti.

Zavedati se velja tudi, da konteksti, navodila in datoteke za usmerjanje agentov predstavljajo svojevrstno vrsto tehničnega dolga. Za razliko od kode, ki ostaja stabilna, dokler se je ne dotikamo, lahko vsaka nova različica modela tiho razvrednoti prej skrbno izdelana navodila [4]. Navodila naj zato temeljijo na konkretnih in preverljivih dejstvih o projektu, izogibamo pa se splošnemu »usmerjanju vedenja« modela.

Spodbudna je ugotovitev, da koda, ki je dobro vzdrževana in razumljiva za človeka, hkrati ostaja bolj zanesljiva tudi za urejanje z umetno inteligenco. Raziskava na 5.000 datotekah je pokazala pomembno povezavo med metriko zdravja kode (CodeHealth) in ohranitvijo pomena po preoblikovanju z jezikovnim modelom [5]; organizacije lahko take metrike uporabijo za presojo, kje je poseg umetne inteligence manj tvegan in kje je potreben dodaten človeški nadzor. Hkrati pa agentno programiranje zahteva več, ne manj discipline – več strukture, jasnejša merila kakovosti in nadzorne mehanizme [6].

Nazadnje ne gre prezreti, da jezikovni modeli ostajajo nezanesljivi na načine, ki niso vedno očitni: slabo obvladujejo zanikanja [7], vrhunski modeli pa si pri preverjanju dejstev pogosto nasprotujejo. Za nas to pomeni, da preverjanje rezultatov proti obstoječi, dokazano pravilni implementaciji ni neobvezen dodatek, temveč nujen del postopka.

Ne smemo pa spregledati stroškov umetne inteligence. Vlagatelji si bodo namreč na dolgi rok želeli povrniti stotine milijard dolarjev, ki jih trenutno vlagajo v podjetja, ki (še) ne poslujejo z dobičkom. Stroške lahko znižamo z naslednjimi ukrepi:

- **Uravnavanje vloženega napora (angl. effort).** Manjši vložen napor lahko prinese boljše razmerje med kakovostjo in ceno; model Fable 5 pri nastavitvi low doseže boljši rezultat (60 % pri 3,76 USD) kot Opus 4.8 pri nastavitvi max (59 % pri 13 USD) [13],
- **Prilagajanje modela zahtevnosti naloge.** Za vse naloge ni treba uporabljati najzmogljivejšega modela. Fable 5 lahko uporabimo za arhitekturo in načrtovanje, samo izvedbo pa glede na njeno zahtevnost prepustimo cenejšim modelom (Opus, Sonnet ali celo kateremu od modelov GPT),
- **Zmanjševanje porabe žetonov (angl. tokens).** Z uporabo namenskih veščin (angl. skills) za zmanjšanje porabe žetonov, kot sta Ponytail [14] in Caveman [15], dosežemo enak rezultat ob manjši porabi.

3.2. Etični pogled: enciklika Magnifica Humanitas

Razprava o umetni inteligenci ni le tehnična, temveč tudi etična. Dober okvir ponuja prva enciklika papeža Leona XIV. z naslovom Magnifica Humanitas: o varovanju človekove osebe v času umetne inteligence, objavljena 25. maja 2026 ob 135-letnici enciklike Rerum Novarum [8]. Enciklika tehnologije ne obravnava kot sovražne ali same po sebi zlonamerne, opozarja pa, da »tehnologija nikoli ni nevtralna, saj prevzame značilnosti tistih, ki jo zasnujejo, financirajo, urejajo in uporabljajo«. Poziva, naj umetna inteligenca služi človeku in skupnemu dobremu ter naj ne koncentrirja moči; med drugim izpostavlja dostojanstvo dela in poziv, naj »ostanemo človeški« [8].

Za razvijalce iz tega sledi konkretna drža. Umetna inteligenca naj bo orodje, ki razvojno ekipo razbremeni rutinskih opravil in ji omogoči osredotočenost na zahtevnejše, ustvarjalne naloge – ne pa nadomestek za človeško presojo in odgovornost. Ta drža se ujema z našo osrednjo ugotovitvijo: pri obeh primerih je vodilno vlogo – odločanje, presojo in končno odgovornost za rezultat – ohranil razvijalec.

Ta drža postaja še pomembnejša, ker razvoj umetne inteligence vse bolj poganja umetna inteligenca sama, s čimer se povečuje tudi tveganje izgube človeškega nadzora [9]. Toliko bolj velja poziv enciklike, naj človek ostane v središču odločanja in odgovornosti.

4 Zaključek in nadaljnji koraki

Naša izkušnja kaže, da je umetna inteligenca lahko resnična pomoč pri razvoju in vzdrževanju internih orodij, če nalogo dobro omejimo, jo razdelimo na preverljive enote in rezultate dosledno preverjamo. Pri implementaciji internih orodij je bila najbolj produktivna pri izvedbi, medtem ko sta zasnova in nadzor ostala v rokah razvijalca.

Ker se je pilotni prepis QueryTool-a izkazal za uspešnega, smo s pomočjo umetne inteligence na podoben način na sodobnejšo tehnološko osnovo prenesli tudi nekatera druga interna orodja. V prihodnje načrtujemo širšo uporabo umetne inteligence, in sicer za samostojno programiranje in podporo ter za njeno vpeljavo, po potrebi tudi za nadzor, neposredno v procese sistema ERP. Prispevek tako predstavlja prvi korak v širšem premisleku o tem, kako naj se uveljavljene interne razvojne prakse soočijo z orodji, ki postajajo iz dneva v dan zmogljivejša.

Literatura

- [1] BIZJAK Tomaž, KOVAČ Gregor, KOZLAR Tomaž: Implementacija prilagodljivega ERP sistema z malo ali nič kode. Sodobne informacijske tehnologije in storitve OTS 2025: Zbornik 28. konference, Maribor, 2025.
- [2] NARAYANAN Arvind, KAPOOR Sayash: Why AI hasn't replaced software engineers, and won't. normaltech.ai, 11. 6. 2026.
- [3] BHATKOTI Vineet: Understanding Context Rot - Why Your AI Agent Gets Worse the Longer It Works. DZone, 18. 6. 2026.
- [4] GOEDECKE Sean: Prompts are technical debt too. seangoedecke.com, 2026.
- [5] BORG Markus, HAGATULAH Nadim, TORNHILL Adam, SÖDERBERG Emma: Code for Machines, Not Just Humans: Quantifying AI-Friendliness with Code Health Metrics. arXiv:2601.02200, 2026.
- [6] TORNHILL Adam: Agentic AI Coding Practices for Speed with Quality (predstavitev), 2026.
- [7] MAYNE Harry, MCKINNEY Lev in dr.: Negation Neglect: When models fail to learn negations in training. arXiv:2605.13829, 2026.
- [8] LEON XIV.: Magnifica Humanitas - enciklika o varovanju človekove osebe v času umetne inteligence. Vatikan, 15. 5. 2026.
- [9] ANTHROPIC INSTITUTE: When AI builds itself. anthropic.com, 2026.
- [10] ARJANCODES: AI Writes Code Faster Than You. Now What? (video), 2026.
- [11] ANTHROPIC: Models overview. platform.claude.com/docs/en/about-claude/models/overview
- [12] BBC: Ford rehires human engineers after AI fails to match quality checks, <https://www.bbc.com/news/articles/cgrkd41n2v9o>
- [13] <https://deepswe.datacurve.ai>
- [14] <https://ponytail.dev/>
- [15] <https://caveman.so/>

Nov, na prstnem odtisu temelječ biometrični varnostni sistem

Komlen Lalović,¹ Dalibor Trajkovic,¹ Nebojsa Bacanin Dzakula²

¹ Razvojni in raziskovalni center DG Group, Maribor, Slovenija
komlen@dg-group.si, dalibor@dg-group.si

² Univerza Singidunum, 11000 Beograd, Srbija
nbacanin@singidunum.ac.rs

Članek predstavlja povsem nov sistem biometrične identifikacije in zagotavljanja identitete dojenčka, ki temelji na minucijah prstnih odtisov. Ti so edini biometrični podatki, ki se pri ljudeh oblikujejo prenatalno. Predstavljamo kvalitativno raziskavo, ki dokazuje, da ima dojenček prstni odtis, ki ga je mogoče pridobiti, shraniti, šifrirati in predvsem zagotoviti njegovo identiteto. Otrokove minucije smo preizkusili na vseh možnih vrstah čitalnikov prstnih odtisov in ugotovili, kateri je za ta namen optimalen. S tem smo si postavili trdno osnovo za nadaljnji razvoj naših raziskav. Vse to potrjuje, da gre za velik družbeni problem po vsem svetu, ki zahteva rešitev s 100-odstotno natančnostjo v bolnišnicah in porodnišnicah v vsaki državi, saj so se primeri zamenjave otrok zgodili na skoraj vseh celinah sveta. Prvi korak je bil izdelava prototipa naše patentirane naprave. Gre za dvojni skener prstnih odtisov, ki sočasno ali z nekajsekundnim presledkom skenira dva prsta, materinega in otrokovega.

Ključne besede:

biometrija

prstni odtisi

varne porodnišnice

osebni podatki

identiteta

1 Uvod

Biometrija je znanstvena disciplina, posvečena analizi, merjenju in statističnemu vrednotenju edinstvenih človeških fizičnih in vedenjskih značilnosti. Združuje računalništvo, matematiko in biologijo za avtomatizacijo osebne identifikacije in preverjanja [1]. V tem prispevku jo uporabljamo za zagotavljanje identitete dojenčkov v porodnišnicah, pri čemer se osredotočamo na njen najbolj očitni element, prstne odtise. V nadaljevanju navajamo nekatere primere, ki potrjujejo obstoj tega družbenega in širšega regionalnega problema, kot v znanem primeru⁵.

Vsi ti primeri vzbujajo nezaupanje v javne zdravstvene sisteme in vse več strahu pri nosečnicah na svetu, česar pa ne bi smelo biti. To je glavni razlog za vzpostavitev projekta Varna rojstna mesta v pametnih bolnišnicah, ki temelji na biometriji prstnih odtisov. Z izvedbo projekta pametnih rojstnih mest bomo odpravili takšne napake in zagotovili boljšo raven varnosti. Glavni cilj pa je zagotoviti identiteto vsakega novorojenčka. Dojenčki že imajo svojo identiteto; te ni treba nadomeščati z identifikacijskimi pripomočki, kot so ogrlice, s svinčnikom zapisane številke in podobno.

Slika 1 prikazuje primer skeniranja prstnih odtisov z uporabo tehnologije optičnega skenerja.



Slika 1: Primer skeniranja prstnih odtisov z uporabo tehnologije optičnega skenerja

2 Stanje tehnike

2.1. Opis nove naprave

Naprava se bo nekoliko razlikovala od današnjih klasičnih čitalnikov prstnih odtisov, saj bo imela dve polji za skeniranje prstov dveh različnih oseb (matere in dojenčka). Ti dve polji za skeniranje se lahko fizično ločita med postopkom izdelave naprave ali pa se programsko preslikata na površino za skeniranje, kot to počnejo obstoječi čitalniki. Naprava bo učinkovita, praktična ter preprosta za uporabo in upravljanje. Naprava je prikazana na sliki 2.

⁵ <https://www.bbc.com/news/articles/cp3njqd9nl9o>



Slika 2: Pregled patentirane naprave

2.2. Funkcionalnosti in vzdrževanje nove naprave

Vzdrževanje naprave je enostavno, klasično in podobno kot pri drugih čitalnikih prstnih odtisov. Poleg običajnega namena in hkratnega skeniranja dveh prstov različnih oseb bo zagotavljala edinstveno identifikacijsko referenco (podobno primarnemu ključu), ki bo osnova za vsak skenirani par materinega in otrokovega prsta. Čitalnik prstnih odtisov z dvema poljema (S1 in S2) deluje tako: najprej se izvede skeniranje (prstov matere in dojenčka), nato naprava ustvari edinstveno referenco, ki bo identifikacijska številka za vsak par mati–dojenček v bolnišnici. [2], [3], [4]

Ta pristop je bil v okviru raziskovalne metodologije uporabljen prvič, rezultati pa so predstavljeni izključno v tem članku. Gre za nadaljevanje dela, ki predstavlja popolnoma novo metodo ugotavljanja identitete, ki temelji na minucijah otrokovih prstnih odtisov namesto obstoječih sistemov, pri katerih se pojavljajo napake, in bo preprečila morebitno zamenjavo ali krajo identitete v porodnišnicah. Raziskava omogoča izbiro ustreznega čitalnika prstnih odtisov in kaže, da je mogoče z novo napravo pridobiti ter preveriti minucije otrokovih prstnih odtisov in nazadnje s skoraj 100-odstotno natančnostjo potrditi identifikacijsko povezavo med materjo in otrokom. S tem bi lahko izboljšali varnost porodnišnic in bolnišnic. [5], [6]

3 Algoritmi naprave

Naprava uporablja dva algoritma. Prvi je namenjen zajemu podatkov, drugi pa preverjanju shranjenih podatkov. Pseudokoda algoritma za zajem biometričnih podatkov prstnega odtisa je prikazana v nadaljevanju in na sliki 3 [7], [8].

Pseudokoda:

code line: 001 START

code line: 002 LOOP 1 TO 3

code line: 003 FIELD-1 F1 SCAN

code line: 004 IF F1 OK THEN GOTO **GENERATE UNIQUE ID**

ELSE IF LOOP < 3 GOTO **END**

code line: 005 LOOP 1 TO 3

code line: 006 FIELD-2 F2 SCAN

code line: 007 IF F2 OK THEN GOTO **GENERATE UNIQUE ID**

ELSE IF LOOP < 3 GOTO **END**

code line: 008 **GENERATE UNIQUE ID**

code line: 009 GENERATE PIN

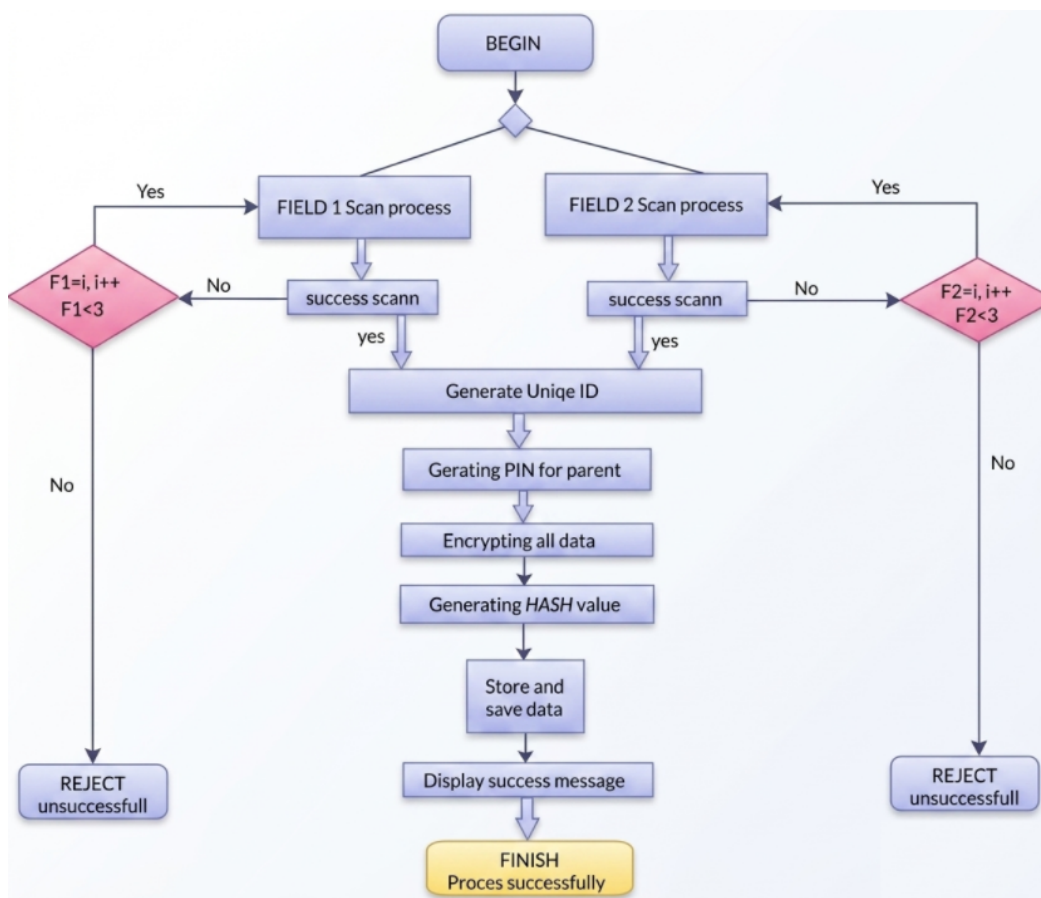
code line: 010 ENCRYPTING DATA

code line: 011 GENERATING HASH VALUE

code line: 012 STORE AND SAVE DATA

code line: 013 DISPLAY SUCCESS MESSAGE

code line: 014 **END.**



Slika 3: Algoritem za zajem podatkov

Algoritem najprej skenira prvi in drugi prst, materin in otrokov. Po treh neuspešnih poskusih se celoten postopek prekine. Če je skeniranje obeh prstov uspešno, se nadaljuje, ustvari enolični ID in kodo PIN za starša, šifrira podatke, izračuna zgoščeno vrednost ter shrani vse podatke in na zaslonu izpiše sporočilo o uspehu skeniranja. S tem se postopek zaključi. [9], [10], [11]

4 Razprava in smernice za nadaljnji razvoj

Poleg svojega glavnega namena imata lahko ta analiza in informacijski sistem, ki temelji na njej, različne možnosti uporabe. Najprej bi šlo za dvojno preverjanje identitete dojenčkov ob prehodu meje, pri katerem slika v potnem listu morda ne zadošča, saj se obraz in videz dojenčka hitro spreminjata. Ta naprava/informacijski sistem lahko zagotovi 100-odstotno zagotovilo, da identiteta dojenčka na meji ustreza podatkom v potnem listu, s 100-odstotno natančnostjo.

Poleg tega omogoča zelo široko uporabo podatkov za zaščito identitete in varnosti otrok, zlasti dojenčkov. Slika 2 prikazuje prototip patentirane naprave, ki jo je mogoče razviti in uporabiti v tovrstnem procesu, prav tako pa je povezan z informacijskim sistemom, izvedenim v objektno usmerjenih programskih jezikih, ki se danes uporabljajo.

5 Zaključek

Nova raziskava naj bi dosegla tri cilje: omogočiti izdelavo patentirane naprave, predstavljene v prejšnjem delu, razvoj informacijskega sistema in uvedbo pristopa varnega porodništva. Lahko pomembno izboljša raven javnega zdravja prihodnjih generacij v vsaki državi. Ta sistem je modularen in ga je mogoče posodabljan in, kar je najpomembnejše, lahko služi kot osnova za prihodnji razvoj biometričnih sistemov. Napravo je mogoče v številnih državah uporabiti za boj proti organiziranemu kriminalu in preprečevanje kraje ali zamenjave novorojenčkov, zlasti na območjih s slabo razvito infrastrukturo IT in tehnološkim razvojem. Ena od možnosti izvedbe so pametni sistemi interneta stvari v zdravstvu.

Vsak biometrični sistem želi zmanjšati tako FAR kot FRR, da bi bil veliko natančnejši in varnejši. Ta naprava je to dosegla, saj združuje podatka dveh skenov, njena natančnost pa eksponentno narašča. V sodobnem internetu stvari (IoT) si večina držav prizadeva zagotoviti povsem novo kakovost zdravstvenih storitev, pomagati osebju v porodnišnicah in olajšati potek poroda in ga narediti manj stresnega za bodoče matere, ginekologe, babice, medicinske sestre ter širšo javnost.

Literatura

- [1] Anil K. Jain-Michigan State University, USA, Patric Flynn-University of Notre Dame, USA, ARUN A. ROSS-West Virginia University, USA (2008): Handbook of Biometrics – Springer, USA
- [2] Komlen Lalović, Ivan Tot, Aleksandra Arsić, Milan Škarić - Security Information System, Based on Fingerprint Biometrics, Acta Polytechnica Hungarica, Volume 16, Issue Number 5, 2019 DOI: 10.12700/APH.16.5.2019.5.6
- [3] Komlen Lalović, Ivana Živić – JAVA GUI APLICATION FOR COMPARATION OF BIOMETRIC SECURITY LEVEL FINGERPRINT vs. IRIS, YU Info 2021 Information technologies Conference, 11th March – Kopaonik Serbia
- [4] Tot Ivan, Mladen Bajčetić, Mladen Trikoš, Komlen Lalović, Bogićević Dušan: Software platform for Learning about Brain Wave (EEG) Acquisition and Analysis - Acta Polytechnica Hungarica - with Imp. F. 1.22, Volume 18, Issue Number 3, 2021 - DOI: 10.12700/APH.18.3.2021.3.8
- [5] A Holobar, D Zazula; Multichannel blind source separation using convolution kernel compensation, IEEE Transactions on Signal Processing 55 (9), 4487-4496
- [6] <https://brandongaille.com/20-babies-switched-at-birth-statistics> - one study based on babyswithch wordwide done by Mr Brandon Gaille, USA bilionaire
- [7] Komlen Lalovic - A Novel Biometric Fingerprint IT System in Determining Newborn Baby Identity, Međunarodna ICT EBT Conference - FON, Beograd – 27. sept 2025.
- [8] Komlen Lalovic - A novel research based on Baby Fingerprint biometrics in service public health system at birthplaces, Međunarodna ICT Konferencija YU Info, Kopaonik - mart 2026.
- [9] Hannah Grace Dahlen, Shea Caplice: “What do midwives fear?”, Published Online: July 24, 2014 – Elsevier, Women and Birth, Journal of Australian College of Midwives
- [10] A Holobar, MA Minetto, D Farina; Accurate identification of motor unit discharge patterns from high density surface EMG and validation with a novel signal-based performance metric, Journal of neural engineering 11 (1), 016008
- [11] Samson Omasirichukwu Offorjindu, Komlen Lalovic, Srdjan Milic, Csaba Varsandan, Miodrag Zivkovic, Zorana Krsmanovic, and Nebojsa Bacanin, Milos Antonijevic - Art Museum Feedback Sentiment Analysis: an Approach Combining Word2Vec, XGBoost and Modified Firefly Algorithm; International Conference on Intelligent Computing and Communication Systems - ICICCS 2026 July 14-16, Burmingham, UK

Samodejna priprava prometnih poročil za obveščanje javnosti prek radijskih postaj

Ruben Ferreira,¹ Rok Kužel,² Timotej Knez,¹ Slavko Žitnik¹

¹ Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Ljubljana, Slovenija
ruben@ferreira.si, slavko.zitnik@fri.uni-lj.si, timotej.knez@fri.uni-lj.si

² Radiotelevizija Slovenija, Ljubljana, Slovenija
rok.kuzel@rtvslo.si

V prispevku predstavljamo razvoj sistema za samodejno generiranje prometnih poročil za radijsko poročanje z uporabo velikih jezikovnih modelov. Sistem temelji na podatkih nacionalnega prometnega portala NAP, iz katerega redno pridobiva podatke o prometnih dogodkih in delih na cesti. Pridobljeni podatki se shranjujejo, normalizirajo in pripravijo za nadaljnjo klasifikacijo ter generiranje besedil. Poseben poudarek namenjamo primerjavi različnih pristopov k uporabi jezikovnih modelov, in sicer tako modelov v oblaku kot lokalno naučenih oziroma prilagojenih modelov. Primerjavo izvajamo za naloge klasifikacije prometnih dogodkov in generiranja prometnih poročil, pri čemer upoštevamo kakovost izhodov, hitrost odziva, stroške uporabe, robustnost in primernost za slovenski jezik. Za generiranje rednih prometnih poročil uporabljamo dvostopenjski pristop, pri katerem se prometni zapisi najprej pretvorijo v strukturirane semantične fragmente, nato pa v končno slovensko besedilo, primerno za radijsko branje. Prispevek obravnava podatkovni cevovod, obdelavo vhodnih podatkov, primerjavo modelov in vlogo človeka pri končni uredniški presoji.

Ključne besede:

veliki jezikovni modeli

generiranje prometnih poročil

prometni podatki

radijsko poročanje

umetna inteligenca

1 Uvod

Prometne informacije so pomemben del vsakodnevnega radijskega poročanja, saj poslušalcem omogočajo hitro prilagoditev poti, izogibanje zastojem, pravočasno zaznavanje zapor ter upoštevanje drugih izrednih razmer na cestnem omrežju. Pri radijskem poročanju je poleg vsebinske pravilnosti ključna tudi časovna ažurnost informacij. Prometna nesreča, zapora voznega pasu ali daljša čakalna doba na mejnem prehodu ima lahko za poslušalce neposredne posledice, zato mora biti informacija pravočasno zaznana, pravilno interpretirana in predstavljena v kratki, razumljivi ter govorno primerni obliki.

V obstoječih uredniških procesih priprava prometnih poročil pogosto temelji na ročnem pregledu več prometnih virov. Urednik oziroma radijski voditelj mora iz razpoložljivih podatkov prepoznati najpomembnejše dogodke, izločiti manj relevantne ali zastarele informacije, združiti vsebinsko sorodne zapise in jih preoblikovati v tekoče besedilo, primerno za radijsko branje. Takšen proces je časovno zahteven, hkrati pa je občutljiv na nepopolnost vhodnih podatkov, podvajanje informacij, različne oblike zapisov in subjektivno presojo pomembnosti posameznega dogodka. Dodaten izziv predstavljajo nujna prometna obvestila, ki morajo biti zaznana in posredovana hitreje kot redna prometna poročila.

Razvoj velikih jezikovnih modelov odpira možnost avtomatizacije priprave prometnih poročil. Jezikovni modeli lahko iz strukturiranih prometnih zapisov tvorijo naravno besedilo, združujejo sorodne dogodke in prilagajajo slog besedila izbranemu komunikacijskemu kanalu. Vendar pa uporaba takšnih modelov v produkcijskem uredniškem okolju zahteva več kot zgolj neposredno pošiljanje surovih podatkov v model. Prometni podatki so časovno občutljivi, pogosto vsebujejo ponovitve, neenotne zapise, zastarele dogodke ali informacije z različno stopnjo pomembnosti. Poleg tega mora sistem preprečevati izmišljanje dejstev, ohranjati sledljivost do izvornih zapisov in omogočati človeku končno uredniško presojo.

V prispevku predstavljamo razvoj sistema za samodejno generiranje prometnih poročil za potrebe radijskega poročanja na RTV Slovenija. Sistem temelji na podatkih nacionalnega prometnega portala NAP, ki zagotavlja sprotne informacije o prometnih dogodkih ter delih na cesti. Razvita rešitev podatke redno pridobiva, jih shranjuje v podatkovno bazo, normalizira in pripravlja kot vhod za klasifikacijo ter generiranje poročil. Pri tem ločujemo med rednim generiranjem prometnih poročil, ki se lahko izvede na zahtevo oziroma v določenih časovnih intervalih, in nujnimi prometnimi obvestili, ki morajo biti po zaznavi čim prej prikazana uporabniku.

Osrednji del rešitve predstavlja kombinacija klasične obdelave podatkov, lokalnega klasifikacijskega modela in velikih jezikovnih modelov. Lokalni klasifikacijski model se uporablja za prepoznavanje nujnih prometnih dogodkov, kar omogoča hitro odločanje brez odvisnosti od zunanjih storitev. Za tvorjenje končnih besedil pa uporabljamo dvostopenjski pristop z velikim jezikovnim modelom. V prvi stopnji se prometni zapisi pretvorijo v strukturirane semantične fragmente, v drugi stopnji pa se ti fragmenti uredijo, združijo in preoblikujejo v slovensko besedilo, primerno za radijsko branje. Takšen pristop zmanjšuje možnost izmišljanja in podvajanja informacij ter omogoča boljši nadzor nad tem, katere informacije so bile uporabljene pri generiranju poročila.

Prispevek se osredotoča predvsem na podatkovni del rešitve in pripravo vhodov za jezikovne modele. Najprej opišemo vrste podatkov, ki so bile uporabljene v projektu: zgodovinske podatke, pridobljene za analizo, testiranje in učenje modelov, ter sprotne podatke iz dostopne točke NAP. Nato predstavimo postopke obdelave, povezovanja zgodovinskih prometnih zapisov z arhiviranimi radijskimi poročili, shranjevanja surovih podatkov, normalizacije zapisov in priprave podatkov za klasifikacijo ter generiranje poročil. S tem pokažemo, da je kakovost izhodnega besedila v veliki meri odvisna od ustrezno zasnovanega podatkovnega cevovoda in ne le od izbire samega jezikovnega modela.

2 Pregled področja

Avtomatizacija priprave besedil v uredniških in informacijskih okoljih ni nov pojav. Že pred širšo uporabo velikih jezikovnih modelov so se v praksi uveljavili sistemi za podatkovno vodeno generiranje besedil, pri katerih je bil vhod praviloma strukturiran podatkovni vir, izhod pa kratek novinarski ali poročevalski prispevek. Takšni sistemi so bili posebej primerni za področja, kjer se vsebina pogosto ponavlja, podatki pa so jasno strukturirani, na primer pri finančnih rezultatih, športnih izidih, volitvah, vremenskih opozorilih ali lokalnih obvestilih. V tem obdobju so rešitve večinoma temeljile na pravilih, predlogah in klasičnih pristopih naravnega jezikovnega tvorjenja, vendar so že uvedle osnovni produkcijski vzorec: zajem podatkov, izbor relevantnih informacij, pretvorbo v besedilo in uredniški nadzor.

Med najbolj znanimi primeri zgodnje produkcijske uporabe je sodelovanje agencije Associated Press s podjetjem Automated Insights in družbo Zacks Investment Research [2]. Sistem je iz strukturiranih finančnih podatkov samodejno pripravljala kratka poročila o poslovnih rezultatih podjetij, s čimer je Associated Press povečal obseg pokrivanja četrtnih rezultatov na več kot 3000 prispevkov na četrtnetje. Podoben primer predstavlja sistem Heliograf [3], ki ga je razvil The Washington Post in ga uporabil za samodejno pripravo kratkih posodobitev pri poročanju o olimpijskih igrah v Rio leta 2016 ter pozneje tudi pri drugih podatkovno intenzivnih temah. Oba primera kažeta, da je avtomatizacija najuspešnejša tam, kjer je mogoče novinarsko nalogo razdeliti na jasno določene korake in kjer je glavni izziv hitra pretvorba strukturiranih podatkov v razumljivo besedilo.

Naslednji razvojni korak predstavljajo sistemi, ki poleg tvorjenja besedila vključujejo tudi zaznavanje, filtriranje oziroma razvrščanje dogodkov. Primer takšnega pristopa je Quakebot [4] časopisa Los Angeles Times, ki spremlja obvestila ameriške geološke službe USGS in ob zaznavi potresa, ki ustreza določenim kriterijem, pripravi osnutek članka. Pri tem objava ni popolnoma avtomatizirana, saj končno odločitev sprejme urednik. Podoben premik od samega pisanja k širši podpori uredniškemu procesu je viden pri Reutersovih sistemih. Reuters Tracer [5] je bil zasnovan za zaznavanje in klasifikacijo potencialno relevantnih dogodkov iz družbenih omrežij v realnem času, Reuters pa je pozneje razvil tudi sistem Lynx Insight [6], ki iz podatkovnih virov pripravlja kratke podatkovno utemeljene vsebine, opozorila in sročilce za novinarje. Ti primeri so pomembni, ker združujejo samodejno obdelavo podatkov, prag pomembnosti in človeka kot končnega odločevalca, kar je neposredno primerljivo s sistemi za časovno občutljive dogodke, kot so prometne nesreče, zapore ali daljši zastoji.

Z razvojem velikih jezikovnih modelov se je težišče avtomatizacije deloma premaknilo od predlog in pravil k bolj prilagodljivemu tvorjenju jezika. Takšni modeli lahko bolje prilagajajo slog, združujejo več vhodnih informacij, krajšajo besedilo in ga oblikujejo glede na ciljni komunikacijski kanal. Hkrati pa uvajajo nova tveganja, predvsem možnost izmišljanja dejstev, izgubo sledljivosti do vhodnih podatkov in težje preverjanje, katere informacije so bile uporabljene pri tvorjenju končnega besedila. Zato se v produkcijskih okoljih pogosto uporabljajo nadzorovani pristopi, pri katerih so vhodni podatki predhodno očiščeni, strukturirani in omejeni, končni rezultat pa ostane predmet človeškega pregleda.

Najbližje področju obravnavanega sistema so rešitve za samodejno pripravo prometnih obvestil za radijsko oziroma medijsko uporabo. Bauer Media in finsko državno podjetje Fintraffic [7] sta predstavila sistem za lokalno usmerjena prometna obvestila na radiu, pri katerem se uporabljajo umetna inteligenca, prometni podatki v realnem času in samodejno generirani glasovi. Namen takšnega pristopa je poslušalcem posredovati bolj lokalno relevantna prometna obvestila, ki so vezana na dejanske prometne razmere in lokacijo. Ta primer je pomemben, ker avtomatizacijo postavlja neposredno v radijski kontekst, kjer mora biti besedilo kratko, govorno naravno, hitro razumljivo in primerno za takojšnjo uporabo v programu.

Še neposrednejši komercialni primer predstavlja rešitev INRIX AI Traffic Reporter [8], ki je namenjena avtomatizaciji prometnega poročanja za televizijo in radio. INRIX jo opisuje kot sistem, ki iz prometnih podatkov in analitike samodejno pripravlja prometna poročila ter jih pretvarja v avdio oziroma video izhode za medijsko uporabo. Pri tem ne gre zgolj za konceptualno rešitev, saj so bila poročila, pripravljena z uporabo umetne inteligence, v Združenem kraljestvu že uvedena v radijskem okolju. Po poročanju portala RadioToday [13] je Bauer Media v sodelovanju z družbo INRIX takšna poročila uvedla na omrežjih Greatest Hits Radio in Hits Radio, kjer

se večina prometnih informacij podaja z računalniško generiranim glasom. Takšne rešitve kažejo, da je avtomatizacija prometnega poročanja že prešla iz raziskovalnega okolja v produkcijsko uporabo. Hkrati pa ostajajo odprta vprašanja glede kakovosti vhodnih podatkov, lokalizacije, jezikovne prilagoditve, uredniškega nadzora in primernosti za posamezne nacionalne prometne vire.

Sistem, predstavljen v tem prispevku, sledi istemu produkcijskemu vzorcu, vendar ga prilagaja slovenskemu prostoru in specifičnim zahtevam radijskega prometnega poročanja. Vhodni podatki izvirajo iz nacionalnega prometnega portala NAP [1], sistem pa jih pred uporabo shrani, normalizira in pripravi za nadaljnjo obdelavo. Posebnost rešitve je kombinacija lokalnega klasifikacijskega modela za zaznavanje pomembnih oziroma nujnih prometnih dogodkov ter dvostopenjskega generiranja z velikim jezikovnim modelom. Pri tem se prometni zapisi najprej pretvorijo v strukturirane semantične fragmente, nato pa v končno slovensko besedilo, primerno za radijsko branje. V primerjavi z obstoječimi primeri je poudarek predvsem na slovenskem jeziku, uporabi nacionalnega prometnega vira, sledljivosti do izvornih prometnih zapisov in ohranjanju človeka kot končnega urednika oziroma uporabnika sistema.

3 Podatki in njihova priprava

Kakovost samodejno generiranega prometnega poročila je neposredno odvisna od kakovosti, ažurnosti in strukture vhodnih podatkov. Pri razvoju sistema smo zato posebno pozornost namenili podatkovnemu cevovodu, ki obsega pridobivanje prometnih podatkov, njihovo shranjevanje, obdelavo, normalizacijo, zaznavanje nujnih dogodkov in pripravo strukturiranih vhodov za jezikovne modele. Podatke, uporabljene v projektu, lahko razdelimo v dve glavni skupini: zgodovinske podatke in sprotne podatke iz nacionalnega prometnega portala NAP [1].

3.1 Zgodovinski podatki

Zgodovinski podatki so bili uporabljeni predvsem v razvojni fazi sistema, in sicer za razumevanje strukture prometnih informacij, analizo uredniškega procesa, testiranje različnih pristopov generiranja poročil ter pripravo učnih in evalvacijskih primerov. Pri tem smo imeli na voljo dva vira podatkov: prometne zapise v obliki Excel datotek ter arhivirana radijska prometna poročila v obliki RTF datotek.

Prometni zapisi iz Excel datotek so vsebovali besedilna polja v obliki HTML, razdeljena na vsebinske sklope, kot so opozorila, vreme, prireditve, delo na cesti in omejitve za tovorna vozila, kot je prikazano v tabeli 2. Ti zapisi so predstavljali arhivsko obliko prometnih informacij, ki so po vsebini primerljive s podatki, pridobljenimi prek programskega vmesnika NAP. Arhivirana radijska poročila pa so predstavljala ciljno obliko izhodnega besedila, saj kažejo, kako uredniki iz širšega nabora prometnih informacij izberejo najpomembnejše dogodke in jih preoblikujejo v kratko, govorno primerno poročilo.

Ker prometni zapisi in radijska poročila niso bili neposredno povezani z enoličnimi identifikatorji, smo pripravili postopek približnega povezovanja. Za vsako radijsko poročilo smo iz glave dokumenta izluščili naslov, datum, čas in program, nato pa poiskali prometne zapise, ki so nastali v časovnem oknu pred objavo poročila. Med kandidati smo nato izvedli besedilno primerjavo. Besedila smo predhodno očistili, pretvorili v male črke, odstranili ločila, kontrolne znake in slovenske zaustavitvene besede (angl. *stop words*), nato pa za razvrščanje uporabili predstavitev TF-IDF [14] in kosinusno podobnost [15].

S takšnim postopkom smo pridobili približne pare med vhodnimi prometnimi zapisi in končnimi radijskimi poročili. Ti pari so bili pomembni iz več razlogov. Prvič, omogočili so testiranje in prilagajanje različnih velikih jezikovnih modelov, saj smo lahko primerjali generirana poročila z dejanskimi uredniškimi poročili. Drugič, ker so bila pri zgodovinskih podatkih na voljo tudi ciljna besedila, je bilo mogoče del naloge obravnavati kot nadzorovano učenje oziroma kot evalvacijo modelov na podlagi znanih izhodov. Tretjič, podatki so omogočili pripravo primerov za učenje in preverjanje klasifikacijskih modelov, namenjenih ločevanju pomembnih oziroma nujnih prometnih dogodkov od manj pomembnih. Poleg tega so zgodovinski podatki služili za analizo vzorcev v prometnih informacijah in boljše razumevanje uredniških odločitev pri pripravi radijskih poročil.

Analiza zgodovinskih podatkov je pokazala, da se izvorni prometni zapisi in končna radijska poročila pomembno razlikujejo. Izvorni zapisi so pogosto daljši, bolj tehnični in vsebujejo več podrobnosti, medtem ko radijsko poročilo zahteva izbor najpomembnejših informacij, krajšanje, združevanje sorodnih dogodkov in prilagoditev jezika govornemu mediju. To je vplivalo na zasnovo sistema, predvsem na odločitev za strukturirano pripravo vhodnih podatkov pred končnim generiranjem besedila.

Tabela 1: Pregled zgodovinskih prometnih zapisov.

Obdobje	Št. zapisov	Povp. zapisov/dan
2022	55.001	150,7
2023	57.094	156,4
2024	58.278	159,2
Skupaj	170.373	155,4

V tabeli 1 »zapis« označuje eno vrstico v arhivski datoteki Excel oziroma časovno označen posnetek prometnih informacij, kot je prikazano v tabeli 2. Posamezen zapis lahko vsebuje več vsebinskih sklopov in več prometnih dogodkov, zato število zapisov ne predstavlja števila enoličnih prometnih dogodkov.

Tabela 2: Struktura enega zgodovinskega zapisa.

Polje v datoteki Excel	Pomen
LegacyId	Enolični identifikator zapisa v arhivskem izvozu
Datum	Čas nastanka oziroma posodobitve prometnega zapisa
Operater	Podatek o osebi oziroma viru, ki je pripravil ali uredil zapis.
Pomembno	Sklop za posebej izpostavljene oziroma nujne prometne informacije.
Nesreče	Informacije o prometnih nesrečah in njihovem vplivu na promet.
Vreme	Prometne informacije, povezane z vremenskimi razmerami.
Ovire	Informacije o ovirah na vozišču ali drugih motnjah v prometu.
DeloNaCesti	Informacije o delih na cesti, zaporah in spremenjeni prometni ureditvi.
Opozorila	Splošna opozorila, prometne napovedi, omejitve in druga obvestila.
MednarodneInformacije	Informacije, povezane s tujino, mejnimi prehodi ali mednarodnimi omejitvami.
Zastoji	Informacije o zastojih, kolonah in daljših potovalnih časih.
Splošno	Splošne prometne informacije, ki niso nujno vezane na en sam konkreten dogodek.

3.2. Sprotni podatki iz dostopnih točk NAP

V produkcijskem delu sistema se podatki pridobivajo iz sistema NAP. Sistem tehnično podpira tri vire: prometne dogodke, dela na cesti in prometna poročila. Za sprotno generiranje radijskih obvestil sta najpomembnejša vira prometni dogodki in dela na cesti, saj vsebujeta posamezne aktualne zapise o razmerah na cestah. Vir prometnih poročil je v sistemu prav tako predviden, vendar se uporablja predvsem za shranjevanje in primerjalno analizo, ne pa kot glavni vhod za generiranje posameznih nujnih obvestil.

Zapisi, pridobljeni iz dostopnih točk NAP, so strukturirani kot posamezni prometni dogodki oziroma zapisi o delih na cesti. Posamezen zapis praviloma vsebuje enolični identifikator, čas posodobitve, podatek o cesti oziroma

cestnem odseku, kategorijo ceste, vzrok dogodka, opis razmer in morebitno dodatno pojasnilo. To je pomembno, ker sistem teh podatkov ne obravnava zgolj kot prosta besedila, temveč kot strukturirane dogodke, ki jih je mogoče shraniti, primerjati, razvrščati in uporabiti kot vhod v nadaljnje modele.

Pri prehodu iz zgodovinskih podatkov na sprotne podatke NAP je bilo pomembno, da se sistem ne opira na podatke, ki so bili prisotni samo v arhivskih izvozih iz Excela. Modeli, razviti ali preverjeni na zgodovinskih podatkih, morajo namreč tudi v produkcijskem delovanju prejemati primerljivo strukturo vhodnih podatkov. Zato smo pri pripravi podatkov uporabili predvsem tiste vsebinske sklope iz zgodovinskih zapisov v Excelu, ki jih je mogoče neposredno ali vsaj delno povezati z vrednostmi polja »vzrok« v sprotnih podatkih NAP.

Primerjava je pokazala, da so neposredno primerljivi predvsem sklopi delo na cesti, zastoji, ovire, nesreče, vreme in omejitve za tovorna vozila. Ti sklopi so bili zato uporabljeni kot glavna osnova za pripravo primerljivih vhodnih podatkov. Sklop »Pomembno« je bil uporaben le delno, saj se v njem v zgodovinskih podatkih pojavljajo varnostno kritični dogodki, kot je vožnja v nasprotno smer, medtem ko se v podatkih NAP tak dogodek pojavi kot samostojna vrednost polja »vzrok«. Sklopi brez neposredne ali delne ustreznice, na primer splošne informacije, mednarodne novice in opozorila, niso bili uporabljeni kot glavna osnova za primerjavo oziroma učenje modelov.

Tabela 3: Primerljivost zgodovinskih podatkov s podatki, pridobljenimi iz dostopne točke.

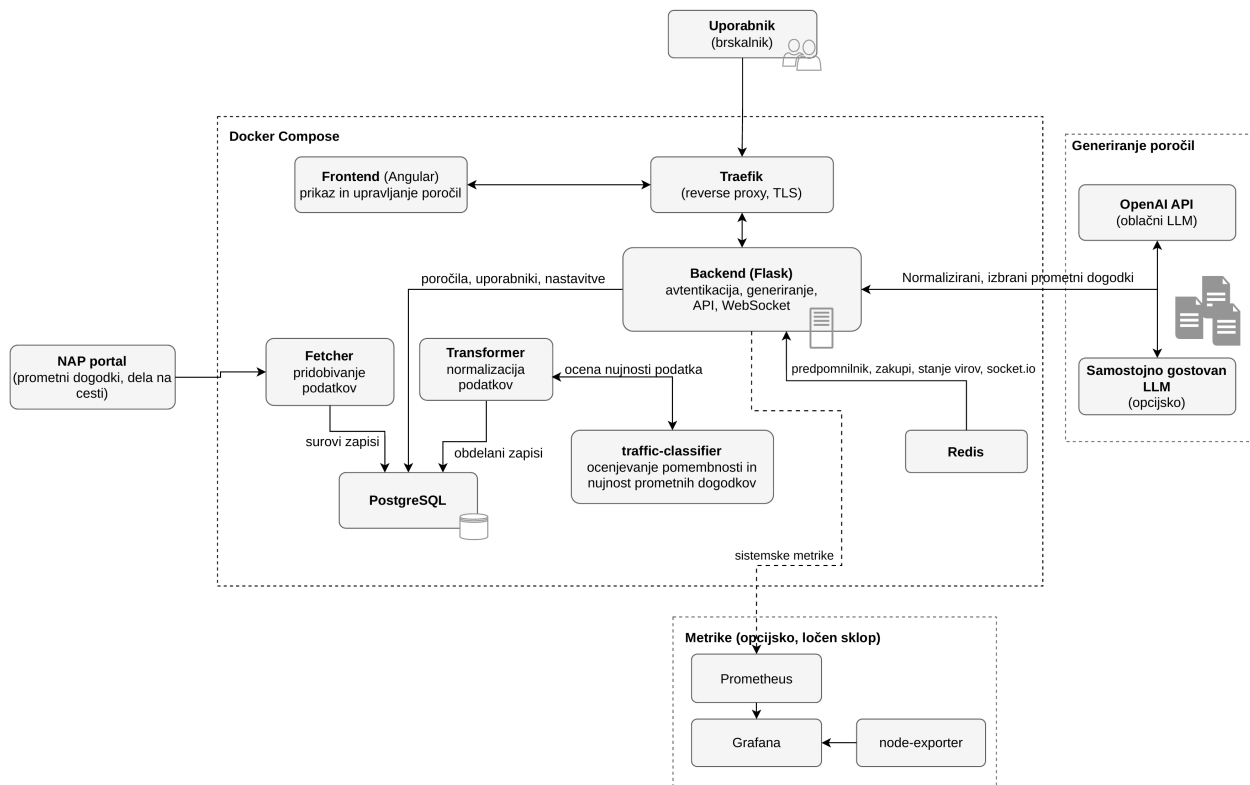
Vsebinski sklop v zgodovinskih podatkih iz Excela	Vrednost »vzrok« v podatkih NAP Primerljivost	Primerljivost
Delo na cesti	Delo	Primerljivo
Zastoji	Zastoj	Primerljivo
Ovire	Ovire	Primerljivo
Nesreče	Nesreča	Primerljivo
Vreme	Vreme	Primerljivo
Omejitve za tovorna vozila	Tovornjaki	Primerljivo
Pomembno	Vožnja v nasprotno smer	Delno primerljivo
Splošno	/	Ni primerljivo
Mednarodne novice	/	Ni primerljivo
Opozorila	/	Ni primerljivo

Pri uporabi sprotnih podatkov je bil pomemben izziv ločevanje nujnih oziroma pomembnih prometnih dogodkov od manj pomembnih. Dostopne točke NAP zagotavljajo aktualne prometne informacije, vendar dogodki, ki zahtevajo takojšnje radijsko obvestilo, niso posebej označeni kot nujni. Prometna nesreča, daljši zastoj, ovira na pomembni cesti ali vožnja v nasprotno smer se zato v viru pojavi kot običajen prometni dogodek. Zaradi tega smo v sistem vključili klasifikacijski model, ki na podlagi vsebine dogodka presodi, ali gre za pomemben dogodek, ki ga je treba nemudoma posredovati uporabniškemu vmesniku.

4 Zasnova in implementacija sistema

4.1. Arhitektura sistema

Sistem je zasnovan kot skupina med seboj ločenih storitev, ki tečejo v samostojnih vsebnikih Docker [16] in med seboj komunicirajo prek notranjega omrežja, kot je prikazano na sliki 1. Takšna zasnova omogoča neodvisno namestitvev, ponovni zagon in skaliranje posameznih delov sistema, hkrati pa loči odgovornosti med pridobivanjem podatkov, njihovo obdelavo, klasifikacijo, generiranjem poročil in predstavitvijo uporabniku. V nadaljevanju so opisane posamezne storitve, kot so definirane v namestitveni konfiguraciji sistema.



Slika 1: Arhitektura sistema za generiranje prometnih poročil.

4.1.1. Reverse proxy (Traefik)

Traefik [17] predstavlja vstopno točko sistema. Poskrbi za zaključevanje povezav TLS (samodejno pridobivanje in obnavljanje potrdil prek storitve Let's Encrypt [18]) ter usmerjanje prihajajočih zahtev glede na domeno in pot: zahteve na “/api” in “/socket.io” preusmeri na zaledno aplikacijo, vse ostale pa na odjemalsko aplikacijo. S tem je edina komponenta, neposredno izpostavljena na javnem omrežju. Traefik na ločeni, izključno interni vstopni točki izpostavlja tudi svoje metrike prometa z uporabo sistema za spremljanje Prometheus [19], dostop do vmesnika Grafana [20] pa usmerja na podpoti “/monitoring” znotraj obstoječe domene, tako da spremljanje delovanja ne potrebuje ločene poddomene ali dodatnega potrdila TLS.

4.1.2. Zaledna aplikacija (angl. backend)

Zaledna aplikacija, razvita z ogrodjem Flask [21], izvaja poslovno logiko sistema in uporabnikom izpostavlja HTTP API ter komunikacijo v realnem času prek protokola WebSocket. Upravlja uporabnike, prometne zapise, generirana poročila, nastavitve sistema in povratne informacije urednikov.

Storitvena plast povezuje klasifikacijo dogodkov, dvostopenjsko generiranje poročil, časovno razporejanje in obveščanje uporabnikov. Redis [22] se uporablja za predpomnjenje rezultatov, koordinacijo razporejenih opravil

in razpošiljanje dogodkov med instancami, vsi klici jezikovnih modelov pa se beležijo zaradi sledljivosti in nadzora nad stroški. Podrobnosti izbire zapisov, predpomnjenja in generiranja so opisane v razdelku 4.4.

4.1.3. *Fetcher in Transformer*

Procesa Fetcher in Transformer sta zagnana iz iste izvorne slike kot zaledna aplikacija, vendar z drugačno vstopno točko, kar omogoča neodvisno skaliranje in ponovni zagon pridobivanja podatkov, ločeno od njihove obdelave.

Proces Fetcher redno dostopa do dostopnih točk portala NAP (prometni dogodki, dela na cesti, prometna poročila), preverja spremembe, da se izogne nepotrebnemu prenosu nespremenjenih podatkov, ter surove zapise shrani v podatkovno bazo. Poleg tega spremlja razpoložljivost posameznega vira in stanje objavlja prek Redisa in protokola WebSocket.

Proces Transformer surove zapise pretvarja in normalizira v obdelane tabele, kliče klasifikacijski model za oceno nujnosti posameznega dogodka ter zaznava dogodke, ki zahtevajo takojšnjo obravnavo in sprožitev nujnega poročila.

4.1.4. *Klasifikator prometnih dogodkov*

Klasifikator je samostojna storitev, razvita z ogrodjem FastAPI [23]. Uporablja naučeni model ModerniBERT, za slovenski jezik prilagojeno različico modela ModernBERT [24], proces Transformer pa jo kliče prek internega zahtevka REST, zaščitenega s ključem API. Storitev je namensko ločena od zaledne aplikacije, saj uporablja drugačen tehnološki sklad (Python z modelom PyTorch [25]) in ima drugačne strojne zahteve. Omogoča hitro lokalno klasifikacijo nujnosti dogodkov brez odvisnosti od zunanjega ponudnika; kot alternativa je na voljo tudi klasifikacija prek oblačnega jezikovnega modela (opisano v razdelku 4.3).

4.1.5. *Odjemalska aplikacija (angl. frontend)*

Odjemalska aplikacija je implementirana kot enostranska spletna aplikacija v ogrodju Angular [27]. Urednikom omogoča pregled prometnih zapisov, generiranje in urejanje poročil, pregled izvornih zapisov ter podajanje povratnih informacij. Nujna poročila, spremembe stanja virov in potek generiranja se posredujejo prek povezave WebSocket, zato ročno osveževanje strani ni potrebno. Administratorski del omogoča upravljanje dostopov ter konfiguracijo modelov in parametrov poročanja. Osnovni uporabniški vmesnik aplikacije je prikazan na sliki 2.

The screenshot displays the AGPI (Avtomatsko Generiranje Prometnih Informacij) web application. The interface is divided into several sections:

- Left Panel (Nujna poročila):** Lists emergency reports with details like incident ID (e.g., #22215, #22204, #22202, #14222), date, and a brief description of the incident (e.g., vehicle breakdown, accident, traffic jam). Each entry has buttons for 'UREDI' (Edit), 'SKRIJ' (Hide), and 'NI URGENTNO' (Not Urgent).
- Center Panel (Generator prometnih poročil):** Features a 'Prometno poročilo' (Traffic Report) section with a 'USTVARI POROČILO' (Create Report) and 'ZAČNI GENERIRANJE' (Start Generation) button. It shows a selected model ('Samodejna izbira') and a date range for report generation. Below, it displays generated traffic information for a specific date and time, including details about road closures and traffic jams.
- Right Panel (Dogodki):** Shows a list of traffic incidents ('Skupno dogodkov: 5') with details like location (e.g., G2-106, Škofljica - Račica), date, time, and cause (e.g., accident, traffic jam). It also includes a table of incidents with columns for 'Redna' (Regular) and 'Nujna' (Emergency) status, and a 'OSVEŽI' (Refresh) button.

The footer of the application includes logos for FRI (Fakulteta za računalništvo in informatiko), RADIO TELEVIZIJA SLOVENIJA, and LLMS4EU.

Slika 2: Uporabniški vmesnik.

Uporabniški vmesnik je razdeljen na tri glavne vsebinske dele. Na levi strani je prikazan seznam nujnih prometnih poročil, na katerem lahko urednik posamezno poročilo uredi, skrije ali označi kot napačno nujno poročilo. Osrednji del je namenjen ustvarjanju, prikazu in urejanju generiranega prometnega poročila ter spremljanju njegovega stanja. Na desni strani so prikazani izvorni prometni dogodki in dela na cesti, ki jih je mogoče uporabiti pri generiranju, pod njimi pa je seznam predhodno ustvarjenih rednih in nujnih poročil. Dostop do nastavitev aplikacije je omogočen v zgornjem desnem delu vmesnika.

4.1.6. Podatkovna in operativna infrastruktura

PostgreSQL [28] predstavlja trajno podatkovno shrambo za prometne zapise, poročila, uporabnike, nastavitve in metapodatke klicev jezikovnih modelov. Redis [22] se uporablja za predpomnjenje, koordinacijo opravil, stanje delovnih procesov in komunikacijo med instancami. Operativno spremljanje zagotavljata Prometheus [19] in Grafana [20], ki prikazujeta aplikacijske in sistemske metrike. Zunanji storitvi sistema sta portal NAP kot podatkovni vir in OpenAI API [26] kot oblaki ponudnik jezikovnih modelov.

4.2. Pridobivanje in shranjevanje podatkov

Proces Fetcher redno dostopa do treh dostopnih točk portala NAP – za prometne dogodke, dela na cesti in prometna poročila – do vsake v svojem, neodvisno nastavljenem časovnem intervalu. Dostop je avtenticiran prek OAuth2: fetcher najprej pridobi dostopni žeton z uporabniškim imenom in geslom, nato pa ga do izteka veljavnosti obnavlja z osvežitvenim žetonom, in sicer proaktivno tik pred iztekom ter reaktivno ob morebitnem odgovoru 401.

Preden fetcher prenese celoten odgovor, s pogojno zahtevo preveri, ali se je nabor podatkov od zadnjega poizvedovanja sploh spremenil. Če vir vrne 304 (ni sprememb), se noben zapis ne prepíše – namesto tega se za vrstice, ki so bile del zadnjega dejansko prenesenega posnetka, le pomakne časovna oznaka zadnje potrditve prisotnosti, s čimer sistem loči med »dogodek se od prejšnjega preverjanja ni spremenil« in »dogodek v tem posnetku ni bil več prisoten«, ne da bi za to moral znova zapisovati nespremenjeno vsebino.

Prejeti zapisi se shranjujejo v ločene surove tabele v PostgreSQL, ločeno za vsak vir, z enoličnim identifikatorjem dogodka, ki ga dodeli NAP. Znotraj enega prenesenega posnetka se zapisi z istim identifikatorjem pred zapisom v bazo združijo, sam zapis v bazo pa se izvede po tem identifikatorju: obstoječa vrstica se prepíše le, če je časovni žig spremembe iz izvirnega sistema resnično novejši od že shranjenega oziroma če ta prej ni bil znan – s čimer poznejša, a vsebinsko zastarela dostava podatkov ne more prepisati novejšega stanja. Poleg žiga zadnje spremembe vsaka vrstica hrani tudi prvi trenutek, ko je bil zapis zaznan in zadnji trenutek, ko je bil še potrjen kot prisoten; ta dva časovna žiga sta osnova tako za filter zastarelosti vira kot za sprotni pogoj pri generiranju poročil (glej razdelek 4.4). Ob dejanski vsebinski spremembi se zapis dodatno označi kot neobdelan, kar zaledni storitvi »Transformer« pove, da ga mora znova pretvoriti in klasificirati. Vzporedno s shranjevanjem storitev »Fetcher« za vsak vir podatkov vodi tudi stanje razpoložljivosti glede na zaporedne uspehe in neuspehe poizvedb, ki ga objavlja prek Redisa in WebSocketa, ter piše lastni srčni utrip, s čimer je mogoče delovanje procesa pridobivanja podatkov spremljati ločeno od delovanja posameznega vira NAP.

4.3. Klasifikacija pomembnosti prometnih dogodkov

Cilj klasifikacije pomembnosti prometnih dogodkov je razločevanje med nujnimi in nenujnimi prometnimi dogodki, kar omogoča njihov prikaz v prednostnem delu uporabniškega vmesnika. Sprva smo za klasifikacijo uporabljali plačljivi model GPT. Zaradi klasifikacije velikega števila dogodkov je ta pristop zahteval veliko plačljivih klicev programskega vmesnika OpenAI. Za izboljšanje hitrosti in zmanjšanje stroškov smo zasnovali lasten namenski model, s katerim lahko klasificiramo dogodke. Da bi zagotovili večjo prilagodljivost in robustnost sistema, nujnosti nismo napovedovali neposredno, temveč smo opredelili več lastnosti dogodkov, na podlagi katerih je mogoče z uporabo pravil določiti, ali je dogodek nujen. Model prepozna naslednje lastnosti: vožnja v napačno smer, zapora pasu zaradi nesreče, popolna zapora ceste zaradi nesreče, zapora ceste iz drugih razlogov, ovira na cesti, velika gneča (več kot 30 minut), varnostno tveganje, načrtovan dogodek in tip ceste. Preizkusili smo

več različnih modelov: diskriminativne modele SloBERTa, CroSloEngualBERT in ModerniBERT ter generativna modela GaMS-2B in GaMS-9B. Najboljše rezultate smo dosegli z modelom ModerniBERT, ki predstavlja različico modela ModernBERT, doučeno za slovenski jezik.

Za učenje namenskega klasifikacijskega modela smo pripravili uravnoteženo učno množico s skupno 58.000 primeri. Postopek je najprej obsegal zajem približno 15.000 dejanskih prometnih dogodkov iz preteklih poročil, ki smo jim vrednosti atributov samodejno določili z uporabo velikega jezikovnega modela GPT-5.2. Ker so bile določene kritične lastnosti (npr. vožnja v napačno smer) v realnih podatkih izjemno redke, smo z jezikovnim modelom sintetično ustvarili dodatne dogodke. Ustvarjanje smo nadaljevali, dokler za vsako vrednost posameznega atributa nismo zbrali vsaj 5.000 primerov, s čimer smo uspešno preprečili neenakomerno porazdelitev razredov in modelu omogočili stabilnejše učenje.

4.4. Generiranje prometnih poročil

4.4.1. Postopek generiranja prometnih poročil

Sistem loči med rednimi poročili, ki se generirajo na zahtevo urednika ali samodejno po nastavljenem urniku, zgodovinskimi poročili za izbran časovni trenutek, ročno sestavljenimi poročili iz izbranih zapisov ter nujnimi poročili, ki so vedno vezana na natanko en dogodek iz vira prometnih dogodkov. Za redna in nujna poročila velja dodatna, na zaledju uveljavljena zahteva po sprotnosti: v poštev pridejo le zapisi, ki so bili v zadnjem posnetku vira NAP zaznani v zadnjih 60 sekundah, s čimer se poročilo ne more sestaviti iz zapisov, ki jih je vmesni cikel pridobivanja podatkov že prehitel.

Kandidatni dogodki in dela na cesti se pred generiranjem izberejo in razvrstijo po administratorsko nastavljivi zgornji meji na posamezen vir ter po dveh dodatnih mehanizmih. Prvi je neobvezni filter zastarelosti vira: zapisi, katerih izvorni podatek je starejši od administratorsko nastavljenega praga v urah, se izločijo, saj lahko vir NAP dlje časa ne osveži dolgoročnih zapisov, kot so dela na cesti, ki pa niso pomembna za takojšnje radijsko poročanje. Drugi mehanizem je deterministična razvrstitev po pomembnosti: za dogodke se namesto preproste razvrstitve po kategoriji ceste uporablja razširjen vrstni red, ki najprej upošteva vzrok dogodka in šele nato skupino ceste (avtoceste/hitre ceste, regionalne ceste, lokalne ceste/javne poti) – vožnja v nasprotno smer je vedno na prvem mestu, sledijo nesreče na vseh cestah, nato zastoji, vreme, okvare in ovire, razvrščeni najprej po pomembnejši skupini cest, na koncu pa še dela na cesti in omejitve za tovorna vozila. Dela na cesti se razvrščajo po enostavnejši shemi, ki temelji neposredno na kategoriji ceste.

Poleg tega smo uvedli deterministični filter pomembnosti zastojev, ki pred generiranjem izloči zastoje pod administratorsko nastavljenim pragom: na avtocestah in hitrih cestah mora biti v zapisu navedena zamuda v minutah, ki dosega vsaj nastavljeni prag, na drugih cestah pa mora dolžina kolone v kilometrih dosegati vsaj nastavljeni prag.

Za nenujna poročila je pred klicem jezikovnega modela vključeno tudi predpomnjenje v Redisu, ki zmanjša odzivni čas in strošek klicev za vsebinsko enake zahteve. Vhodni zapisi in namig za glavo poročila se pred izračunom ključa normalizirajo: iz zapisov se odstranijo časovno spremenljiva polja (npr. čas zadnje zaznave), iz namiga za glavo pa datum in ura, tako da dva vsebinsko enaka zahtevka, ki se razlikujeta le v trenutku klica, ustvarita isti ključ; vrstni red zapisov se pri tem ohrani, saj neposredno določa oštevilčenje v prvi stopnji generiranja. Normaliziran zapis se skupaj z izbrano strategijo in različico shematike predpomnilnika kanonično serializira in zgosti v vsebinsko naslovljen ključ. Pod tem ključem se v Redisu z administratorsko nastavljenim časom veljavnosti shrani že izračunano poročilo; pri zadetku se preskočita obe stopnji klica jezikovnega modela in poročilo le dobi nov datum in uro glave.

Generiranje besedila poteka v dveh stopnjah z velikim jezikovnim modelom. V prvi stopnji model za vsak vhodni zapis neodvisno izdelava en normaliziran element: strukturirana semantična polja (vrsta dogodka, razred ceste, oznake) ter kratke, za radio pripravljene slovenske besedilne fragmente (uvodna fraza, fraza vzroka, fraza lokacije, fraza smeri, fraza učinka in morebitna dodatna fraza, npr. za zamudo ali obvoz), ki jih druga stopnja sestavi v celotne povedi. Model v tej stopnji ne sme razvrščati, združevati ali primerjati zapisov med seboj, ne sme si

izmišljati vzroka, ki v vhodu ni izrecno naveden, in ne sme v izhod vključiti cestnih oznak, evropskih številc cest ali oznak odsekov oziroma kilometrskih oznak – vhodni zapisi so bili namreč že izbrani z determinističnim filtriranjem na zaledju, zato prva stopnja ne sme dodatno uredniško presojeti pomembnosti posameznega dogodka, temveč sme element označiti kot neuporaben le, če je strukturno neuporaben (npr. brez vsebinsko smiselne podatka o prometu). Druga stopnja iz teh normaliziranih elementov sestavi končno besedilo: uredi vrstni red, združi elemente na isti cesti v en odstavek in za vsak vključen element ohrani sklic na izvorni zapis.

Klic vsake stopnje je zasnovan z rezervnim mehanizmom (angl. *fallback*) med ponudniki jezikovnih modelov: če je konfiguriran lasten (samostojno gostovan) model, se ta poskusi najprej, ob napaki pa se klic samodejno ponovi z oblaknim ponudnikom OpenAI, razen če je administrator izrecno izbral izključno lastni model. Odgovor vsake stopnje je strukturiran skladno z vnaprej opredeljeno shemo JSON, ki jo zaledje dodatno preveri in po potrebi popravi. Vsak klic jezikovnega modela – sistemsko sporočilo (in njegov zgoščen odtis), uporabniški poziv, odgovor ter morebitna napaka – se zabeleži ločeno za vsako stopnjo, kar omogoča sledljivost, nadzor nad stroški in analizo napak.

Nujna poročila imajo lastno logiko zunaj zgoraj opisanega predpomnjenja: sprožitev, generiranje in oddaja so poenoteni v eni storitvi, ki prek protokola WebSocket naznani začetek in konec generiranja, po uspešnem zaključku izvorni zapis označi kot uporabljen (da se v naslednjih poročilih ne podvaja) ter pred oddajo izrecno preveri, da poročilo dejansko vsebuje natanko en sprožitveni dogodek in nobenega dela na cesti. Shranjevanje poročila, povezav do izvornih zapisov in zapisov o klicih jezikovnega modela poteka v eni sami podatkovni transakciji, tako da je poročilo bodisi v celoti shranjeno z vsemi sledljivimi povezavami bodisi sploh ne.

Vsako shranjeno poročilo poleg besedila in glave hrani tudi metapodatke, potrebne za poznejšo rekonstrukcijo in vpogled: strategijo generiranja, identifikatorje uporabljenih izvornih zapisov po viru ter posnetek vseh administratorsko nastavljivih parametrov generiranja, ki so veljali v trenutku nastanka poročila – ali je bil vključen filter zastarelosti vira in s kakšnim pragom, ali je bilo predpomnjenje omogočeno in s kakšnim časom veljavnosti, ter ali je bil vključen filter pomembnosti zastojev in z ustreznimi pragovi. Ta posnetek je potreben, ker so ustrezne nastavitve sistemsko globalne in se ob vsaki spremembi prepišejo, zato bi bila brez posnetka izgubljena informacija o tem, s kakšnimi nastavitvami je bilo generirano posamezno starejše poročilo.

4.4.2. Primer rednega prometnega poročila

Pri prikazanem primeru rednega poročila sta bila za generiranje izbrana dva prometna dogodka, ki se nanašata na različni cesti. Prva stopnja generiranja je za vsak vhodni zapis pripravila ločen strukturiran element z normaliziranimi podatki in besedilnimi fragmenti. Druga stopnja je iz obeh elementov oblikovala dve samostojni povedi. Ker se dogodka nanašata na različni cesti, ju sistem ni združil v isti odstavek.

Tabela 4: Primer izhodov prve in druge stopnje generiranja prometnega poročila.

Vhod	Stopnja 1 (izhod)	Stopnja 2 (izhod)
3958: "A4, Maribor - Gruškovje (Hrvaška), priključek Marjeta - priključek Zlatoličje v smeri Podlehnik, okvara vozila, oviran promet."	<pre>{ "source_ref": "3958", "road": "podravska avtocesta", "road_class": "avtocesta_hitra", "event_type": "drugo", "lead_phrase": "na podravski avtocesti", "cause_phrase": null, "location_phrase": "med priključkoma Marjeta in Zlatoličje", "direction_phrase": "proti Podlehniku", "effect_phrase": "oviran promet zaradi okvare vozila", "tags": ["okvara", "oviran_promet"] }</pre>	Na podravski avtocesti je med priključkoma Marjeta in Zlatoličje proti Podlehniku oviran promet zaradi okvare vozila.

3960: "G1-11, Šmarje - Koper, zastoj, dolžina: 2 km in 700 m."	<pre>{ "source_ref": "3960", "road": "Šmarje-Koper", "road_class": "regionalna_glavna", "event_type": "zastoj", "lead_phrase": "na glavni cesti Šmarje-Koper", "cause_phrase": null, "location_phrase": null, "direction_phrase": null, "effect_phrase": "zastoj", "tags": ["zastoj"] }</pre>	Na glavni cesti Šmarje-Koper je zastoj.
--	---	---

Končno poročilo, prikazano uredniku, je nato oblikovano na naslednji način:

Prometna informacija 19. 7. 2026 21.36. Podatki o prometu.

Na podravski avtocesti je med priključkoma Marjeta in Zlatoličje proti Podlehniku oviran promet zaradi okvare vozila.

Na glavni cesti Šmarje-Koper je zastoj.

4.4.3. Primer nujnega prometnega poročila

Pri nujnem poročilu je postopek generiranja vezan na en sam prometni dogodek, ki ga je klasifikator označil kot pomembnega. V prikazanem primeru je šlo za okvaro vozila na primorski avtocesti pred počivališčem Ravbarkomanda v smeri Ljubljane. Iz dogodka je prva stopnja pripravila normalizirane besedilne fragmente, druga stopnja pa jih je sestavila v končno poved nujnega poročila.

- Vhod: "A1, Koper - Ljubljana, pred počivališčem Ravbarkomanda v smeri Ljubljane, okvara vozila, oviran promet."
- Stopnja 1: {"road": "primorska avtocesta", "road_class": "avtocesta_hitra", "cause_phrase": "zaradi okvare vozila", "location_phrase": "pred počivališčem Ravbarkomanda", "direction_phrase": "proti Ljubljani", "effect_phrase": "oviran promet", "tags": ["okvara", "oviran_promet"]}
- Stopnja 2: "Na primorski avtocesti je zaradi okvare vozila pred počivališčem Ravbarkomanda proti Ljubljani oviran promet."

Končno poročilo, prikazano uredniku, je nato oblikovano tako:

Prometna informacija 22. 7. 2026 18.43. Nujna prometna informacija.

Na primorski avtocesti je zaradi okvare vozila pred počivališčem Ravbarkomanda proti Ljubljani oviran promet.

4.5. Namestitev in operativni vidiki

Sistem se namešča kot skupina vsebnikov Docker, opisana v razdelku 4.1, ločeno za razvojno in produkcijsko okolje: razvojna konfiguracija izpostavi vrata za neposreden dostop in uporablja kodo iz delovnega imenika, produkcijska pa storitve izpostavi izključno prek storitve Traefik s samodejno pridobljenim potrdilom TLS. Vsaka storitev ima nastavljeno politiko samodejnega ponovnega zagona ter, kadar je smiselno, tudi preverjanje stanja (angl. *healthcheck*) na aplikacijski ravni. Spremembe podatkovne sheme se uveljavljajo prek migracijskega orodja Alembic [29], kar omogoča nadzorovan in ponovljiv prehod med različicami sheme brez ročnih posegov v bazo.

Del obvladovanja tveganj je vgrajen že v samo aplikacijo, neodvisno od infrastrukturnega spremljanja: dvostopenjsko generiranje poročil ima rezervni mehanizem med ponudniki jezikovnih modelov, predpomnjenje v Redisu je zasnovano tako, da vsaka napaka pri dostopu do Redisa generiranja poročila ne prekine, temveč ga le izvede brez predpomnilnika, vsak klic jezikovnega modela pa se zabeleži zaradi sledljivosti in naknadne analize napak. Prav tako se ob vsakem generiranju v metapodatkih poročila shrani posnetek takrat veljavnih nastavitvev (filter zastarelosti vira, predpomnjenje, filter pomembnosti zastojev), kar omogoča naknadno razumevanje delovanja sistema, tudi če so se administratorske nastavitve medtem spremenile.

Za operativno spremljanje delovanja je vzpostavljen sklop orodij Prometheus, Grafana in node-exporter, ki v realnem času prikazuje zasedenost niti in bazena povezav do baze glede na njune konfigurirane zgornje meje, trajanje in prepustnost generiranja poročil ter osnovne metrike gostitelja. To upravljanje tveganj je trenutno pretežno opazovalno: nadzorna plošča je na voljo administratorju, ki jo mora sam pregledati, obveščanje o presežnih vrednostih pa ni samodejno – Prometheus ne izvaja pravil za alarme (angl. *alerting rules*), v namestitev pa ni vključena komponenta za obveščanje, prek katere bi bila anomalija (npr. izčrpan bazen povezav do baze ali neobičajno visok delež neuspešnih tvorjenj poročil) samodejno sporočena administratorju, namesto da jo ta opazi šele ob naslednjem pregledu nadzorne plošče.

Nekateri operativni vidiki sistema ostajajo predmet nadaljnjega dela. Trenutno se surovi in obdelani prometni zapisi v podatkovni bazi hranijo brez samodejnega arhiviranja ali brisanja, zato bo treba pri daljši produkcijski uporabi uvesti politiko hrambe podatkov. Prav tako je predvidena vzpostavitev rednega varnostnega kopiranja podatkovne baze in dokumentiranega postopka obnovitve po izpadu. Namestitev sistema za zdaj poteka ročno, zato bi nadaljnja avtomatizacija uvajanja omogočila bolj ponovljive in varnejše posodobitve. Dodatna možnost izboljšave je tudi centralizirano zbiranje dnevnikov posameznih storitev, kar bi olajšalo naknadno analizo napak in spremljanje delovanja sistema.

5 Vrednotenje sistema

Za generiranje prometnih poročil smo primerjali komercialni oblačni model GPT-5.1 in dve različici lokalnega modela GaMS3-12B-Instruct, prilagojenega za slovenščino (ena različica je bila učena na izvlečkih zgodovinskih poročil, druga pa na vhodno-izhodnih parih modela GPT). Strokovnjaki z RTV Slovenija so na podlagi evalvacijskega dokumenta z vzporednimi prikazi poročil ugotovili, da druga različica modela GaMS ustvarja jezikovno in slovnično najbolj kakovostna besedila. Kljub boljši jezikovni ustreznosti pa se je pri nadaljnjem delu lokalni model izkazal za manj prožnega. Zaradi strogega prilagajanja na učno množico je pri poskusih spreminjanja poziva (angl. *prompt*) občasno še vedno sledil prvotno naučenim vzorcem in s tem izgubil del splošnosti, medtem ko je bil model, dostopen prek programskega vmesnika OpenAI API, bistveno bolj prilagodljiv sprotnim spremembam pozivov.

Ko je aplikacijo v produkciji začelo uporabljati več uporabnikov (od 8. do 21. julija 2026 – to je najbolj reprezentativno obdobje, saj se je takrat začela intenzivna produkcijska uporaba), je povprečni dnevni strošek znašal približno 12,97 € na dan, kar ob mesečni ekstrapolaciji pomeni približno 389 € na mesec oziroma 4.670 € na leto. Za lokalno tvorjenje besedila smo že preizkusili model GaMS3-12B-Instruct (kvantiziran Q6_K), ki pri približno 6,56 bita na utež zavzame približno 10 GB pomnilnika za uteži [9]. Nekvantizirana različica bi predvidoma potrebovala približno 32 GB pomnilnika VRAM, kvantizirana pa pri skoraj enaki kakovosti besedila deluje na grafični kartici s 16 GB pomnilnika VRAM; v praksi uporabljamo kartico NVIDIA GeForce RTX 5070 Ti (16 GB, 300 W, priporočena maloprodajna cena približno 750 USD) [10]. Ob neprekinjenem 24-urnem delovanju in ceni električne energije za poslovne odjemalce približno 0,18 € na kWh (referenčna cena za negospodinske odjemalce v prvem četrtletju 2026) [11], [12] bi tekoči strošek električne energije za kartico RTX 5070 Ti znašal približno 39 € na mesec oziroma 467 € na leto, kar je bistveno manj od mesečnega stroška klicev storitve OpenAI pri polni produkcijski uporabi. Lastna strojna oprema (približno 700–900 €) bi se ob trenutnem tempu rasti povrnila v manj kot dveh mesecih, pri čemer je treba upoštevati še dodaten čas in stroške vzdrževanja ter nadgradenj in odsotnost podpore, ki jo ponuja OpenAI.

Aplikacija trenutno omogoča lektorjem, da pri posameznem poročilu označijo besedilo kot ustrezno, uredijo generirano besedilo, označijo tip napake in dodajo prost komentar; ob ponovnem odprtju poročila se namesto izvirnega generiranega besedila prikaže zadnja lektorjeva popravljena različica, kar velja tudi za druge uporabnike, ki si isto poročilo ogledajo pozneje. Povezava med zbranimi povratnimi informacijami in dejanskim izboljševanjem poziva trenutno še ni samodejna, temveč zahteva ročni pregled in vključevanje ugotovitev v nadaljnji razvoj. V prihodnje bi bilo smiselno raziskati možnosti za delno ali popolnoma samodejno izvedbo tega procesa ter vzpostaviti sistematično povezavo med povratnimi informacijami uporabnikov in inženirstvom pozivov.

6 Zaključek

V prispevku smo predstavili zasnovo in implementacijo sistema za samodejno generiranje prometnih poročil za radijsko poročanje na RTV Slovenija. Sistem povezuje sprotne prometne podatke iz nacionalnega portala NAP, podatkovni cevovod za shranjevanje in normalizacijo zapisov, klasifikacijski model za zaznavanje pomembnih oziroma nujnih dogodkov ter dvostopenjsko generiranje besedila z velikimi jezikovnimi modeli. Pokazali smo, da kakovost končnega poročila ni odvisna samo od izbire jezikovnega modela, temveč predvsem od kakovosti, strukture, filtriranja in razvrščanja vhodnih podatkov.

Pomemben del razvoja je predstavljala povezava med zgodovinskimi in sprotnimi podatki. Zgodovinski prometni zapisi in arhivirana radijska poročila so omogočili razumevanje uredniškega procesa, pripravo evalvacijskih primerov in razvoj pristopov za klasifikacijo ter generiranje besedila. Hkrati se je pokazalo, da zgodovinskih podatkov ni mogoče uporabiti neposredno v celoti, saj morajo modeli v produkcijskem delovanju prejemati vhodne podatke, ki so primerljivi s podatki iz NAP dostopnih točk. Zato je bila posebna pozornost namenjena izboru tistih vsebinskih sklopov in atributov, ki jih je mogoče zanesljivo pridobiti tudi iz sprotnega vira.

Rešitev je bila preizkušena in se že uporablja tudi v dejanski uredniški praksi. Povratne informacije uredništva so pokazale, da sistem lahko uspešno podpira pripravo prometnih poročil, hkrati pa so razkrile pomen sprotnega prilagajanja pravil, pragov in pozivov jezikovnemu modelu. V resničnem obratovanju so se pokazali primeri, ki jih samo vrednotenje na zgodovinskih podatkih ne bi zadostno razkrilo, na primer obravnava vožnje v nasprotno smer, zastaranje nujnih obvestil, dvoumna krajevna imena, manjkajoči vzroki dogodkov in pragovi za vključevanje zastojev. Ti primeri potrjujejo, da mora biti sistem zasnovan kot nastavljiva in sledljiva uredniška podpora, ne kot popolnoma zaprt samodejni generator.

Primerjava med oblračnimi in lokalno prilagojenimi jezikovnimi modeli je pokazala tudi praktičen kompromis med kakovostjo besedila, stroški, odzivnostjo in neodvisnostjo od zunanjih ponudnikov. Oblačni modeli trenutno dosegajo boljšo slogovno kakovost, vendar njihova uporaba prinaša operativne stroške in odvisnost od zunanje storitve. Zato zbrani popravki, komentarji in uredniške oznake predstavljajo pomembno osnovo za nadaljnje prilagajanje lokalnega modela, s katerim bi bilo mogoče postopoma zmanjšati odvisnost od plačljivih oblračnih rešitev.

Na podlagi razvoja in pilotne uporabe lahko sklenemo, da je samodejno generiranje prometnih poročil izvedljivo in uporabno, kadar je podprto z zanesljivim podatkovnim cevovodom, determinističnim filtriranjem, sledljivostjo do izvornih zapisov in možnostjo končne uredniške presoje. Sistem zato ni namenjen popolni zamenjavi urednika, temveč predvsem zmanjšanju ročnega dela, hitrejšemu zaznavanju pomembnih dogodkov in pripravi kakovostnega poročila. Nadaljnje delo vključuje dodatno prilagoditev lokalnega jezikovnega modela na zbranih produkcijskih podatkih, bolj sistematično vrednotenje kakovosti generiranih poročil ter nadgradnje uporabniškega vmesnika glede na potrebe poročanja.

Zahvala

Delo je nastalo v sodelovanju z RTV Slovenija ter v okviru projekta European Union GA 101198470 »Digital Europe Programme: LLMs4EU« in projekta Slovenian Artificial Intelligence Factory (SLAIF, 101254461), ki ga sofinancira Evropska unija (HORIZON-JU-EUROHPC-2025-AI-01-IBA). Izražena stališča in mnenja so izključno stališča avtorjev in ne odražajo nujno stališč Evropske unije ali EuroHPC JU.

Literatura

- [1] <https://nap.si/sl>, Nacionalna točka dostopa, obiskano 21. 5. 2026.
- [2] MEIR Nicole, »Automated earnings stories multiply«, *The Associated Press*, 15. 6. 2015, <https://www.ap.org/the-definitive-source/announcements/automated-earnings-stories-multiply/>, obiskano 4. 6. 2026.
- [3] THE WASHINGTON POST, »The Washington Post experiments with automated storytelling to help power 2016 Rio Olympics coverage«, *The Washington Post*, 5. 8. 2016, <https://www.washingtonpost.com/pr/wp/2016/08/05/the-washington-post-experiments-with-automated-storytelling-to-help-power-2016-rio-olympics-coverage/>, obiskano 4. 6. 2026.
- [4] LOS ANGELES TIMES, »What is the Quakebot and how does it work?«, *Los Angeles Times*, <https://www.latimes.com/la-me-quakebot-faq-20190517-story.html>, obiskano 4. 6. 2026.
- [5] LIU Xiaomo, NOURBAKSH Armineh, LI Quanzhi, SHAH Sameena, MARTIN Robert, DUPREY John, »Reuters Tracer: Toward Automated News Production Using Large Scale Social Media Data«, *2017 IEEE International Conference on Big Data*, IEEE, 2017, str. 1483–1493, doi: <https://doi.org/10.1109/BigData.2017.8258082>.
- [6] REUTERS COMMUNICATIONS, »How AI helps power trusted news at Reuters«, *Reuters*, 12. 7. 2023, <https://www.reuters.com/media-center/how-ai-helps-power-trusted-news-at-reuters/>, obiskano 4. 6. 2026.
- [7] FINTRAFFIC, »Bauer Media to start broadcasting targeted traffic bulletins produced locally using artificial intelligence on the radio«, *Fintraffic*, 8. 11. 2023, zadnjič spremenjeno 9. 1. 2026, <https://www.fintraffic.fi/en/news/bauer-media-start-broadcasting-targeted-traffic-bulletins-produced-locally-using-artificial>, obiskano 4. 6. 2026.
- [8] INRIX, »INRIX Launches AI-Powered Traffic Reports for TV and Radio – Fully Replacing Human Broadcasters«, *INRIX*, 18. 11. 2025, <https://inrix.com/press-releases/inrix-launches-ai-powered-traffic-reports-for-tv-and-radio-fully-replacing-human-broadcasters/>, obiskano 4. 6. 2026.
- [9] GGML-ORG, »llama.cpp – tools/quantize README (K-quants)«, GitHub, <https://github.com/ggml-org/llama.cpp/blob/master/tools/quantize/README.md>, obiskano 23. 7. 2026.
- [10] NVIDIA, »GeForce RTX 5070 Ti«, NVIDIA, <https://www.nvidia.com/en-us/geforce/graphics-cards/50-series/rtx-5070-family/>, obiskano 23. 7. 2026.
- [11] ENERGETIKA-PORTAL.SI, »Cene električne energije v prvem četrtletju 2026«, *Energetika-portal.si*, <https://www.energetika-portal.si/nc/novica/n/cene-elektricne-energije-v-prvem-cetrletju-2026/>, obiskano 23. 7. 2026.
- [12] GEN-I, »Nove cene tarif električne energije za male poslovne odjemalce«, GEN-I, <https://gen-i.si/novice/nove-cene-tarif-elektricne-energije-za-male-poslovne-odjemalce/>, obiskano 23. 7. 2026.
- [13] RADIOTODAY, »AI-powered traffic reports have arrived at Bauer«, *RadioToday*, 14. 11. 2025, <https://radiotoday.co.uk/2025/11/ai-powered-traffic-reports-have-arrived-at-bauer/>, obiskano 23. 7. 2026.
- [14] SALTON Gerard, BUCKLEY Christopher, »Term-weighting approaches in automatic text retrieval«, *Information Processing & Management*, let. 24, št. 5, 1988, str. 513–523, doi: [https://doi.org/10.1016/0306-4573\(88\)90021-0](https://doi.org/10.1016/0306-4573(88)90021-0).
- [15] SALTON Gerard, WONG Anita, YANG Chung-Shu, »A Vector Space Model for Automatic Indexing«, *Communications of the ACM*, let. 18, št. 11, 1975, str. 613–620, doi: <https://doi.org/10.1145/361219.361220>.
- [16] DOCKER, »What is Docker?«, *Docker Docs*, <https://docs.docker.com/get-started/docker-overview/>, obiskano 26. 7. 2026.
- [17] TRAEFIK LABS, »Configuration Introduction«, *Traefik Documentation*, <https://doc.traefik.io/traefik/getting-started/configuration-overview/>, obiskano 26. 7. 2026.
- [18] INTERNET SECURITY RESEARCH GROUP, »Documentation«, *Let's Encrypt*, <https://letsencrypt.org/docs/>, obiskano 26. 7. 2026.

- [19] PROMETHEUS AUTHORS, »Overview«, *Prometheus Documentation*, <https://prometheus.io/docs/introduction/overview/>, obiskano 26. 7. 2026.
- [20] GRAFANA LABS, »About Grafana«, *Grafana Documentation*, <https://grafana.com/docs/grafana/latest/introduction/>, obiskano 26. 7. 2026.
- [21] PALLETS PROJECTS, »Welcome to Flask«, *Flask Documentation*, <https://flask.palletsprojects.com/en/stable/>, obiskano 26. 7. 2026.
- [22] REDIS, »Redis Documentation«, *Redis Docs*, <https://redis.io/docs/latest/>, obiskano 26. 7. 2026.
- [23] RAMÍREZ Sebastián, »FastAPI«, *FastAPI Documentation*, <https://fastapi.tiangolo.com/>, obiskano 26. 7. 2026.
- [24] WARNER Benjamin et al., »Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference«, arXiv:2412.13663, 2024, <https://arxiv.org/abs/2412.13663>.
- [25] PASZKE Adam et al., »PyTorch: An Imperative Style, High-Performance Deep Learning Library«, *Advances in Neural Information Processing Systems* 32, 2019, str. 8024–8035, <https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>.
- [26] OPENAI, »OpenAI API Documentation«, <https://developers.openai.com/api/docs/>, obiskano 26. 7. 2026.
- [27] GOOGLE, »What is Angular?«, *Angular Documentation*, <https://angular.dev/overview>, obiskano 26. 7. 2026.
- [28] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, »PostgreSQL 17 Documentation«, <https://www.postgresql.org/docs/17/>, obiskano 26. 7. 2026.
- [29] BAYER Mike, »Alembic Documentation«, <https://alembic.sqlalchemy.org/en/latest/>, obiskano 26. 7. 2026.

Od podatkov do prvakov: integracija podatkov za podporo odločanja v športu

Jernej Flisar

Twelve Football, Stockholm, Švedska
jernej@twelve.football

Uporaba podatkov je v zadnjih letih postala pomemben del vodenja profesionalnih športnih ekip. Nogomet se je v ta podatkovni razvoj vključil nekoliko pozneje kot nekateri drugi športi, predvsem zaradi same narave igre, ki je dinamična, manj strukturirana in vključuje veliko število igralcev, ki se hkrati gibljejo po velikem prostoru. V prispevku predstavljamo, kako lahko iz surovih nogometnih podatkov pridemo do ključnih informacij, ki jih trenerji, analitiki in drugi člani strokovnega štaba uporabijo v svojih procesih odločanja. Trenerji in analitiki pri tem niso samo zunanji opazovalci, temveč so aktivno vključeni v proces oblikovanja metrik in modelov. Osrednji del prispevka predstavlja praktičen primer, v katerem smo trenerjev princip obrambne kompaktnosti prevedli v podatkovni model in oblikovali metrike, s katerimi je bilo mogoče spremljati, ali ekipa dogovorjene principe dejansko izvaja na igrišču in kje prihaja do odstopanj. Za obdelavo, transformacijo in modeliranje podatkov smo uporabili kombinacijo sistemov DuckDB in MotherDuck ter programskega jezika Python, rezultate pa predstavili prek interaktivnih pregledov v Streamlitu in samodejno pripravljenih poročil v formatu PDF. S prispevkom želimo pokazati, da podatki v nogometu ne nadomeščajo trenerjevega znanja, temveč ga dopolnjujejo tako, da taktične ideje naredijo merljive, preverljive in uporabne v vsakodnevnem delu.

Ključne besede:

nogomet

dogodkovni in sledilni podatki

taktična analiza

DuckDB

podpora odločanju

1 Uvod

Uporaba podatkov je v zadnjih letih postala pomemben del vodenja profesionalnih športnih ekip. Medtem ko so se nekateri športi, predvsem bejzbol in košarka, na podatkovno analitiko naslonili razmeroma zgodaj, se je nogomet v ta podatkovni razvoj vključil nekoliko pozneje, predvsem zaradi same narave igre. Igra je dinamična, manj strukturirana in vključuje veliko število igralcev, ki se hkrati gibljejo po velikem prostoru, zato zbiranje, obdelava in interpretacija podatkov v nogometu predstavljajo poseben izziv. Igra je le redko razdeljena na jasno ločene in ponovljive situacije, ki bi jih bilo mogoče neposredno prešteti in primerjati, posamezni dogodek z žogo pa pove razmeroma malo o tem, kaj se je v istem trenutku dogajalo na preostalem delu igrišča [1].

Dodatna posebnost nogometa je majhno število zadetkov. Ker je zadetkov na tekmo malo, je izid močno odvisen od naključja, kar pomeni, da rezultat sam po sebi ni zanesljiv pokazatelj kakovosti prikazane igre. Prav zato so se v nogometu uveljavile metrike, ki namesto izida merijo proces, ki do izida pripelje, in s tem omogočajo bolj stabilno oceno uspešnosti ekipe skozi daljše obdobje.

V zadnjih letih se je dostopnost podatkov bistveno spremenila. Poleg dogodkovnih podatkov, ki opisujejo akcije z žogo, so postali široko dostopni tudi sledilni podatki, ki zajemajo položaje vseh igralcev in žoge v kratkih časovnih intervalih. Takšni podatki omogočajo analizo kolektivnega taktičnega vedenja ekipe, torej tistega dela igre, ki se dogaja brez žoge in je bil prej dostopen le z videoanalizo in subjektivno presojo [2]. Sistematični pregledi raziskav kažejo, da prav pozicijski podatki odpirajo največ novih možnosti za razumevanje kolektivnega vedenja ekip, hkrati pa opozarjajo, da metodološka raznolikost in pomanjkanje standardizacije otežujeta prenos raziskovalnih ugotovitev v vsakodnevno delo strokovnih štabov [3].

Prav ta prenos je po naših izkušnjah osrednji izziv. Tehnične možnosti za izračun naprednih metrik so danes dostopne tudi manjšim klubom, vendar analitika pogosto ostane ločena od dela strokovnega štaba. Metrike, ki niso povezane z igralnim modelom ekipe in s konkretnimi vprašanji trenerjev, se v praksi ne uporabijo, ne glede na to, kako dovršena je njihova statistična osnova. Dodatno omejitev predstavlja časovni okvir: ker se tekme igrajo vsakih nekaj dni, so roki za pripravo analiz zelo kratki, zahteve pa se hitro spreminjajo glede na nasprotnika, stanje ekipe in ugotovitve iz prejšnjih tekem.

V prispevku najprej podrobneje opišemo glavne sklope nogometnih podatkov, razlike med njimi, njihove ponudnike in osnovne metrike, ki jih iz njih izpeljemo. Nato predstavimo področja, na katerih klubi podatke uporabljajo v vsakdanjem delu, postopek prevajanja taktičnih principov v merljive metrike in podatkovno arhitekturo, na kateri rešitev temelji. Osrednji del prispevka predstavlja študija primera iz sodelovanja s strokovnim štabom kluba angleške Premier lige.

2 Nogometni podatki

Nogometne podatke najpogosteje delimo v tri glavne sklope: dogodkovne, sledilne in fizične. Sklopi se med seboj dopolnjujejo in odgovarjajo na različne vrste vprašanj, zato izbira podatkovnega vira ni tehnična, temveč izhaja iz vprašanj, na katera želimo odgovoriti.

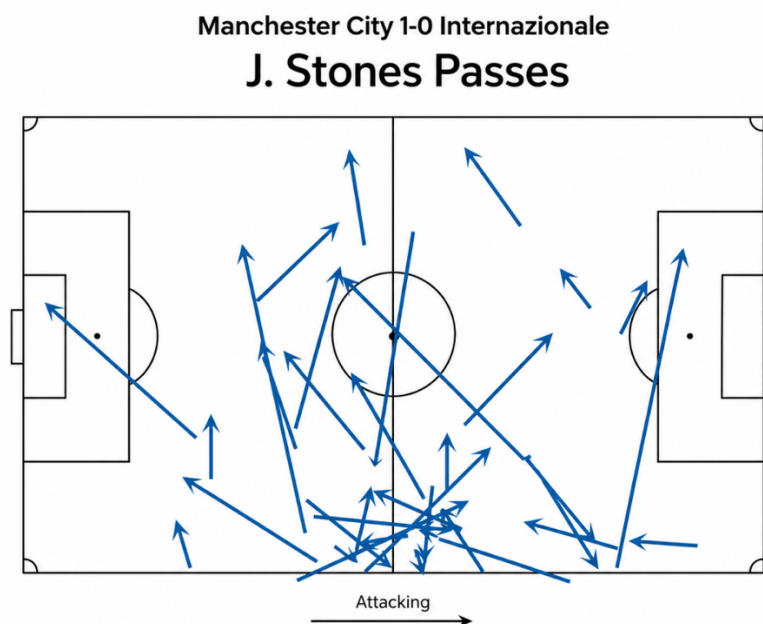
2.1. Dogodkovni podatki

Dogodkovni podatki (angl. event data) opisujejo posamezne akcije akterjev (igralcev) z žogo. Vključujejo podaje, strele, preigravanja, odvzeme žoge, prekrške, prekinitve in zadetke, vsaka akcija pa je umeščena v prostor na igrišču in v čas tekme.

Posamezen zapis običajno vsebuje vrsto dogodka, igralca, ekipo, časovno oznako, koordinate na igrišču in izid akcije, pogosto pa tudi dodatne attribute, na primer telesni del, s katerim je bila akcija izvedena, vrsto podaje ali oznako, ali je bila akcija izvedena pod pritiskom. Na eni tekmi je zabeleženih približno dva do tri tisoč dogodkov.

Zajem je bil dolgo pretežno ročen: operaterji so ob spremljanju posnetka označevali dogodke po vnaprej določenem klasifikacijskem sistemu. V zadnjih letih se ročno označevanje vse bolj dopolnjuje s samodejnim prepoznavanjem, kar znižuje stroške in skrajšuje čas do razpoložljivosti podatkov.

Ključna omejitev dogodkovnih podatkov je, da opisujejo le akterja z žogo. O tem, kje so bili v istem trenutku preostali igralci, ne povedo ničesar. Ker ima igralec žogo v posesti le kratek čas, dogodkovni podatki zajamejo razmeroma majhen del dogajanja. Za analizo obrambne organizacije, ki je po definiciji kolektivna in se večinoma odvija brez žoge, so zato sami po sebi nezadostni.



Slika 1: Dogodkovni podatki – podaje posameznega igralca na eni tekmi. Vsaka puščica prikazuje mesto, smer in dolžino podaje.

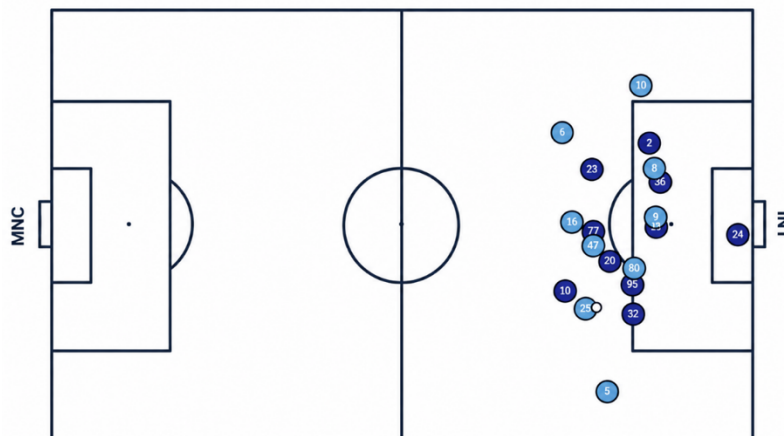
2.2. Sledilni podatki

Sledilni podatki (angl. tracking data) zajemajo položaje, hitrosti in pospeške vseh igralcev ter žoge v kratkih časovnih intervalih, običajno petindvajsetkrat na sekundo. Ena tekma tako obsega več milijonov podatkovnih točk, kar je za nekaj velikostnih redov več od dogodkovnih podatkov in ima neposredne posledice za izbiro podatkovne arhitekture.

Zajem poteka na tri načine. Optični sistemi s kamerami, nameščenimi na stadionu, zagotavljajo najvišjo kakovost in zajamejo vse igralce skozi celotno tekmo. Sistemi, ki sledenje izračunajo iz televizijskega signala (angl. broadcast tracking), so cenejši in dostopni za bistveno več tekmovalj, vendar zajamejo le tiste igralce, ki so v danem trenutku v kadru, zato je treba manjkajoče položaje ocenjevati. Naprave GPS in druge nosljive naprave zagotavljajo natančne podatke o gibanju posameznega igralca, vendar praviloma le za lastno ekipo in predvsem za namene spremljanja obremenitve.

Prav sledilni podatki omogočajo analizo kolektivnega taktičnega vedenja: medsebojnih razdalj med igralci, oblike in višine obrambne linije, kompaktnosti ekipe, gostote pokrivanja prostora ter usklajenosti gibanja. Pregled raziskav o analizi obrambne igre s sledilnimi podatki kaže, da metrike segajo od individualne ravni, kamor sodita obrambni pritisk in razdalja do žoge, do ekipne ravni, kjer se merijo razdalje med linijami, številčna premoč in kompaktnost [4]. Na sledilnih podatkih temeljijo tudi prostorski modeli, ki ocenjujejo, kateri del igrišča posamezna ekipa v danem trenutku obvladuje (angl. pitch control) [5]. Kot je zapisal Johan Cruyff, igralca opredeljuje predvsem to, kaj počne v tistih minutah, ko žoge nima; prav ta del igre postane merljiv šele s sledilnimi podatki.

Manchester City 1-0 Internazionale



Slika 2: Sledilni podatki – položaji vseh igralcev obeh ekip in žoge v enem samem trenutku tekme.

2.3. Fizični podatki

Iz sledilnih podatkov in nosljivih naprav izpeljemo fizične podatke o aktivnosti igralcev: pretečeno razdaljo po hitrostnih conah, najvišjo hitrost, število sprintov, število tekov z visoko intenzivnostjo in število pospeškov. Dopolnjujejo jih meritve srčnega utripa in drugi podatki, ki jih spremlja kondicijski del strokovnega štaba.

Fizične podatke je smiselno vedno obravnavati ločeno za faze s posestjo in za faze brez nje, saj sta profil obremenitve in njegova razlaga v teh dveh fazah povsem različna. Sama pretečena razdalja je brez taktičnega konteksta zavajajoča: večja razdalja ni nujno znak boljše igre, temveč je lahko posledica slabše postavitve, zaradi katere mora igralec pokrivati več prostora.

2.4. Razlike med dogodkovnimi in sledilnimi podatki

Razlika med obema sklopoma ni le v obsegu, temveč predvsem v tem, na katera vprašanja lahko odgovorita. Dogodkovni podatki opisujejo, kaj se je zgodilo z žogo, sledilni podatki pa prikazujejo položaj vseh igralcev v trenutku posameznega dogodka. Zaradi različnih informacij, ki jih zagotavljata, ima vsak sklop določene omejitve. Šele njuna kombinacija omogoča celovitejšo analizo igre. Primerjavo povzemamo v tabeli 1.

Tabela 1: Primerjava dogodkovnih in sledilnih podatkov ter kombinacija obeh.

	Dogodkovni podatki		Sledilni podatki	Dogodkovni in sledilni podatki skupaj
Kaj zajemajo	akcije z žogo (angl. on-ball), ki se zgodijo med tekmo		položaje, hitrosti in pospeške vseh igralcev ter žoge med tekmo	združitev obeh sklopov v enotno predstavitev
Primeri vsebine	zadetki, streli, odvzemi, prekrški	podaje,	gibanje ekipe brez žoge, razdalje med linijami, pokrivanje prostora	akcija z žogo skupaj s postavitvijo vseh igralcev v istem trenutku
Način zajema	označevanje posnetka, bolj samodejno	vse	zajem iz TV signala, kamer na stadionu, naprav GPS	povezava obeh virov po času in identiteti igralcev
Obseg na tekmo	2–3 tisoč dogodkov		več kot 2 milijona podatkovnih točk	obseg sledilnih podatkov, obogaten z oznakami dogodkov
Česa ne pove	kaj se v istem trenutku dogaja s preostalimi igralci		katera akcija z žogo v danem trenutku poteka	

Povezava obeh sklopov je v praksi pogosta, saj šele ta omogoča vprašanja, na katera noben vir sam ne odgovori: koliko nasprotnikov je bilo v bližini podajalca, ali je bil strel izveden pod pritiskom, kateri soigralci so bili v tistem trenutku prosti in katera podajna možnost je ostala neizkoriščena. Nekateri ponudniki takšen kontekst ponujajo neposredno, tako da vsakemu dogodku priložijo položaje igralcev v tistem trenutku, kar predstavlja vmesno obliko med obema sklopoma.

Sklopa se razlikujeta tudi po dostopnosti. Dogodkovni podatki so na voljo za bistveno več tekmovalj, vključno z nižjimi ligami, in so zato običajna osnova za skavtiranje v širokem naboru trgov. Sledilni podatki so dražji in omejeni predvsem na višja tekmovalja, so pa nepogrešljivi za taktično in fizično analizo, kakršno opisujemo v študiji primera.

2.5. Ponudniki podatkov

Trg podatkov je razdeljen med več ponudnikov, ki se razlikujejo po pokritosti tekmovalj, načinu zajema in podrobnosti klasifikacijskega sistema. Med ponudniki dogodkovnih podatkov so najbolj uveljavljeni Stats Perform s podatki pod znamko Opta, StatsBomb in Wyscout, pri čemer se razlikujejo predvsem po številu zabeleženih atributov na dogodek in po pokritosti nižjih tekmovalj.

Pri sledilnih podatkih ločimo optične sisteme, nameščene na stadionih, med katerimi so Second Spectrum, TRACAB in Hawk-Eye, ter sisteme, ki sledenje izračunajo iz televizijskega signala, med katerimi je najbolj razširjen SkillCorner. Prvi zagotavljajo večjo natančnost in popolno pokritost igralcev, drugi pa bistveno širšo pokritost tekmovalj ob nižjih stroških. Fizične podatke s treningov in tekem zagotavljajo proizvajalci nosljivih naprav, med katerimi sta najbolj razširjena Catapult in STATSports.

Za klub izbira ponudnika ni le stroškovna odločitev. Formati in klasifikacijski sistemi se med ponudniki razlikujejo, zato menjava vira pomeni, da zgodovinskih podatkov ni mogoče neposredno primerjati z novimi. Prav to je eden od razlogov, da v podatkovni arhitekturi, opisani v poglavju 5, podatke različnih ponudnikov čim prej pretvorimo v enotno interno shemo.

2.6. Osnovne metrike

Iz opisanih podatkov gradimo metrike, ki surove zapise pretvorijo v oceno vrednosti.

Pričakovani zadetki (angl. expected goals, xG) so najbolj uveljavljena metrika v nogometu in za večino klubov prvi korak v podatkovno analitiko. Model vsakemu strelu pripiše verjetnost, da se konča z zadetkom, pri čemer se opira na okoliščine, ki so bile ob strelu znane: razdaljo in kot do gola, del telesa oziroma način izvedbe, vrsto predhodne akcije, položaj branilcev in vratarja ter podatek, ali je bil strel izveden pod pritiskom. Vrednost xG je torej ocena, kako pogosto bi povprečen igralec iz enake situacije dosegel zadetek: strel z bele točke ima približno 0,76 xG, strel z roba kazenskega prostora pa pogosto manj kot 0,05 xG.

Slika 3 prikazuje, kako verjetnost zadetka pada z oddaljenostjo in manjšim kotom. Območja z enako verjetnostjo tvorijo krivulje okrog gola, pri čemer je razlika med strelom iz osrčja kazenskega prostora in strelom z njegovega roba bistveno večja, kot se zdi na prvi pogled.

Vsota vrednosti xG čez tekmo je precej bolj stabilen pokazatelj kakovosti ustvarjenih priložnosti kot število doseženih zadetkov, saj ni odvisna od naključja pri zaključku. Prav zato xG omogoča ločevanje med kakovostjo igre in trenutno učinkovitostjo: ekipa, ki dosega manj zadetkov, kot znaša njena vrednost xG, praviloma ni igrala slabo, temveč je imela slabši izkoristek, kar se v daljšem obdobju običajno izravna. Enaka logika velja v obrambni fazi, kjer vrednost xG nasprotnika pove, kako nevarne priložnosti je ekipa dopustila.

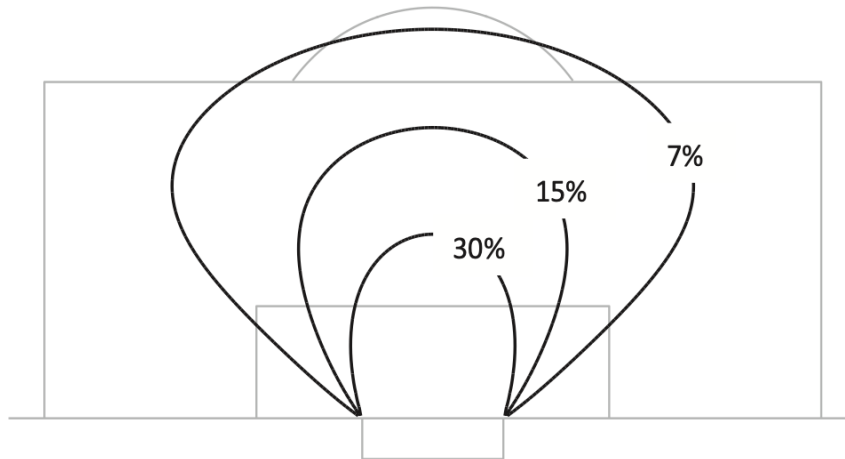
Pričakovane asistenc (angl. expected assists, xA) na enak način ovrednotijo podajo pred strelom in omogočajo, da ustvarjalnost podajalca ločimo od uspešnosti soigralca pri zaključku. Igralec ima lahko visoko vrednost xA brez ene same asistenc, kar pomeni, da redno ustvarja priložnosti, ki jih soigralci ne izkoristijo, ali obratno.

Pričakovana nevarnost (angl. expected threat, xT) razširi vrednotenje strelav in asistenc na napredovanje žoge. Vsaki podaji ali vodenju pripiše spremembo verjetnosti, da bo posest pripeljala do zadetka, in s tem omogoči

merjenje progresivnosti [6]. Metrika pokaže tudi primere, ko ekipa žogo redno prenaša v nevarna območja, a si iz tega ne ustvari priložnosti.

Isto logiko posploši pristop VAEP, ki vsaki akciji na igrišču pripiše spremembo verjetnosti, da bo ekipa v kratkem dosegla oziroma prejela zadetek, in z enotnim merilom ovrednoti tako napadalne kot obrambne akcije [7].

Poleg vrednotenja akcij so v uporabi tudi metrike, ki opisujejo **slog igre**. Metrika PPDA, ki meri število nasprotnikovih podaj na eno obrambno akcijo, na primer pove, kako visoko in kako agresivno ekipa pritiska. Na strani sledilnih podatkov med osnovne metrike sodijo kompaktnost ekipe, višina obrambne linije, razdalje med linijami in nadzor prostora.



Slika 3: Model pričakovanih zadetkov – krivulje enake verjetnosti zadetka glede na oddaljenost in kot strela.

3 Kako klubi uporabljajo podatke

Po naših izkušnjah pri sodelovanju s klubi različnih velikosti in ravni tekmovanja se uporaba podatkov osredotoča na štiri področja, ki se razlikujejo po časovnem okviru in po tem, kdo je naročnik analize.

Analiza lastne ekipe je najbolj dolgoročna in najtesneje povezana z igralnim modelom. Njeno izhodišče ni vprašanje, kaj je v podatkih razvidno, temveč kaj klub sploh želi meriti: kakšen slog igre zasleduje, koliko posesti želi imeti, kako visoko želi pritiskati in kako želi izvajati prehode med posameznimi igralnimi fazami. Šele iz teh odgovorov je mogoče izpeljati smiselne ključne kazalnike uspešnosti. Bistveno je razlikovanje med metrikami, ki merijo slog, in tistimi, ki merijo rezultat: PPDA ne pove, ali ekipa pritiska dobro, temveč le, ali sploh pritiska in kako visoko. Kadar se ti dve vrsti metrik zamešata, se v ocenjevanje ekipe vgradi napačna povratna informacija.

Analiza tekme in nasprotnika je najkrajši in najbolj ponovljiv cikel, saj se ponovi vsakih nekaj dni. Pri analizi nasprotnika je prvo vprašanje, katere njegove tekme sploh pregledati; smiselne so predvsem tiste proti podobnim nasprotnikom in tiste, ki napovedujejo, kako bo nasprotnik igral prav proti naši ekipi. Zanima nas, kako gradi igro in po katerih conah prihaja v napadalno tretjino, kje si ustvarja priložnosti in kje so si priložnosti proti njemu ustvarili drugi.

Tehnično skavtiranje podatke uporablja najprej za zožitev trga. Izhodišče je profil, ki opredeli, katere kakovosti so za določeno vlogo bolj in katere manj pomembne, in šele nato sledi iskanje igralcev, ki temu profilu ustrezajo. Ključno vprašanje ni le, kako dober je igralec, temveč v kakšni ekipi in v kakšnem slogu je igral doslej ter kako bi se njegove kakovosti prenesle v našo ekipo in v naše prvenstvo.

Vsem področjem je skupno, da podatki sami po sebi ne dajo odgovora. Dajo ga šele v kombinaciji z vprašanjem, ki ga postavi strokovni štab, in s kontekstom, ki ga prispeva nogometno znanje.

4 Od igralnega modela do metrik

Izhodišče našega dela ni bil nabor razpoložljivih podatkov, temveč igralni model ekipe. Strokovni štab je imel jasno predstavo o tem, kako naj ekipa deluje, in to predstavo je znal ubesediti v nogometnem jeziku. Naloga podatkovnega dela ni bila predlagati nove koncepte, temveč obstoječe taktične principe prevesti v merljivo obliko, ki jo je mogoče spremljati skozi čas.

Ta prevod poteka v več korakih. Najprej princip skupaj s trenerji razgradimo na merljive elemente: kaj natančno naj bi bilo vidno na igrišču, ko ekipa princip izvaja pravilno, in kaj, ko ga ne. Nato določimo, v katerih fazah igre je princip sploh relevanten, saj je vsaka obrambna metrika smiselna le v ustreznem kontekstu posesti in položaja žoge. Šele nato izberemo podatkovni vir in opredelimo konkreten izračun.

Ključno pri tem je, da trenerji in analitiki niso samo zunanji opazovalci, temveč so aktivno vključeni v proces oblikovanja metrik in modelov. Te morajo biti povezane z igralnim modelom, cilji ekipe in konkretnimi vprašanji strokovnega štaba. Vsako vmesno različico smo predstavili strokovnemu štabu na konkretnih posnetkih tekem in preverili, ali se vrednost metrike ujema s trenersko presojo iste situacije. Kadar se ni, to praviloma ni pomenilo, da se trener moti, temveč da opredelitev metrike še ne zajema nečesa, kar je za presojo bistveno. Takšen iterativen postopek je bistveno povečal verjetnost, da bo metrika pozneje dejansko uporabljena, saj so jo člani štaba razumeli in ji zaupali.

Drugi pomemben dejavnik je hitrost. Ker se tekme igrajo vsakih nekaj dni, mora biti celotna pot od prejema podatkov do pripravljenega pregleda dovolj kratka, da rezultati pridejo do trenerjev, preden se pozornost preusmeri na naslednjega nasprotnika.

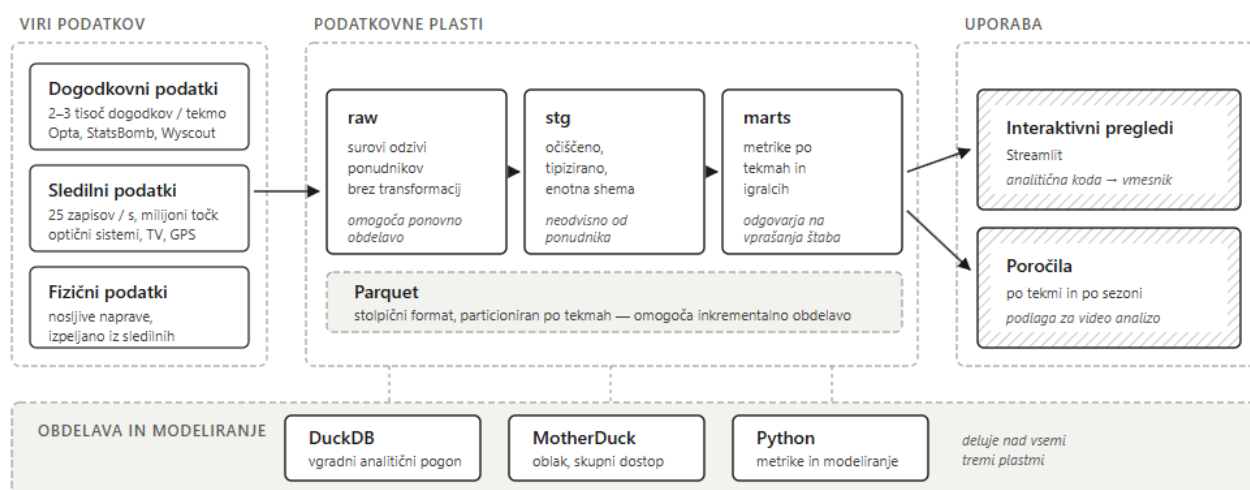
5 Podatkovna arhitektura

Za obdelavo, transformacijo in modeliranje podatkov uporabljamo kombinacijo sistemov DuckDB in MotherDuck ter programskega jezika Python. Izbira je posledica dveh zahtev: obsega sledilnih podatkov, ki na ravni sezone hitro preseže milijardo vrstic, in potrebe po hitri iteraciji, opisani v prejšnjem poglavju.

Sorodne arhitekturne vzorce so na tej konferenci obravnavali že drugi avtorji. Nered in Jeraj sta predstavila podatkovno platformo za zajem biosenzoričnih podatkov v realnem času, ki obravnava podoben razred problemov na področju športa in zdravja [8]. Sestak in Turkanović sta obravnavala nadgradnjo podatkovnih jezer s formatom Delta Lake in razlike med odprtimi formati za shranjevanje v podatkovnem kolišču [9], Heričko s soavtorji pa vpeljavajo tabelaričnih tokov v podatkovno arhitekturo [10]. Naša rešitev je v primerjavi s temi pristopi namerno preprostejša: ker gre za analitično obdelavo omejenega števila virov brez zahtev po sprotni obdelavi v realnem času, se izognemo porazdeljenim okvirjem in ostanemo pri vgrajenem pogonu nad datotekami v objektni shrambi.

DuckDB služi kot učinkovit analitični SQL-pogon za lokalno obdelavo večjih količin podatkov. Gre za vgrajeni podatkovni sistem, ki poizvedbe izvaja v procesu same aplikacije in je zasnovan za stolpčno analitično obdelavo [11], zaradi česar omogoča neposredno poizvedovanje po datotekah v formatu Parquet brez predhodnega nalaganja v ločen strežnik. MotherDuck isti pogon razširja v oblako okolje za shranjevanje, deljenje in izvajanje analiz, kar pomeni, da analitiki in drugi člani štaba dostopajo do istih pripravljenih tabel brez lokalne namestitve [12]. Python uporabljamo za izračun naprednejših metrik, modeliranje ter pripravo vizualizacij in poročil. Takšna arhitektura je omogočila hitro iteracijo od surovih prostorsko-časovnih podatkov do uporabnih vpogledov brez potrebe po kompleksni klasični podatkovni infrastrukturi.

Podatke organiziramo v tri plasti. V plasti *raw* hranimo surove odzive ponudnikov v izvorni obliki, brez transformacij, kar omogoča ponovno izvedbo celotne obdelave, kadar se opredelitev metrike spremeni. V plasti *stg* so podatki očiščeni, tipizirani in normalizirani v enotno shemo, neodvisno od ponudnika; ta korak je ključen prav zaradi razlik med ponudniki, opisanih v poglavju 2.5. V plasti *marts* so pripravljene tabele, ki neposredno odgovarjajo na vprašanja strokovnega štaba, na primer vrednosti metrik po tekmah in igralcih. Vsaka plast je shranjena v stolpcnem formatu Parquet [13] in razdeljena na naravne particije, običajno po tekmah, kar omogoča inkrementalno obdelavo: ob novi tekmi ni treba ponovno obdelati celotne sezone. Celotno arhitekturo prikazuje slika 4.



Slika 4: Arhitektura podatkovne rešitve od virov podatkov prek podatkovnih plasti do uporabe.

Rezultate predstavljamo prek interaktivnih pregledov v Streamlitu in samodejno pripravljenih poročil v formatu PDF. Streamlit omogoča, da se analitična koda v Pythonu neposredno pretvori v uporabniški vmesnik brez ločenega razvoja spletne aplikacije [14]. Poročila pripravljamo v dveh oblikah: poročilo o posamezni tekmi, namenjeno pripravi na videoanalizo v dneh po tekmi, in sezonsko poročilo, ki prikazuje razvoj metrik v daljšem obdobju. To je bistveno skrajšalo pot od nove ideje do pregleda, ki si ga lahko ogleda strokovni štab, in omogočilo, da so vmesne različice metrik nastajale v tesnem sodelovanju s trenerji.

Med razvojem se je pokazalo, da je prav ločitev plasti *raw* in *stg* tista, ki omogoča pogoste spremembe opredelitev metrik, saj lahko izračune ponovimo nad že zajetimi podatki, ne da bi jih morali znova pridobivati od ponudnika.

6 Študija primera: obrambna kompaktnost

Osrednji del prispevka predstavlja praktičen primer uporabe podatkov pri razvoju metrike, ki povezuje igralni model ekipe z njenimi taktičnimi principi. Sodelovali smo s strokovnim štabom kluba angleške Premier lige v okviru nekaj tekmovalnih sezon; primer, ki ga opisujemo v nadaljevanju, izhaja iz ene izmed njih.

6.1 Taktični princip

Strokovni štab je imel jasno predstavo o tem, kako želi, da ekipa deluje v obrambi, predvsem kako povezano mora biti gibanje obrambnih igralcev glede na položaj žoge in nasprotnika. Princip je bil ubeseden v obliki preprostega pravila: branilci naj v fazi brez žoge od žoge nikoli ne bodo oddaljeni več kot tri navpične pasove igrišča (angl. lanes). Pravilo je bilo igralcem razumljivo in ga je bilo preprosto posredovati, hkrati pa je bilo dovolj natančno, da ga je bilo mogoče prevesti v merljivo obliko. To nogometno idejo smo prevedli v podatkovni model in oblikovali metrike, s katerimi je bilo mogoče spremljati, ali ekipa dogovorjene principe dejansko izvaja na igrišču, kje prihaja do odstopanj in ali se skozi čas približuje zelenemu načinu igre.

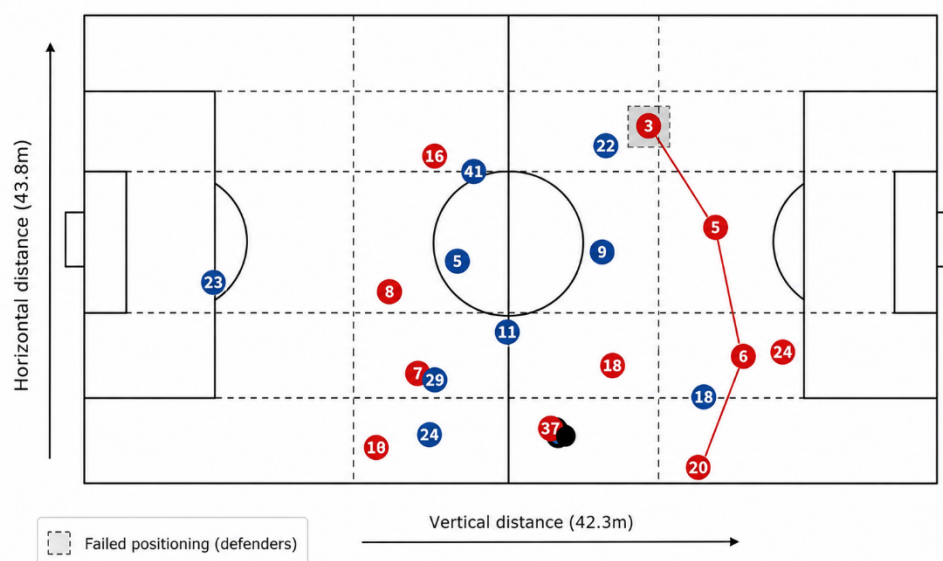
Prevelika oddaljenost obrambne skupine od žoge pomeni, da se odpre prostor, ki ga lahko nasprotnik izkoristi za prodor ali za podajo za hrbet obrambne linije, hkrati pa oteži medsebojno pokrivanje branilcev.

Trenerjeva zahteva se dobro ujema z ugotovitvami raziskav. Analiza sledilnih podatkov nemške Bundeslige je pokazala, da je bila skupina petih branilcev, ki so bili najbližje žogi, ob uspešnih obrambnih akcijah bolj kompaktna kot ob neuspešnih, medtem ko kompaktnost celotne ekipe za odvzem žoge ni bila pomembna [15]. Merjenje kompaktnosti glede na položaj žoge in ne na celotno ekipo je torej utemeljeno tudi z vidika raziskav, kar je bil dodaten argument pri izbiri definicije metrike.

6.2. Definicija metrike

Princip smo operacionalizirali kot **horizontalno kompaktnost obrambne skupine**, merjeno v metrih in izračunano izključno v fazah, ko ekipa ni imela posesti žoge. Metrika meri, koliko je obrambna skupina raztegnjena po širini igrišča, torej med najbolj oddaljenima bočnima branilcema; nižja vrednost pomeni bolj kompaktno postavitvev. Prav ta razdalja neposredno ustreza trenerjevemu pravilu o treh navpičnih pasovih, saj se pasovi merijo po širini igrišča. Izračunali smo jo iz sledilnih podatkov, ki zagotavljajo položaje vseh igralcev skozi celotno tekmo [16].

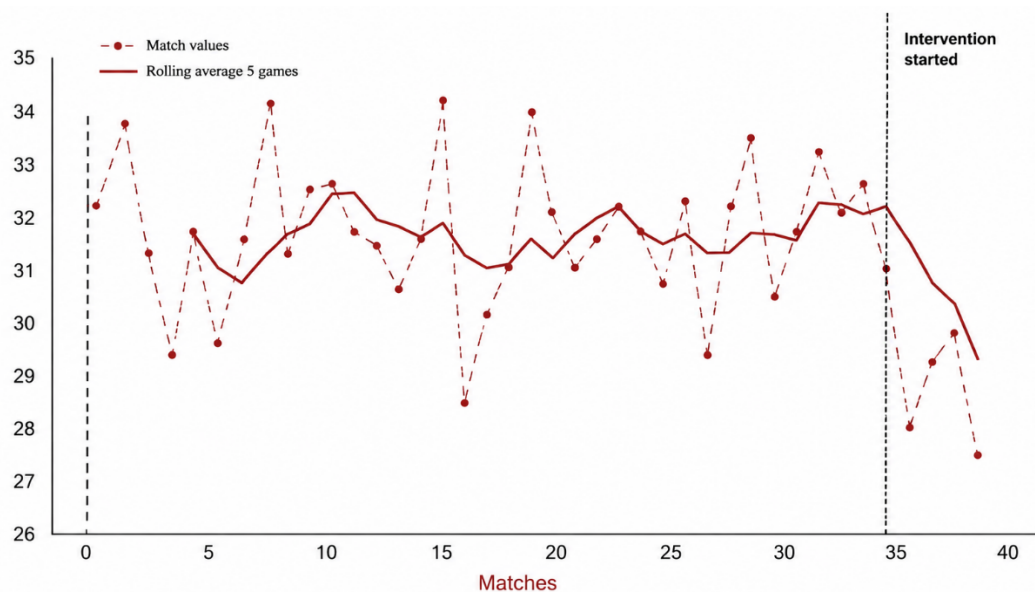
Poleg agregirane vrednosti smo razvili tudi diagnostični pogled, ki posamezne trenutke označi kot primere neustrezne postavitve (angl. failed positioning). Za vsak tak trenutek prikažemo postavitve obeh ekip na igrišču skupaj s horizontalno in navpično razdaljo obrambne skupine, kar omogoča, da se agregirana številka takoj poveže s konkretno situacijo na posnetku. Prav ta povezava med vrednostjo in posnetkom se je v pogovorih s štabom izkazala za bistveno, saj trenerji ne delajo z metrikami, temveč z nogometnimi situacijami.



Slika 5: Prikaz trenutka z neustrezno postavitvijo obrambne skupine ter horizontalno in navpično razdaljo med igralci.

6.3. Spremljanje skozi sezono

Metriko smo po vsaki tekmi izračunali in jo prikazali v kontekstu celotne sezone. Poleg vrednosti posamezne tekme smo prikazovali drseče povprečje petih tekem, saj posamična tekma niha glede na nasprotnika in potek srečanja, trajnejši vzorec pa je razviden šele iz daljšega obdobja. Prav ločevanje med posamičnim odstopanjem in trajnejšim vzorcem je bilo za strokovni štab najbolj uporabno.



Slika 6: Sezonski trend horizontalne kompaktnosti v fazi brez žoge z drsečim povprečjem petih tekem in označeno točko intervencije.

6.4. Ugotovitev in intervencija

Spremljanje postavitev posameznih igralcev je pokazalo, da je bil eden od branilcev v fazah brez žoge sistematično postavljen predaleč od žoge glede na preostanek obrambne skupine. Odstopanje ni bilo posledica posamične napake, temveč se je ponavljalo skozi več tekem, kar je bilo v podatkih razvidno kot vztrajno povečana horizontalna kompaktnost v določenih fazah igre.

Strokovni štab je ugotovitev uporabil neposredno v delu z igralcem. Pri videoanalizi so mu prikazali konkretne situacije iz diagnostičnega pogleda in mu pojasnili, kakšna postavitev se od njega pričakuje glede na položaj žoge. Po tej intervenciji se je vrednost metrike opazno znižala (slika 6). Ekipa je v fazi brez žoge torej postala bolj kompaktna in bližje želenemu izvajanju dogovorjenega principa.

Ta primer po našem mnenju najbolje ponazarja vlogo podatkov v opisanem procesu. Metrika sama po sebi ni odkrila ničesar, česar trenerji ne bi mogli opaziti tudi z videoanalizo. Njena vrednost je bila v tem, da je vzorec pokazala sistematično in količinsko, na več tekmah hkrati, in s tem omogočila, da je bil pogovor z igralcem podprt s konkretnim dokazom namesto s splošnim vtisom. Prav tako je omogočila, da je bil učinek intervencije po njeni izvedbi tudi izmerjen. K temu je treba dodati še obseg: metriko izračunamo za vse tekme sezone hkrati, medtem ko bi ročno pregledovanje posnetkov za enak nabor zahtevalo več ur dela, zato takšna odstopanja prej opazimo.

7 Zaključek

V prispevku smo predstavili celotno pot od surovih nogometnih podatkov do konkretne odločitve strokovnega štaba. Opisali smo glavne sklope nogometnih podatkov, razlike med dogodkovnimi in sledilnimi podatki, ponudnike in osnovne metrike, področja, na katerih klubi podatke uporabljajo, postopek prevajanja taktičnih principov v merljive metrike, podatkovno arhitekturo, zgrajeno na kombinaciji DuckDB, MotherDuck in Pythona, ter študijo primera, v kateri smo trenerjev princip obrambne kompaktnosti operacionalizirali kot horizontalno kompaktnost obrambne skupine v fazah brez žoge.

Izbrana arhitektura se je izkazala za primerno. Ločitev podatkov v plasti *raw*, *stg* in *marts* je omogočila, da smo opredelitev metrike med razvojem večkrat spremenili in izračune ponovili nad že zajetimi podatki, particioniranje po tekmah pa je omogočilo inkrementalno obdelavo v časovnem okviru, ki ga narekuje tekmovalni ritem. Za obseg

podatkov, s katerim smo delali, klasična podatkovna infrastruktura ne bi prinesla bistvenih prednosti, prinesla pa bi znatno večjo kompleksnost vzpostavitve in vzdrževanja.

S prispevkom želimo pokazati, da podatki v nogometu ne nadomeščajo trenerjevega znanja, temveč ga dopolnjujejo. Njihova vrednost je v tem, da taktične ideje naredijo merljive, preverljive in uporabne v vsakodnevnem delu. V predstavljenem primeru je takšen pristop strokovnemu štabu omogočil spremljanje izvajanja obrambnih principov, prepoznavanje odstopanj in pripravo bolj ciljno usmerjenih sprememb, utemeljenih na podatkih. Prednost podatkov v primerjavi zgolj z videoanalizo ni v tem, da bi podatki pokazali kaj, česar trener ne bi videl, temveč v obsegu in hitrosti: isto vprašanje lahko preverimo na vseh tekmah sezone hkrati v realnem času.

Literatura

- [1] R. Rein in D. Memmert, "Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science," SpringerPlus, let. 5, št. 1410, 2016, doi: <https://doi.org/10.1186/s40064-016-3108-2>.
- [2] F. R. Goes idr., "Unlocking the potential of big data to support tactical performance analysis in professional soccer: A systematic review," European Journal of Sport Science, let. 21, št. 4, str. 481–496, 2021, doi: <https://doi.org/10.1080/17461391.2020.1747552>.
- [3] B. Low idr., "A Systematic Review of Collective Tactical Behaviours in Football Using Positional Data," Sports Medicine, let. 50, str. 343–385, 2020, doi: <https://doi.org/10.1007/s40279-019-01194-7>.
- [4] L. Forcher, S. Altmann, L. Forcher, D. Jekauc in M. Kempe, "The use of player tracking data to analyze defensive play in professional soccer – A scoping review," International Journal of Sports Science and Coaching, 2022, doi: <https://doi.org/10.1177/17479541221075734>.
- [5] J. Fernández in L. Bornn, "Wide Open Spaces: A statistical technique for measuring space creation in professional soccer," v MIT Sloan Sports Analytics Conference, 2018.
- [6] K. Singh, "Introducing Expected Threat (xT)," 2019. [Na spletu]. Dostopno na: <https://karun.in/blog/expected-threat.html>, obiskano [julij 2026].
- [7] T. Decroos, L. Bransen, J. Van Haaren in J. Davis, "Actions Speak Louder than Goals: Valuing Player Actions in Soccer," v Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, str. 1851–1861, doi: <https://doi.org/10.1145/3292500.3330758>.
- [8] U. Nered in J. Jeraj, "Podatki v utripu srca: podatkovna platforma za zajem biosenzoričnih podatkov v realnem času," v Sodobne informacijske tehnologije in storitve OTS 2023: Zbornik 26. konference, Maribor, 2023.
- [9] M. Sestak in M. Turkanović, "Nadgradnja podatkovnih jezer s pomočjo formata za shranjevanje Delta Lake," v Sodobne informacijske tehnologije in storitve OTS 2023: Zbornik 26. konference, Maribor, 2023, doi: <https://doi.org/10.18690/um.feri.8.2023.21>.
- [10] T. Heričko, S. Brdnik in M. Turkanović, "Vpeljava tabelaričnih tokov v podatkovno arhitekturo," v Sodobne informacijske tehnologije in storitve OTS 2025: Zbornik 28. konference, Maribor, 2025, str. 13–24.
- [11] M. Raasveldt in H. Mühleisen, "DuckDB: an Embeddable Analytical Database," v Proceedings of the 2019 International Conference on Management of Data (SIGMOD), 2019, str. 1981–1984, doi: <https://doi.org/10.1145/3299869.3320212>.
- [12] motherduck.com, MotherDuck Documentation, obiskano 15. 2. 2026.
- [13] parquet.apache.org, Apache Parquet Documentation, obiskano 13. 2. 2026.
- [14] streamlit.io, Streamlit Documentation, obiskano 15. 2. 2026.
- [15] L. Forcher, L. Forcher, S. Altmann, D. Jekauc in M. Kempe, "Is a compact organization important for defensive success in elite soccer? – Analysis based on player tracking data," International Journal of Sports Science and Coaching, let. 19, št. 2, str. 757–768, 2024, doi: <https://doi.org/10.1177/17479541231172695>.
- [16] statsperform.com, Opta Tracking Data, obiskano 10. 2. 2026.

OTS 2026 Sodobne informacijske tehnologije in storitve

Zbornik devetindvajsete konference

Luka Pavlič, Tina Beranič, Marjan Heričko (ur.)

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija

luka.pavlic@um.si, tina.beranic@um.si, marjan.hericko@um.si

V zborniku devetindvajsete konference OTS 2026 so objavljeni prispevki strokovnjakov s področja informatike, v katerih so predstavljena nova spoznanja in trendi razvoja, vpeljave, prilagajanja ter upravljanja informacijskih rešitev, kot tudi konkretni uspešni pristopi in dobre prakse. Prispevki naslavljajo področja sodobnih arhitekturnih izzivov, klasične, generativne in globoke umetne inteligence, sodobnih spletnih ali mobilnih uporabniških vmesnikov, kot tudi tradicionalnih, brezstrežniških in decentraliziranih zalednih sistemov v oblaku. Tematike prispevkov obsegajo tudi kibernetsko varnost, zagotavljanje kakovosti, sodobne pristope k hranjenju in obdelavi podatkov, ter predpise in standardizacijo, ki smo ji informatiki izpostavljeni vsakodnevno. Prispevki tako še naprej omogočajo boljšo povezanost IT strokovnjakov, informatikov, arhitektov in razvijalcev IT rešitev in storitev, ter akademske sfere in gospodarstva.

Ključne besede:

informatika in informacijske tehnologije,
programsko inženirstvo,
informacijski sistemi in rešitve,
digitalna preobrazba,
razvoj mobilnih in spletnih rešitev,
arhitekture v oblaku,
podatkovne tehnologije,
poslovna inteligenca,
umetna inteligenca in strojno učenje,
inteligentni agenti,
obdelava velepodatkov in podatkovnih tokov,
metode agilnega razvoja,
tehnologije veriženja blokov,
kibernetska varnost.

GENERALNI POKROVITELJ



POKROVITELJI



MEDIJSKI POKROVITELJ





Tehnologije, ki povezujejo fizični in digitalni svet

Pametne naprave povezujemo v zanesljive ekosisteme, ki zajemajo in obdelujejo podatke ter omogočajo avtomatizacijo in odzivanje v realnem času.

Razvijamo rešitve z umetno in generativno inteligenco, ki povezujejo podatke, avtomatizirajo procese ter omogočajo napovedovanje in podporo pri odločanju.

Razvijamo digitalne dvojčke, ki povezujejo podatke, procese in umetno inteligenco ter omogočajo spremljanje, simuliranje in napovedovanje realnih sistemov.

Oblikujemo in razvijamo napredne spletne, mobilne in namizne produkte, ki združujejo uporabniško izkušnjo, zanesljivo tehnologijo in dolgoročno poslovno vrednost.

IoT & CONNECTED SYSTEMS
Povezani sistemi v realnem času.

AI & INTELLIGENT SYSTEMS
Od podatkov do avtonomnih odločitev.

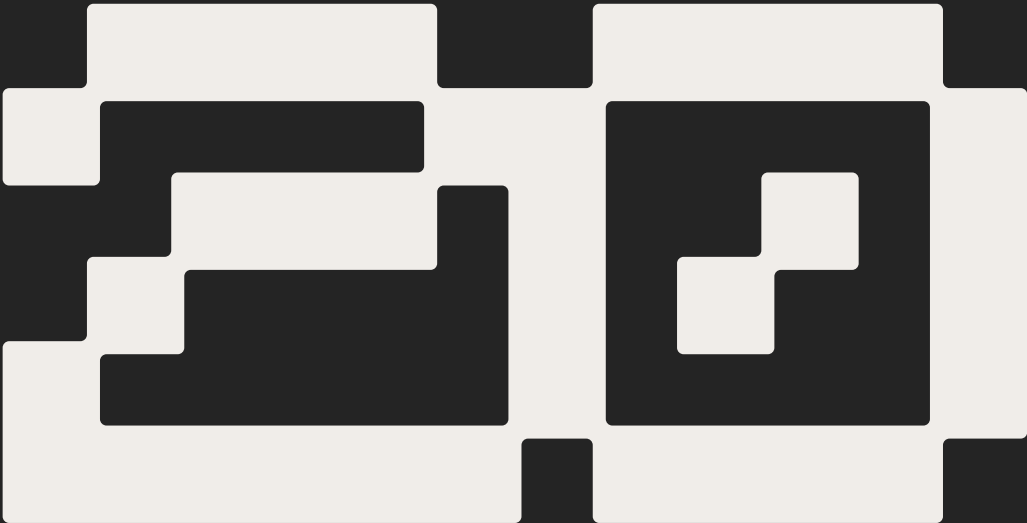
DIGITAL TWINS
Simuliramo. Napovedujemo.

DIGITAL PRODUCTS
Od ideje do delujoče rešitve.

inova.si



INOVA



founded 2006
uptime 20 years
status [RUNNING]
next_target ∞



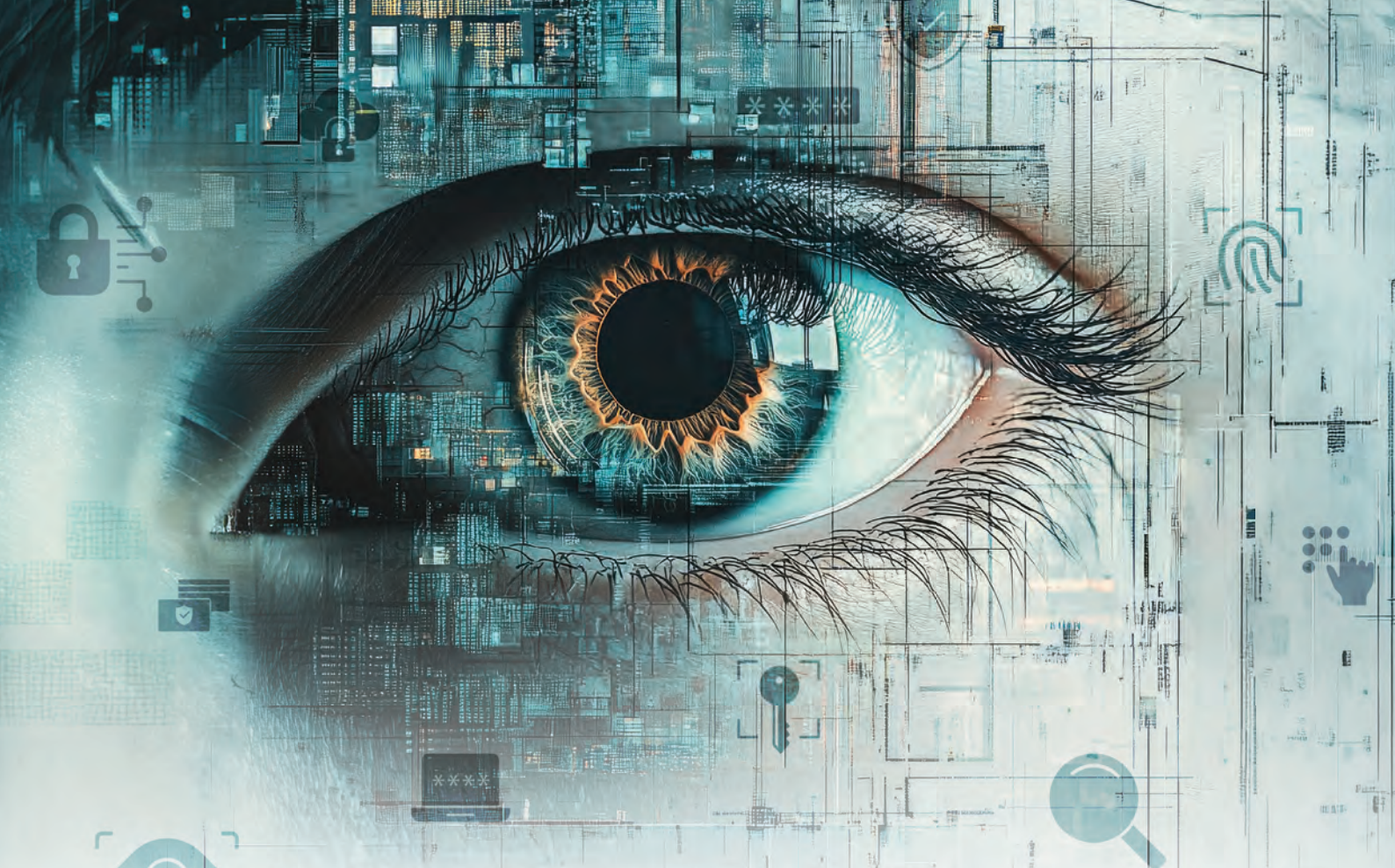
Dravske elektrarne Maribor

Skupina  hse 25^{let}

Z energijo narave ustvarjamo zeleno prihodnost.

Odgovorni do okolja,
predani družbi.





LABORATORIJ ZA KIBERNETSKO VARNOST IN DIGITALNO FORENZIKO



Certifikata laboratorija:



ISO/IEC 27001:2022
Sistemi vodenja informacijske varnosti



ISO 9001:2015
Sistemi vodenja kakovosti

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.

INFORMATIKA

pravi partner za vas



**LASTNO
ZNANJE**

+



**IZBRANI
PARTNERJI**

+



**DIALOG Z
NAROČNIKOM**

=



**OPTIMALEN
REZULTAT ZA VSE**

Razvoj rešitev po meri
*z uporabo najnovejših
tehnologij in varnostnih
standardov*

**Uvajanje standardnih
programskih rešitev**
za optimizacijo poslovanja

Podatkovna analitika
*za pravilno in
pravočasno odločanje*

Varnostni operativni center
za 24/7 kibernetko varnost

Gostovanje rešitev
z E2E podporo

Informatika je razvojno naravnano IKT podjetje in cenjen partner v slovenskem elektroenergetskem prostoru. Kontaktirajte nas in skupaj bomo našli rešitev za vaše izzive.



info@informatika.si |



+386 2 707 10 00

TELEKOMUNIKACIJE · ENERGETIKA · INTEGRACIJA · AI

Poslovne aplikacije in integracije

Preverjene tehnologije za sisteme, ki morajo delovati brez prekinitev.



Akruma.SI — spletno računovodstvo
za mikro do srednje velika podjetja — akruma.si

LASTEN PRODUKT

REFERENCE · OD 2011

A1 Slovenija · Telemach Slovenija · Telekom Slovenije · ELES

ROBOTIKA IN AI

Razvoj na štirinožnem robotu Unitree Go2
NVIDIA DGX Spark gruča — lokalno AI okolje, podatki ostanejo pri vas



ODPRTA KODA

Red Hat OpenShift · Java · Liferay · Podman
Apache Camel · Apache Karaf · Quarkus

»Imate sistem, ki mora delovati?«

Pogovorimo se — poskenirajte ali pišite.



mibesis.com
spoznajte nas



akruma.si
oglejte si produkt



info@mibesis.com
informacije in povpraševanje

INOVACIJE, KI SPREMINJAJO SVETOVNE TRGE

Mikropis Holding je evropsko podjetje z 38-letno tradicijo uvajanja inovativnih rešitev na svetovne trge. Specializirani za poslovne rešitve, smo znani in dolgoročni partner mnogih pomembnih podjetij. Ponujamo prilagojene rešitve za stranke na področjih trgovine, gostinstva, samopostrežbe, financ, e-marketinga, upravljanja človeških virov, zdravja in dobrega počutja. Prav tako smo krovno podjetje za globalno rešitev 24alife, usmerjeno v preprečevanje bolezni in dobro počutje, ter Connected mHealth, digitalno rešitev za virtualno rehabilitacijo.



Ustvarjamo svet, kjer se ljudje povezujejo, delijo in kolektivno skrbijo za svoje zdravje in dobro počutje. Znanstvena dognanja na področju zdravja in dobrega počutja, so razvita v sodelovanju s kliniko Mayo Clinic, kar zagotavlja visoko raven strokovnosti in veljavnosti.



Razvoj mobilnih aplikacij, ki izboljšujejo zdravje in dobro počutje uporabnikov.



Razvoj naprednih platform, ki optimizirajo zdravje, in rehabilitacijo



Razvoj prilagojenih rešitev za trgovine, gostinstvo, samopostrežbo, finance...

TE ZANIMA VEČ? PRIDRUŽI SE NAM.

Za več informacij nam pišite na: zaposlitev@mikropis.si

MSG PLAUT UAP

ZAVAROVALNIŠTVO. NADGRAJENO.

Zanesljiv temelj sodobnega zavarovalništva

Več kot 25 let izkušenj pri digitalizaciji poslovnih procesov, razvoju informacijskih sistemov ter implementaciji rešitev za zavarovalnice.

STORITVE



Informacijski sistemi

razvoj prilagojenih poslovnih rešitev



Mobilne aplikacije

iOS in Android rešitve za uporabnike in agente



Rešitve v oblaku

mikrostoritvena arhitektura (Docker)



Digitalizacija procesov

avtomatizacija in optimizacija poslovanja



API integracije

povezovanje sistemov z odprto arhitekturo



Migracija podatkov

zanesljiv prenos in integracija podatkov

Cloud ali lokalna namestitvev

Visoka stopnja avtomatizacije

Odprta arhitektura (API)

Celovito upravljanje podatkov

ZAČNIMO PROJEKT

+386 2 2356200
andrej.kline@msg-plaut.com
www.msg-plaut-uap.com



THE HASHRATE ● MARKETPLACE

*Vodilni ponudnik platforme
za rudarjenje kriptovalut*

www.nicehash.com



NOVUM•RGI CODE•Dare

We embrace challenges with
courage and determination.

Z inženirsko natančnostjo

gradimo digitalno prihodnost.

Združujemo strokovno znanje,
inovacije in najnovejše tehnologije.



Spoznajte več o
nas in področjih

www.src.si



Ljubljana-Naklo-Maribor-Beograd

SRC MB, Gosposvetska cesta 29, 2000 Maribor
+386 1 600 7000, www.src.si



RAZVIJAMO ŽE OD LETA 2007

V Mariboru razvijamo lastna produkta, s katerima podpiramo podjetja v telekomunikacijah, energetiki, e-mobilnosti ter ponudnike SaaS storitev.

50+ LJUDI, ENA GLOBALNA EKIPA

Naša mednarodna ekipa povezuje razvojno, produktno in poslovno znanje iz Evrope, Azije in Severne Amerike.

100+ PODJETIJ NA 6 CELINAH

Naša produkta Tridens Monetization in Tridens EV Charge podpirata podjetja in njihove uporabnike po vsem svetu.

OD PRAKSE DO KARIERE

Pri nas lahko opravljaš prakso, se vključiš v delo na resničnih produktih in ob podpori izkušene ekipe pridobiš dragocene izkušnje. Številni naši sodelavci so svojo karierno pot v Tridensu začeli prav na praksi.



www.tridenstechnology.com

 **tridens**
technology

NAROČNIŠKA AKCIJA

NAROČI SE NA REVIGO

Računalniške novice

www.racunalniske-novice.com

AKCIJSKA CENA

OB LETNI NAROČNINI Z DARILOM



Letna naročnina na revijo Računalniške novice
(21 števil) **skupaj z darilom** za samo **33,50 €**

Akcija velja do razprodaje zalog! Ponudba NE velja za podaljšanje naročnine oz. za naročnike, ki so že bili naročeni na revijo v zadnjih 2 letih.



IZBERI DARILO:
USB ključek,
PowerBank
ali **Bluetooth**
slušalke



USB ključ Apacer AH355
128 GB 3.2



PowerBank Joyroom
JR-PBF19 10 000mAh



Bluetooth slušalke Joyroom
ANC JR-FN3



NAROČI SE ZDAJ

T: 01 620 88 00

M: narocnine@stromboli.si

GENERALNI POKROVITELJ



POKROVITELJI



MEDIJSKI POKROVITELJ



www.racunalniske-novice.com



INŠTITUT ZA
INFORMATIKO

ISBN 978-961-299-188-3



9 789612 991883



Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Podatkovne tehnologije

Varnost
Kriptografija
Web 3

Strojno učenje
Poslovna analitika