Univerzitetna založba
Univerze v Mariboru

28. konferenca sodobne informacijske tehnologije in storitve

ots 2025

Oblačne rešitve
Mobilne aplikacije
Spletne aplikacije
Agilni razvoj
Optimizacija
Mikrostoritve
Strojno učenje
Podatki
Kakovost
Zabojniki

Zbornik prispevkov

Uredniki
Luka Pavlič
Tina Beranič
Marjan Heričko

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

OTS 2025

Sodobne informacijske tehnologije in storitve

Zbornik 28. konference

Uredniki
Luka Pavlič
Tina Beranič
Marjan Heričko

September 2025

Naslov / *Title*
OTS 2025 Sodobne informacijske tehnologije in storitve
OTS 2025 Advanced Information Technologies and Services

Podnaslov / *Subtitle*
Zbornik 28. konference
Conference Proceedings of the 28th Conference

Uredniki / *Editors*
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 / *Technical editors*
Luka Pavlič
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Jan Perša
(Univerza v Mariboru, Univerzitetna knjižnica Maribor)

Oblikovanje ovitka / *Cover designer*
Špela Čučko
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Grafika na ovitku / *Cover graphics*
OTS 2025, Čučko, 2025

Grafične priloge / *Graphics material*
Vsi viri so lastni, če ni navedeno drugače. Avtorji in Pavlič, Beranič, Heričko (uredniki), 2025

Konferenca / *Conference*
OTS 2025 Sodobne informacijske tehnologije in storitve

Kraj in datum / *Location and date*
Maribor, Slovenija, 3. in 4. september 2025

Programski odbor / *Program committee*
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 / *Organizing committee*
Tina Beranič (vodja), Luka Pavlič, Špela Čučko, Luka Četina, Lucija Brezočnik, Saša Brdnik, Mitja Gradišnik, Tjaša Heričko, Nika Jeršič, Katja Kous, Vasilka Saklamaeva, Nadica Uzunova Petrič, Lidija Vinceković (vsi Univerza v Mariboru, Slovenija).

Založnik / *Published by*
Univerza v Mariboru
Univerzitetna založba
Slomškov trg 15, 2000 Maribor, Slovenija
https://press.um.si, zalozba@um.si

Izdajatelj / *Issued by*
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 / *Edition*
Prva izdaja

Vrsta publikacije / *Publication type*
E-knjiga

Izdano / *Published at*
Maribor, Slovenija, september 2025

Dostopno na / *Available at*
https://press.um.si/index.php/ump/catalog/book/1000

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.946.5:004.7(082)

SODOBNE informacijske tehnologije in storitve (konferenca) (28 ; 2025 ; Maribor)
OTS 2025 [Elektronski vir] : sodobne informacijske tehnologije in storitve : zbornik 28. konference : [Maribor, 3. in 4. september 2025] / uredniki Luka Pavlič, Tina Beranič, Marjan Heričko. - 1. izd. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2025

Dostopno tudi na: https://press.um.si/index.php/ump/catalog/book/1000

ISBN 978-961-299-036-7 (WEB, pdf)
doi: 10.18690/um.feri.7.2025
COBISS.SI-ID 246681859

ISBN 978-961-299-036-7 (pdf)
978-961-299-037-4 (mehka vezava)
978-961-299-038-1 (USB ključ)

DOI https://doi.org/10.18690/um.feri.7.2025

Cena *Price* Brezplačni izvod

Odgovorna oseba založnika *For publisher* Prof. dr. Zdravko Kačič, rektor Univerze v Mariboru

Citiranje *Attribution* Pavlič, L., Beranič, T., Heričko, M. (ur.). (2025). *OTS 2025 Sodobne informacijske tehnologije in storitve: Zbornik 28. konference.* Univerza v Mariboru, Univerzitetna založba. doi: 10.18690/um.feri.7.2025

https://www.ots.si

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 2025 vsebuje zanimive prispevke, ki iz različnih zornih kotov osvetljujejo tehnične in organizacijske vidike razvoja, nadgradnje, prilagajanja in upravljanja informacijskih rešitev in storitev. Klasična, generativna in globoka umetna inteligenca je v zadnjem letu že postala stalnica IS/T projektov. 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, ter predpisi in standardizacija, ki smo ji informatiki izpostavljeni vsakodnevno.

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.

izr. prof. dr. Luka Pavlič	doc. dr. Tina Beranič	prof. dr. Marjan Heričko
vodja programskega odbora OTS 2025	vodja organizacijskega odbora OTS 2025	predsednik konference OTS 2025

KAZALO

Vloga MLOps pri zagotavljanju kakovosti napovednih modelov

Zala Lahovnik, Grega Vrbančič, Vili Podgorelec

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
zala.lahovnik1@um.si, grega.vrbancic@um.si, vili.podgorelec@um.si

Z razvojem umetne inteligence in strojnega učenja ter vse pogostejšo integracijo v programske rešitve, postaja eden izmed ključnih vidikov zagotavljanje zanesljivosti in kakovosti napovednih modelov strojnega učenja. Kljub napredku se številni projekti, ki vključujejo inteligentne komponente soočajo z neuspehi zaradi pomanjkljivega testiranja in neustrezne integracije v produkcijska okolja. Pogosti razlogi za neuspeh vključujejo slabo integracijo modelov v informacijske sisteme, pomanjkljivo testiranje in odsotnost sistematičnega nadzora kakovosti modelov. Ključno vlogo pri uspešnosti modelov ima kakovost vhodnih podatkov – napačni, nepopolni ali pristranski podatki zmanjšajo robustnost napovedi in posledično zaupanje uporabnikov. Pomemben izziv predstavlja tudi pojav spremembe porazdelitev vrednosti podatkov, ki lahko vodi v postopno degradacijo modela. Te izzive naslavlja MLOps pristop, ki združuje pristop DevOps z značilnostmi razvoja in upravljanja napovednih modelov strojnega učenja. MLOps omogoča avtomatizacijo, ponovljivost, transparentnost in nadzor celotnega življenjskega cikla napovednih modelov. Ključni gradniki vključujejo verzioniranje podatkov in modelov, validacijo, avtomatizirano ponovno učenje ter sprotno spremljanje uspešnosti delovanja v produkciji. V prispevku so ti vidiki osvetljeni skozi študijo primera Airly platforme, ki predstavlja celovit primer MLOps podpore.

Ključne besede:
MLOps,
zagotavljanje kakovosti,
napovedni modeli,
strojno učenje,
inteligentni sistemi.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.1

1 Uvod

V zadnjem desetletju je razvoj umetne inteligence (angl. Artificial Intelligence, AI) in strojnega učenja (angl. Machine Learning, ML) povzročil pomembne premike na številnih področjih, kot so medicina, industrija, finance in transport. Kljub temu pa številni AI/ML projekti v praksi ne dosegajo pričakovanih rezultatov in pogosto ne uspejo preiti iz raziskovalne faze v produkcijska okolja. Raziskave kažejo, da do 85 % projektov na področju umetne inteligence ne uspe doseči trajne implementacije ali želenih poslovnih učinkov. Glavni vzroki neuspeha so pogosto povezani z neustrezno integracijo modelov v obstoječe informacijske sisteme, pomanjkljivim testiranjem in odsotnostjo sistematičnega nadzora kakovosti modelov v realnem času [1].

Posebna značilnost inteligentnih sistemov je izrazit vpliv kakovosti vhodnih podatkov na učenje in posledično tudi na uspešnost napovedovanja modelov. Napačni, nepopolni ali pristranski podatki lahko pomembno poslabšajo uspešnost in robustnost napovednih modelov, kar vodi v nezanesljive rezultate in zmanjšano zaupanje uporabnikov [2]. Pogosto se modeli soočajo s tudi pojavom spremembe porazdelitve vrednosti podatkov (angl. data drift). To je pojav pri katerem se s časom spreminjajo porazdelitve vrednosti vhodnih podatkov ali razmerja med vhodnimi in izhodnimi značilnicami. Brez ustreznega nadzora in prilagajanja lahko takšne spremembe povzročijo postopno degradacijo napovedne uspešnosti modela, kar lahko pripelje do napačnih odločitev ter posledično do finančnih izgub [3].

Kot odziv na te izzive se razvija praksa MLOps, ki združuje metodologije DevOps z značilnostmi razvoja in vzdrževanja ML modelov, s ciljem povečanja stopnje avtomatizacije, nadzora kakovosti in zanesljivosti napovednih modelov. MLOps omogoča avtomatizacijo in standardizacijo procesov ter zagotavlja transparentnost, ponovljivost in zanesljivost celotnega življenjskega cikla ML sistemov [4]. Ključne komponente MLOps vključujejo upravljanje verzij modelov in podatkov, avtomatizirane postopke za ponovno učenje modelov, validacijo in testiranje podatkov ter neprekinjeno dostavo in spremljanje delovanja modelov kot tudi kakovosti vhodnih podatkov v produkcijskih okoljih.

V prispevku podrobneje obravnavamo ključne izzive in rešitve za zagotavljanje kakovosti napovednih modelov z vidika MLOps praks. Posebno pozornost namenjamo predvsem validaciji, testiranju in nadzoru podatkov, verzioniranju modelov in podatkov ter tehnikam spremljanja modelov v produkcijskih okoljih. Za ilustracijo praktične uporabe predstavljamo študijo primera platforme Airly, ki vključuje celovito MLOps podporo za vse ključne korake v ciklu neprekinjenega učenja napovednih modelov in zagotavljanja njihove zanesljivosti v realnem času.

2 Neprekinjeno učenje

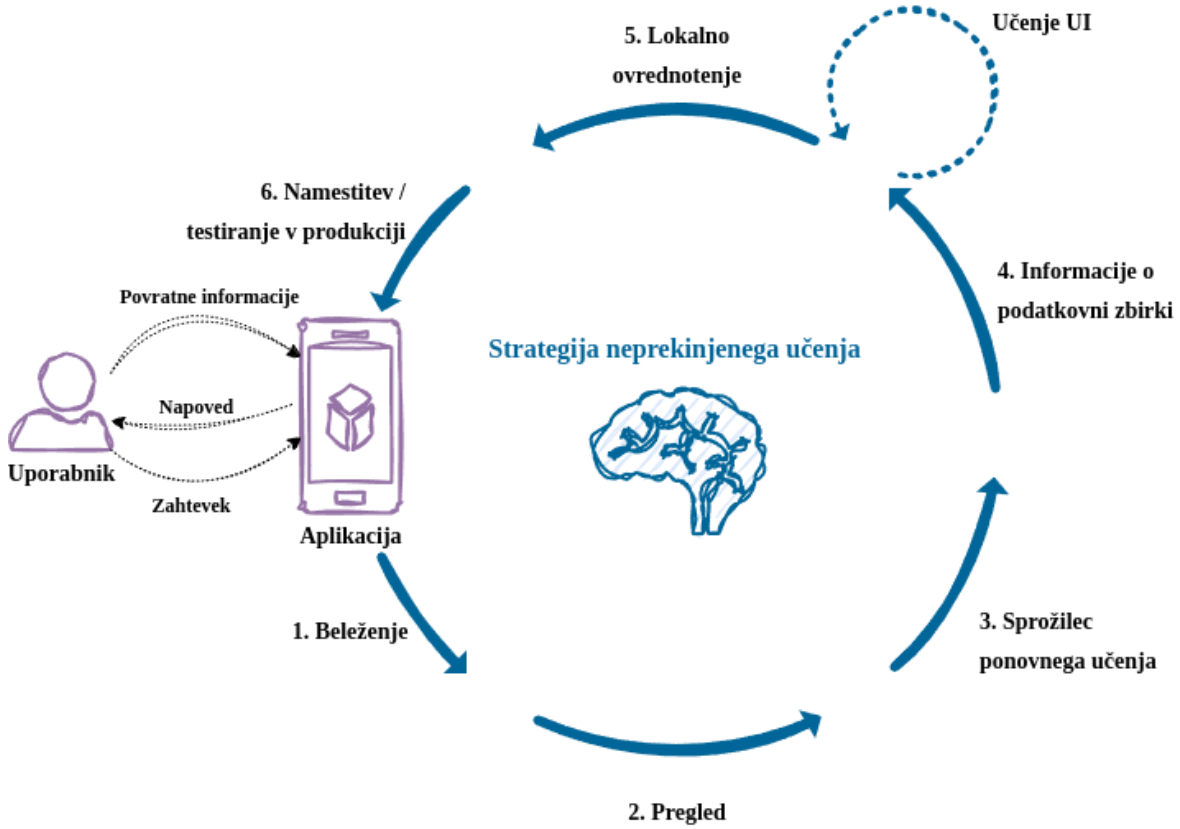
V klasičnem programskem inženirstvu končni izdelek običajno predstavlja statično različico programske rešitve. V nasprotju s tem pa so napovedni modeli strojnega učenja dinamični, njihova napovedna uspešnost pa se pogosto degradira skozi čas zaradi sprememb porazdelitev vrednosti v podatkih, poslovnih pravilih, sprememb v vedenju uporabnikov ali sprememb v okolju. Takšne spremembe zahtevajo neprekinjen nadzor podatkov, posodabljanje napovednih modelov, v mnogih primerih pa tudi ponovno učenje ali do-učenje modelov. Neprekinjeno učenje (angl. Continuous Learning) v kontekstu produkcijskih ML sistemov pomeni vzpostavitev avtomatiziranega sistema, ki vključuje zajem in verzioniranje podatkov, validiranje in testiranje podatkov, ponovljivost in sledenje eksperimentom, ovrednotenje in verzioniranje napovednih modelov, namestitev v produkcijo ter nadzorovanje delovanja modelov v realnem času.

Celoten cikel mora biti ponovljiv, sledljiv in kar se da avtomatiziran, kar predstavlja temelj sodobnih MLOps praks. Takšen pristop omogoča, da modeli ostanejo posodobljeni, prilagojeni trenutnim razmeram in kar se da zanesljivi za uporabo v realnih okoljih. Slika 1 prikazuje ključne korake strategije neprekinjenega učenja, ki so temelj sodobnih

MLOps pristopov. Namen takšne strategije je zagotoviti, da napovedni modeli ohranjajo svojo kakovost tudi po implementaciji v produkcijsko okolje, kjer se podatki in ostali dejavniki pogosto spreminjajo.

Proces se začne z beleženjem podatkov o interakciji uporabnikov z aplikacijo (točka 1), kjer se zbirajo vhodni podatki, napovedi modela in povratne informacije. Ti podatki se nato periodično pregledajo (točka 2) z namenom preverjanja sprememb v porazdelitvi vrednosti podatkov ali sami strukturi podatkov.

Ko sistem zazna pomembne spremembe ali padec napovedne uspešnosti modela, se sproži proces (ponovnega) učenja (točka 3). To vključuje pripravo in verzioniranje nove (posodobljene) verzije podatkovne zbirke (točka 4), kar je ključno za zagotavljanje ponovljivosti eksperimentov in sledljivosti spremembam. Zatem sledi lokalno ovrednotenje posodobljenega napovednega modela (točka 5), pri čemer se nova različica modela primerja z obstoječo glede na izbrane metrike napovedne uspešnosti. Če novi model dosega boljše rezultate, se namesti oziroma posodobi v produkcijskem okolju (točka 6). V okviru MLOps se pri tem uporabi infrastruktura za avtomatizirano testiranje, sledenje modelov in nadzorovanje njihovega vedenja v realnem času. Sistem lahko tako neprekinjeno spremlja napovedno uspešnost modela in pravočasno zazna morebitno degradacijo, kar omogoča proaktivno vzdrževanje kakovosti napovedne uspešnosti ML modelov.



Slika 1: Primer strategije neprekinjenega učenja napovednega modela.

2.1. MLOps

MLOps je skovanka za ML operacije (Ops) in razvoj (Dev), kjer se aplicirajo DevOps prakse na ML metode. DevOps je skupek metod, ki zaznavajo spremembe v programski opremi in upravljajo zanesljivost drugih orodij, ki so s tem povezane. Neprekinjena integracija in neprekinjena dostava uporablja vrsto strategij za zagotovitev, da so izdaje programske opreme hitrejše in zanesljivejše. MLOps metoda vključuje v proces programskega razvoja tudi ML modele. MLOps proces integrira ML značilnosti z DevOps koncepti, s čimer se omogoči avtomatska namestitev in učinkovito nadzorovanje ML modelov v času razvoja. MLOps sisteme sestavljajo 3 glavne komponente, in sicer: model, koda in podatki. Proces MLOps pa nam omogoča tudi enostavnejše nadzorovanje in upravljanje večjega števila napovednih modelov [5].

Ključen dejavnik, ki ima velik vpliv na napovedno uspešnost ML modela, so podatki. Zaradi napak ali anomalij, ki se lahko pojavijo v podatkih, se morajo ti filtrirati in procesirati preden so uporabljeni za učenje napovednih modelov. V samem procesu učenja napovednega modela so vključeni tudi konvencionalni integracijski testi in testi enot. Validacija in analiza ovrednotenja napovedne uspešnosti modela je nujno potrebna za preverjanje delovanja ključne funkcije modela - napovedovanja. Ko model prestane faze ovrednotenja napovedne uspešnosti ter vse integracijske teste in teste enot, se ga namesti v testno okolje, ki simulira delovanje produkcijskega okolja [5]. Grajenje ali učenje napovednega modela zahteva več komponent, med ključne pa spadajo: podatkovna zbirka namenjena za učenje, izbrane metrike napovedne uspešnosti, izbran algoritem strojnega učenja, vrednosti hiperparametrov algoritma strojnega učenja in podatkovna zbirka namenjena za ovrednotenje napovedne uspešnosti (testiranje) naučenega modela. Že samo število komponent, ki sestavlja proces gradnje, je pomemben razlog zakaj je vpeljava MLOps procesa velik izziv. Kompleksnost posameznih komponent pa proces samo še oteži [6].

Modeli, ne glede na uporabljen algoritem, so lahko samo le tako »dobri« kot so »dobri« učni podatki. Različni dejavniki lahko povzročijo, da so podatki neuporabni za namen učenja napovednih modelov, med ključne pa spadajo nepopolnost, nenatančnost in nekonsistentnost podatkov. V ta namen se podatki v procesu MLOps validirajo, testirajo in predprocesirajo. Omenjeni koraki zajemajo pregled distribucije vrednosti podatkov, čiščenje, zapolnjevanje manjkajočih vrednosti, transformacija v drugačno obliko, filtriranje, vzorčenje, itd. [6].

Modeli morajo biti tudi testirani oz. ovrednoteni. To je proces, v katerem preverjamo tudi, ali model sploh dela to, kar bi naj in kako dobro to delo opravlja. Ovrednotiti ga je potrebno v kontekstu njegovega delovanja, pri čemer moramo imeti možnost primerjave z nečim, kar je obstajalo pred modelom – ali prejšnja verzija modela ali rešitev na podlagi pravil, da lahko preverimo kaj bi bil rezultat, če bi nov model zamenjal prejšnjo rešitev. Pri tem je pomembna tudi izbira primerne metrike s katero bomo ocenili in primerjali različne modele glede na dan problem, ki ga rešujejo. Da preverimo sposobnost generalizacije (sposobnosti posplošitve znanja) modela, mora biti metrika izračunana na podlagi podatkovne zbirke, ki ni bila uporabljena v procesu učenja napovednega modela [6].

2.2. Verzioniranje in ponovljivost eksperimentov

Verzioniranje in ponovljivost eksperimentov naslavljata dve različni potrebi in sicer potrebo, da imajo podatkovni znanstveniki možnost vračanja na različne verzije modelov ter da bodo v neki točki razvoja, podatkovni znanstveniki skoraj zagotovo morali poustvariti pogoje, ki so vodili do posodobljene različice napovednega modela, tudi nekaj let po dejanski namestitvi. Predvsem slednja služi za namen primerjave novo razvitih napovednih modelov s predhodnimi.

Verzioniranju je sicer do določene mere že zadoščeno, v kolikor vse temelji na programski kodi, pri čemer so v uporabi orodja za verzioniranje programske kode. Seveda, pa je tipično za namen uspešne poustvaritve pogojev potrebno poskrbeti tudi za verzioniranje podatkovnih zbirk, ki so v nekem trenutku bile uporabljene za učenje napovednega modela. Takšno poustvaritev pogojev z drugimi besedami imenujemo tudi ponovljivost eksperimentov in je ena izmed ključnih lastnosti MLOps paradigme. Torej v vsaki točki v razvoju, ko podatkovni znanstveniki ustvarijo (naučijo) model, ki uspešno rešuje problem, je potrebno zagotoviti, da je model mogoče poustvariti. V namen zagotovitve je potrebno zadostiti številnim vidikom. Prvi je zagotovitev, da so predpostavke podatkovnih znanstvenikov zabeležene. Na ta način poskrbimo, da lahko drug podatkovni znanstvenik brez težav poustvari katerikoli eksperiment, brez dodatnih informacij. Drug vidik je vpliv naključnosti. Mnogi algoritmi strojnega učenja kot tudi sami procesi znotraj razvoja, kot je na primer naključna razdelitev podatkovnih zbirk, uporabljajo psevdo-naključna števila. Da lahko natančno reproduciramo eksperiment, moramo imeti kontrolo tudi nad temi števili. To največkrat kontroliramo z nastavljanjem fiksne vrednosti semena (angl. seed) v generatorju števil. S stališča podatkov, kot omenjeno potrebujemo podatke v obliki kot so bili v tistem času. Prav tako ne smemo zanemariti raznih nastavitev metod ali algoritmov, ki so uporabljeni v celotnem procesu. Tudi te je potrebno skrbno shranjevati saj lahko zgolj na tak način zagotovimo ponovljivost eksperimentov. Za namen

primerjave napovednih uspešnosti posameznih verzij napovednih modelov je potrebno beleženje oz. shranjevanje tudi teh. Nepresenetljivo tudi različne implementacije enakega modela vodijo v različne modele, ki lahko nato v določenih primerih ob istih vhodnih podatkih vrnejo napovedi, ki so si med seboj različne. In nenazadnje je potrebno tudi zagotoviti enako izvajalno okolje. Že nekoliko različne verzije Python knjižnic lahko vodijo v drugačne rezultate, kar pa je težko napovedati in nadzorovati [6].

Verzioniranje modelov je pomemben vidik MLOps, ki omogoča, da modelom sistematično sledimo in jih nadziramo. Uporaba verzioniranja modelov je nujna za zagotavljanje ponovljivosti, nadzorovanja posodobitev ali sprememb in izvedbo vrnitve na predhodno verzijo (angl. rollback). Poznamo različne strategije poimenovanja verzij modela, kot je semantično verzioniranje, ki uporablja shemo za številko verzije (npr. major.minor.patch) ali pa se uporablja strategija z uporabo metapodatkov, ki vključujejo različne podrobnosti o modelu, kot je arhitektura, učni podatki in hiperparametri. Slednji pristop, ki je tudi med bolj uporabljanimi, v ta namen uporablja t.i. register modelov. Register modelov služi kot odložišče vseh prej omenjenih informacij oz. podatkov, ki so potrebni za uspešno reproduciranje nekega eksperimenta [7]. Ena izmed bolj popularnih odprtokodnih rešitev je kot je MLFlow, ki ponuja centralizirano orodje za shranjevanje, upravljanje in dostop do različic modelov, obstaja pa tudi množica lastniških in plačljivih rešitev kot so Neptune, Metaflow, ipd..

Verzioniranje podatkov sledi enaki paradigmi kot verzioniranje programske kode, vendar za podatkovne zbirke. Sistemi z živimi podatki konstantno pridobivajo nove podatke, kar lahko vodi v različne verzije enake podatkovne zbirke. Verzioniranje podatkov omogoča sledenje podatkovnim zbirkam, kjer se zabeleži vsaka sprememba na posamezni podatkovni zbirki. S tem procesom pridobimo vpogled v spremembe in razvoj podatkov v projektu in zmanjšamo tveganja za nastanek napak. Če se pojavi napaka z najnovejšo verzijo podatkov, se lahko enostavno vrnemo na zadnjo delujočo verzijo. Z uporabo tehnologij verzioniranja podatkov lahko ekipe zajamejo verzije podatkov in modelov z Git commit-i, kar predstavlja mehanizem za spremembo med različnimi podatki. Rezultat takšnih tehnologij je ena zgodovina tako za podatke, kot za kodo in za modele strojnega učenja, skozi katere se lahko pomikajo člani ekipe [8]. Med pogosteje uporabljanimi orodji za namen verzioniranja podatkovnih zbirk so Git-LFS in DVC, ki se oba relativno enostavno integrirata v orodje Git.

2.3. Učenje napovednih modelov

Uporaba napovednih modelov strojnega učenja je iz dneva v dan bolj vseprisotna. Specifična lastnost napovednih modelov pa je, da tipično niso statični in jih je tega potrebno pogosto ponovno učiti, da se prilagodijo na spremembe v podatkih [9]. Ponovno učenje modela je nujno, ko se pojavi sprememba v porazdelitvi vrednosti podatkov (angl. Data Drift). Ta pojav se lahko pripeti že zaradi sezonskosti podatkov ali nenazadnje sprememb v uporabniških nastavitvah. Ko pride do takšnega pojava se tipično napovedna uspešnost modela zmanjša [11].

Ponovno učenje modelov je pomembna komponenta za prilagajanje modelov. Modele lahko ponovno učimo po vnaprej določenem časovnem obdobju na enake intervale ne glede na napovedno uspešnost modelov. Takšen pristop je sicer predvidljiv (načrtovan), vendar lahko vodi v zakasnjeno prilagoditev napovednega modela na spremembe v okolju (podatkih). Načrtovano ponovno učenje se dogaja v določenih intervalih, kot je tedensko, mesečno, glede na zgodovinske trende v podatkovnih premikih [10]. Takšen pristop je uporaben, kadar je podatkovni premik možno napovedati in se zgodi postopoma. Pri takšnem podatkovnem premiku lahko pripravimo sistematične posodobitve, ki ne bodo močno vplivale na produkcijski cevovod. Omejitev omenjenega pristopa pa je, da se ne more dinamično odzivati na spremembe v podatkih. Če se podatkovni premik zgodi prej kot pričakovano bo model slabše deloval do naslednje načrtovane posodobitve. Podobna omejitev se zgodi tudi, če je interval prevelik. V takšnem primeru model morda ne bo uspel zajeti pomembnih sprememb v podatkih, zaradi česar bo dalj časa deloval s slabšo točnostjo. Zaradi takšnih pomanjkljivosti omenjenega pristopa so bile predlagane metode, kjer se modeli dinamično ponovno učijo glede na določene prage napovedne uspešnosti (angl. performance thresholds) ali zaznanih sprememb v porazdelitvah vrednosti podatkovnih. Slednji pristop pa seveda zahteva, neprekinjeno nadzorovanje in ovrednotenje napovednih modelov v produkciji (angl. online evaluation),

saj sicer nimamo vzvoda, na podlagi katerega, bi lahko objektivno ovrednotili uspešnost delovanja napovednega modela [12].

2.4. Namestitev v produkcijo

Ko je model ponovno naučen, sledi namestitev v izvajalno okolje. Preden ga namestimo je potrebno preveriti, da model ne vpeljuje nazadovanja v smiselne napovedne uspešnost ali da ne vpeljuje pristranskosti. S tem namenom izvajamo ovrednotenje modela pred dejansko izpostavitvijo izbranega modela vsem končnim uporabnikom. V ta namen lahko uporabimo obstoječe pristope testiranja, kot na primer AB testiranje kjer nov model izpostavimo manjši podmnožici uporabnikov, ostali pa še vedno uporabljajo trenutno verzijo napovednega modela. S tem zmanjšamo tveganje za nastanek poslovne škode v primeru slabšega delovanja posodobljenega napovednega modela, saj mu je izpostavljen le manjši delež uporabnikov. Poleg omenjenega AB testiranja je priročna in pogosto uporabljana tudi namestitev v senci (angl. shadow deployment). S takšnim pristopom lahko preverimo in primerjamo uspešnost in učinkovitost novega modela z obstoječo verzijo brez, da bi posodobljen model imel vpliv na same uporabnike. Namestitev v senci namreč deluje nekoliko drugače kot AB testiranje, saj se model izvaja vzporedno s trenutnim produkcijskim, na način, da se vhodni podatki pošiljajo v oba modela pri čemer se napovedi produkcijskega posredujejo uporabniku, napovedi modela v senci pa se le shranijo za kasnejšo analizo in primerjavo. S tem lahko dobimo realen vpogled v delovanje posodobljenega napovednega modela, brez da bi vplival na delovanje produkcije [12].

2.5. Nadzorovanje

Nadzorovanje je v kontekstu napovednih modelov namenjenih sledenju delovanja modelov v produkcijskih okoljih, kjer se spremljajo vrednosti metrik napovedne uspešnost kot tudi propustnost ter latence pri pridobivanju napovedi. Slednje so ključne za zgodnjo zaznavanje ozkih grl, ki so pogost pojav predvsem pri kompleksnih modelih globokega učenja. Na ta način se nam tudi omogoči vpogled v realno časovne vrednosti metrik posameznega modela, s pomočjo česar lahko hitreje zaznamo tudi prisotnost spremembe porazdelitev vrednosti vhodnih podatkov. Dodatno nam neprekinjeno nadzorovanje omogoča tudi hitrejšo zaznavanje morebitnega poslabšanja napovedne uspešnosti modelov ali zaznavo prisotnih nepravilnosti, kar nam omogoča pravočasno posredovanje in prilagoditev napovednega modela [7].

3 Validiranje, testiranje in nadzorovanje podatkov

Validiranje, testiranje in nadzorovanje podatkov so ključni procesi v vsakem MLOps življenjskem ciklu. Uspešnost teh je namreč ključna za uspešnost učenja napovednega modela kot tudi kasnejšo napovedno uspešnost. V splošnem validacija podatkov predstavlja preverjanje same strukture podatkovne zbirke ter zalog vrednosti posameznih značilnic. Na drugi strani testiranje v splošnem služi za namen identifikacije potencialne spremembe v porazdelitvi vrednosti vhodnih značilnic. Morebitna nezaznava spremembe v porazdelitve vrednosti vhodnih značilnic lahko prav tako ključno vpliva na napovedno uspešnost modela v produkcijskem okolju.

3.1. Validiranje podatkov

Obstajajo različni pristopi validacije podatkov, ki jih v splošnem delimo na validacijo znotraj paketa podatkov (angl. Single-batch) in validacijo med paketi podatkov (angl. Inter-batch). Validacija znotraj paketa se uporablja za naslavljanje problema anomalij v sklopih podatkov. Cilj v tej fazi je zaznati anomalijo in obvestiti odgovorne osebe o napaki in začeti z preiskavo. Na drugi strani pa je validacija med paketi namenjena za preverjanje obstoja signifikantnih razlik med različnimi sklopi ali verzijami podatkov [15]. Za preverjanje anomalij v podatkih se tipično

uporablja pristop validacije na podlagi sheme. Sistem preverja strukturo novih podatkov s predpisano shemo, ki služi kot logični model za podatke. V kolikor se pojavijo kakršne koli razlike od pričakovane strukture, so podatki označeni kot anomalija in se sproži opozorilo, ki zahteva intervencijo [15]. Dodatno se validacija med paketi uporablja tudi za namen iskanja napak, ki nastanejo zaradi spremembe značilnic, ki so lahko posledica spremembe v okolju.

3.2. Testiranje podatkov

Sčasoma je pričakovano, da se napovedna uspešnost modelov poslabša (degradira), kar lahko pripišemo dejstvu, da je večina modelov strojnega učenja zgrajena nad zgodovinskimi podatki, okolje v katerem napovedni model deluje pa se konstanto spreminja. Posledično pride do pojava, ki ga imenujemo sprememba porazdelitve vrednosti značilnic. Poznamo tri glavne vrste takšnih sprememb in sicer kovariatno spremembo (angl. Covariate shift), spremembo predhodnih verjetnosti (angl. Prior probability shift) ter konceptualno spremembo (angl. Concept shift). Kovariatna sprememba označuje spremembe distribucij vhodnih podatkov oziroma značilnic. Sprememba predhodnih verjetnosti predstavlja spremembo distribucije izhodnih podatkov oziroma spremenljivk. Konceptualna sprememba pa predstavlja spremembo statistične povezave med vhodnimi in izhodnimi podatki [13].

Kovariatna sprememba se navezuje na spremembe v distribuciji vrednosti vhodnih podatkov. Definirana je na več različnih načinov, bistveno pa je, da se skozi čas populacija spreminja in z njo vrednosti oziroma porazdelitve vhodnih podatkov. Govorimo torej o pojavu, ki nastane kadar pride do večje razlike med podatki, ki so bili uporabljeni za učenje modela, in podatki, ki se uporabljajo za ustvarjanje napovedi [13].

Sprememba predhodnih verjetnosti predstavlja pojav, v katerem se porazdelitev ciljnih razredov med učenjem napovednega modela in samo uporabo modela v produkciji spremeni. To pomeni, da so značilnosti posameznih razredov v učni podatkovni zbirki in vhodnimi podatki v produkciji enake, vendar se je delež posameznih razredov med tema dvema fazama spremenil. Tak pojav lahko pomembno vpliva na delovanje modela v praksi, saj večina nadzorovanih modelov implicitno predpostavlja, da so podatki v učnem in testnem okolju iz iste porazdelitve. Če ta predpostavka ne drži, so lahko napovedi sistematično pristranske [13].

Konceptualna sprememba se zgodi, ko se razmerje med vhodnimi značnilnicami in ciljno značilnico spremeni, kar predstavlja najtežji izziv med predstavljenimi tipi podatkovnih sprememb. V takšnem primeru ostajajo porazdelitve vrednosti značilnic enake, tako pri vhodnih kot izhodnih značilnicah, spremeni se pa funkcija preslikave iz vhodnih v izhodne vrednosti. Takšna sprememba pomeni, da enaki vhodni podatki skozi čas vodijo do različnih izhodov. Na primer, v sistemu za zaznavanje goljufij se lahko vedenjski vzorci uporabnikov ne spremenijo, vendar lahko nova oblika zlorabe povzroči, da isti vzorec, ki je bil v preteklosti označen kot legitimen, v sedanjosti predstavlja goljufijo. Konceptualne spremembe so posebej problematične zato, ker jih ni mogoče preprosto zaznati zgolj z analizo porazdelitev vrednosti značilnic. Zahtevajo stalno nadzorovanje delovanja napovednega modela v realnem okolju in pogosto vključujejo uporabo detekcijskih mehanizmov, temelječih na naprednejših statističnih testih ali periodično ponovno učenje modela. Če takšnih sprememb ne zaznamo pravočasno, lahko pride do občutnega poslabšanja kakovosti napovedi in s tem do napačnih odločitev, kar je še posebej kritično v varnostno občutljivih ali reguliranih okoljih.

3.3. Nadzorovanje podatkov

Sprememba v porazdelitvi vrednosti značilnic se lahko zazna, kadar se statistične značilnosti vhodnih podatkov razlikujejo od porazdelitve vrednosti učnih podatkov. Tipično jih lahko zaznamo že na podlagi razlik v povprečnih vrednostih značilnic, variance ali oblike porazdelitve vhodnih podatkov. Zaznava takšnih pojavov zahteva neprekinjen nadzor in statistično analizo novih podatkov v primerjavi z zgodovinskimi podatki [12]. Za zaznavo

tovrstnih sprememb v realnem času poznamo različne tehnike, kot so statistične metode, tehnike procesiranja z uporabo UI mehanizmov, metrike na podlagi razdalje in vizualizacijske tehnike [14].

V ta namen pogosto uporabljamo statistične teste kot so Kolmogorov-Smirnov test, indeks populacijske stabilnosti (PSI) in Chi-kvadrat test (angl. Chi-Square Test). Kolmogorov-Smirnov test se uporablja za preverjanje enakosti dveh distribucij populacij, z namenom ugotovitve ali sta populaciji pomembno različni. Indeks populacijske stabilnosti se uporablja za ustvarjanje statistike, ki prikazuje relativno gibanje vrednosti značilnic v in med razredi. Ko je PSI vrednost višja od 0,25, potem so zaznane velike ravni spremembe v porazdelitvi podatkov. Chi-Square test se uporablja v primeru kategoričnih spremenljivk. Testira porazdelitve opazovanih vrednosti z že znanimi oziroma pričakovanimi vrednostmi [14].

Poleg omenjenih statističnih metod se pogosto uporabljajo tudi metode, ki delujejo na podlagi razdalje, kot na primer Kullback-Leiblerjeva (KL) divergenca in Jensen-Shannonova divergenca. Kullback-Leiblerjeva divergenca razloži do kakšne mere se razlikuje ena porazdelitev verjetnosti od druge. Jensen-Shannonova divergenca pa je zrcalna slike KL divergence [14].

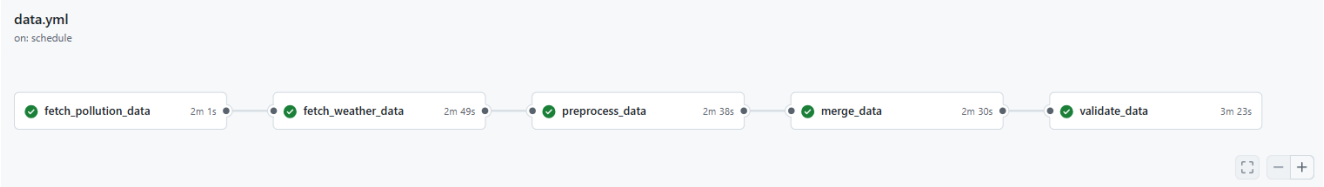
V procesu MLOps je ključno, da se tovrstna testiranja implementirajo v cevovod in izvajajo ob vsaki posodobitvi podatkov, ki jih uporabljamo za učenje napovednih modelov ter na tak način preprečimo morebitno spremembo porazdelitve vrednosti podatkov, ki bi lahko negativno vplivala na napovedno uspešnost modela v produkciji.

4 Študija primera: Airly

Airly je na MLOps pristopu temelječa rešitev, ki omogoča ogled napovedi kakovosti zraka v Sloveniji. Natančneje, omogoča pregled trenutnega stanja kot tudi napovedi za delce PM10 in PM2.5. Podatki se kontinuirano zbirajo in prikazujejo za 21 meteoroloških postaj na urnem intervalu. Poleg vpogleda v trenutno stanje in napovedi, rešitev omogoča tudi nadzorovanje delovanja napovednih modelov, kjer so prikazane metrike napovedne uspešnosti za posamezni model in primerjave med napovedanimi ter dejanskimi vrednostmi delcev v zraku za vsako meteorološko postajo. Neprekinjeno zbiranje, validiranje in testiranje podatkov nam omogoča zaznavanje sprememb v porazdelitvah vrednosti podatkov, na podlagi katerih lahko potem sprejmemo odločitev ali model zamenjamo z novim ali ne. Dodatno je neposredno podprto tudi upravljanje z vsemi modeli v sklopu rešitve.

4.1. Cevovod za podatke

Vsaka MLOps rešitev vsebuje različne cevovode za izvedbo nalog. Takšen pristop je uporabljen tudi v naši rešitvi. MLOps proces Airly-ja se začne s cevovodom za podatke (Slika 2), ki jih pridobimo iz različnih virov in jih nato predprocesiramo (shranimo podatke na urni interval, odstranimo duplikate). Sledi združevanje podatkov, kjer združimo vrednosti PM delcev z vremenskimi podatki. Zatem sledi validacija in testiranje podatkov z uporabo statističnih testov. Če so testi uspešno prestani, se cevovod nadaljuje z razdelitvijo podatkov na učno in testno množico. V našem primeru se ta cevovod izvaja periodično in sicer enkrat na dan.



Slika 2: Grafični prikaz podatkovnega cevovoda.

Podatki se shranjujejo in verzionirajo z uporabo orodja DVC, ki omogoča shranjevanje velike količine podatkov in razbremenitev orodja za verzioniranje kode, v našem primeru orodja Git. V vsakem stanju cevododa se le ti dodajo na oddaljen strežnik s pomočjo DVC ukazov, ki preverja in beleži spremembe v posameznem direktoriju.

V kolikor spremembe obstajajo se za direktorij generira nova datoteka oziroma posodobi obstoječa, kamor se zapiše meta podatke, kot so število datotek, vrednost zgoščevalne funkcije (angl. hash), velikost direktorija, s čimer je enostavno omogočeno sledenje verzijam podatkov. Nato se spremembe naložijo na oddaljen DVC podatkovni strežnik, spremenjena datoteka z metapodatki pa se objavi na Git strežnik.

4.2. Cevovod za učenje modelov

Ker se zavedamo, da lahko pride do spremembe v porazdelitvi podatkov, smo razvili tudi avtomatiziran cevovod za učenje modelov. Cevovod se trenutno izvaja periodično in sicer enkrat mesečno ali pa na zahtevo. Najprej se pridobi koda celotnega repozitorija skupaj z ».dvc« datotekami, ki vsebujejo potrebne metapodatke, zatem pa se pridobijo najnovejše verzije podatkov iz DVC strežnika, ki so uspešno prestale proces validiranja in testiranja. Šele nato se izvede dejansko proces učenja novih modelov, pri čemer so vsi koraki zabeleženi na način, da lahko zagotovimo reprodukcijo eksperimentov. Ko so modeli naučeni, se nova verzija, skupaj z metapodatki in trenutno verzijo podatkov, shrani na MLFlow strežnik in se ji dodeli nova verzija.

4.3. Nadzorovanje v produkciji

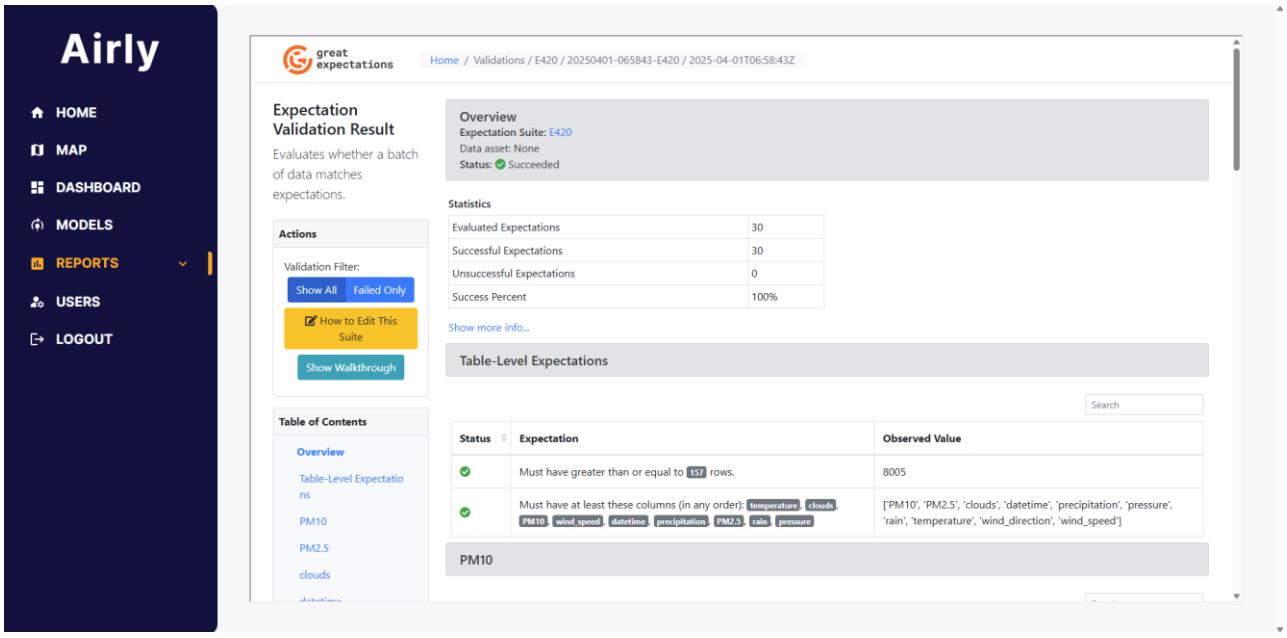
Za vsako postajo si lahko ogledamo napovedane vrednosti s strani različnih modelov ter dejanske vrednosti delcev PM v zraku. Slika 3 prikazuje nadzorno ploščo, ki omogoča vpogled v uspešnost delovanja napovednega modela. Siva krivulja predstavlja realne vrednosti delcev PM10 in PM2.5 za izbrano meteorološko postajo, medtem ko črna krivulja predstavlja napovedane vrednosti. Dodatno ima platforma implementirano še zmožnost testiranja napovednega modela v produkciji z uporabo pristopa testiranja v senci. V našem primeru rumena krivulja predstavlja model v senci, katerega napovedi uporabniki ne vidijo, nam pa služi za namen primerjave napovedne uspešnosti s trenutnim produkcijskim napovednim modelom.

Slika 3: Prikaz pogleda za nadzorovanje napovedi.

Poleg vpogleda v primerjave napovedanih vrednosti onesnaženja zraka, rešitev omogoča tudi vpogled v vrednosti izbranih regresijskih metrik s katerimi lahko kvantitativno ocenimo napovedno uspešnost modela. Slika 4 prikazuje vpogled v vrednosti regresijskih metrik povprečna absolutna napaka (angl. Mean absolute error, MAE) ter povprečna kvadratna napaka (angl. Mean squared error, MSE) za izbrano meteorološko postajo.

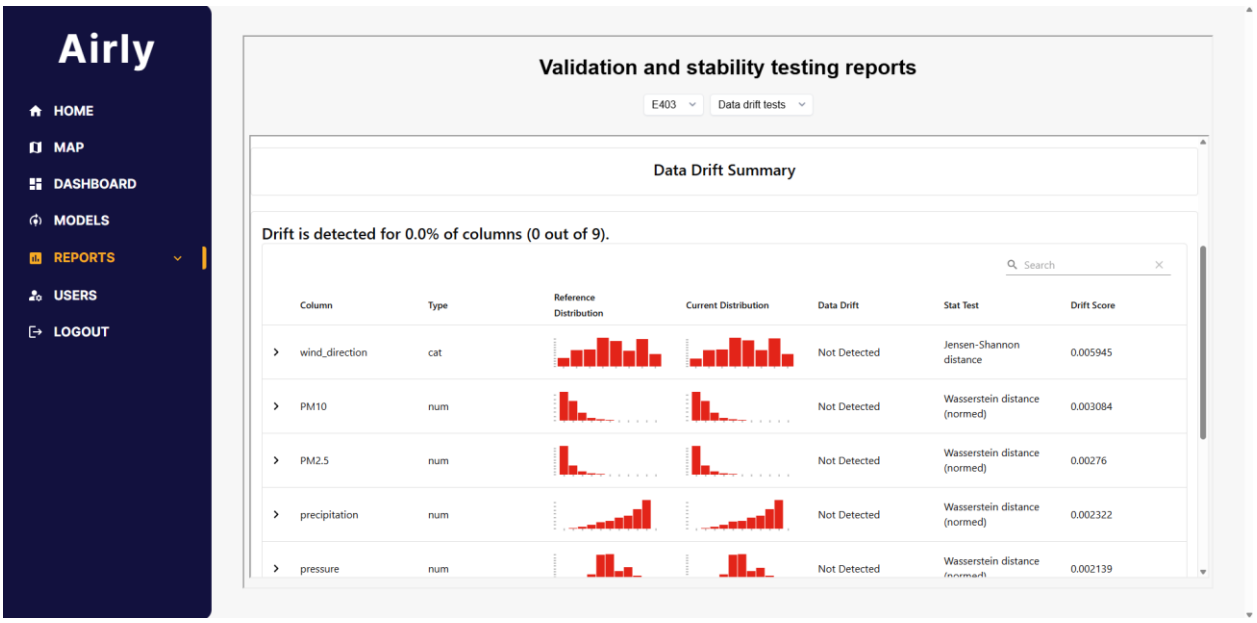
Slika 4: Prikaz pogleda vrednosti regresijskih metrik.

V sklopu nadzorovanja si lahko tudi ogledamo poročila o izvedenih procesih validacije in testiranja podatkov. V poročilu o validaciji podatkov (Slika 5) lahko vidimo število vseh vrstic, stolpcev, zalogo vrednosti posameznega stolpca ter koliko vrednosti je bilo nepričakovanih oziroma odstopalo od zastavljene sheme. Prav tako pa imamo omogočen vpogled v vsa zgodovinska validacijska poročila.



Slika 5: Primer vpogleda v validacijsko poročilo.

V poročilu o testiranju podatkov (Slika 6) lahko preverimo, ali je prišlo do spremembe porazdelitve vrednosti podatkov za katero izmed značilnic, vrsto podatkov, referenčno distribucijo podatkov, trenutno distribucijo podatkov, kateri statistični test je bil uporabljen za testiranje podatkov ter dejanski rezultat spremembe porazdelitve podatkov. Poleg pregleda zadnje aktualnega poročila o testiranju imamo tudi vpogled v vsa predhodna poročila o testiranju.



Slika 6: Primer vpogleda v poročilo testiranja podatkov.

Vpogled v metrike napovedne uspešnosti modelov, kot tudi v poročila o validaciji in testiranju podatkov, nam omogočajo celovito razumevanje delovanja napovednega modela. S tem lahko ugotovimo, kako dobro se model generalizira na še nevidene podatke, kako občutljiv je na odstopanja v vhodnih podatkih ter ali obstajajo znaki degradacije napovedne uspešnosti skozi čas. Spremljanje teh metrik omogoča pravočasno odkrivanje težav, kot so premiki v podatkovni porazdelitvi, zmanjšana točnost napovedi ali porast neustreznih vhodov, kar je ključno za zagotavljanje stabilnosti in zanesljivosti produkcijskega modela. Takšen vpogled je temelj za informirano odločanje o ponovnem učenju modela, prilagoditvah cevovoda za obdelavo podatkov ali izboljšavah validacijskih pravil.

5 Zaključek

Vzpostavitev kakovostne MLOps rešitve zahteva premišljeno uporabo širokega nabora orodij in tehnologij, ki morajo med seboj učinkovito sodelovati. Najpogosteje uporabljena orodja so odprtokodne rešitve, kot so MLFlow za sledenje poskusom in verzioniranje modelov, DVC za upravljanje z različicami podatkov in cevovodov ter TensorFlow za gradnjo in učenje modelov. Vendar pa tehnična kompleksnost integracije teh orodij pogosto zahteva poglobljeno razumevanje posameznih komponent ter njihovo natančno uskladitev znotraj celotne arhitekture. Vzpostavitev takšne infrastrukture ni trivialna naloga. Vključuje tako načrtovanje ustreznega podatkovnega cevovoda kot tudi zagotavljanje zadostnih računalniških virov za izvajanje učenja napovednih modelov. MLOps ni enkraten projekt, temveč živ proces, ki se mora prilagajati novim podatkom in spreminjajočim se okoliščinam. Zato je ključno, da se podatki neprekinjeno nadzorujejo, saj lahko že manjše spremembe v vhodnih podatkih ali strukturi povzročijo odstopanja, ki lahko ključno vplivajo na delovanje modela.

Kljub številnim prednostim pa MLOps prinaša tudi nekatere izzive. Integracija odprtokodnih orodij lahko naleti na težave zaradi nepopolne dokumentacije, pomanjkljive podpore ali nedoslednosti med knjižnicami. Poleg tega so potrebe po računski moči in podatkovni shrambi lahko visoke, še posebej v okoljih z obsežnimi podatki in kompleksnimi modeli. Vse to povečuje časovno kompleksnost za vzpostavitev delujoče in stabilne infrastrukture.

Kljub relativno zahtevni vzpostavitvi pa dobro implementirana MLOps rešitev prinaša pomembne dolgoročne koristi saj omogoča zanesljivo in ponovljivo razvijanje, nameščanje ter vzdrževanje napovednih modelov v produkcijskih okoljih. S tem ne le dviguje kakovost napovednih modelov, temveč tudi omogoča njihovo proaktivno spremljanje, lažje odpravljanje napak in hitrejšo prilagoditev na spremembe. MLOps tako postaja ključen gradnik sodobnih inteligentnih sistemov, kjer je zaupanje v delovanje modela enako pomembno kot njegova napovedna uspešnost.

Literatura

[1] Jameel Francis, Forbes, „Forbes,“ 15 11 2024. [Elektronski]. Dostopno na: https://www.forbes.com/councils/forbestechcouncil/2024/11/15/why-85-of-your-ai-models-may-fail/.

[2] E. Breck, S. Cai, E. Nielsen, M. Salib in D. Sculley, „The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction,“ v Proceedings of IEEE Big Data, 2017.

[3] J. Gama, I. Žliobaite, A. Bifet, M. Pechenizkiy in A. Bouchachia, „A Survey on Concept Drift Adaptation,“ ACM Computing Surveys, Izv. 1, št. 1, 2013.

[4] J. Kazmierczak, K. Salama in V. Huerta, „MLOps: Continuous delivery and automation pipelines in machine learning,“ 28 08 2024. [Elektronski]. Dostopno na: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[5] S. Wazir, G. S. Kashyap, in P. Saxena, „MLOps: A Review“, 2023. [Elektronski]. Dostopno na: https://arxiv.org/abs/2308.10908

[6] M. Treveil idr., Introducing MLOps. O'Reilly Media, 2020.

[7] S. K. Thota, V. R. R. Alluri, V. Kumar, V. K. R. Vangoor, in S. Chitta, „MLOps: Streamlining Machine Learning Model Deployment in Production“, Journal of Artificial Intelligence, let. 2, str. 186–205, apr. 2022.

[8] E. Orr, „Data Version Control: What It Is and How It Works“. Pridobljeno: 23. marec 2025. [Na spletu]. Dostopno na: https://lakefs.io/data-version-control/

[9] Y. Wu, E. Dobriban, in S. Davidson, „DeltaGrad: Rapid retraining of machine learning models“, v Proceedings of the 37th International Conference on Machine Learning, H. D. III in A. Singh, Ur., v Proceedings of Machine Learning Research, vol. 119. PMLR, apr. 2020, str. 10355–10366. [Na spletu]. Dostopno na: https://proceedings.mlr.press/v119/wu20b.html

[10] A. Mahadevan in M. Mathioudakis, „Cost-Effective Retraining of Machine Learning Models“, 2023. [Na spletu]. Dostopno na: https://arxiv.org/abs/2310.04216

[11] K. Rahmani idr., „Assessing the effects of data drift on the performance of machine learning models used in clinical sepsis prediction“, Int J Med Inform, let. 173, str. 104930, 2023, doi: https://doi.org/10.1016/j.ijmedinf.2022.104930.

[12] A. Balasubramanian, „End-to-end model lifecycle management: An MLOPS framework for drift detection, root cause analysis, and continuous retraining“.

[13] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V Chawla, in F. Herrera, „A unifying view on dataset shift in classification“, Pattern Recognit, let. 45, št. 1, str. 521–530, 2012, doi: https://doi.org/10.1016/j.patcog.2011.06.019.

[14] N. Kodakandla, „Data drift detection and mitigation: A comprehensive MLOps approach for real-time systems“, International Journal of Science and Research Archive, let. 12, str. 3127–3139, apr. 2024, doi: 10.30574/ijsra.2024.12.1.0724.

[15] N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, in S. Whang, „Data validation for machine learning“, Proceedings of machine learning and systems, let. 1, str. 334–347, 2019.

Vpeljava tabelaričnih tokov v podatkovno arhitekturo

Tjaša Heričko, Saša Brdnik, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
tjasa.hericko@um.si, sasa.brdnik@um.si, muhamed.turkanovic@um.si

Sodobne podatkovne arhitekture se vse bolj usmerjajo k agilnemu modelu ELT, temelječemu na podatkovnih jezerih in koliščih. Ključna prednost takšnega pristopa je uporaba odprtih tabelaričnih formatov, kot so Apache Iceberg, Hudi in Delta Lake, ki temeljijo na odprtih datotečnih formatih, kot so Avro, ORC in Parquet. V prispevku predstavljamo Tableflow – novo rešitev iz ekosistema Confluent, ki omogoča neposredno predstavitev podatkov v Kafka temah kot odprte tabele v formatu Iceberg ali Delta. S tem se podatki, pridobljeni iz virov OLTP, že v fazi zajema in vnosa pretvorijo v format, primeren za poizvedovanje in učinkovitejše shranjevanje neobdelanih podatkov v podatkovno jezero/kolišče. Kafka v tem kontekstu ne služi le pretočni obdelavi, temveč tudi kot mehanizem za zajem in vnos podatkov, skladen s sodobno velepodatkovno arhitekturo. To bistveno zmanjša izgubo konteksta in sheme, ki se pogosto pojavlja pri klasičnih prenosih med operativnimi in analitičnimi sistemi. V prispevku bomo predstavili uporabnost tabelaričnega toka, prikazali praktično uporabo rešitve Tableflow znotraj platforme Confluent Cloud in integracijo s sodobnimi podatkovnimi arhitekturami ter izvedli primerjalno analizo z obstoječimi pristopi materializacije pretočnih podatkov v tabelarno obliko.

Ključne besede:
odprti tabelarični formati,
podatkovni tokovi,
pretočna obdelava podatkov,
Apache Kafka,
Confluent Tableflow.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.2

1 Uvod

V zadnjem desetletju smo priča preobrazbi podatkovnih arhitektur, ki se od tradicionalnega pristopa **ETL** (*angl. Extract-Transform-Load*) vse bolj nagibajo k agilnejšemu modelu **ELT** (*angl. Extract-Load-Transform*). Namesto da bi podatke najprej preoblikovali in jih šele nato naložili v strogo shematizirana relacijska podatkovna skladišča (*angl. relational data warehouses*) po principu ***"Schema-on-Write"***, sodobne podatkovne arhitekture pogosteje temeljijo na stolpčnih (NoSQL) podatkovnih bazah (*angl. columnar databases*) ter **podatkovnih jezerih** (*angl.* ***data lakes***). V primeru slednjih se izvorni podatki v svoji nestrukturirani ali delno strukturirani obliki najprej naložijo v ciljno hrambo, nato pa se po potrebi preoblikujejo za analizo po principu ***"Schema-on-Read"***. To pomeni, da se struktura podatkom določi šele ob njihovem branju, namesto ob njihovem nalaganju v podatkovno hrambo [1,2].

Tako stolpčno kot vrstično orientirana (tj. klasična) podatkovna skladišča, kakor podatkovna jezera, so se v praksi izkazala kot učinkovita pri izpolnjevanju določenih potreb. Klasična podatkovna skladišča so še posebej primerna za obdelavo strukturiranih podatkov in zagotavljajo zanesljivo upravljanje podatkov. Stolpčno orientirana podatkovna skladišča so optimizirana za analitične poizvedbe nad velikimi količinami podatkov, saj omogočajo hitrejše branje in agregiranje podatkov po stolpcih, kar povečuje učinkovitost pri kompleksnih analizah in poročanju. Medtem ko pa so podatkovna jezera učinkovita pri delu z raznovrstnimi podatki in so stroškovno učinkovita. V zadnjih letih se vse bolj uveljavljajo **podatkovna košišča** (*angl.* ***data lakehouses***), ki združujejo ključne prednosti vseh pristopov in hkrati naslavljajo njihove omejitve v enotni rešitvi. Ko govorimo o pristopih, moramo izpostaviti tudi dejstvo, da so vsi omenjeni pristopi podatkovnih skladišč namenjeni analitični obdelavi podatkov (tj. *On-Line Analytical Processing* – ***OLAP***), medtem ko podatkovna košišča dodajajo hkratno podporo transakcijski obdelavi masovne količine podatkov (tj. *On-Line Transactional Processing* – ***OLTP***). Konceptualno gre pri podatkovnih košiščih torej za uvedbo modela podatkovnega skladišča nad podatkovnim jezerom, kjer specializirane komponente omogočajo podporo transakcijskega modela pri delu z velepopodatki in upravljanje ter uveljavljanje sheme podatkov. S tem se ohranjajo zanesljivost, doslednost in zmogljivost navkljub inherentni prilagodljivosti podatkovnih jezer. V nasprotnem primeru namreč ta brez ustreznih mehanizmov pogosto sčasoma postanejo podatkovna močvirja (*angl. data swamps*). Podatkovno košišče omogoča shranjevanje strukturiranih, delno strukturiranih in nestrukturiranih podatkov, podobno kot podatkovno jezero, a hkrati podpira napredno obdelavo in upravljanje podatkov, kot jo poznamo pri podatkovnih skladiščih. Takšen pristop organizacijam omogoča, da znotraj enotne podatkovne infrastrukture učinkovito upravljajo celoten spekter (vele)podatkov – ne glede na njihovo izvorno obliko ali stopnjo strukturiranosti – in zadovoljujejo različne analitične potrebe, tako pri **paketni obdelavi podatkov** (*angl.* ***batch processing***) kot tudi pri **pretočni obdelavi podatkov** (*angl.* ***stream processing***). Takšna arhitektura omogoča večjo prilagodljivost, boljšo razširljivost ter zmanjšuje kompleksnost pri obdelavi podatkov [2,3,4].

Vse večji porast virov podatkov, ki podatke generirajo neprekinjeno, pričakovanj po svežih podatkih, še posebej za umetnointeligenčne (*angl. artificial intelligence* – ***AI***) aplikacije, in potreba po analitičnem vpogledu v podatke ter sprejemanje odločitev v realnem času postavlja v ospredje **podatkovnega inženirstva** (*angl. data engineering*) pretočno obdelavo podatkov, nepogrešljivi del katere najpogosteje predstavlja **Apache Kafka** s svojim ekosistemom. Kafka zagotavlja neprekinjeno obdelavo tokov podatkovnih zapisov, razvrščenih v teme (*angl. topics*), ob nizkih zakasnitvah in visoki prepustnosti. **Realnočasovno** obdelavo podatkovnih tokov – neprekinjeno, sočasno in na nivoju posameznih podatkovnih zapisov (tj. dogodkov) – nato tradicionalno omogočajo komponente Kafkinega ekosistema, kot sta domorodni komponenti **Kafka Streams API** in **ksqlDB** [5]. V tem kontekstu se pri prenosu tokovnih podatkov iz transakcijskih v analitične sisteme pojavlja izziv kompleksnosti procesov ETL/ELT, ki poskrbijo, da so podatki pripravljeni v obliki, ki je uporabna za analitiko ter **strojno učenje** (*angl.* ***machine learning – ML***). Pri tem so podatki izvzeti iz konteksta, v katerem so nastali in bili upravljani, zato med prenosom iz operativnega v analitično okolje pogosto prihaja do izgube konteksta in sheme podatkov. Eden izmed novejših pristopov k naslavljanju tega izziva je uvedba **tabelaričnih tokov** s pomočjo nedavno predstavljene rešitve **Tableflow**, ki je trenutno na voljo v okviru ekosistema **Confluent**. Ta rešitev omogoča neposredno predstavljanje pretočnih podatkov iz Kafka tem kot **odprte tabele** v formatih **Apache Iceberg** ali **Delta Lake**. S

tem so podatki iz operativnih sistemov že ob zajemu in vnosu pripravljeni v formatu, ki je primeren za učinkovito shranjevanje neobdelanih podatkov v podatkovnem jezeru/košišču in za nadaljnjo obdelavo z izbranim **poizvedovalnim/računskim pogonom/mehanizmom** (*angl.* ***query/compute engine***), ki podpira izbrane odprte tabelarične formate. To omogoča bolj prilagodljivo, razširljivo in učinkovito t. i. **brezglavno podatkovno arhitekturo** (*angl.* ***headless data architecture***), kjer so storitve shranjevanja, upravljanja in dostopanja do podatkov formalno ločene od storitev, ki obdelujejo in poizvedujejo po podatkih. Hkrati je takšna arhitektura skladna s principi sodobnega podatkovnega inženirstva *"**Shift Left**"* (tj. premik obdelave in upravljanja podatkov bližje izvornemu podatkovnemu viru) [6,7,8]. V nadaljevanju prispevka se bomo osredotočili na predstavitev rešitve Tableflow in izpostavili prednosti uvedbe tabelaričnih tokov v podatkovno arhitekturo napram tradicionalnemu prenosu pretočnih podatkov med operativnim in analitičnim svetom. Zraven konceptualnega pregleda bomo praktično prikazali uporabo rešitve Tableflow znotraj platforme Confluent Cloud in integracijo s platformo Databricks ter izvedli primerjalno analizo z obstoječimi pristopi materializacije pretočnih podatkov v tabelarno obliko in poizvedovanja po tokovnih podatkih. Izpostavili bomo, kdaj in zakaj pristop z uporabo tabelaričnih tokov predstavlja boljšo alternativo.

2 Podatkovna jezera in košišča

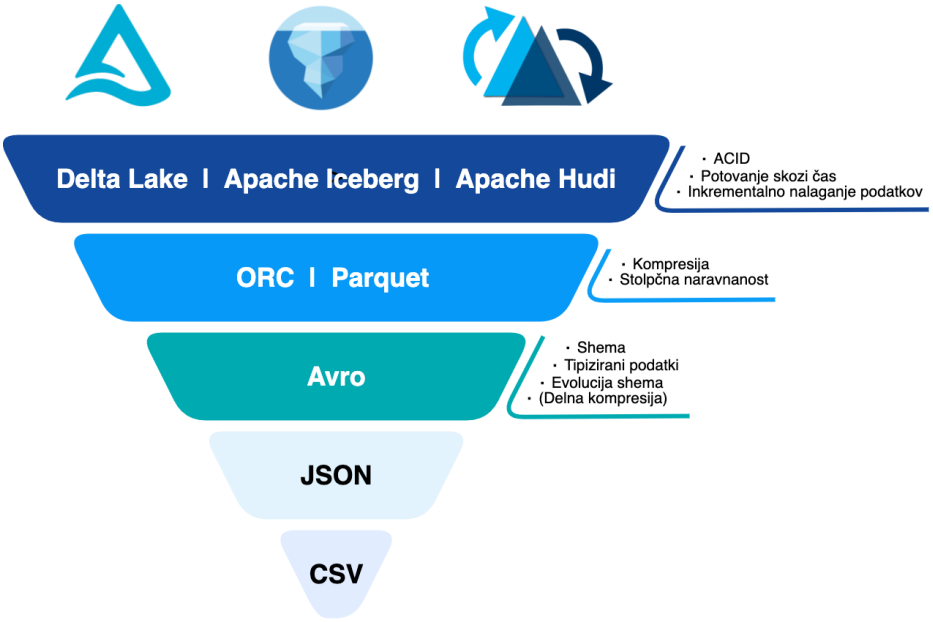
Podatkovna jezera predstavljajo jedro sodobnih podatkovnih arhitektur za shranjevanje podatkov v izvorni, neobdelani obliki, ki v ciljno hrambo vstopajo bodisi kot paketi (*angl.* ***batches***) bodisi kot podatkovni tokovi (*angl. data* ***streams***). V praksi so se izkazala za uporabna predvsem zato, ker podpirajo shranjevanje velike količine podatkov raznovrstnih tipov in različnih velikosti v svoji **izvorni obliki**. Pomemben premik v paradigmi je prehod iz pristopa *"Schema-on-Write"*, kjer je bilo potrebno podatke prilagoditi vnaprej določeni shemi že ob shranjevanju, na pristop *"Schema-on-Read"*, pri katerem se shema uporabi šele ob branju podatkov. Ta pristop omogoča večjo prilagodljivost in enostavnejše delo z nepredvidljivimi ali hitro spreminjajočimi se podatkovnimi strukturami [1,2].

S pomočjo pristopa ELT so podatki iz operativnih sistemov preneseni neposredno v podatkovna jezera, brez predhodne transformacije (T), pri čemer se transformacije izvajajo šele ob branju podatkov. Namesto uporabe zunanjega sistema za izvedbo transformacij podatkov ali izvedbe transformacij pred nalaganjem podatkov, tj. pristop ETL, se transformacije izvajajo znotraj podatkovnega jezera po potrebi. Takšen pristop je botroval t. i. **medaljonski arhitekturi** *(angl.* ***medallion architecture***), kjer imamo znotraj jezera več navideznih con shranjevanja podatkov, tj. bronasta, srebrna in zlata cona, pri čemer podatke v svoji izvorni obliki zmeraj shranjujemo v bronasti coni ter jih nato z enkratno ali večkratno transformacijo (T iz ELT) počistimo, obogatimo in uredimo za srebrno in zlato cono. Prednost takšnega pristopa je izkoriščanje zmogljivosti oblačnih podatkovnih jezer in orodij za obdelavo podatkov, kot so **Amazon S3** (**AWS**), **Google Cloud Storage** (**Google Cloud Platform**) in **Azure Data Lake Storage** (**Microsoft Azure**).

Podatkovna jezera pogosto nimajo urejenega sistema za upravljanje shem, ne podpirajo transakcij, nimajo dobre podpore za analitiko in niso primerna za strukturirane poizvedbe. Omenjeni izzivi so bili v preteklih letih naslovljeni z vpeljavo **podatkovnih košišč** v podatkovne arhitekture, ki v hibridni rešitvi združujejo lastnosti podatkovnih jezer in podatkovnih skladišč. Košišča omogočajo večji nadzor nad upravljanjem podatkov, pri čemer temeljijo na odprtih podatkovnih formatih z neposrednim dostopom preko poizvedb SQL in podporo za enotno obdelavo podatkov za podatkovno analitiko ter strojno učenje. Rešujejo predvsem izziv zastarelih, izgubljenih in neobvladljivih podatkov in pri tem ohranjajo nizke stroške ter interoperabilnost [1,2,9].

Ključno vlogo pri vzpostavitvi podatkovnih košišč igrajo **odprti tabelarični formati** (*angl.* ***open table formats***), ki omogočajo vzpostavitev košišča iz podatkovnega jezera ter tako nadgradijo podatkovno jezero z zmožnostmi podatkovnega skladišča. Najvidnejši predstavniki so **Delta Lake**, **Apache Hudi** in **Apache Iceberg** (Slika 1). Ti temeljijo na **odprtih datotečnih formatih** (*angl.* ***open file formats***), kot so **Apache Avro**, **Apache Parquet** in **Apache ORC**. Odprti tabelarični formati omogočajo ključne funkcionalnosti, kot so na lastnostih ACID temelječe transakcije, možnost potovanja skozi čas (*angl. time travel*) med poizvedovanjem, podpora za evolucijo sheme,

inkrementalno nalaganje podatkov in optimizacija shranjevanju ter branju podatkov [1]. Hkrati omogočajo prilagodljivost glede uporabe različnih **poizvedovalnih/računskih pogonov/mehanizmov** – **Presto**, **Trino**, **Apache Spark**, **Apache Flink** in drugi – za obdelavo podatkov na istem naboru podatkov glede na konkretne primere uporabe. S tem igra uporaba odprtih formatov pomembno vlogo pri demokratizaciji uporabe podatkov, zmanjšuje tveganja omejenosti na določenega ponudnika rešitev in daje večji nadzor organizacijam, kako so podatki shranjeni in uporabljeni [2,3,4].



Slika 1: Odprti formati tabel, temelječi na odprtih formatih datotek, kot osnova za podatkovno kolišče.

Hudi kombinira različne serializacijske tehnike s ciljem podpore izvedbi stolpčnih podatkovnih poizvedb ter hkratne podpore atomarnih posodobitev. Tipična uporaba Hudi je tabela, ki se posodablja s podatki CDC (*angl. Change Data Capture*) iz transakcijske podatkovne baze. Podatki se najprej shranijo v vrstični obliki, nato pa se prepakirajo v stolpčno obliko za izboljšanje učinkovitost poizvedb. Hudi omogoča poizvedovanje po najnovejših podatkih in podpira analitične ter transakcijske posodobitve. Podobno posodobitve v realnem času ter učinkovite poizvedbe omogoča tudi Iceberg, pri čemer ta posebej izstopa s podporo za evolucijo shem in upravljanje tabel pri velikosti podatkov na ravni petabajtov. Delta pa je zasnovan kot dodatna plast na oblaku temelječimi podatkovnimi jezeri, kot je Amazon S3, kjer z uporabo transakcijskega dnevnika omogoča lastnosti ACID, časovno sledenje podatkov ter bistveno hitrejše metapodatkovne operacije pri velikih tabelah [1,10].

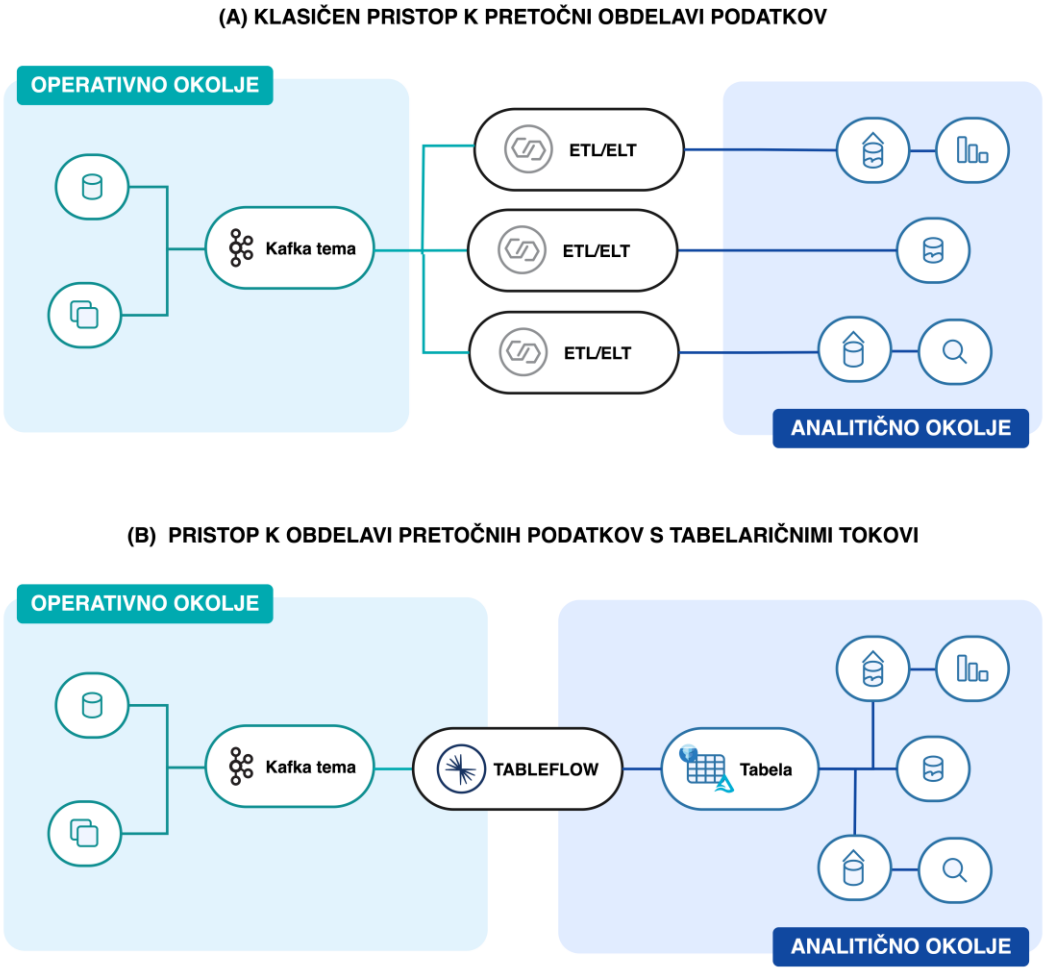
Pomemben trend je vedno tesnejša integracija podatkovnega inženirstva, podatkovne znanosti in umetne inteligence. Napredni sistemi kolišč ter odprti formati omogočajo, da podatkovni znanstveniki in inženirji uporabljajo isti nabor podatkov za analitiko, napovedne modele in AI rešitve brez podvajanja ali premikanja podatkov. Integracija z orodji za strojno učenje, kot sta **TensorFlow** in **PyTorch**, ter platformami za **MLOps**, kot so **MLflow**, **Vertex AI** (Google Cloud), **SageMaker** (AWS) in **Azure Machine Learning**, omogoča celoten življenjski cikel modelov – od zbiranja podatkov, učenja, ovrednotenja, do uvajanja v produkcijo. Ta povezava ustvarja inteligentne podatkovne platforme, kjer so podatki, analitika in AI neločljivo povezani, kar omogoča proaktivno prilagajanje procesov na podlagi napovedi in avtomatiziranega odločanja. Tukaj zaseda posebno mesto prav **Apache Spark** – porazdeljen računski pogon, ki združuje paketno in pretočno obdelavo podatkov, omogoča izvajanje poizvedb SQL (**Spark SQL**), vključuje knjižnice za strojno učenje (**MLlib, PySpark**), grafovno obdelavo (**GraphX**) ter integracijo z različnimi podatkovnimi viri. Zaradi teh lastnosti Spark pogosto deluje kot *"hrbtenica"* podatkovnih platform, kjer povezuje podatkovna jezera, kolišča in analitična orodja.

3 Tabelarični podatkovni tokovi (Tableflow)

Tabelarični podatkovni tokovi rešitve Tableflow predstavljajo nedavno inovacijo v sodobnih podatkovnih arhitekturah, ki omogoča neposredno **preslikavo neprekinjenih tokov dogodkov**, kot so **Kafka teme**, **v odprte tabelarične formate**, kot sta **Iceberg** in **Delta**. Uvedba tabelaričnih tokov stremi k združevanju realnočasovnih podatkovnih tokov z zajemom in vnosom podatkov v ciljno podatkovno hrambo oziroma pripravo podatkov za analitiko. Tako so podatki, ki tečejo v realnem času po Kafkah temah, z uporabo rešitve Tableflow dostopni tudi kot analitične tabele v podatkovnem jezeru/košišču. S tem se omogoča možnost uporabe istih podatkov tako v realnočasovnih aplikacijah kot v analitičnih poizvedbah na dosleden način [6,7].

Klasičen pristop k pretočni obdelavi podatkov tradicionalno vključuje relativno kompleksne procese za prenos podatkov iz operativne v analitične sisteme. Potrebna je vzpostavitev ustrezne infrastrukture (tipično ustrezno konfiguriranje povezovalnikov za Kafko) ter implementacija nizov procesov za ustrezno pripravo podatkov. Ti procesi vključujejo transformacijo dogodkov v tabelarične formate, skrb za sheme, vključno z uporabo registrov shem in upravljanjem evolucije shem, ter neprestano združevanje množice majhnih datotek, ki nastanejo zaradi kontinuiranega toka, s ciljem ohranjanja sprejemljive hitrosti poizvedb. Če se dogodki pretakajo iz procesov CDC, je treba dodatno materializirati in uporabiti te spremembe za pridobivanje relevantnega stanja podatkov. Vse to zahteva veliko inženirskega truda in računskih virov, pri čemer so takšni pristopi nagnjeni k napakam, (le) za to, da se tok podatkov pretvori v obliko, uporabno za hrambo v podatkovnem jezeru in analitiko [7].

Za razliko od klasičnega pristopa k pretočni obdelavi podatkov pristop z uporabo tabelaričnih tokov združuje operativno in analitično okolje, s čimer se odpravlja zgodovinska ločnica med njima, kot je ilustrirano na Sliki 2. Namesto ročnega branja podatkov iz Kafka teme in nalaganja podatkov v podatkovno hrambo, Tableflow poenoti tok in tabelo. Isti podatki so sočasno dostopni kot podatkovni tokovi in kot tabela, brez potrebe po dodatnih procesih ETL. Pri tem funkcionalnosti rešitve Tableflow znotraj ekosistema Confluent v celoti poskrbijo za avtomatizacijo procesov, ki jih je treba pri klasičnem pristopu ročno obvladovati. Tableflow prinaša inovacije v Confluentovi shrambi **Kora**, na osnovi katerih so Kafka teme neposredno materializirane v datotečnem formatu **Parquet**, ki oblikujejo bodisi tabelo **Iceberg** bodisi tabelo **Delta**. Pri tem se sheme ne podvajajo, prav tako jih ni treba ročno preslikovati. Namesto tega rešitev Tableflow uporablja register shem kot vir resnice in poskrbi za skladnost shem ter njihovo evolucijo brez prekinitev delovanja ob morebitnih spremembah. Posledično ni potrebnega ročnega ujemanja polj in tipov, kar v klasičnem pristopu povzroča izzive ob vsaki spremembi v izvorni operativni aplikaciji. Pri pristopu z uporabo tabelaričnih tokov se pravila kakovosti podatkov uveljavljajo v podatkovnem toku pri izvoru, nezdružljivi podatki pa so zavrnjeni zgodaj. Prav tako rešitev Tableflow neprekinjeno v ozadju združuje manjše datoteke, da zagotovi učinkovitost bralnih poizvedb na nastalih tabelah, kar je z uporabo povezovalnikov v sklopu klasičnih pristopov težko doseči v realnem času. Razlika je tudi v dostopu do podatkov, in sicer pri tradicionalnem pristopu so isti podatki razpršeni v dveh sistemih, v Kafki za podatkovne tokove in v podatkovnem jezeru, na primer Hadoop HDFS ali S3, za tabelo, medtem ko pristop s tabelaričnimi tokovi omogoča, da porabniki v operativnem okolju še naprej berejo realnočasovne podatkovne tokove (Kafka API), analitična orodja pa istočasno berejo Iceberg/Delta tabelo preko poizvedb SQL. Pri tem obe predstavitvi temeljita na enem samem, usklajenem naboru podatkov. To je možno zaradi dualnosti tok-tabela, kjer sta tok in tabela dva pogleda na iste podatke, pri čemer rešitev Tableflow to dualnost realizira na ravni hrambe podatkov [7].



Slika 2: Primerjava med (A) klasičnim pristopom k pretočni obdelavi podatkov in (B) pristopom s tabelaričnimi tokovi (povzeto po [7]).

Vpeljava tabelaričnih tokov v podatkovne arhitekture, tj. neposredno preslikovanje Kafka tem v tabele Iceberg/Delta, na področju podatkovnega inženirstva prinaša številne ključne prednosti, ki jih lahko povzamemo v naslednjih vidikih:

- ***Kompleksnost podatkovne arhitekture***. Kafka teme se neposredno predstavijo kot Iceberg/Delta tabele, kar močno poenostavi podatkovno arhitekturo in odpravi potrebo po kompleksnih prilagojenih procesih ETL. Tabelarični tokovi tako omogočajo t. i. pristop *"Zero ETL"*, saj so podatki iz podatkovnih tokov na avtomatiziran način brez dodatnih procesov na voljo kot analitične tabele z minimalno inženirskega truda za organizacije. Pri tem je poskrbljeno za skladnost sheme med podatkovnim tokom in tabelo – vsaka Kafka tema, vključena v tabelarične tokove, mora imeti definirano shemo, ki se nato uporablja tudi pri ustvarjanju tabele. S tem se zagotovi strukturiranost in konsistentnost podatkov v tabeli, avtomatsko pa se podpira tudi evolucija sheme z ohranjanjem združljivosti. Rešitev Tableflow avtomatsko upravlja formatiranje in vzdrževanje tabel. Dogodke v vhodnih formatih sproti pretvarja v stolpčne datoteke Parquet, ki tvorijo podlago tabelam Iceberg/Delta. Pri neprestanem dotoku podatkovnih tokov nastajajo številne majhne datoteke, ki jih Tableflow samodejno združuje in optimizira, briše zastarele datoteke ter tako ohranja hitro branje in učinkovito porabo prostora. Vse to poteka brez podvajanja podatkov, saj Kafka in tabela delita isto osnovo podatkov, tabela pa je običajno dostopna kot *samo-za-branje* (*angl. read-only*), kar zagotavlja konsistentnost in integriteto. Z vsem tem se z vpeljavo tabelaričnih tokov zmanjša inženirski napor, s čimer se razbremenijo ekipe podatkovnih inženirjev, odpravljajo se

nekateri podvojeni koraki obdelave in hrambe podatkov, kar se odraža v manj kompleksni podatkovni arhitekturi napram klasičnim pristopom k pretočni obdelavi podatkov [6,7].

- ***Aktualnost in relevantnost podatkov za analitiko***. Ker se dogodki iz operativnih sistemov v realnem času pretakajo v odprte tabelarične formate, so podatki iz operativnih virov v podatkovnem jezeru/košču neprestano sveži, brez bistvene zakasnitve med dogodkom in njegovo razpoložljivostjo za poizvedovanje. To omogoča hitrejši dostop do svežih podatkov, saj podatki prispejo, so obdelani in na voljo v skoraj realnem času za takojšnje poizvedbe. Prav tako z vpeljavo tabelaričnih tokov zaradi dualnosti tok-tabela ne prihaja do izgube konteksta in sheme podatkov. Pri tem Kafka ne služi le pretočni obdelavi, temveč tudi kot mehanizem za zajem in vnos podatkov. Analitiki in podatkovni znanstveniki lahko tako obravnavajo aktualne in relevantne podatke, kar bistveno zmanjša t. i. *"Time-to-Insight"* in poveča vrednost podatkovne analitike, kakor tudi pospešuje uvajanje umetnointeligenčnih rešitev ter izboljšuje proces podpore odločanju [6,7].
- ***Integracija z obstoječimi orodji in okolji***. Ker sta Iceberg in Delta odprta standarda, Tableflow močno zmanjšuje odvisnost od posameznih orodij in ponudnikov. Iceberg ali Delta imata široko podporo v analitičnih ogrodjih in orodjih, kot so Spark, Trino, Flink, Snowflake in Databricks. S preslikovanjem Kafka tem v Iceberg/Delta se izognemo omejenosti na enega ponudnika rešitev. Podatki so shranjeni v odprti obliki, ki jo lahko bere katerikoli ustrezen pogon, ki je združljiv z odprtimi tabelami. Tabelarični tokovi tako omogočajo, da isti nabor podatkov takoj izkoriščajo različna orodja znotraj različnih okolij, ki lahko dostopajo do tabel brez posebnih prilagoditev [6,7].

4 Tehnološki pregled rešitve Tableflow

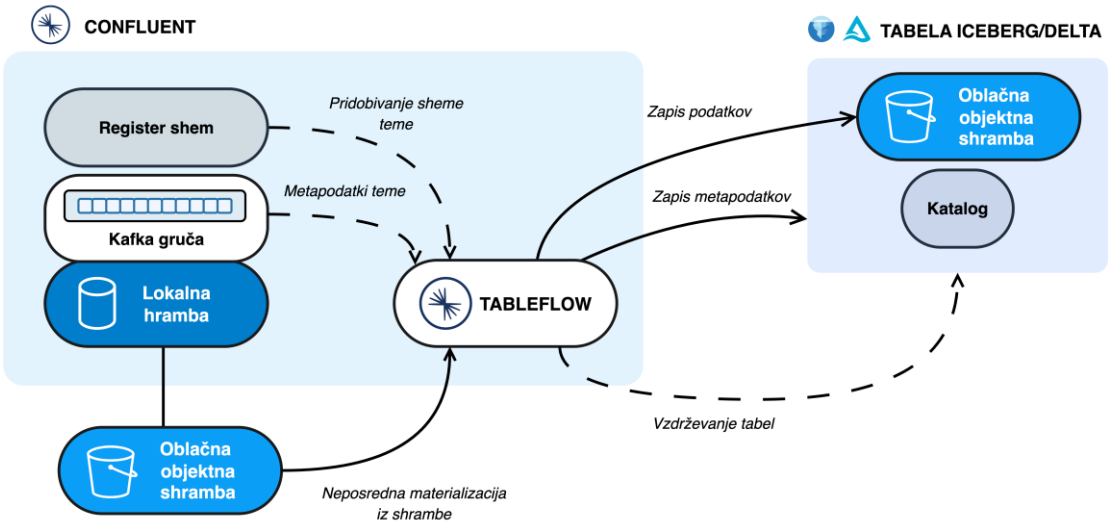
Rešitev Tableflow je storitev znotraj ekosistema Confluent za pretočno podatkovno platformo in je postala splošno dostopna v začetku leta 2025. Gre za popolno vodeno storitev v okviru **Confluent Cloud**, ki nadgrajuje jedro Apache Kafka z novim slojem za materializacijo podatkov. Pri tem Tableflow ni zgolj še en povezovalnik, temveč je globoko integriran v platformo, tako v podatkovno shrambo kot tudi v upravljanje shem in obdelavo tokov. Pri tem Tableflow predstavlja korak v smeri, da podatkovna platforma učinkovito služi tako operativnim aplikacijam kakor analitičnim potrebam na skupnih podatkih v odprtem formatu [6].

V Confluent Cloud konzoli, tj. preko grafičnega uporabniškega vmesnika, ali preko API/CLI lahko podatkovni inženir na izbranih Kafka temah omogoči Tableflow v zgolj nekaj konfiguracijskih korakih. Za vsako takšno temo sistem avtomatsko vzpostavi spremljajočo tabelo, pri čemer podatkovni inženir predhodno izbere željen format, ki se sproti posodablja z novimi dogodki. Tableflow se opira na odprte tabelarične formate, trenutno na Iceberg (splošno dostopen) in Delta (v odprti predogledni fazi). V ozadju Confluent Cloud upravlja infrastrukturo, kar pomeni, da uporabniku ni treba skrbeti za strežnike, shrambo datotek ali usklajevanje – Tableflow deluje kot platformna funkcionalnost. Trenutno je Tableflow na voljo izključno v okolju Confluent Cloud pri določenih ponudnikih storitev v oblaku, na primer AWS, in ni del klasične namestitve Confluent Platform On-Premises, kar poudarja njegovo odvisnost od upravljalnega okolja Confluent in novega shranjevalnega sloja. Kot del ponudbe Confluent se Tableflow obračunava glede na čas uporabe teme in obseg obdelanih podatkov (GB), podobno kot njihove preostale oblačne storitve [6].

Tableflow je zasnovan kot večslojna nadgradnja Kafka infrastrukture. Ključna inovacija je v nadgradnji shranjevalnega podsistema ***Kora Storage Layer***. Kora, ki že sicer skrbi za t. i. *"Tiered Storage"*, tj. premikanje starejših dogodkov Kafka teme na poceni objektno shrambo, je razširjena tako, da lahko iz dogodkov v Kafka temi sproti ustvarja datoteke Parquet. S tem Kafka dogodki postanejo gradniki tabele Iceberg/Delta. Namesto da bi ločen povezovalnik zunaj Kafke bral dogodke in pisal nove kopije v datotečni sistem, Tableflow doseže ***"zero-copy"* materializacijo**. Isti podatki, ki jih Kafka hrani (v Confluent Cloud so Kafka podatki v S3), se predstavijo v obliki datotek Parquet in pripadajočih metapodatkov tabel. Komponenta *Metadata Materializer* v Tableflow se

navezuje na *Confluent Schema Registry*, iz katerega dobi shemo dogodkov, in na podlagi te ustvarja Iceberg metapodatke (manifesti, razvezave) ali transakcijske zapise Delta. Sproti se izvajajo pretvorbe podatkovnih tipov med Kafka shemo in tabelo ter sledi pravilom združljivosti shem (na primer, če se shema nadgradi z novim poljem, se ta sprememba zabeleži v Iceberg/Delta shemi, skladno z nastavitvami kompatibilnosti) [6,7].

Podrobnejša tehnična arhitektura in pretok podatkov s pomočjo Tableflow je prikazan na Sliki 3. Ko dogodki prihajajo v Kafka temo, jih shranjevalni sloj rešitve Tableflow zajeme v obliki mini-datotek odprtega datotečnega formata Parquet na objektni shrambi. Zaradi zagotavljanja nizke zakasnitve se ti zapisi lahko sprva zapišejo kot manjše datoteke. Vzporedno pa procesi v ozadju izvajajo združevanje – manjše datoteke združujejo v večje posnetke datotek, kar izboljšuje poizvedbe. Tableflow s tem rešuje znano omejitev ločenih povezovalnikov, kjer se je bilo treba odločati med hitrostjo prenosa (majhne datoteke, a slabše poizvedbe) in učinkovitostjo poizvedb (velike datoteke, a višja zakasnitev). Z integracijo v shrambo Kafka se lahko optimizira oboje – piše se pogosto za sproten dotok, a v ozadju optimizira za branje. Ko se zapiše nov segment/tabela, *Metadata Materializer* posodobi posnetek stanja Iceberg ali dnevniški zapis Delta, tako da je nov paket podatkov takoj viden analitičnim porabnikom. Kot posledica takšna arhitekturna zasnova povzroča, da obstaja en sam vir podatkov – Kafka posredniki (*angl. broker*) in objektna shramba skupaj hranijo podatke, tabela in Kafka tema pa predstavljata *"dve strani istega kovanca"*. Porabniki lahko torej ali berejo celoten tok prek Kafka API (za minimalno zakasnitev) ali pa dostopajo do posameznih dogodkov oziroma celotne zgodovine preko poizvedb SQL na tabeli (za *ad-hoc* analize) [11].



Slika 3: Tehnična arhitektura rešitve Tableflow v ekosistemu Confluent (povzeto po [11]).

V Confluentoven ekosistemu se Tableflow povezuje s storitvami **kataloga podatkov**, ki vsebuje vgrajen katalog **Iceberg REST** za dostop do tabel. To pomeni, da lahko uporabnik v kateremkoli združljivem pogonu, na primer Spark, Trino, Flink, Presto, ustvari Iceberg tabelo z uporabo URL-ja Confluentovega kataloga in API ključev ter tako poizveduje neposredno po Kafkah podatkih v tabelarni obliki. Hkrati pa podpira integracijo z zunanjimi kataloškimi storitvami, ko so **AWS Glue**, **Snowlake/Apache Polaris** ipd., da se tabela lahko registrira tudi v obstoječe metapodatkovne sisteme organizacije [6,7].

5 Primeri uporabe in integracija s sodobnimi podatkovnimi arhitekturami

Eden od pogostih primerov uporabe je obdelava podatkov o uporabniških interakcijah iz več različnih aplikacij, kot so mobilne aplikacije in spletni brskalniki. Ti dogodki se v realnem času pošiljajo v Kafka temo in so lahko ob podpori Tableflow nemudoma preneseni in dostopni kot analitične tabele v okolju Databricks. Tako lahko analitične ekipe gradijo nadzorne plošče in analize v realnem času brez ločenih postopkov ETL in brez skrbi glede zadrževanja (*angl. retention policy*) podatkov v Kafki. Podoben pristop se uporablja pri aplikacijah strojnega učenja, kjer je cilj ne le obdelava dogodkov, ampak tudi njihovo trajno shranjevanje za kasnejše ponovno učenje modelov. Če bi se pri tem zanašali le na podatke v Kafka temi, bi bili omejeni z nastavljenim zadrževanjem in potencialno nepopolno zgodovino. Tableflow omogoča, da se vsi dogodki zapisujejo v podatkovno jezero v tabelo Delta z neomejeno hrambo, kar daje podatkovnim znanstvenikom popoln dostop do zgodovinskih podatkov ter hkrati vso moč podatkovnih kolišč.

Za bolj poglobljeno ponazoritev si poglejmo primer podjetja, ki uporablja platformo **Databricks** kot svoje podatkovno kolišče in osrednjo platformo za analitiko ter strojno učenje. Databricks združuje lastnosti podatkovnega jezera in skladišča, hkrati pa z **Unity Catalog** nudi centralizirano upravljanje metapodatkov, varnosti in sledljivosti. V takšni arhitekturi Tableflow materializira podatke iz Kafka tem neposredno v tabelo Delta, shranjeno v zunanjem oblaku (npr. Amazon S3) v načinu *Bring-Your-Own-Storage*. Databricks to tabelo registrira kot zunanjo tabelo, kar pomeni, da se tabela Delta piše direktno v zunanje vedro (*angl. bucket*) in je potem na voljo kot zunanja tabela v okolju Databrick [6,7]. To omogoča izvajanje poizvedb SQL s pomočjo **Spark SQL**, integracijo z ML cevovodi prek **PySpark** ter uporabo funkcionalnosti BI.

V številnih primerih je pristop **samo Tableflow** (t. i. *"Zero ETL"*) povsem zadosten – podatki se iz Kafke takoj zapišejo v odprt tabelarični format (tj. Delta), kar omogoča analitikom in podatkovnim znanstvenikom neposreden dostop, pri čemer se prečiščevanje in obogatitev podatkov izvedeta šele znotraj okolja podatkovnega kolišča (tj. ELT in prenos podatkov iz bronaste v srebrno ali zlato cono). Tak pristop je optimalen, ko ni nujne potrebe po takojšnjem odstranjevanju občutljivih podatkov, deduplikaciji ali zapletenih agregacijah pred vnosom v jezero.

Kadar pa so potrebni **kritični postopki že pred materializacijo** (*"Shift Left"*), se v arhitekturo lahko vključi procesor podatkovnih tokov, kot je **Flink** ali **Kafka Streams**. Flink je primernejši za kompleksne transformacije z zahtevno obdelavo dogodkov po času dogodka (*angl. event time*), uporabo naprednih časovnih oken itn. Kafka Streams pa je primernejša izbira, kadar so transformacije tesno vezane na domeno aplikacije, obsegajo manjša stanja in želimo logiko izvajati znotraj obstoječih mikrostoritev brez dodatne infrastrukture. V obeh primerih transformiran tok preide v novo Kafka temo, iz katere Tableflow izvede materializacijo prav tako v novo odprto tabelo.

V praksi se lahko uporabi **dvojna pot**: surova pot, kjer Tableflow neposredno materializira izvorni tok v **bronasto** tabelo, in *curated* pot, kjer se podatki predhodno obdelajo (npr. s Flink ali Kafka Streams) ter nato materializirajo v **srebrno** tabelo. Tak pristop zagotavlja popolno arhivsko sled surovih podatkov, hkrati pa ponuja prečiščene in optimizirane podatke za hitrejše poizvedbe ter zmanjšanje stroškov obdelave v okolju Databricks. Takšna arhitektura omogoča jasno ločitev odgovornosti: **Flink** ali **Kafka Streams za transformacijo, Tableflow za materializacijo, Databricks** skupaj s **Spark za analitiko in strojno učenje**.

Ključna prednost uporabe Tableflow je torej ta, da podatki iz realnočasovnih tokov postanejo neposredno del *lakehouse* ekosistema v obliki odprtih tabel, kot je Delta, kjer so takoj dostopni prek standardnih poizvedb SQL ter brez težav vključeni v obstoječe analitične procese in procese strojnega učenja. Namesto da bi jih morali nekoč brati neposredno iz Kafke – s čimer je bil obseg zgodovinskih podatkov morda omejen zaradi retencijskih politik in so bile kompleksne analize otežene – ali pa graditi lastne cevovode ETL, ki so zahtevali dodatno infrastrukturo, vzdrževanje in uvajali latenco, so ti podatki lahko zdaj neomejeno dostopni, konsistentni in optimizirani za uporabo. Prednost tega, da so sedaj tokovni podatki del podatkovnih kolišč z odprtimi tabelami, je med drugim podpora ACID transakcijam, ki zagotavljajo zanesljivo in predvidljivo stanje podatkov tudi ob sočasnih obdelavah. V klasičnih podatkovnih jezerih (brez odprtih tabel) je bilo to težko zagotoviti, ker si delal z datotekami (Parquet, ORC) brez transakcijskega dnevnika – kar je vodilo do nepopolnih naborov podatkov. Prav tako je prednost kolišč

ta, da omogočajo napredne mehanizme optimizacije poizvedb, kot so združevanje datotek (*angl. file compaction*), indeksiranje in fizično urejanje (*angl. z-ordening, sorting, clustering*) po pogosto uporabljenih ključih, kar bistveno skrajša čas poizvedb in zniža stroške. V jezerih brez te logike je bil dostop do podatkov pogosto drag, ker si moral prebrati ogromno nepotrebnih datotek. Odprti formati z metapodatkovnim slojem vedo, **kje** so relevantni podatki. Košišča hkrati omogočajo t. i. *time travel*, torej vpogled v pretekla stanja tabele in povrnitev na prejšnje verzije, kar podpira revizije, reproducibilnost modelov strojnega učenja in obnovo po napakah. Lahko rečeš: *"Daj mi stanje tabele, kot je bilo včeraj ob 15:00"* ali *"pri verziji commit-a št. 57"*. Prej si moral te scenarije reševati z ročnimi kopijami datotek (verzije map), kar je bilo težko in neučinkovito. Z odprtimi tabelami imaš to vgrajeno. Takšna arhitektura ne združuje le tokovnih in paketnih podatkov v enoten analitični prostor, temveč omogoča tudi enostavno in neposredno učenje modelov na celotni zgodovini dogodkov, kar vodi v hitrejše odločitve, boljše napovedne modele in manj tehničnega dolga.

6 Primerjalna analiza rešitve Tableflow s klasičnimi pristopi

Rešitev **Tableflow** v okolju Confluent Cloud je zasnovana za neposredno materializacijo Kafka tem v odprte tabelarične formate, kot sta Iceberg in Delta, z visoko stopnjo avtomatizacije. Njena naloga je zagotoviti, da so podatki, ki prihajajo iz tokov, sočasno dostopni kot analitične tabele, optimizirane za poizvedbe, brez potrebe po ročnih postopkih ETL. Da bi objektivno ocenili njen doseg in prednosti, jo je smiselno primerjati s pristopi, ki zasledujejo enak cilj – materializacijo pretočnih podatkov v tabelarno obliko.

Kafka Connect Object-Storage Sink predstavlja klasičen pristop materializacije, kjer se Kafka tokovi zapisujejo v podatkovna jezera in objektne shrambe (npr. S3, HDFS, Azure Data Lake/Blob, GCS) kot Parquet/Avro/ORC. Prednost je enostavna konfiguracija in široka podpora ciljnih sistemov, slabost pa, da ponor (*angl. sink*) praviloma ne vzpostavi metapodatkov odprtih tabel (Iceberg/Delta) in ne izvaja samodejne optimizacije datotek, kot so združevanje majhnih datotek (*angl. small files compaction*), gručanje po pogosto uporabljenih stolpcih (*angl. clustering*) ali dodajanje particij. Posledica tega je, da ob neprestanem zapisovanju majhnih paketov podatkov v shrambo nastajajo tisoči majhnih datotek, kar pri analitičnih poizvedbah povzroča veliko število bralnih operacij in podaljša odzivni čas. Na primer, poizvedba v Spark SQL, ki bere tabelo z 10 milijoni zapisov, razpršenih v 200.000 majhnih datotek, se lahko izvaja več minut, medtem ko enaka poizvedba nad optimizirano tabelo, kjer so datoteke združene in ustrezno particionirane (npr. po datumu ali regiji), zaključi v nekaj sekundah. Ker ponor ne upravlja particij, indeksov ali metapodatkov, mora uporabnik samostojno izvajati naknadne postopke optimizacije, pogosto z dodatnimi procesi ETL ali orodji (npr. Spark Jobs). Upravljanje evolucije shem in registracija tabel v analitičnih okoljih tako ostane na strani uporabnika ali pa zahteva integracijo z ločenimi metapodatkovnimi storitvami.

Apache Hudi DeltaStreamer je orodje, ki omogoča pretok podatkov iz Kafke neposredno v Hudi tabele, z osnovno podporo za evolucijo shem in optimizacijo podatkovnih datotek. Je odprtokodna rešitev, ki zahteva samostojno namestitev, konfiguracijo in vzdrževanje infrastrukture. Čeprav ponuja integrirano obdelavo in zapis v format, primeren za analitiko, je upravljanje sistema zahtevnejše in manj primerno za ekipe, ki nimajo izkušenj z administracijo Hadoop/Spark okolij.

Apache Flink v kombinaciji s **SQL** in ustreznim povezovalnikom *sink* za Iceberg ali Delta omogoča, da se podatki iz Kafka tokov neposredno pretvorijo v odprt tabelarični format. Prednost te rešitve je visoka fleksibilnost – uporabnik lahko izvede kompleksne transformacije, filtriranje in obogatitev podatkov že pred zapisom v tabelo. Slabost pa je višja zahtevnost pri postavitvi in upravljanju okolja, potreba po ločeni Flink infrastrukturi in dodatno kodiranje oziroma konfiguracija za upravljanje shem ter optimizacijo datotek.

Tableflow vse zgoraj naštete korake poenostavi z oblačno, v celoti upravljano storitvijo. Integriran je s **Confluent Schema Registry**, kar omogoča avtomatsko upravljanje in evolucijo shem brez prekinitve delovanja.

Optimizacijo datotek (združevanje majhnih datotek, odstranjevanje zastarelih segmentov) izvaja samodejno v ozadju, kar zagotavlja stalno visoko učinkovitost poizvedb. Podpira formate Iceberg in Delta ter omogoča *"Zero ETL"* pristop, kjer je isti nabor podatkov dostopen tako preko Kafka API kot tudi preko poizvedb SQL na materializirani tabeli.

Za boljši pregled nad razlikami med pristopi, ki omogočajo materializacijo podatkovnih tokov v analitične tabele, podajamo primerjalno analizo ključnih lastnosti in zmogljivosti posamezne rešitve v Tabeli 1.

Tabela 1: Primerjalna tabela pristopov materializacije pretočnih podatkov v tabelarno obliko.

Pristop / Lastnost	**Kafka Connect Object-Storage Sink (S3/HDFS/ADLS/GCS)**	**Apache Hudi DeltaStreamer**	**Flink SQL + Iceberg/Delta Sink**	**Tableflow**
Stopnja avtomatizacije	Srednja – brez avtomatskih metapodatkov	Srednja	Nizka	Visoka (*"Zero ETL"*)
Upravljanje shem	Omejeno / ročno	Osnovno podprto	Ročna/konfiguracijska	Avtomatsko (Schema Registry)
Podpora odprtim tabelam	Ne (le datoteke)	Da – Hudi	Da – Iceberg/Delta	Da – Iceberg/Delta
Optimizacija za poizvedbe	Ne	Delno/ročno	Delno/ročno	Da – samodejno
Kompleksnost	Srednja	Visoka	Visoka	Nizka
Prilagodljivost obdelave	Nizka	Srednja	Zelo visoka	Omejena (razširitev s Flink/Streams)

Čeprav **Kafka Streams** in **ksqlDB** niso neposredni konkurenti Tableflow, igrata pomembno vlogo kot komplementarni orodji v arhitekturah, kjer je potrebna kompleksna transformacija podatkov pred materializacijo. Primer smo predstavili tudi v prejšnjem poglavju. **Kafka Streams** omogoča popolnoma prilagojene procesne tokove na ravni kode, saj v lastni aplikaciji zapišeš logiko, ki prebere podatke iz ene ali več Kafka tem, jih transformira in nato rezultate pošlje v drugo Kafka temo. Od tam naprej lahko to temo preko povezoavlnika *sink* (npr. Kafka Connect S3/HDFS Sink) ali lastne kode zapisujemo v podatkovno jezero. Streams sam po sebi ne materializira podatkov v odprti tabelarični format (Iceberg, Delta) niti ne vzpostavi metapodatkov in optimizacij za poizvedbe, zato je ta korak treba izvesti z dodatnimi orodji ali kodo. **Tableflow** ta zadnji del – "**tema → optimizirana tabela v *odprtem tabelaričnem formatu***" – izvede avtomatsko in v oblaku, brez dodatnih korakov. **ksqlDB** pa ponuja preprost vmesnik SQL za filtriranje in agregacijo podatkov v realnem času ter se pogosto uporablja v scenarijih, kjer želimo hitro definirati transformacije brez razvoja lastne kode. V praksi se torej lahko uporabi kombinacija teh orodij s Tableflow, kjer Streams ali ksqlDB pripravita podatke, Tableflow pa jih nato samodejno materializira v odprtem tabelaričnem formatu.

7 Sklep

V prispevku predstavljamo novo rešitev Tableflow ekosistema Confluent, ki s svojimi inovacijami s tabelaričnim tokom omogoča neposredno predstavitev Kafka tem v odprtih tabelaričnih formatih Iceberg/Delta. Sledenje je ključno za integracijo pretočnih podatkov v sodobne arhitekture, temelječe na podatkovnih jezerih, saj odprti formati omogočajo, da lahko več različnih analiznih pogonov sočasno dostopajo do podatkovne tabele na objektni shrambi. S tem Tableflow omogoča, da podatkovna platforma učinkovito služi tako operativnim aplikacijam kakor analitičnim potrebam na skupnih podatkih v odprtem formatu.

Vpeljava tabelaričnih tokov rešitve Tableflow v podatkovno arhitekturo se odraža v številnih prednostih tako za podatkovne inženirje kot za podatkovne analitike, predvsem v manj kompleksni podatkovni arhitekturi, ki zmanjšuje potrebo po kompleksnih in k napakam nagnjenih procesih ETL, hitrejšem dostopu do aktualnih in relevantnih podatkov, pri čemer zaradi dualnosti tok-tabela ne prihaja do izgube konteksta in sheme podatkov ter enostavnosti integracije z obstoječimi orodji in okolji, pri čemer lahko isti nabor podatkov takoj izkoriščajo različna orodja znotraj različnih okolij, ki lahko dostopajo do podatkov brez posebnih prilagoditev. Podobno kot nas je evolucija sodobnih podatkovnih arhitektur pripeljala od jezer do kolišč – vse z namenom zmanjševanja kompleksnosti ter zbiranja številnih funkcionalnosti in paradigem pod eno streho – tako se tudi tabelarični tokovi ponujajo kot ena izmed takšnih evolucijskih korakov, ki pa se še morajo primerno uveljaviti in dokazati.

Literatura

[1] REIS Joe, HOUSLEY Matt. *Fundamentals of Data Engineering: Plan and Build Robust Data Systems*, O'Reilly Media, 2022.

[2] SERRA James. *Deciphering Data Architectures: Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh*, O'Reilly Media, 2024.

[3] MAZUMDAR Dipankar, GOVINDARAJAN Vinoth. *Engineering Lakehouses with Open Table Formats: Build scalable and efficient lakehouses with Apache Iceberg, Apache Hudi, and Delta Lake*, O'Reilly Media, 2025.

[4] ŠESTAK Martina, TURNAKOVIĆ Muhamed. "Nadgradnja podatkovnih jezer s pomočjo formata za shranjevanje Delta Lake", *OTS 2023 : Sodobne informacijske tehnologije in storitve : zbornik 26. konference*, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 2023, str. 243–254.

[5] ŠESTAK Martina. "Prehod na realno-časovno obdelavo podatkovnih tokov s pomočjo Kafka Streams in KSQL/ksqlDB", *OTS 2022 : Sodobne informacijske tehnologije in storitve : zbornik petindvajsete konference*, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 2022, str. 118–127.

[6] SELWAN Marc. https://www.confluent.io/blog/introducing-tableflow, *Introducing Tableflow*, 2024, obiskano 21. 7. 2025.

[7] Confluent, Inc. https://docs.confluent.io/cloud/current/topics/tableflow, *Tableflow Documentation*, 2025, obiskano 21. 7. 2025.

[8] BELLEMARE Adam. *Shift Left: Unifying Operations and Analytics With Data Products*.

[9] ČUŠ Blaž, GOLEC Darko, STRUGAR Ivan. "Data Lakehouse: Benefits in small and medium enterprises", *Journal of Innovative Business and Management*, letnik 14, številka 2, 2023, str. 1–10.

[10] ARMBRUST Michael, DAS Tathagata, SUN Liwen, YAVUZ Burak, ZHU Shixiong, MURTHY Mukul, TORRES Joseph, VAN HOVELL Herman, IONESCU Adrian, ŁUSZCZAK Alicja, ŚWITAKOWSKI Michał, SZAFAŃSKI Michał, LI Xiao, UESHIN Takuya, MOKHTAR Mostafa, BONCZ Peter, GHODSI Ali, PARANJPYE Sameer, SENSTER Pieter, XIN Reynold, ZAHARIA Matei. "Delta Lake: High-performance ACID table storage over cloud object stores", *Proceedings of the VLDB Endowment*, letnik 13, številka 12, 2020, str. 3411–3424.

[11] BUKTA Victoria, INDRASIRI Kasun. https://www.youtube.com/watch?v=js2vL5TNHB0, *Streaming Meets Governance: Building AI-Ready Tables With Confluent Tableflow and Unity Catalog*, obiskano 21. 7. 2025.

Kibernetska obramba prihodnosti: inteligentni agenti z močjo velikih jezikovnih modelov

Jani Dugonik, Damijan Novak

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
jani.dugonik@um.si, damijan.novak@um.si

V prispevku predstavljamo razvoj simulacijskega okolja za preučevanje kibernetskih napadov in obrambnih strategij z uporabo inteligentnih agentov, podprtih z velikimi jezikovnimi modeli. Okolje omogoča realistično simulacijo napadov, kot je spletno ribarjenje, kjer napadalni agenti generirajo prepričljiva sporočila, obrambni agenti pa jih analizirajo in zaznavajo znake prevare. Takšna interakcija omogoča varno testiranje v nadzorovanem okolju ter prispeva k razumevanju učinkovitosti obrambnih mehanizmov, izboljšanju algoritmov za zaznavanje prevar in izobraževanju uporabnikov o sodobnih kibernetskih grožnjah.

Ključne besede:
inteligentni agenti,
kibernetska varnost,
obrambne strategije,
simulacijsko okolje,
socialni inženiring,
spletno ribarjenje,
veliki jezikovni modeli.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.3

1 Uvod

V zadnjih letih se je področje kibernetske varnosti soočilo z izjemnim porastom kompleksnosti in raznolikosti groženj, ki ogrožajo informacijske sisteme, infrastrukturo in uporabnike. Napadi postajajo vse bolj sofisticirani, prilagodljivi in težko predvidljivi, kar zahteva nove pristope k razumevanju, testiranju in razvoju obrambnih strategij. Eden izmed obetavnih pristopov je uporaba inteligentnih agentov, ki omogočajo simulacijo realističnih scenarijev v nadzorovanem okolju, brez tveganja za dejanske sisteme.

Inteligentni agenti so napredni programski sistemi, zasnovani za delovanje v kompleksnih okoljih, kjer neprestano zaznavajo dogajanje v okolju, sprejemajo samostojne odločitve in izvajajo akcije glede na zastavljene cilje. Njihova sposobnost prilagajanja na podlagi zaznanih podatkov in rezultatov lastnega delovanja omogoča dinamično učenje in izboljševanje odzivov, kar je ključno za učinkovito modeliranje napadov in obrambnih strategij. V simulacijskih okoljih, kot so industrijski procesi ali kibernetski sistemi, agenti omogočajo varno testiranje različnih scenarijev, kar prispeva k razvoju robustnejših varnostnih rešitev.

Pomemben napredek na področju umetne inteligence, ki je bistveno prispeval k zmogljivosti inteligentnih agentov, predstavljajo *veliki jezikovni modeli* (angl. *Large Language Models*, LLM). Ti modeli, ki temeljijo na nevronskih mrežah in strojnem učenju, omogočajo razumevanje in tvorbo naravnega jezika na ravni, ki se približuje človeški komunikaciji. S tem agentom omogočajo bolj naravno, vsebinsko in učinkovito interakcijo z okoljem in uporabniki. Sodobni LLM-ji niso več omejeni zgolj na obdelavo besedil, temveč vključujejo tudi slike, videe in druge vrste podatkov, kar jih uvršča med ključne gradnike več modalnih inteligentnih sistemov.

Na področju kibernetske varnosti imajo LLM-ji posebno vlogo pri simulaciji napadov, kot je *spletno ribarjenje* (angl. *phishing*), kjer je cilj napadalca pridobiti občutljive informacije preko manipulativnih sporočil. Ti napadi temeljijo na socialnem inženiringu, kjer se izkoriščajo psihološki mehanizmi, kot so občutek nujnosti, avtoriteta ali radovednost. Sporočila, uporabljena v teh napadih, so pogosto izjemno verodostojna, kar otežuje njihovo zaznavanje. Kljub naprednim tehničnim zaščitam ostaja človeški dejavnik najšibkejši člen v verigi kibernetske varnosti. Najnovejše raziskave kažejo, da agenti, podprti z LLM-ji, omogočajo bolj proaktivno in kontekstualno zaznavanje groženj [1].

Zato je ključnega pomena razvoj simulacijskih okolij, ki omogočajo preučevanje interakcij med napadalci in branilci ter identifikacijo vedenjskih vzorcev, ki vodijo do uspešnih napadov. V tem članku predstavljamo zasnovo simulacijskega okolja, v katerem inteligentni agenti, podprti z LLM-ji, prevzemajo vloge napadalcev in branilcev. Napadalni agenti generirajo prepričljiva sporočila, medtem ko obrambni agenti analizirajo prejeta sporočila, iščejo znake prevare in se ustrezno odzivajo. Takšna interakcija omogoča realistično simulacijo, ki služi kot učni primer in raziskovalno orodje za nadaljnji razvoj varnostnih rešitev.

2 Kibernetska varnost in sodobne grožnje

Kibernetska varnost je danes eno izmed ključnih področij informacijske tehnologije, saj varuje digitalne sisteme, omrežja in podatke pred nepooblaščenim dostopom, zlorabo, krajo ali uničenjem. Z razvojem digitalne družbe in vse večjo odvisnostjo od informacijsko-komunikacijskih tehnologij se povečuje tudi število in kompleksnost kibernetskih groženj. Te grožnje niso več omejene zgolj na tehnične ranljivosti, temveč vključujejo tudi psihološke in družbene dimenzije, kar zahteva celostne in prilagodljive pristope k zaščiti.

Ena izmed najpogostejših in hkrati najuspešnejših oblik napadov je že prej omenjeno spletno ribarjenje. Gre za tehniko socialnega inženiringa, pri kateri napadalec z uporabo lažnih, a prepričljivih sporočil poskuša žrtev prepričati, da razkrije občutljive informacije, kot so prijavni podatki, številke bančnih kartic ali dostop do internih sistemov. Takšna sporočila pogosto izkoriščajo psihološke mehanizme, kot so občutek nujnosti (npr. "vaš račun bo zaklenjen"), avtoriteta (npr. "sporočilo od direktorja") ali radovednost (npr. "prejeli ste nagrado"), kar povečuje verjetnost, da bo uporabnik nasedel.

Kljub številnim tehničnim zaščitnim ukrepom, kot so požarni zidovi, sistemi za zaznavanje vdorov in večfaktorska avtentikacija, ostaja človeški dejavnik najranljivejši element v verigi kibernetske varnosti. Uporabniki pogosto niso dovolj usposobljeni za prepoznavanje prevarantskih sporočil, kar napadalci s pridom izkoriščajo. Zato postaja vse pomembnejše razvijati orodja in metode, ki omogočajo izobraževanje, ozaveščanje in simulacijo napadov v varnem okolju.

V tem kontekstu se kot izjemno uporabni izkažejo simulacijska okolja, ki omogočajo realistično modeliranje napadov in obrambnih odzivov. Takšna simulacijska okolja omogočajo preučevanje vedenjskih vzorcev uporabnikov, testiranje učinkovitosti varnostnih politik ter razvoj in optimizacijo obrambnih strategij. Še posebej dragoceni so v okoljih, kjer je varnost ključnega pomena (npr. bančništvo, zdravstvo, državna uprava), saj omogočajo testiranje brez ogrožanja dejanskih sistemov.

Zaradi vse večje izpopolnjenosti napadov in hitro spreminjajočega se kibernetskega okolja postaja jasno, da tradicionalni pristopi niso več zadostni. Potrebujemo inteligentne in prilagodljive rešitve, ki temeljijo na umetni inteligenci, strojni analizi podatkov in simulacijah realnih scenarijev. V nadaljevanju bomo predstavili, kako lahko inteligentni agenti, podprti z LLM-ji, prispevajo k razvoju takšnih rešitev.

3 Inteligentni sistemi v simulacijskih okoljih

Inteligentni agenti predstavljajo temelj sodobne umetne inteligence in omogočajo razvoj sistemov, ki zaznavajo okolje, se iz njega učijo ter sprejemajo odločitve glede na zastavljene cilje. V simulacijskih okoljih lahko agenti raziskujejo, testirajo in se izboljšujejo (tj., njihovi algoritmi se prilagajajo okolju v katerem delujejo) brez neposrednega vpliva na resnični svet. Agenti lahko skozi simulacije preizkušajo različne strategije, zaznavajo posledice svojih dejanj in se učijo na podlagi povratnih informacij. Tovrsten pristop je še posebej uporaben na področjih, kjer bi bilo učenje v realnem svetu predrago, nevarno ali neetično – na primer pri simulacijah množičnih dogodkov.

Ena izmed ključnih prednosti simulacijskih okolij je njihova prilagodljivost. Raziskovalci lahko spreminjajo parametre okolja, dodajajo nove scenarije in spremljajo, kako se agenti prilagajajo. To omogoča razvoj robustnih in prilagodljivih agentov, ki so sposobni delovati v kompleksnih in dinamičnih okoljih. V kontekstu učenja z okrepitvijo deluje inteligentni agent v zanki zaznava–odločanje–delovanje. Agenti s pomočjo zaznavanja informacij iz okolja izbirajo dejanja, za katera pričakujejo največjo nagrado. Sčasoma razvijejo *politike* (angl. *policy*), ki vodijo do želenih ciljev. Simulacije omogočajo hiter potek časa, več paralelnih epizod učenja in spreminjanje pogojev, kar dodatno pospeši učenje.

Kljub številnim prednostim pa obstajajo tudi izzivi. Eden izmed njih je prenos znanja iz simulacijskega okolja v resnični svet, saj lahko razlike med simulacijo in realnostjo vplivajo na učinkovitost agentov. Zato je pomembno, da so simulacije čim bolj realistične in da se uporabljajo metode za zmanjšanje razkoraka med simulacijo in realnostjo. V prihodnosti lahko pričakujemo še večjo uporabo inteligentnih agentov v simulacijskih okoljih, zlasti v povezavi z napredkom na področju strojnega učenja, računalniškega vida in obdelave naravnega jezika. Ta razvoj bo omogočil ustvarjanje še bolj učinkovitih agentov, ki bodo igrali ključno vlogo v številnih panogah.

Inteligentni agenti se v simulacijskih okoljih danes uporabljajo v mnogih praktičnih primerih. Med najpogostejše spadajo testiranja algoritmov za avtonomna vozila v simuliranih prometnih situacijah, razvoj in preizkušanje robotskih sistemov v industrijskih ali reševalnih scenarijih, učne simulacije za medicinsko izobraževanje, ekonomske simulacije za analizo tržnega vedenja ter vojaške in varnostne simulacije za strateško načrtovanje. Ti primeri potrjujejo raznolikost uporabe in pomen simulacij za področja, kjer je delovanje v realnem času preveč tvegano ali zapleteno.

Tehnično gledano razvoj takšnih agentov temelji na različnih pristopih. Med najpomembnejšimi je ena izmed vej strojnega učenja imenovana *okrepitveno učenje* (angl. *reinforcement learning*), ki agentu omogoča učenje na podlagi pridobljene nagrade iz okolja. Na primer, pozitivna nagrada za uspešno dosežen cilj, negativna za nedoseganje cilja,

kot tudi možnost nevtralne nagrade, če agentovo dejanje ne vpliva bistveno na stanje okolja ali če rezultat ni jasno pozitiven ali negativen. Pomembno vlogo imajo tudi algoritmi *globokega učenja* (angl. *deep learning*), saj omogočajo obdelavo kompleksnih vhodnih podatkov, kot so slike, zvok ali naravni jezik. V zadnjem času pa postaja vse bolj pomembna tudi arhitektura *Transformer* (angl. *Transformer architecture*), ki omogoča razumevanje konteksta in zaporedij, ter več agentni sistemi, kjer več agentov deluje v istem okolju. Ključni elementi uspešnega razvoja so tudi zmogljive simulacijske platforme, kot so OpenAI Gym [4], CARLA [5], Gazebo [6] in Unity ML-Agents [7], ki raziskovalcem nudijo prilagodljivo, razširljivo in realistično okolje za učenje in testiranje.

Skupaj tvorijo inteligentni agenti in simulacijska okolja temelj za varno, učinkovito in etično razvijanje kompleksnih sistemov umetne inteligence. Zaradi svoje fleksibilnosti, sposobnosti delovanja v realnem času (pomemben aspekt!) in možnosti integracije z drugimi naprednimi tehnologijami, so inteligentni agenti postali nepogrešljiv del sodobnih simulacijskih okolj. Njihova uporaba se širi od akademskih raziskav do industrijskih aplikacij, kjer prispevajo k optimizaciji procesov, izboljšanju varnosti in podpori pri odločanju. V prihodnosti lahko pričakujemo še večjo vlogo teh agentov pri razvoju inteligentnih, prilagodljivih in varnih digitalnih okolij.

4 Veliki jezikovni modeli

LLM-ji predstavljajo enega najpomembnejših dosežkov sodobne umetne inteligence. Gre za napredne algoritme, ki temeljijo na globokem učenju, zlasti na arhitekturi Transformer (npr. BERT, GPT), in omogočajo razumevanje, generiranje ter prevajanje naravnega jezika. Zaradi svoje sposobnosti obdelave ogromnih količin podatkov in učenja kompleksnih jezikovnih vzorcev so LLM-ji postali ključni v številnih aplikacijah, od pogovornih sistemov in prevajalnikov do orodij za pisanje besedil, analizo sentimenta ter iskanje informacij.

Osnova delovanja LLM je učenje z nadzorom in učenje iz številnih besedilnih virov, kot so knjige, članki, spletne strani in drugi javno dostopni podatki. Modeli se naučijo napovedovati naslednjo besedo v povedi ali razumeti pomen celotnega besedila. S tem razvijejo notranje predstavitve jezika, ki omogočajo vsebinsko razumevanje in generiranje besedil, ki so po slogu in vsebini zelo podobna človeškemu jeziku.

Zaradi svoje vsestranskosti se LLM-ji uporabljajo na različnih področjih. V zdravstvu pomagajo pri analizi zdravniških poročil, v pravu pri iskanju relevantne sodne prakse, v izobraževanju pa kot osebni tutorji ali orodja za izboljšanje pisanja. V podjetništvu omogočajo avtomatizacijo podpore strankam in analizo mnenj uporabnikov. Vendar pa se ob tem pojavljajo tudi izzivi, povezani z etiko, pristranskostjo modelov ter možnostjo zlorabe, kot so generiranje dezinformacij ali avtomatizirani napadi prek socialnega inženiringa.

Med najbolj znanimi pogovornimi sistemi, ki temeljijo na LLM-jih, so ChatGPT (razvit pri OpenAI) [8], Gemini (prej Bard, razvit pri Googlu) [9], Claude (Anthropic) [10], Copilot (Microsoft) [11] ter DeepSeek [12]. Ti sistemi omogočajo interaktivno komunikacijo v naravnem jeziku in so optimizirani za različne naloge – od splošnega klepeta in odgovarjanja na vprašanja do pisanja besedil, programiranja ter reševanja kompleksnih problemov.

Za boljši pregled lastnosti teh sistemov je v Tabela 1 podana primerjava ključnih značilnosti: razvijalec, uporabljeni model, načini dostopa, tipične uporabe ter posebnosti posameznega sistema. Primerjava pokaže, da se sistemi med seboj razlikujejo predvsem glede integracije z drugimi orodji, odprtosti, varnostnih mehanizmov in namenskosti. Medtem ko komercialni modeli dajejo poudarek na uporabniško izkušnjo in produktivnost, odprtokodni modeli, kot sta DeepSeek in Mistral, ponujajo več svobode za raziskovanje in prilagoditve.

ChatGPT, ki temelji na modelih GPT-3.5 in GPT-4, je dostopen preko spletnega vmesnika in kot integracija v Microsoftova orodja, kot sta Word in Excel. Uporablja se v izobraževanju kot osebni tutor, pomočnik pri pisanju

seminarskih nalog ali razlagi kompleksnih pojmov. V poslovnem okolju omogoča generiranje poročil, povzetkov sestankov ter pomoč uporabnikom prek klepetalnih vmesnikov.

Gemini, Googlov naslednik sistema Bard, je močno povezan z iskalnikom in omogoča napredno iskanje po spletu z razumevanjem konteksta. Omogoča tudi ustvarjanje vizualnih vsebin, risanje diagramov in deluje kot razširjen pomočnik v okolju Google Workspace. V šolskem okolju se uporablja za razlago pojmov, ustvarjanje kvizov in pripravo učnega gradiva.

Claude, razvit pri podjetju Anthropic, je osredotočen na varnost, transparentnost in etično uporabo umetne inteligence. Njegova zasnova daje poudarek pojasnjevanju odločitev in zmanjševanju halucinacij. Claude je pogosto uporabljen v okoljih, kjer je pomembna zanesljivost informacij – na primer v pravnih raziskavah ali občutljivih notranjih komunikacijah.

Microsoft Copilot pa LLM-je vključuje neposredno v obstoječa poslovna orodja (npr. PowerPoint, Outlook, Teams), kjer uporabnikom omogoča samodejno pisanje elektronske pošte, povzemanje sej, pripravo predstavitev in analiz podatkov.

DeepSeek, razvit na Kitajskem, predstavlja odprtokodno alternativo velikim komercialnim modelom in združuje zmogljivost s stroškovno učinkovitostjo. Modeli DeepSeek-VL in DeepSeek-Coder so namenjeni predvsem več modalni obdelavi (besedilo plus slike) in programerskim nalogam. DeepSeek je posebej zanimiv za raziskovalce in razvijalce, saj omogoča lokalno uporabo in prilagajanje, hkrati pa ohranja kakovost primerljivo z največjimi zaprtimi modeli.

Tabela 1: Primerjava ključnih značilnosti sistemov.

Sistem	Razvijalec	Model	Dostopnost	Primarne uporabe	Posebnosti
ChatGPT	OpenAI	GPT-3.5, GPT-4	Spletna aplikacija, mobilna aplikacija, MS Office	Pisanje besedil, klepet, programiranje, povzemanje vsebin	Zelo vsestranski, široko dostopen, hiter odziv, odlična prilagoditev kontekstu
Gemini (Bard)	Google DeepMind	Gemini 1.5 Ultra	Splet, Google Search, Gmail, Docs	Iskanje informacij, razlaga pojmov, ustvarjanje gradiv	Močna integracija z Googlovimi storitvami, dostop do spletnih virov
Claude	Anthropic	Claude 3 Opus	Spletna aplikacija	Pravna analiza, občutljive vsebine, varna raba	Poudarek na varnosti, pojasnjevanje odločitev, ni neposrednega dostopa do interneta
Copilot	Microsoft+OpenAI	GPT-4 (prek OpenAI)	Spletna aplikacija, vgrajen v Microsoft 365	Pisarniška avtomatizacija, pisanje e-pošte, analiza dokumentov	Tesna integracija z Word, Excel, Outlook; usmerjeno v produktivnost
DeepSeek	DeepSeek AI	DeepSeek-VL, DeepSeek-Coder	Odprtokodno (HuggingFace, lokalna raba, API)	Programiranje, več modalne naloge, raziskave	Odprtokoden, lokalna uporaba možna, hiter razvoj, primerljiv z vodilnimi komercialnimi modeli

ChatGPT je priljubljen zaradi svoje vsestranskosti, zanesljivega razumevanja konteksta in široke dostopnosti prek spletnih ter mobilnih aplikacij. Gemini izstopa s tesno integracijo z Googlovim iskalnikom in oblačnimi storitvami, kar omogoča napredno iskanje in delo z dokumenti v realnem času. Claude daje poudarek varnosti, preglednosti odločitev in zmanjševanju halucinacij, zato je posebej primeren za uporabo v občutljivih okoljih. Microsoftov Copilot je neposredno vgrajen v pisarniška orodja, kot so Word, Excel in Outlook, kjer avtomatizira številne rutinske naloge. DeepSeek pa predstavlja odprtokodno alternativo, ki omogoča lokalno uporabo in je posebej učinkovit pri programiranju ter večmodalnih nalogah, kot je obdelava besedil in slik.

Ti pogovorni sistemi vse pogosteje postajajo nepogrešljiv del vsakdanjih orodij v poslovnem svetu, izobraževanju, raziskovanju in zasebni rabi. Z nenehnim razvojem postajajo bolj zanesljivi, večjezični in uporabniku prijazni, kar

odpira nove možnosti za interakcijo z umetno inteligenco ter spodbuja njeno odgovorno in učinkovito vključevanje v različna področja družbe.

Razvoj LLM-jev je povezan tudi z visokimi računalniškimi zahtevami. Uporaba grafičnih procesnih enot in specializiranih procesorjev omogoča učinkovito učenje modelov z milijardami parametrov. S tem pa se povečuje tudi energetska poraba in potreba po trajnostnih pristopih k razvoju umetne inteligence.

V prihodnosti lahko pričakujemo nadaljnji napredek na področju personalizacije modelov, večjezičnosti, zmanjševanja pristranskosti ter razvoja manjših, učinkovitejših modelov, ki bodo dostopni širši javnosti. LLM-ji tako ne oblikujejo le načina, kako uporabljamo jezik v digitalnem okolju, temveč tudi na novo opredeljujejo razmerje med človekom in strojem.

Z vključitvijo LLM-jev v inteligentne agente pridobimo bistveno večjo funkcionalnost, prilagodljivost in realističnost simulacij. V nasprotju z agenti brez LLM-jev, ki temeljijo na vnaprej določenih pravilih in omejenih vzorcih, agenti z LLM-ji omogočajo dinamično generiranje naravnega jezika, globoko razumevanje vsebine ter več modalno zaznavanje. Napadalni agenti lahko ustvarjajo raznolika in prepričljiva prevarantska sporočila, medtem ko obrambni agenti izvajajo napredno analizo in zaznavanje prevar na podlagi semantičnih in stilističnih značilnosti. Ključne razlike med obema pristopoma so povzete v Tabela 2, ki prikazuje primerjavo med inteligentnimi agenti z vključenimi LLM-ji in tistimi, ki temeljijo na klasičnih metodah. Primerjava zajema vidike, kot so sposobnost generiranja vsebin, razumevanje jezika, prilagodljivost komunikacije, zaznavanje prevar, učenje iz interakcij, več modalna obdelava podatkov, realističnost simulacije ter uporabnost za izobraževalne namene. Rezultati jasno kažejo, da agenti z LLM-ji omogočajo bistveno bolj napredno in učinkovito delovanje v simulacijskih okoljih za kibernetsko varnost. V nadaljevanju bomo predstavili arhitekturo takšnega simulacijskega okolja in način, kako agenti medsebojno delujejo v simuliranem okolju.

Tabela 2: Primerjava med agenti brez in z LLM.

Lastnost	Agent brez LLM-jev	Agent z LLM-ji
Generiranje vsebin	Omejeno na vnaprej definirana sporočila ali predloge.	Dinamično generiranje naravnega jezika, prilagojenega vsebini.
Razumevanje jezika	Pogosto odvisno od ključnih besed.	Globoko razumevanje semantike, sintakse in vsebine.
Prilagodljivost komunikacije	Težko se prilagaja različnim slogom.	Lahko oponaša različne stile, tone in namene.
Zaznavanje prevar	Temelji na pravilih ali preprostih vzorcih.	Napredna analiza besedilnih značilnosti in anomalij.
Učenje iz interakcij	Zahteva ročno posodabljanje.	Samodejno učenje.
Več modalna obdelava podatkov	Ni podprta.	Poleg besedila še podpora slikam, videom in zvoku.
Realističnost simulacije	Omejena variabilnost.	Raznoliki scenariji in odzivi.
Uporabnost za izobraževanje	Primerna za osnovno učenje.	Primerna za kompleksne delavnice in raziskave.

5 Simulacijsko okolje

Simulacijsko okolje je namenski digitalni prostor, zasnovan za modeliranje, testiranje in analizo interakcij med različnimi entitetami – v našem primeru med napadalnimi in obrambnimi agenti. Takšna okolja omogočajo varno in nadzorovano izvajanje eksperimentov, kjer se lahko preučujejo učinki različnih strategij, algoritmov in odzivov na simulirane grožnje. Ključne značilnosti simulacijskih okolij vključujejo:

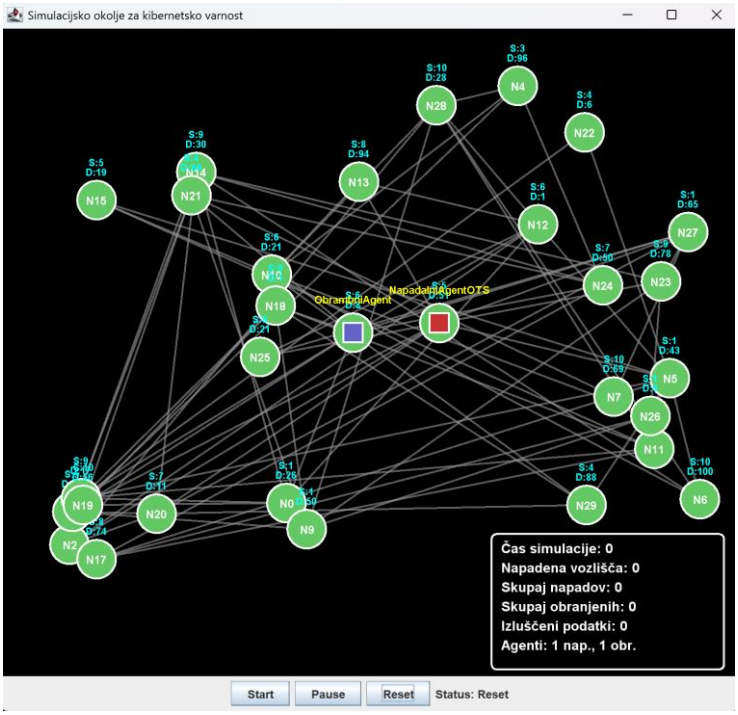
- **Interaktivnost**: omogočajo izmenjavo podatkov med agenti.
- **Merljivost**: beležijo uspešnost posameznih strategij.
- **Prilagodljivost**: omogočajo testiranje različnih scenarijev.
- **Varnost**: vse aktivnosti potekajo v izoliranem okolju brez tveganja za dejanske sisteme.

Takšna okolja so nepogrešljiva pri razvoju in testiranju kibernetskih obrambnih mehanizmov, saj omogočajo realistično simulacijo napadov in odzivov nanje.

Slika 1 prikazuje prototipno zasnovo simulacijskega okolja, kjer se odvijajo interakcije med napadalnimi in obrambnimi agenti. Vsako vozlišče v okolju predstavlja posameznega uporabnika, ki je lahko tarča napada. Napadalni agenti imajo možnost napasti katerokoli vozlišče, kar simulira realne scenarije kibernetskih groženj, kot so napadi socialnega inženiringa. Vsako vozlišče je lahko zaščiteno z enim ali več obrambnimi agenti, ki analizirajo prejeta sporočila in se odzivajo na potencialne grožnje. Obrambni agenti delujejo neodvisno ali v skupinah, kar omogoča testiranje različnih obrambnih strategij – od osnovne zaščite do napredne analize s pomočjo LLM-jev. Vizualna postavitev omogoča jasno predstavo o:

- topologiji okolja, kjer so vozlišča med seboj povezana,
- tokovih napadov, ki se lahko usmerijo na katerokoli točko,
- porazdelitvi obrambnih agentov, ki varujejo posamezne uporabnike.

Takšna struktura omogoča dinamično simulacijo, kjer se lahko meri učinkovitost obrambe glede na število agentov, vrsto uporabljenega modela in kompleksnost napada.



Slika 1: Simulacijsko okolje.

Napadalni agent

Napadalni agent deluje kot proaktivna entiteta, katere cilj je simulirati realne grožnje s področja socialnega inženiringa. Zaradi varnostnih omejitev LLM-jev, se ne uporablja generativnega modela, temveč črpa sporočila iz vnaprej pripravljenega nabora podatkov [13]. Ta vsebuje kategorizirana sporočila, prevedena iz arabščine in angleščine v slovenščino, razvrščena kot:

- *Prevarantska* (angl. *malicious*)
- *Neškodljiva* (angl. *benign*)

Sporočila vključujejo scenarije, kot so lažne nagrade, nujna obvestila, ponarejena bančna sporočila in poskusi kraje gesel. Vsako sporočilo se obravnava kot samostojna enota, kar omogoča natančno testiranje obrambnih mehanizmov.

Obrambni agenti

V simulaciji nastopata dva obrambna agenta z implementiranimi različnimi pristopi:

- Klasični strojni model (Agent brez LLM*):* Temelji na logistični regresiji [14] in pomembnosti izraza v določenem dokumentu glede na celotno zbirko dokumentov (angl. *Term Frequency–Inverse Document Frequency,* TF-IDF) [15]. Postopek vključuje:
  - Učenje na 80 % podatkov, testiranje na 20 %.
  - Klasifikacijo sporočil kot *True* (prevarantsko) ali *False* (neškodljivo). Ta pristop je hiter in enostaven, a ne zazna globljih semantičnih vzorcev.
- Model LLM (Agent z LLM): Uporablja napredni jezikovni model, ki razume kontekst, stil in semantiko sporočil. Postopek vključuje:
  - Pripravo ukaza z 160 označenimi sporočili.
  - Napoved za novo sporočilo (*True* ali *False*). Prednost je boljša natančnost in razlaga odločitev, slabost pa večja računska zahtevnost in občutljivost na strukturo ukaza.

Slika 2 prikazuje mikro-simulacijo znotraj širšega simulacijskega okolja, kjer se osredotočimo na eno vozlišče – torej enega uporabnika. V tem primeru se odvija neposredna interakcija med napadalnim agentom, ki poskuša izvesti napad s prevarantskimi sporočili, in obrambnim agentom, ki ima nalogo prepoznati, ali gre za prevaro. Napadalni agent pošlje sporočilo, ki vsebuje značilne elemente socialnega inženiringa, kot so:

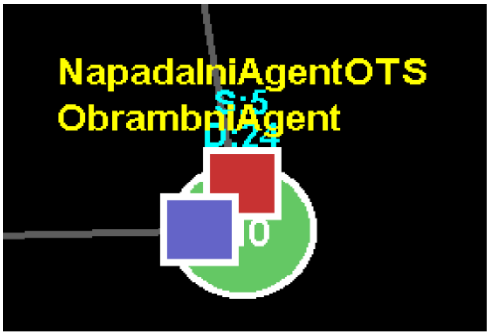
- pozivi k takojšnjemu ukrepanju,
- lažne varnostne grožnje,
- ponarejena obvestila bank ali storitev.

Obrambni agent nato analizira vsebino sporočila z uporabo izbranega pristopa (klasični model ali LLM) in oceni verjetnost, da gre za prevarantsko vsebino. Če zazna anomalije v jeziku, slogu ali semantiki, sproži ustrezen odziv – opozorilo, blokado ali poročanje o incidentu.

Ta vizualizacija omogoča vpogled v:

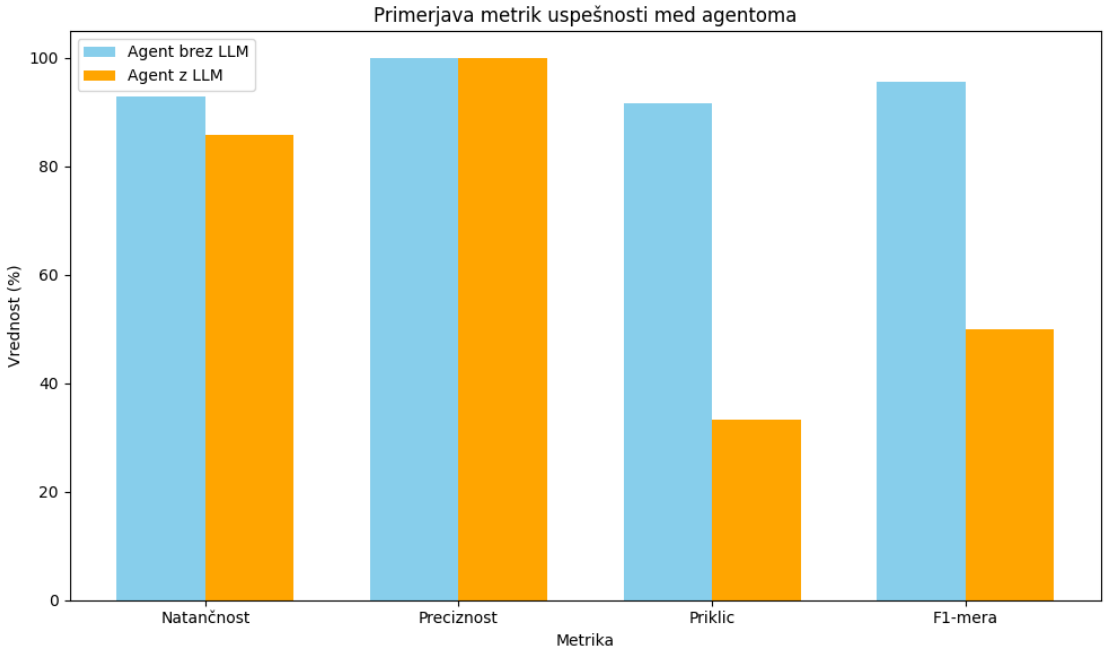
- potek napada na ravni posameznega uporabnika,
- reakcijo obrambnega sistema v realnem času, in
- učinkovitost klasifikacije glede na uporabljeni model.

Takšna predstavitev je ključna za razumevanje delovanja agentov v konkretnih situacijah in za primerjavo različnih obrambnih strategij.



Slika 2: Mikro-simulacija znotraj širšega simulacijskega okolja (tj., eno vozlišče).

Tabela 3 prikazuje konkretne primere klasifikacije posameznih sporočil s strani obeh agentov. Primerjava omogoča vpogled v to, kako se agent brez LLM in agent z LLM odzivata na različne tipe vsebin – od očitno neškodljivih do preudarno zavajajočih. Vidimo lahko, da agent brez LLM dosledno prepoznava prevarantska sporočila, medtem ko agent z LLM pogosto spregleda nevarnost, kar se odraža v napačnih negativnih klasifikacijah.
Napaka! Vira sklicevanja ni bilo mogoče najti. in Tabela 4 prikazujeta primerjavo uspešnosti obeh agentov na podlagi šestih ključnih metrik. Vizualizacija jasno poudari razliko v priklicu in F1-meri, kjer agent brez LLM izstopa kot bolj uravnotežen in zanesljiv. Agent z LLM sicer dosega popolno preciznost (ni napačnih pozitivnih), vendar zaradi nizkega priklica spregleda večino prevarantskih sporočil, kar zmanjšuje njegovo uporabnost v realnih varnostnih scenarijih.



Graf 1: Primerjava metrik uspešnosti med agentoma.

Tabela 3: Primeri klasifikacije (prikaz sporočila omejeno na 100 znakov).

Sporočilo	Agent brez LLM	Agent z LLM
Tu so naše najnovejše ponudbe za nepremičnine, obiščite naše spletno mesto.	PRAVILNO	PRAVILNO
To je kapital. Zaznali smo novo prijavo iz vaše običajne naprave. Lažno dejanje je bilo potrebno, če...	PRAVILNO	PRAVILNO
Kvalificirani ste, da povečate svojo kreditno kartico, zdaj vnesite podatke o svojem računu.	PRAVILNO	NAROBE
Sporočilo Banque Misr: Imate davčno okrevanje, ki vas še ni prejelo, zdaj vnesite podatke o svojem r...	PRAVILNO	PRAVILNO
Hej bratec, tvoja Spotify Premium je še vedno aktivna. Nadaljujte z zagozditvijo!	PRAVILNO	PRAVILNO
To je nujna falsetice iz IRS. Dolgujete neplačane davke. Takoj zagotovite svoj SSN, da se izognete a...	PRAVILNO	NAROBE
Vaša zahteva za posojilo je odobrena, za več podrobnosti nas kontaktirajte.	PRAVILNO	PRAVILNO
Vaša mobilna storitev bo jutri prekinjena, razen če ne posodobite informacij o obračunu.	NAROBE	NAROBE
Dober večer, mi smo iz elektroenergetskega podjetja, imate neplačani račun, plačati morate takoj, si...	NAROBE	NAROBE
Ali morate pomagati pri uporabi bančne aplikacije? Tu smo, da vam pomagamo.	PRAVILNO	PRAVILNO

Tabela 4: Primerjava metrik uspešnosti med agentoma.

Metrika	Opis	Agent brez LLM	Agent z LLM
Natančnost	Delež pravilno zaznanih med vsemi sporočili.	92,86 %	85,71 %
Preciznost	Delež pravilno zaznanih prevarantskih sporočil med vsemi, ki jih je agent označil kot prevarantska.	100 %	100 %
Priklic	Delež pravilno zaznanih prevarantskih sporočil med vsemi dejanskimi prevarantskimi sporočili.	91,67 %	33,33 %
F1-mera	Uravnotežena metrika med preciznostjo in priklicem.	95,65 %	50 %
Napačni pozitivni	Število neškodljivih sporočil, ki so bila napačno označena kot škodljiva.	0	0
Napačni negativni	Število škodljivih sporočil, ki jih agent ni zaznal.	2	16

6 Zaključek

V prispevku smo predstavili uporabo simulacijskega okolja za kibernetsko varnost, v katerem delujejo inteligentni agenti z možnostjo vključitve LLM-jev. Okolje omogoča realistično simulacijo napadov socialnega inženiringa, kot je spletno ribarjenje, ter analizo odzivov obrambnih agentov na različne tipe sporočil. V ta namen smo primerjali dva pristopa: klasični strojni model (agent brez LLM) in napredni model, ki uporablja LLM (agent z LLM).

Rezultati eksperimenta so pokazali, da kljub naprednim zmožnostim LLM-jev, kot so razumevanje konteksta, semantike in stilskih značilnosti, agent brez LLM trenutno dosega boljše rezultate pri zaznavanju prevarantskih sporočil. Klasični model, ki temelji na logistični regresiji in TF-IDF, je dosegel višjo natančnost, boljši priklic in višjo F1-mero. Še posebej pomembno je, da je agent brez LLM zaznal skoraj vse prevarantske primere, medtem ko je agent z LLM spregledal večino napadov, kar se je odrazilo v visokem številu napačnih negativnih klasifikacij.

Analiza konkretnih primerov klasifikacije je dodatno potrdila, da je agent brez LLM bolj dosleden in zanesljiv pri prepoznavanju nevarnih vsebin. Vizualizacije in metrika uspešnosti jasno kažejo, da je klasični pristop v trenutni implementaciji bolj primeren za naloge, kjer je ključna visoka občutljivost na prevarantske vzorce.

Kljub temu pa ostaja vključitev LLM-jev pomembna smer razvoja, saj ti modeli ponujajo večjo prilagodljivost, možnost generiranja naravnega jezika in potencial za naprednejše oblike analize. Vendar pa je za njihovo učinkovito uporabo v varnostnih okoljih potrebna nadaljnja optimizacija, predvsem na področju zmanjševanja napačnih negativnih odločitev.

Kljub napredku se je treba zavedati tudi omejitev:

- Velikost modelov vpliva na kakovost, a tudi na zahteve po strojni opremi.
- Varnostne omejitve v večini LLM-jev preprečujejo generiranje napadalnih vsebin (tj., tudi v raziskovalne ali simulacijske namene).
- Filtri za zavračanje zlonamernih ukazov pogosto onemogočajo igranje vlog napadalcev, kar omejuje realističnost simulacij.

Na podlagi analize napačnih klasifikacij in vedenjskih vzorcev agentov z LLM predlagamo naslednje izboljšave:

- Izboljšanje strukture ukaza: Dodajanje jasnih navodil, primerov in pričakovanih odgovorov, da model bolje razume nalogo.

- Uporaba več raznolikih primerov prevarantskih sporočil: Vključitev stilistično različnih sporočil – formalna, neformalna, z zavajajočimi izrazi.
- Dodajanje razlage klasifikacije v ukaz: Zahteva, da model pojasni svojo odločitev, povečuje razumevanje konteksta in omogoča boljšo sledljivost.
- Preverjanje občutljivosti na dolžino in kompleksnost sporočil: Nekatera daljša ali bolj zapletena sporočila so bila napačno klasificirana, kar kaže na potrebo po dodatnem uravnoteženju.
- Fino nastavljanje lokalnega modela z dodatnimi slovenskimi podatki: Prilagoditev modela s podatki, ki bolje odražajo jezikovne in kulturne značilnosti slovenskega okolja.
- Uporaba drugih, predvsem večjih in zmogljivejših modelov: Večji modeli imajo potencial za boljše razumevanje kompleksnih jezikovnih vzorcev in subtilnih manipulacij, kar lahko izboljša zaznavanje prevarantskih sporočil.

Simulacijsko okolje, kot je bilo predstavljeno, se izkazuje kot dragoceno orodje za raziskave in izobraževanje na področju kibernetske varnosti. Omogoča varno testiranje različnih scenarijev, primerjavo pristopov in razvoj novih strategij za zaščito pred sodobnimi grožnjami. V prihodnosti bo takšno okolje ključno za razvoj inteligentnih, prilagodljivih in robustnih obrambnih sistemov.

Literatura

[1] Q. Zhu, »Game Theory Meets LLM and Agentic AI: Reimagining Cybersecurity for the Age of Intelligent Threats«, arXiv, 2025.
[2] P. R. Norvig in S. A. Intelligence, »A modern approach«, Upper Saddle River, NJ, USA: Prentice Hall, 2002.
[3] P. A. Andersen, M. Goodwin in O. C. Granmo, »Deep RTS: a game environment for deep reinforcement learning in real-time strategy games«, V: 2018 IEEE conference on computational intelligence and games (CIG), IEEE, str. 1–8, avgust 2018.
[4] OpenAI, "OpenAI Gym", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://github.com/openai/gym
[5] CARLA, "CARLA", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://carla.readthedocs.io/en/0.9.12/adv_agents
[6] Gazebo, "Gazebo Sim", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://github.com/openai/gymhttps://github.com/gazebosim
[7] Unity, "Unity ML Agents", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://github.com/Unity-Technologies/ml-agents
[8] OpenAI, "ChatGPT", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://chatgpt.com
[9] Google, "Gemini", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://gemini.google.com
[10] OpenAI, "OpenAI Gym", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://github.com/openai/gym
[10] Claude, "Claude AI", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://claude.ai
[11] Microsoft, "Copilot", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://copilot.microsoft.com
[12] OpenAI, "OpenAI Gym", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://github.com/openai/gym
[12] DeepSeek AI, "DeepSeek", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://chat.deepseek.com
[13] Kaggle, " Social Engineering Detection Benchmark with LLMs", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://www.kaggle.com/datasets/dohaalqurashi/social-engineering-detection-benchmark-with-llms
[14] Wikipedia, "Logistic Regression", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://en.wikipedia.org/wiki/Logistic_regression
[15] Wikipedia, "TF-IDF", Dostopano: Jul. 28, 2025. [Online]. Dosegljivo na: https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Strategije in izzivi pri prehodu dolgo živečih projektov iz SVN na GIT

Gregor Novak, Grega Krajnc, Izidor Pokrivač

SRC d.o.o., Ljubljana, Slovenija
gregor.novak@src.si, grega.krajnc@src.si, izidor.pokrivac@src.si

Migracija starejših, a še aktivnih projektov iz sistema za nadzor različic Subversion (SVN) na Git je lahko zahteven, a dolgoročno zelo koristen proces. V prispevku predstavljamo praktično izkušnjo selitve večletnih projektov iz SVN na Git in opišemo izzive, s katerimi smo se srečali pri preslikavi zgodovine revizij, kot so ohranjanje metapodatkov in pravilne strukture vej, oznak ter združitev. Opišemo uporabo orodij, kot so git-svn, git filter-repo in lastni skripti, s katerimi smo avtomatizirali in poenostavili korake urejanja zgodovine. Selitev smo izkoristili tudi za prenovo razvojnega procesa. Uvedli smo delovni tok na osnovi uveljavljenih praks za Git, integracijo zahtev za združitev in novo CI/CD infrastrukturo. Posebej naslavljamo vzporedno podporo dveh aplikacijskih platform z ločenimi procesi znotraj Git okolja. Prehod na Git nam je omogočil uporabo bolj prilagodljive, pregledne in sodobne razvojne infrastrukture, lažjo integracijo z orodji za sledenje nalogam ter lažjo vključenost pregledov kode v vsakdanji proces. Migracijo razumemo kot izhodišče za uvajanje naprednejših praks, saj Git s svojo fleksibilnostjo podpira nenehno izboljševanje delovnega toka in sledenje sodobnim tehnologijam ter pristopom razvoja programske opreme.

Ključne besede:
nadzor različic,
Subversion,
SVN,
GIT,
delovni proces,
modernizacija,
razvojna orodja.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.4

1 Uvod

V sodobnem razvoju programske opreme je učinkovito upravljanje različic kode ključnega pomena za uspešno sodelovanje in hiter razvoj. Medtem ko se je Git uveljavil kot prevladujoče orodje za nadzor različic, številne ekipe še vedno uporabljajo starejše sisteme, kot je Subversion (v nadaljevanju SVN), zlasti pri starejših, a še vedno aktivnih projektih.

Git je bil razvit leta 2005 za potrebe razvoja jedra Linuxa [1]. Gre za distribuiran sistem za upravljanje različic, pri katerem ima vsak razvijalec lokalno kopijo celotne zgodovine projekta. To omogoča delo tudi brez stalne internetne povezave. Nasprotno pa SVN temelji na centraliziranem modelu, kjer se zgodovina sprememb nahaja na strežniku. Posledično so določene operacije počasnejše in odvisne od omrežne povezave.

Ena izmed ključnih prednosti Gita je učinkovito delo z vejami (angl. branches). Razvijalci lahko z njimi preprosto ločijo nove funkcionalnosti, eksperimentalne spremembe ali popravke napak od glavne kode, ne da bi ogrozili stabilnost projekta. Čeprav tudi SVN omogoča uporabo vej, je delo z njimi bolj zapleteno in manj učinkovito kot pri Gitu. Veje v SVN se tako po navadi uporabljajo le izjemoma, medtem ko je njihova uporaba v Gitu stalnica.

Poleg tehničnih prednosti Git izstopa tudi po močni podpori gostiteljskih platform, kot so GitHub, GitLab in Bitbucket. Te ponujajo sodobna orodja za pregledovanje kode, sodelovanje, avtomatizacijo CI/CD procesov in upravljanje projektov, kar izboljšuje razvojni proces. SVN sicer ostaja zanesljiv sistem, a njegova manj prilagodljiva arhitektura in nižja razširjenost sodobnih orodij je razlog, da vse bolj izgublja stik s potrebami sodobnih razvojnih ekip.

1.1. Motivacija za selitev starejših projektov

V podjetju SRC d.o.o. za vse novejše projekte uporabljamo Git kot privzeti sistem za upravljanje različic, hkrati pa še vedno vzdržujemo starejše projekte, pri katerih se je do sedaj uporabljal SVN. Kljub njihovi starosti so ti projekti v vsakodnevni produkcijski uporabi, njihova stabilnost pa je kritična za delo končnih uporabnikov. Ti projekti niso zgolj pasivno vzdrževani – redno jih nadgrajujemo in razvijamo naprej. Ravno zaradi tega je selitev iz SVN na Git imela smisel: kljub njihovi starosti, so projekti živi, uvedba Gita pa zanje pomeni tudi uvedbo sodobnega razvojnega okolja, boljša orodja in višjo prilagodljivost razvojnega cikla.

O migraciji na Git smo razmišljali že dlje časa, a selitev ni bila prioriteta, saj je delovni proces s SVN-jem deloval stabilno, robustno in varno. Zaradi drugih nalog vzdrževanja in nadgrajevanja projektov pa selitvi na Git tudi nismo uspeli nameniti primernega časovnega okna, da bi se migracije lotili na temeljit in strukturiran način, kljub temu da bi nam uporaba Gita lahko že v preteklosti prinesla dodano vrednost k razvojnemu procesu.

Projekti, pri katerih se je uporabljal SVN, so v večini javanski zaledni sistemi. Tekom njihovega vzdrževanja se je pojavila potreba po selitvi zalednih sistemov iz ene aplikacijske platforme na drugo (v nadaljevanju platforma A in platforma B). V času prehoda smo morali zagotavljati hkratno izdajo posameznih različic, kot tudi pravilno delovanje sistemov za obe platformi. Git ponuja učinkovito upravljanje z vejami, hkrati pa omogoča tudi pripravo kopij celotnih repozitorijev. Zaradi tega je delovni proces, ki smo ga morali vzpostaviti, lažje doseči z Gitom kot s SVN-jem. Menjava aplikacijskih platform in vzporedno vzdrževanje sistemov na obeh platformah je bila tako naravna prelomnica projektov, pri kateri je postala selitev na Git ne le smiselna, temveč tudi potrebna za lažje upravljanje razvoja.

2 Naš pristop k selitvi na Git

Ob prehodu iz SVN na Git smo si zadali jasen cilj: selitev mora biti premišljena, varna in brez vpliva na stabilnost projektov. Poseben poudarek smo namenili ohranjanju celotne zgodovine sprememb, saj se ta vsakodnevno uporablja pri analizah, odpravljanju težav in razumevanju konteksta odločitev iz preteklosti. Naša zgodovina obsega več let razvoja, številne spremembe in tudi menjave članov ekip. Podatki v zgodovini, kot so denimo številke nalog v revizijskih sporočilih, so tako pogosto edina vez z razvojnimi odločitvami iz preteklosti.

Zavedali smo se tudi, da selitev ni le tehnični prenos repozitorija, temveč vključuje spremembo razvojne kulture in orodij, ki zajema drugačen potek razvoja, drugačno integracijo CI/CD in predvsem drugačen način vsakodnevnega dela. Želeli smo, da je ob izvedbi selitve vse pripravljeno za nemoten prehod – brez potrebe po vrnitvi v SVN-okolje ali ročnih prilagoditev v zadnjem trenutku. V nadaljevanju poglavja tako na kratko predstavljamo ključne faze, ki so zaznamovale naš prehod na Git.

2.1. Izbira primernega trenutka za selitev

Selitve nismo izvedli impulzivno. Odločitev o prehodu smo sprejeli premišljeno in v pravem trenutku – ko so se potrebe po razvejanem razvoju in večji fleksibilnosti ujemale s prelomnico v razvojnem ciklu: selitvijo projektov na novo aplikacijsko platformo. Želeli smo, da bo v trenutku prehoda na Git vzpostavljeno vse potrebno: prenesena zgodovina repozitorijev, preizkušena integracija CI/CD in jasno definiran delovni proces. Le na ta način smo lahko zagotovili, da se bo razvoj lahko nadaljeval brez prekinitev ali dodatnih tveganj. Prehod smo torej načrtovali vnaprej in na način, da bi bila selitev za ekipo čim manj moteča, razvojna okolja pa kar se da usklajena.

2.2. Prenos zgodovine projektov na Git

Zgodovina razvoja je za nas izjemno dragocena. Vsebovala je več let sprememb, pogosto kompleksne in povezane z različnimi zahtevami. Ena ključnih zahtev pri selitvi je bila torej ohraniti čim bolj dosledno, pregledno in uporabno zgodovino v Gitu. Pri prenosu zgodovine smo se soočili z izzivi – zaradi arhitekturnih razlik SVN-ja in Gita je avtomatska migracija pogosto prinesla popačen potek zgodovine. Za ustrezno predstavitev zgodovine v Gitu smo zato izvajali ročne posege, preuredili strukturo commitov, da izraža dejansko stanje iz SVN-ja in pripravili skripte za avtomatizacijo prilagoditev. S tem smo zagotovili, da migrirana zgodovina izraža dejansko stanje poteka razvoja iz preteklosti, kar omogoča njeno nadaljnjo uporabnost v Gitu.

2.3. Načrtovanje delovnega procesa

Migracija je bila tudi priložnost za prenovo razvojnega procesa. Eden od ciljev je bil, da ne podvajamo navad iz SVN-ja v Gitu, temveč se premaknemo k uveljavljenim praksam za Git, kot sta razvejano razvijanje in združevanje kode preko zahtev za združitev. Ker smo morali vzdrževati dve vzporedni različici aplikacije – za platformo A in platformo B – smo za čas hkratne podpore oblikovali proces, ki izkorišča zmožnost podvajanja Git-repozitorijev s celotno zgodovino, s čimer je tudi CI/CD proces izoliran zgolj na dotično platformo. Z načrtnim pristopom smo tako ustvarili okolje, v katerem lahko razvoj poteka učinkoviteje, pregledneje in z boljšimi orodji kot prej, predvsem pa smo s prenovljenim delovnim tokom zadostili novim potrebam razvojnega procesa.

3 Selitev zgodovine

Selitev zgodovine revizij iz SVN-ja na Git se je izkazala za najzahtevnejši in časovno najintenzivnejši del celotnega procesa. Začetna preslikava revizij iz SVN-ja v Git-commite je bila sicer preprosta, a smo hitro ugotovili, da rezultati niso popolnoma ustrezni. Pridobljena zgodovina v Gitu je bila v več primerih popačena ne glede na izbrano orodje za migracijo. Razlika med arhitekturama SVN-ja in Gita se namreč odraža tudi v tem, kako se interpretirajo veje in oznake (angl. tags). Posledično prenesen repozitorij ni v celoti izražal stanja zgodovine, kot bi ga naravno pričakovali v Gitu.

Za doseganje zgodovine, ki izraža pravilno stanje in pričakovan potek smo morali izvajate ročne posege v strukturo commitov. Kljub temu, da je bilo takšno dodatno urejanje zgodovine časovno zamudno, smo se za to odločili, ker smo se s selitvijo na Git želeli znebiti odvisnosti od SVN-ja, a hkrati tudi ohraniti pravilnost podatkov, ki nam jih nudi obstoječa zgodovina. V tem poglavju tako podrobneje predstavljamo orodja, ki smo jih uporabili pri prenosu in urejanju zgodovine ter korake, ki smo jih izvedli, da smo dosegli repozitorij s pravilno strukturo vej, oznak in metapodatkov commitov.

Zaradi obsežnosti in časovne zahtevnosti postopka smo ugotovili, da je prenos zgodovine smiselno izvesti dovolj zgodaj – še preden so pripravljeni vsi drugi deli migracije. Glavni del zgodovine lahko namreč prenesemo vnaprej, aktiven razvoj pa do dejanskega prehoda na Git še vedno poteka v SVN-ju. Kasnejše revizije iz SVN-ja lahko še vedno uvozimo in jih dodamo k že preneseni zgodovini, vse do zaključka migracije na Git.

3.1. Uporabljena orodja

git-svn

git-svn je orodje, ki omogoča kloniranje SVN-repozitorijev v Git in preslikavo SVN-revizij v Git-commite [2]. Gre za ukaz, vgrajen v Git, ki omogoča delo z obstoječimi SVN-repozitoriji kar znotraj Gitove infrastrukture. Preizkusili smo tudi druga orodja, kot sta svn2git [3] in SubGit [4], vendar se je git-svn izkazal za najprimernejšo izbiro. Orodje je enostavno za uporabo, ne zahteva dodatnih odvisnosti in je v celoti brezplačno. Na našo odločitev pa je prav tako vplivalo dejstvo, da smo tudi pri ostalih orodjih naleteli na podobne težave pri kakovosti migrirane zgodovine. Zato smo git-svn prepoznali kot rešitev, ki je najbolj preprosta za uporabo, hkrati pa nam še vedno omogoča zadostno preglednost in prilagodljivost pri nadaljnji obdelavi.

Lastni namenski skripti

Za poenostavitev postopkov čiščenja zgodovine in rekonstrukcijo logične strukture repozitorija smo pripravili lastne skripte. Njihov namen je bil odprava potrebe po ročnem izvajanju ponavljajočih, daljših operacij, s čimer smo skrajšali čas, potreben za urejanje zgodovine.

git filter-repo

Orodje git filter-repo je sodoben in zmogljiv pripomoček za prečiščevanje zgodovine repozitorijev v Gitu. Predstavlja uradno priporočeno zamenjavo zastarelega ukaza `git filter-branch` [5]. Orodje smo uporabili predvsem za popravljanje metapodatkov po operacijah, kot je rebase, da smo v končnem repozitoriju ohranili dosledne in točne podatke o zgodovini sprememb. Drug razlog za uporabo orodja pa je bila prilagoditev datotečne strukture repozitorija, saj so bile v starejših delih zgodovine zaradi preteklih praks prisotne binarne datoteke, ki jih v Gitu nismo želeli ohranjati. Git namreč ni optimalen za verzioniranje binarnih vsebin, saj ob vsaki spremembi shrani celotno vsebino datoteke. Ohranjanje takšnih datotek v zgodovini bi povzročilo nesorazmerno povečanje velikosti repozitorija na disku in otežilo njegovo vsakodnevno uporabo.

Grafični vmesnik za Git

Za hitrejši pregled in primerjavo commitov, vizualizacijo grafa in preverjanje metapodatkov smo uporabljali tudi grafični vmesnik za Git. Ta ni bil nujno potreben, vendar nam je prihranil precej časa predvsem zato, ker nam je omogočil neposreden pregled stanja brez vnašanja dodatnih ukazov v ukazni vrstici.

Uporabljali smo integriran vmesnik za delo z Gitom znotraj okolja IntelliJ IDEA [6], ki je sicer enoten za vsa JetBrainsova orodja [7]. Ta izbira je bila predvsem stvar priročnosti, saj to okolje uporabljamo že pri samem razvoju. Seveda pa enake funkcionalnosti nudijo tudi številna druga grafična orodja za Git. Ključno je bilo, da smo z vmesnikom hitro preverjali:

- strukturo grafa commitov (veje, oznake in združitve),
- razlike med vsebino commitov ter
- metapodatke commitov (avtorji, časi, sporočila).

3.2. Začetni prenos zgodovine

Priprava datoteke s podatki o avtorjih

Za prenos zgodovine iz SVN-ja v Git smo uporabili ukaz `git svn clone`. Pred tem je bilo potrebno pripraviti datoteko z avtorji, saj git-svn brez nje ne more pravilno preslikati avtorjev iz SVN-ja.

Za pridobitev osnovnega seznama avtorjev iz SVN-repozitorija smo uporabili naslednji cevovod v Bashu.

```
svn log -q <svn-repo-url> | \
    awk -F '|' '/^r/ {gsub(/ /, "", $2); sub(" $", "", $2); \
    print $2" = todo-username <todo-email@todo.com>"}' | \
    sort -u > authors.txt
```

Cevovod na mestu, kjer smo ga pognali, ustvari datoteko z vsebino avtorjev iz SVN-ja v obliki, kot jo prikazuje spodnji primer.

```
1  MartinKrpan = todo-username <todo-email@todo.com>
2  PeterKlepec = todo-username <todo-email@todo.com>
3  peterklepec = todo-username <todo-email@todo.com>
4  VeronikaCeljska = todo-username <todo-email@todo.com>
5  VeronikaDeseniska = todo-username <todo-email@todo.com>
```

Datoteko smo pred njeno uporabo vsebinsko uredili, da smo uskladili imena avtorjev in določili njihove dejanske elektronske naslove. Pri tem smo uporabili isto obliko podatkov, kot jo razvijalci že uporabljamo pri drugih projektih v Gitu.

```
1  MartinKrpan = MartinKrpan <martin.krpan@src.si>
2  PeterKlepec = PeterKlepec <peter.klepec@src.si>
3  peterklepec = PeterKlepec <peter.klepec@src.si>
4  VeronikaCeljska = VeronikaCeljska <veronika.celjska@src.si>
5  VeronikaDeseniska = VeronikaCeljska <veronika.celjska@src.si>
```

Prikazana primera prenesene in urejene datoteke prikazujeta tudi možnost povezovanja različnih uporabniških imen iz SVN-ja z enotnim avtorjem v Gitu. S takšno ureditvijo datoteke dosežemo, da so commiti iz SVN-ja v Gitu pripisani pravim osebam ne glede na to, s kakšnim uporabniškim imenom so v preteklosti delale v SVN-ju.

Kloniranje SVN-repozitorija z git-svn

Ko je bila datoteka z avtorji pripravljena, smo jo podali ukazu `git svn clone` skupaj z naslovom SVN-repozitorija in imenom lokalnega direktorija, kamor smo želeli klonirati projekt.

```
git svn clone -s --authors-file=<authors-file-path> <svn-repo-url> <cloned-repo-dir-name>
```

Sami smo uporabili zastavico `-s` (sinonim za `--stdlayout`), ker so bili naši SVN-repozitoriji organizirani po standardni strukturi z mapami trunk, branches in tags. Če projekt odstopa od standardne postavitve, je potrebno poti do posameznih delov določiti ročno z zastavicami `--trunk`, `--branches` in `--tags`.

Prenos lahko pri obsežnejših projektih z daljšo zgodovino traja več ur ali celo dni. Med prenosom lahko pride do napak, recimo v primeru prekinjene omrežne povezave. V takšnih primerih lahko postopek ponovno zaženemo z enakim ukazom – git-svn bo nadaljeval tam, kjer je prenos prekinilo, pri tem pa ne potrebujemo izvesti dodatnih ukrepov nad že ustvarjenimi datotekami.

Commiti preneseni, na tak način, so vsebovali metapodatke v obliki, kot je predstavljena z naslednjim primerom.

```
$ git log -1 cc215e71
commit cc215e71797ae4a22df036f1c26907fe0c3bf9be
Author: MartinKrpan <martin.krpan@src.si>
Date:   Wed Jan 1 08:51:10 2025 +0000

    Initial version

    git-svn-id: https://example.svn.repo/trunk@1111 198ea1cc-1dab-4807-93c9-788d0729d413
```

Na dnu sporočila vsakega commita je viden dodatni zapis git-svn-id, ki predstavlja povezavo s pripadajočo SVN-revizijo. Ta metapodatek omogoča nadaljnjo sinhronizacijo z izhodiščnim repozitorijem – na primer za prenos novejših revizij ali dvosmerne integracije.

Če metapodatkov ne želimo ohraniti, lahko pri zagonu ukaza uporabimo še zastavico `--no-metadata`. Vendar v tem primeru izgubimo možnost nadaljnje integracije z izvornim SVN-repozitorijem. Če nam je ta integracija pomembna, metapodatkov pa v končnem repozitoriju vseeno ne želimo, zastavice ne uporabimo, prisotne metapodatke pa nato odstranimo z orodjem git filter-repo, ko integracije več ne potrebujemo. Mi se te integracije sicer nismo posluževali, smo pa metapodatke vseeno obdržali, da lahko jasneje ločimo med commiti, ki so nastali še v SVN-ju in kasnejšimi commiti, ki so bili dodani neposredno v Gitu.

3.3. Koraki po začetnem prenosu

Priprava lokalnega repozitorija brez reference na SVN

Ker povezave do izvornega SVN-repozitorija nismo več potrebovali, smo pred nadaljnjo obdelavo zgodovine najprej pripravili lokalni repozitorij, ki reference na SVN ne vsebuje več. Po prenosu zgodovine z git-svn je v pridobljenem repozitoriju izvorni SVN-repozitorij obravnavan podobno kot navaden oddaljen Git-repozitorij, z določenimi razlikami. Medtem ko lahko dostopamo do vseh commitov preko referenc na oddaljene veje, samega oddaljenega repozitorija ne moremo odstraniti enako, kot če bi šlo za navaden oddaljen repozitorij. Zaradi tega smo se reference na SVN znebili tako, da smo na novo klonirali obstoječ lokalni git-svn-repozitorij. Nov klon tako za oddaljen repozitorij ne bo več imel reference na SVN-repozitorij, temveč bo imel referenco na repozitorij, ki se nahaja v našem lokalnem datotečnem sistemu.

Najprej smo v kloniranem git-svn-repozitoriju za vse oddaljene veje iz SVN-ja ustvarili lokalne veje. S tem smo poskrbeli, da bodo vse veje vidne v klonu, ki je kloniran iz lokalnega git-svn-repozitorija.

```
git branch -r --no-color | grep -v '\->' | \
    while read remote_branch; do git branch "${remote_branch#origin/}" "$remote_branch"; done
```

Za tem smo izvedli kloniranje lokalnega git-svn-repozitorija.

```
1  git clone <local-git-svn-clone> <new-clone>
2  cd <new-clone>
3  git branch -r --no-color | grep -v '\->' | \
4      while read remote_branch; do git branch "${remote_branch#origin/}" "$remote_branch"; done
5  git remote rm origin
```

Pri tem smo v novem klonu morali ponovno izvesti ukaz za kreiranje lokalnih referenc vej, saj so pri svežem klonu veje ponovno dostopne zgolj kot oddaljene reference, s tem da je bil oddaljen repozitorij v tem primeru lokalni git-svn-klon. Na koncu smo odstranili še referenco oddaljenega repozitorija na lokalni git-svn-klon in s tem pridobili repozitorij, ki nima nobene reference oddaljenih repozitorijev.

Prvotni git-svn-klon smo obdržali kot rezervno kopijo, ker je lahko nadaljnje urejanje zgodovine destruktivno glede na prvotno preneseno zgodovino. Z ohranitvijo originalnega klona pa imamo možnost začeti znova brez ponovnega prenašanja z git-svn, če bi ob urejanju zgodovine morebiti izvedli nepopravljivo napako.

Zamenjava vej za oznake z dejanskimi oznakami

Zaradi načina predstavitve oznak v SVN-ju so te pri prenosu repozitorija z git-svn predstavljene kot veje, ki v imenu vsebujejo še predpono s potjo do oznake v datotečni strukturi SVN-repozitorija. Ker so v našem primeru imeli SVN-repozitoriji standardno strukturo, so veje, ki so predstavljale oznake, v imenu vsebovale predpono »tags/«. Oznake smo v Gitu želeli predstaviti kot dejanske oznake, ne kot veje s posebnimi imeni. Zato smo pripravili funkcijo v Pythonu, ki za vsako takšno vejo ustvari ustrezno oznako. Ime oznake pridobi iz imena veje, oznako pa postavi na isti commit, na katerega kaže veja. Po ustvarjeni oznaki funkcija vejo še izbriše, saj ta ni več potrebna.

```python
import subprocess


def tags_from_branches():
    # Ensure the current branch is main
    subprocess.run(["git", "checkout", "main"], check=True)

    # Get all local branches that include 'tags/' in their name
    output = subprocess.check_output(["git", "branch", "--list", "tags/*"], text=True)
    branches = [b.strip() for b in output.splitlines()]

    for tag_branch in branches:
        # Example: "tags/v1.0/v1.0.5" -> "v1.0.5"
        tag_name = tag_branch.split("/")[-1]

        print(f"Creating tag '{tag_name}' from branch '{tag_branch}'")
        subprocess.run(["git", "tag", tag_name, tag_branch], check=True)
        print(f"Deleting branch '{tag_branch}'")
        subprocess.run(["git", "branch", "-D", tag_branch], check=True)
```

Odločili smo se, da oznake iz SVN-ja v Gitu kreiramo kot lahke oznake (angl. lightweight tags), kar pomeni, da oznake ne vsebujejo anotacij s podatki o avtorju, sporočilu in času kreiranja oznake.

3.4. Ureditev zgodovine

Če bi bila prenesena zgodovina brez napak, bi bil repozitorij po kreiranju lokalne kopije in ustvarjanju pravih oznak že pripravljen za nadaljnjo uporabo. V našem primeru pa so se pojavile nepravilnosti v zgodovini, ki so izvirale iz načina, kako git-svn pretvori SVN-strukturo v Git. Zato smo preneseno zgodovino dodatno uredili. Ta faza je bila najzahtevnejši del selitve zgodovine in glavni razlog, da je ta trajala več časa. Kljub temu je bil ta korak nujen, da smo zagotovili Git-repozitorij s pregledno zgodovino, ki ohranja dejanski potek razvoja. V nadaljevanju opisujemo najpomembnejše korake, ki smo jih izvedli za pridobitev urejene zgodovine prenesenega repozitorija v Gitu.

Premik oznak na ustrezne commite

V SVN-ju so oznake mape, ki vsebujejo kopijo vsebine iz določene revizije repozitorija. Oznako se ustvari s commitom, ki zabeleži dejanje kopiranja. Pri prenosu v Git se to izrazi kot samostojen commit, ki v resnici ne vsebuje nobenih sprememb – tako imenovani prazen commit (angl. empty commit).

V Gitu pa oznake niso mape ali commiti, temveč kazalci (angl. pointers) na že obstoječe commite. Zato je oznake primerneje usmeriti neposredno na commit, ki odraža dejansko vsebino določene prelomnice – torej tisti commit,

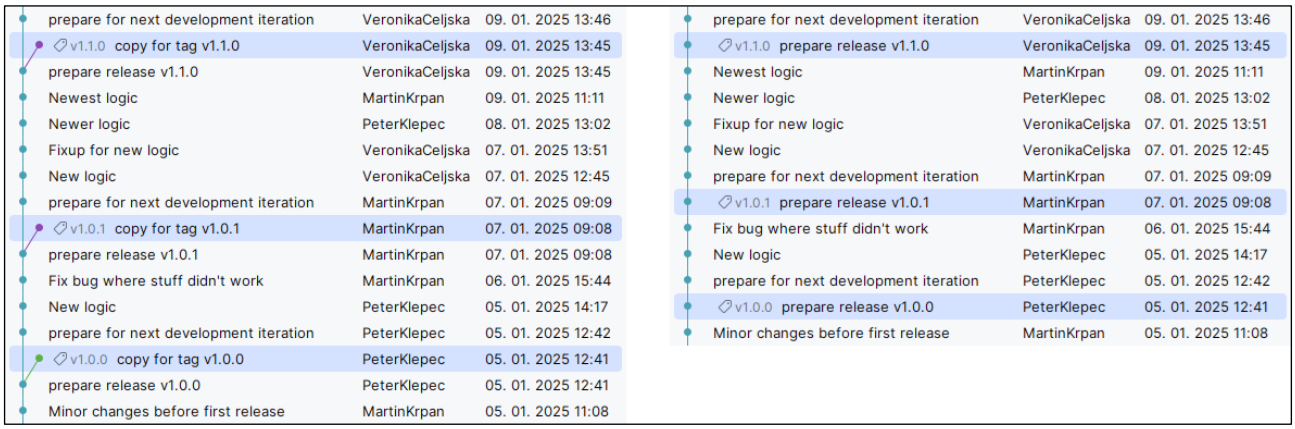
iz katerega je bila v SVN-ju ustvarjena kopija. Če oznake ostanejo na omenjenih umetnih praznih commitih, graf repozitorija v Gitu postane nepregleden, saj so oznake prikazane na stranskih vejah, ločeno od glavne zgodovine. S premikom oznak na pravilnejši commit izboljšamo berljivost grafa in ohranimo logičen potek zgodovine.

Naši projekti so zaradi daljše zgodovine vsebovali večje število oznak, zato smo za njihovo premikanje pripravili funkcijo v Pythonu.

```
1  import subprocess
2
3
4  def move_tags_to_nth_parent_commit(tags, n):
5      for tag in tags:
6          nth_parent_commit_hash = subprocess.check_output(
7              ["git", "rev-list", f"--skip={n}", "-1", tag], text=True
8          ).strip()
9          subprocess.run(["git", "tag", "-f", tag, nth_parent_commit_hash], check=True)
```

Funkcija prejme seznam oznak za premik in število *n*, ki predstavlja globino predhodnega commita, na katerega je potrebno prestaviti vsako podano oznako. Možnost poljubne globine za določen seznam oznak smo podprli zato, ker je bilo v določenih primerih potrebno oznake prestaviti za več kot en predhodni commit, npr. če so bili prisotni dodatni prazni commiti, ki so nastali zaradi sprememb lokacije oznak v datotečni strukturi.

Na Sliki 1 je prikazan primer grafa repozitorija pred in po premiku oznak. Levo vidimo stanje pred premikom, kjer se oznake nahajajo na ločenih commitih, ki se vejijo iz glavne veje repozitorija. Desno pa je prikazano stanje po popravku. Oznake so premaknjene na primernejše commite na glavni veji, v tem primeru na njihove direktne predhodnike. Graf ima sedaj znižano kompleksnost in pravilnejši potek glede na način uporabe oznak v Gitu. V obeh primerih so za lažjo primerjavo označene vrstice commitov z oznakami.



Slika 1: Primerjava grafa repozitorija pred (levo) in po (desno) premiku oznak na ustrezne commite.

Obnovitev pravilnega poteka grafa zgodovine

Po začetnem prenosu zgodovine iz SVN-ja pogosto ni bila pravilno ohranjena struktura vejitev in združitev (angl. branching and merging). Pogosto se je namreč zgodilo, da določene združitve niso bile pravilno predstavljene z združitvenimi commiti, ali pa da veje niso izvirale iz commita, ki je predstavljal njihov pravi izhodiščni commit (angl. base commit) v SVN-ju. Zgodovina je bila posledično neustrezna za dolgoročno vzdrževanje in preglednost. Takšno stanje smo morali ročno popraviti, kar je vključilo rekonstrukcijo poteka vej z operacijo rebase in rekreacijo združitvenih commitov, kjer so bili ti napačno predstavljeni kot navadni commiti.

Takoj ko spremenimo potek grafa, moramo izvesti operacijo rebase še za vse commite od točke spremembe do aktualnega HEAD-commita. To velja tudi za veje in združitvene commite, ki so sicer pravilno zaznani. Ker so commiti v Gitu nespremenljivi, z operacijo rebase ustvarimo novo različico commita, če z operacijo spremenimo vsaj eno lastnost commita, kot je denimo njegov predhodni commit. Zaradi tega je izvajanje operacije rebase, še posebej globoko v zgodovini, lahko zelo zamudno, saj pomeni rekreiranje commitov, skupaj z vejitvami in

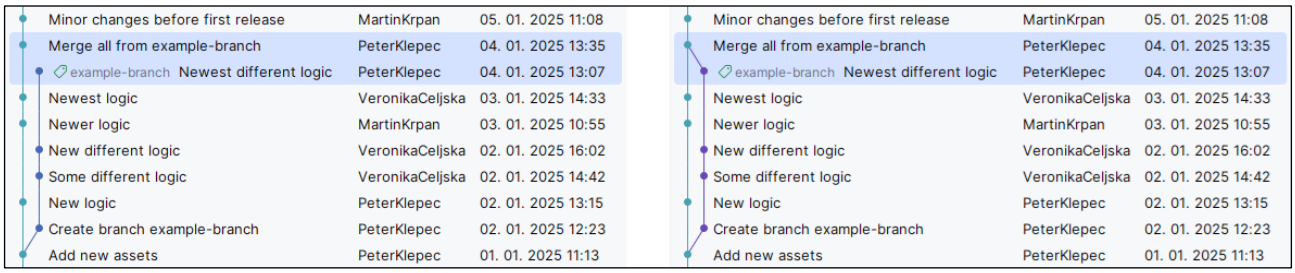
združitvami, na celotni poti do HEAD-commita. Če se na spremenjeni poti grafa nahajajo oznake, jih je potrebno po spremenjenem poteku grafa prestaviti na ustrezne nove commite, ki smo jih ustvarili z operacijo rebase, saj se lokacije oznak samodejno ne spreminjajo.

Operacijo rebase smo vedno izvajali z zastavico `--committer-date-is-author-date`, saj bi novi commiti v nasprotnem primeru za datum ustvarjanja commita imeli nastavljen datum, ko smo izvedli rebase. Na podoben način Git ločuje tudi med avtorjem commita in tistim, ki je commit dejansko izvedel (angl. committer). Slednji podatek se ob izvedbi operacije rebase za nove commite nastavi na vrednosti iz naših lokalnih nastavitev imena in elektronskega naslova za repozitorij. Za ohranjanje enakosti med avtorjem in committerjem ne potrebujemo skrbeti sproti, saj lahko to dosežemo za celotno zgodovino z uporabo git filter-repo, kar je tudi opisano kasneje v poglavju.

Nezaznani združitveni commiti

Ob prenosu zgodovine smo srečevali več ponavljajočih anomalij, ki smo jih morali odpravljati. V nadaljevanju pa podrobneje opisujemo napako, ki je bila po našem mnenju nabolj kritična Pri določenih združitvah git-svn ob prenosu ni zaznal, da gre za združitveni commit. Čeprav je stanje projekta pravilno ustrezalo rezultatu združitve, je bil commit za združitev predstavljen kot običajen linearen commit, kar pomeni, da se podrobnejša zgodovina združene veje na glavni veji ni ohranila. Commiti združene veje so bili sicer normalno preneseni z git-svn, a ti niso bili povezani s celoto. Rešitev je bila, da te združitvene commite ročno rekreiramo, nato pa vse nadaljnje commite z operacijo rebase prekopiramo na nov združitveni commit.

Nepravilno stanje združitve je prikazano na levi strani Slike 2. Na desni strani Slike 2 pa je prikazano stanje na grafu, ki smo ga želeli doseči z rekreacijo združitvenega commita. Razlika na samem grafu vizualno ni drastična, vendar pa je vsebinsko pomembna, da lahko na glavni veji pravilno prepoznamo dejanski izvorni commit posameznih sprememb preko ukazov, kot je `git blame`.



Slika 2: Primerjava grafa zgodovine pred (levo) in po (desno) rekreaciji združitvenega commita.

Pri rekreaciji združitvenega commita je bilo potrebno razrešiti tudi vse morebitne konflikte. Ker je vsak nepravilno zaznan združitveni commit, predstavljen kot navaden commit, vseboval pravilno stanje ob posamezni združitvi, je tako vseboval tudi stanje razrešenih konfliktov. Ker je nemogoče zagotavljati, da bi ob vsaki rekreaciji združitve pravilno razrešili konflikte, smo vsebino novega združitvenega commita z ukazom `git read-tree --reset -u` vedno nastavili na zgodovino prvotnega nepravilno zaznanega združitvenega commita. Ker je bila vsebina novega združitvenega commita s tem identična izvornemu linearnemu commitu, so bili vsi konflikti razrešeni na enak način, kot so bili razrešeni v prvotnem commitu. Pri ustvarjanju novega združitvenega commita pa smo prav tako morali vse metapodatke, kot so avtor, čas in sporočilo, nastaviti na vrednosti, ki so prisotne na prvotnem commitu. Pri tem smo uporabljali `git commit --amend`. Za avtomatizacijo opisanega postopka smo implementirali skript v Pythonu, ki glede na izvoren commit rekreira združitveni commit z enako vsebino in metapodatki na novi lokaciji.

```
1  import argparse
2  import os
3  import subprocess
4
5
6  def run_git_command(cmd, *, capture_output=False, text=True, env=None):
7      std = subprocess.PIPE if capture_output else None
8      return subprocess.run(["git"] + cmd, stdout=std, stderr=std, text=text, env=env, check=True)
```

```
 9
10
11  def get_git_output(cmd):
12      return subprocess.check_output(["git"] + cmd, text=True).strip()
13
14
15  def amend_commit(original_commit):
16      author = get_git_output(["log", "-1", "--pretty=%an <%ae>", original_commit])
17      date = get_git_output(["log", "-1", "--pretty=%ci", original_commit])
18
19      env = os.environ.copy()
20      env["GIT_COMMITTER_DATE"] = date
21      run_git_command(
22          ["commit", "--amend", "--no-edit", f"--author={author}", f"--date={date}"],
23          env=env
24      )
25
26
27  def recreate_merge(what_to_merge, where_to_merge, original_merge_commit, branch_to_reset=None):
28      # Use commit hash to avoid updating branch HEAD if 'where_to_merge' is a branch name
29      where_to_merge_hash = get_git_output(["rev-parse", where_to_merge])
30      run_git_command(["checkout", where_to_merge_hash])
31
32      # Merge but don't commit yet
33      run_git_command(["merge", "--no-commit", "--no-ff", what_to_merge])
34      # Replace working tree and index to match the original merge commit exactly
35      run_git_command(["read-tree", "--reset", "-u", original_merge_commit])
36
37      orig_commit_message = get_git_output(["log", "-1", "--pretty=%B", original_merge_commit])
38      run_git_command(["commit", "-m", orig_commit_message])
39      amend_commit(original_merge_commit)
40
41      # Reset working tree to clean state
42      run_git_command(["reset", "--hard", "HEAD"])
43
44      if branch_to_reset is not None:
45          run_git_command(["checkout", "-B", branch_to_reset])
46
47
48  if __name__ == "__main__":
49      parser = argparse.ArgumentParser(description="Recreate a merge commit at a new base.")
50
51      parser.add_argument("what", help="The branch or commit to merge (the 'merged' side)")
52      parser.add_argument("into", help="The branch or commit to merge into (the base)")
53      parser.add_argument("orig", help="The original merge commit to recreate")
54      parser.add_argument(
55          "--B", help="Branch name to update to the new merge result (optional)", default=None
56      )
57
58      args = parser.parse_args()
59      recreate_merge(args.what, args.into, args.orig, branch_to_reset=args.B)
```

Skript prejme tri parametre. S prvim parametrom povemo, kaj želimo združiti. To je v večini primerov HEAD-commit veje, ali pa ime veje, ki jo želimo združiti. Drug parameter pove, kam združujemo oz. kje želimo izvesti združitev. Ta commit predstavlja predhodnika združitvenega commita na veji, v katero združujemo željeno drugo vejo. Kot tretji parameter podamo referenco na obstoječ združitveni commit, iz katerega se bo črpalo vsebino sprememb in metapodatke commita. V danem primeru je to združitveni commit, ki je nepravilno predstavljen kot linearen commit. Skript prejme še dodaten opcijski parameter za ime veje, za katero se bo izvedlo njeno kreiranje ali posodobitev lokacije na novo ustvarjen združitveni commit.

Skript je sposoben rekreacije združitve tako iz linearnih commitov kot tudi iz pravih združitvenih commitov. Zaradi tega smo ga uporabljali tudi ob rekreaciji vseh pravilno zaznanih združitvenih commitov na poteh, za katere je bilo potrebno izvesti operacijo rebase. Operacija rebase sicer podpira rekreiranje združitev z uporabo zastavice `--rebase-merges`, vendar je bila slednja za nas neuporabna, saj moramo z njeno uporabo vse konflikte ob združevanjih vseeno razreševati ročno.

Premik oznak na nove commite

Ko smo z operacijo rebase zaključili urejanje celotne zgodovine, smo morali obstoječe oznake prestaviti na ustrezna mesta v novi strukturi repozitorija. V ta namen smo pripravili dve funkciji v Pythonu. Prva v datoteko JSON shrani podatek o trenutni lokaciji oznak, druga pa na podlagi te datoteke določi ustrezne nove commite, dostopne iz podane veje, na katere prestavi oznake.

```python
import json
import subprocess


def extract_svn_revision(commit_msg):
    return commit_msg.split("git-svn-id:")[1].split()[0].split("@")[-1]


def save_revisions_tags_json(json_save_path):
    tags = subprocess.check_output(["git", "tag"], text=True).splitlines()
    tags = [t.strip() for t in tags if t.strip()]

    revisions_tags = {}
    for t in tags:
        log_output = subprocess.check_output(["git", "log", "-1", t], text=True)
        revision = extract_svn_revision(log_output)
        revisions_tags[revision] = t

    with open(json_save_path, "w") as f:
        json.dump(revisions_tags, f, indent=4)


def tags_from_revisions_json(target_branch, revisions_json_file):
    subprocess.run(["git", "checkout", target_branch])

    with open(revisions_json_file) as f:
        revisions_tags = json.load(f)

    target_revisions = set(revisions_tags)
    found_revisions = set()

    git_log = subprocess.check_output(["git", "log"], text=True).split("commit ")
    git_log = [commit for commit in git_log if commit]

    for commit in git_log:
        svn_revision = extract_svn_revision(commit)
        if svn_revision not in target_revisions:
            continue

        commit_hash = commit.splitlines()[0]
        tag = revisions_tags[svn_revision]
        subprocess.run(["git", "tag", "-f", tag, commit_hash], check=True)

        found_revisions.add(svn_revision)
        if len(found_revisions) == len(target_revisions):
            break

    print("Done.")
```

Za podatek o lokaciji oznak smo izkoristili SVN-metapodatke o revizijah v sporočilih commitov, ki se ohranijo tudi v novih commitih, ustvarjenih z operacijo rebase. Če bi se odločili, da SVN-metapodatkov ob prenosu zgodovine z git-svn ne uporabimo, pa bi za enolično določanje commitov za lokacije oznak morali uporabiti drug nabor metapodatkov, kot je recimo avtor commita v kombinaciji s časom in sporočilom commita.

Odstranitev nepotrebnih vej

V SVN-ju so veje predstavljene kot mape, zato v zgodovini ostane podatek o veji tudi po tem, ko je bila njena mapa izbrisana. Posledično se pri prenosu v Git ohranijo vse veje, ki so kadar koli obstajale v razponu prenesenih revizij – tudi tiste, katerih mape v trenutnem stanju repozitorija ne obstajajo več. Zato smo po zaključku ureditve grafa repozitorija v Gitu izbrisali vse veje, ki jih nismo več potrebovali. S tem smo zagotovili, da se te ne prenašajo naprej in ne ustvarjajo zmede pri nadaljnjem razvoju.

Dodatna ureditev s pomočjo git filter-repo

S predhodnimi koraki smo zgodovino repozitorija že skoraj v celoti uredili. Obstaja pa še nekaj dodatnih korakov, ki smo jih po končanem urejanju izvedli z orodjem git filter-repo, da smo dosegli zares čisto zgodovino. Pri uporabi ukaza `git filter-repo` smo vedno uporabljali zastavico `--force`, saj orodje git filter-repo ne dopušča destruktivnih akcij v repozitorijih, za katere zazna, da niso sveži. V naših primerih smo zgodovino že ročno urejali, tako da ni šlo več za sveže klonirane repozitorije. Kljub temu smo pred vsako uporabo orodja ustvarili varnostno kopijo repozitorija, da nismo izgubili celotnega napredka v primeru, da z uporabljenim ukazom ne bi pridobili željenega rezultata.

Prazni commiti

Omenili smo že, da so oznake in veje v SVN-ju predstavljene kot mape. Kreira se jih tako, da se določeno revizijo kopira na potrebno mesto, kar se izvede z novim commitom. Ker v Gitu oznak in vej ne predstavljamo z mapami, so takšni commiti, kjer se zgolj ustvarja novo mesto v strukturi SVN-repozitorija, prisotni, a so vsebinsko prazni (angl. empty commits). Takšne commite za oznake smo že odstranili s premikom oznak. V repozitoriju pa so lahko še vedno prisotni drugi prazni commiti, kot so commiti za ustvarjanje vej. S spodnjim ukazom takšne commite odstranimo iz repozitorija.

```
git filter-repo --force --prune-empty always
```

Ukaz filter-repo bo poskrbel, da je potek grafa ohranjen, kljub temu, da so prazni commiti odstranjeni. Zaradi odsotnosti praznih commitov pa bodo vsi commiti, ki jim sledijo, v resnici novi commiti z drugo hash-vrednostjo, podobno kot če bi izvedli operacijo rebase. Prednost uporabe git filter-repo pa je, da se vse združitve, lokacije oznak in metapodatki ohranijo, ne da bi jih potrebovali dodatno urejati sami.

Enakost med avtorjem in committerjem

Git ločuje med avtorjem commita in tistim, ki je dejansko ustvaril commit (angl. committer). Ob izvedbi operacije rebase se lahko za novo nastale commite podatka o avtorju in committerju razlikujeta, saj je podatek o committerju nastavljen na lokalne podatke tistega, ki je izvedel rebase, medtem ko se vrednost avtorja ohrani. Takšni metapodatki pa ne predstavljajo realnega stanja prenesene zgodovine iz SVN, zato smo s pomočjo git filter-repo vrednosti za committerja nastavili nazaj na vrednosti avtorja za celoten repozitorij, pri čemer smo uporabili spodnji ukaz.

```
git filter-repo --force --commit-callback '
    commit.committer_name = commit.author_name;
    commit.committer_email = commit.author_email;'
```

Manipulacija datotečne strukture projekta

Ker gre pri projektih, ki smo jih prenašali na Git, za starejše projekte, so se v preteklosti zaradi takratnih tehnoloških omejitev uporabljale prakse, ki danes niso več aktualne. Binarne datoteke, potrebne za delovanje aplikacije, se je tako dodajalo v repozitorij, njihovo vsebino pa se je skupaj z razvojem aplikacije tudi spreminjalo. Git spremembe binarnih datotek obravnava tako, da ob vsaki spremembi shrani celotno vsebino datoteke. To lahko privede do visoke velikosti repozitorija na disku, kar otežuje vsakodnevno delo razvijalcev, saj imamo v Gitu celotno zgodovino repozitorija preneseno lokalno. Zaradi tega smo se odločili, da sledenje takšnim datotekam v Gitu odstranimo skozi celotno zgodovino. Spodaj sta primera ukazov, ki jih lahko uporabimo za odstranitev datoteke ali direktorija iz zgodovine.

```
# Odstranitev datoteke
git filter-repo --force --path path/to/file --invert-paths
# Odstranitev direktorija
git filter-repo --force --path path/to/dir/ --invert-paths
```

Orodje git filter-repo bo sledenje podani datoteki ali direktoriju odstranilo skozi celotno zgodovino, hkrati pa se bodo odstranili tudi vsi commiti, ki so se nanašali izključno samo na to datoteko ali direktorij.

Ker smo določene binarne datoteke odstranili iz zgodovine, smo lahko prilagodili tudi datotečno strukturo projekta, da je bila ta primerna za posodobljen nabor datotek. Z orodjem git filter-repo smo tako določenim direktorijem spremenili lokacijo znotraj projekta. Naslednja primera ukazov prikazujeta dve možnosti premika – prvi primer predstavlja premik vsebine mape na popolnoma novo pot, drug primer pa predstavlja premik vsebine mape na korensko pot repozitorija.

```
# Premik na novo mesto
git filter-repo --force --path path/to/dir/ --path-rename path/to/dir/:new/path/to/dir/
# Premik na koren repozitorija
git filter-repo --force --path path/to/dir/ --path-rename path/to/dir/:
```

3.5. Objava oddaljenega Git-repozitorija

S predhodnimi koraki smo dosegli čisto zgodovino repozitorija s pravilnim potekom grafa. Takšen repozitorij smo nato objavili na interni platformi za gostovanje Git-repozitorijev. Na platformi najprej ustvarimo nov prazen repozitorij, v lokalnem repozitoriju pa zanj dodamo novo referenco oddaljenega repozitorija. Na dodan oddaljen repozitorij izvedemo push vseh vej in vseh oznak.

```
1  git remote add origin <new-remote-repo-url>
2  git push origin --all
3  git push origin --tags
```

Oddaljen repozitorij lahko ustvarimo tudi, če zaenkrat razvojnega procesa še nimamo namena v celoti preseliti na Git. Zgodovino nastalega repozitorija lahko namreč do končne migracije sproti posodabljamo, hkrati pa nam bo nov repozitorij omogočal testiranje vseh CI/CD procesov, ki jih moramo posodobiti, da ti pravilno delujejo v Gitu.

3.6. Posodabljanje prenesene zgodovine

Če po začetni selitvi zgodovine razvoj še vedno poteka v SVN-ju, lahko zgodovino v Gitu posodabljamo z novejšimi revizijami iz SVN-ja, ki so nastale po začetnem prenosu zgodovine. Tako zagotovimo, da bo repozitorij v Gitu ob prehodu že vseboval celotno zgodovino, ne da bi bilo potrebno čakati še na njeno dokončno pripravo.

Če bi datotečna struktura repozitorija ostala neobdelana, bi lahko repozitorij posodabljali kar preko orodja git-svn. Pogoj za to je sicer, da sporočila v commitih vsebujejo potrebne metapodatke o številki revizije posameznega commita in da lokalne veje v Gitu sledijo oddaljenim vejam v SVN-repozitoriju. Ker pa smo v našem primeru datotečno strukturo spreminjali, to ni bilo več mogoče, ker stanje projekta v Gitu ni bilo več skladno s stanjem projekta v SVN-ju. Zato smo zgodovino do časa prehoda na Git posodabljali tako, da smo delno prenašali novo nastalo zgodovino, jo uredili na enak način kot obstoječo zgodovino in jo po ureditvi dodali k skupni zgodovini.

Delno zgodovino smo z ukazom `git svn clone` prenesli tako, da smo z uporabo zastavice `-r` podali razpon revizij za prenos.

```
git svn clone -s -r <revision-number>:HEAD --authors-file=<authors-file-path> \
    <svn-repo-url> <cloned-repo-dir-name>
```

Za začetno revizijo smo izbrali takšno, ki je že prisotna v prvotni zgodovini, da smo lahko s tem primerjali vsebino commitov in se prepričali, da je nova zgodovina po njenem urejanju v pravilnem vsebinskem stanju. Za končno revizijo smo podali HEAD, torej smo zgodovino prenesli vse od podane do zadnje nastale revizije. Uporabili smo obstoječo datoteko avtorjev, ki smo jo ustvarili že ob prvotnem prenosu zgodovine. Pomembno je, da v takšnem primeru v datoteko dodamo vse morebitne avtorje, ki so od zadnjega prenosa naredili svoj prvi prispevek v določenem projektu in zato še niso bili prisotni v prvotni različici datoteke.

Ko je bila nova zgodovina urejena, smo jo vključili v obstoječ repozitorij s pomočjo začasne oddaljene povezave na lokalni repozitorij z novo zgodovino. Nato smo pridobili oznake in vse relevantne veje, za slednje pa ustvarili

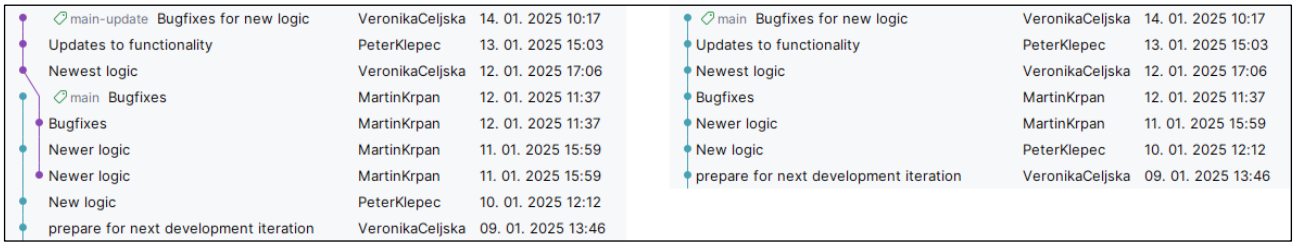
še lokalne reference. Po kreiranju lokalnih vej smo referenco oddaljene povezave odstranili, ker je nismo več potrebovali.

```
git remote add update <local-repo-with-new-history>
git fetch update main --tags
git checkout -b main-update update/main
git remote rm update
```

Zgornji nabor ukazov prikazuje ustvarjanje lokalne veje samo za nove commite na glavni veji, ki je v našem primeru poimenovana »main«. Če so v novi zgodovini nastale nove veje, ali pa so bili dodani novi commiti na obstoječe veje, moramo ustvariti lokalne reference tudi za takšne veje.

Novo ustvarjene veje bodo v obstoječem repozitoriju prikazane kot osamele veje (angl. orphan branches), saj zaradi delnega prenosa nimajo skupnega izhodišča (angl. base) z obstoječo zgodovino, temveč je izhodišče commit za tisto revizijo, od katere smo prenesli delno zgodovino. Nove commite dodamo k obstoječi zgodovini tako, da izvedemo rebase. Če smo prenesli tudi takšne commite, ki so v obstoječi zgodovini že prisotni, pazimo, da takšnih commitov ob izvedbi operacije rebase ne ohranjamo.

Slika 3 na levi strani prikazuje graf repozitorija, kjer je v repozitorij že dodana veja z novo zgodovino kot osamela veja. Ta osamela veja vsebuje tudi dva commita, ki sta na obstoječi veji že prisotna. Na desni strani pa je prikazan graf repozitorija, kjer se je z operacijo rebase commite iz nove veje dodalo na obstoječo vejo. Z operacijo rebase so bili dodani zgolj trije commiti, ki na obstoječi veji predhodno niso obstajali.



Slika 3: Dodajanje nove zgodovine k obstoječi zgodovini.

4 Priprava delovnega procesa

Selitev na Git smo izkoristili kot priložnost za prenovo razvojnega procesa. Ker Git temelji na drugačnih konceptih kot SVN, smo si prizadevali zasnovati delovni tok, ki izkorišča prednosti novega sistema, namesto da bi vanj preslikali stare prakse. Če Git obravnavamo na enak način kot SVN, hitro izgubimo njegove ključne prednosti in s tem tudi smisel same selitve. Pri snovanju novega delovnega procesa smo se zgledovali po uveljavljenih tokovih za Git, kot so GitFlow [8], GitHub Flow [9], GitLab Flow [10] in trunk-based development [11]. Te pristope smo nato prilagodili lastnim potrebam in značilnostim projektov. Ključno vodilo je bilo, da izkoristimo lahkotnost in fleksibilnost, ki jo Git ponuja pri delu z vejami, hkrati pa izkoriščamo njegove naprednejše možnosti, kot je operacija rebase, kjer je to smiselno ali mogoče.

4.1. Uporaba zahtev za združitev za integracijo pregleda kode

Najpomembnejša razlika med Gitom in SVN-jem je izredno učinkovito in hitro upravljanje z vejami. V Gitu ustvarjanje, preklapljanje in združevanje vej predstavlja osnovo vsakodnevnega dela, medtem ko je bilo v SVN-ju to pogosto nepraktično in rezervirano za večje spremembe. Ta lastnost je upoštevana v večini najpogosteje uporabljanih delovnih tokovih, saj omogoča izoliran razvoj funkcionalnosti in popravkov, večjo preglednost sprememb in lažje sodelovanje med člani ekipe.

Večina sodobnih platform za Git, kot so GitHub, GitLab in Bitbucket, dodatno omogočajo zahteve za združitev sprememb iz ene veje v drugo vejo (angl. pull requests ali merge requests). V sklopu zahteve za združitev je možno pregledati vse spremembe, ki bi se z združitvijo dodale, pred samo združitvijo pa zahtevati popravke, ali pa zahtevo celo zavrniti. To nam omogoča neposreden pregled kode ob vsaki zahtevi za združitev. Ker smo se odločili, da naš delovni tok osnujemo glede na že uveljavljene tokove, smo s tem tudi začeli uporabljati ločene veje za razvoj sprememb. Njihovo združevanje v glavno vejo smo izvajali izključno preko zahtev za združitev.

Z začetkom uporabe zahtev za združitev smo uvedli tudi sistemsko zahtevo po pregledu vsake spremembe pred vključitvijo v glavno vejo razvoja. Pregledi kode tako niso bili več prepuščeni dosledni disciplini sprotnega pregledovanja novih revizij, temveč so postali obvezen in naravno vgrajen korak v razvojnem ciklu. S tem smo povečali robustnost procesa, zmanjšali možnost spregledanih napak ter izboljšali skupno kakovost kode.

4.2. Definicija delovnega toka in CI/CD procesa

V uvodnem delu prispevka smo že omenili, da je bila selitev projektov iz platforme A na platformo B pomembna prelomnica, ob kateri smo se zaradi praktičnih razlogov odločili tudi za selitev na Git. Glavni razlog je bil v tem, da smo morali v prehodnem obdobju istočasno podpirati delovanje aplikacij na obeh platformah. Nove funkcionalnosti so se razvijale na platformi A, saj so aplikacije še vedno delovale na njej, a so morale biti hkrati pripravljene tudi za delovanje na platformi B.

Prenova delovnega toka pa je zahtevala tudi prilagoditev CI/CD-procesa. Že v času SVN-ja smo uporabljali avtomatizirano gradnjo in nameščanje, zato smo želeli to prakso ohraniti in nadgraditi. Ker smo vedeli, da bo podpora platforme A v prihodnosti ukinjena, smo se odločili, da CI/CD proces za platformo B v celoti prenovimo, medtem ko za platformo A ni bilo smiselno uvajati večjih sprememb. V Gitu smo to lahko bistveno lažje izvedli z uporabo ločenih razvojnih kopij repozitorija za vsako platformo posebej. Tako je imel vsak repozitorij svoj lasten CI/CD proces, popolnoma ločen in prilagojen specifičnim zahtevam platforme, ne da bi en proces vplival na drugega.

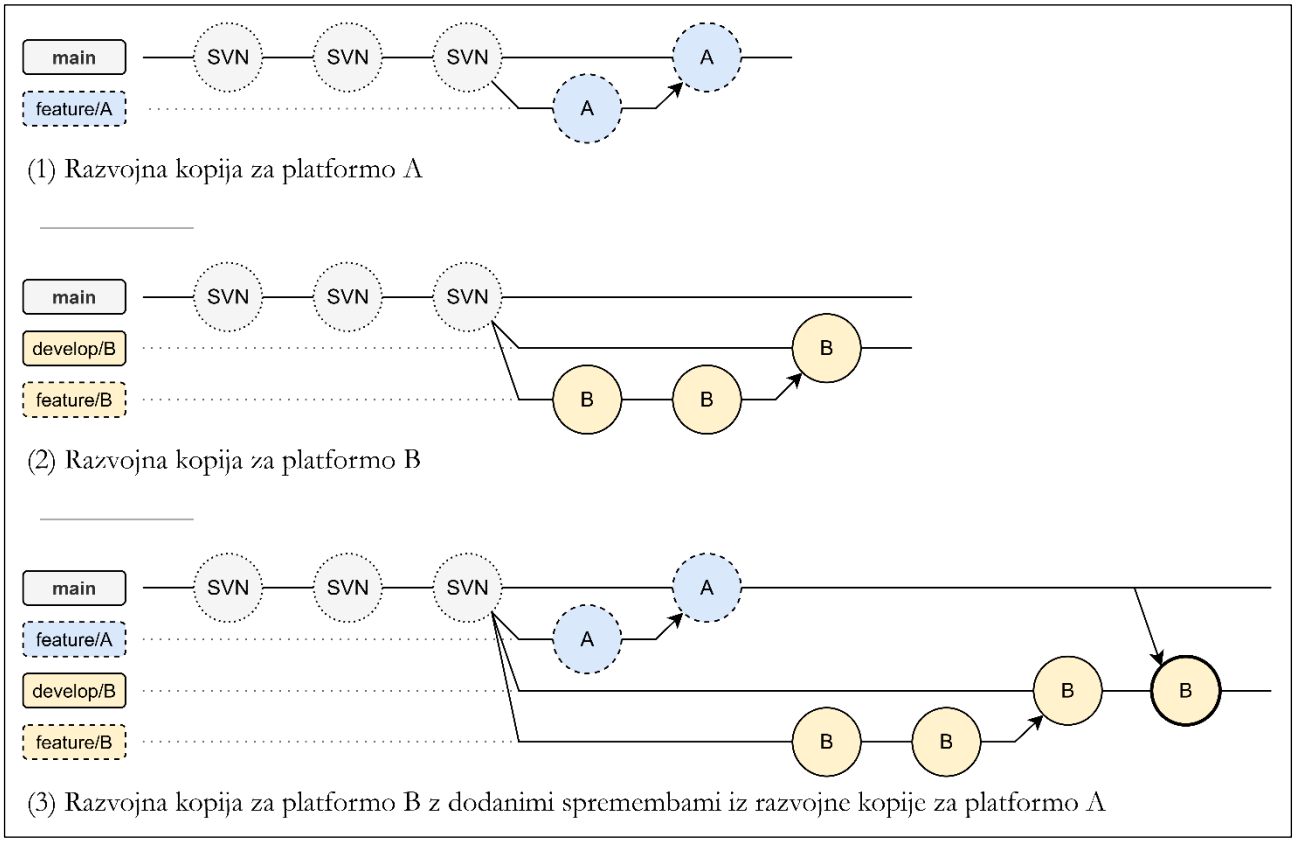
Za platformo A smo CI/CD proces prilagodili zgolj minimalno – toliko, da je ta pravilno deloval z Git-repozitorijem. Za delo z Gitom smo uporabili delovni tok, ki temelji na pristopu, imenovanem trunk-based development, podobno kot pri delu s SVN. Kljub temu smo v Gitu uvajali ločene veje za posamezne funkcionalnosti, ki smo jih nato dodajali v glavno vejo s pomočjo zahtev za združitev. Na ta način smo obdržali prednosti, ki jih Git prinaša, kot je vgrajen pregled kode, ne da bi pretirano spreminjali obstoječ proces razvoja na platformi A. Na ta način smo omogočili lažji prehod z minimalnim vložkom, saj platforma A dolgoročno ni bila predvidena za razvoj.

Za platformo B pa smo uvedli bolj strukturiran delovni tok, ki temelji na pristopu GitFlow. Platforma B je imela svojo ločeno razvojno vejo, v katero smo dodajali tehnične spremembe, potrebne za pravilno delovanje aplikacij na novi platformi. Poslovna logika se je še vedno v celoti razvijala na platformi A, deljena izvorna zgodovina obeh kopij repozitorijev pa nam je omogočila, da smo spremembe poslovne logike periodično sinhronizirali s tehničnimi spremembami za platformo B. CI/CD proces smo za platformo B vzpostavili povsem na novo, skladno z izhodišči pripravljenega delovnega toka. Postavili smo novo strukturo cevovodov in poskrbeli za testiranje vseh ključnih funkcionalnosti.

Tak pristop nam je omogočil stabilno in pregledno upravljanje sprememb v obeh platformah, hkrati pa smo lahko CI/CD procese za posamezno platformo neodvisno razvijali, vzdrževali in testirali. Tovrstna prilagodljivost razvojnih tokov in CI/CD procesov bi bila bistveno težje izvedljiva v SVN-ju, kjer delo z več vejami in več repozitoriji ni podprto na enako učinkovit način kot v Gitu. Primer opisanega postopka prav tako ponazoruje Slika 4, ki predstavlja tri dele postopka:

- pod (1) je prikazana razvojna kopija repozitorija za platformo A,
- pod (2) razvojna kopija repozitorija za platformo B,

- pod (3) pa je prikazan prenos sprememb iz razvojne kopije za platformo A v razvojno kopijo za platformo B.



Slika 4: Hkratni razvoj za obe platformi v ločenih kopijah repozitorija.

4.3. Testiranje CI/CD procesa

Ker je bil CI/CD proces pomemben del prehoda na Git in hkrati ključen za zanesljiv potek razvoja, smo ga morali ustrezno preveriti, še preden smo omogočili dejanski razvoj v Gitu. V ta namen smo pripravili testne kopije repozitorijev za platformi A in B, s katerimi smo simulirali običajen potek razvoja, objavljanja in izdaje novih različic glede na posamezen delovni tok.

Testni repozitoriji so bili pripravljeni z namenom omogočanja izvajanja vseh CI/CD korakov, ne da bi s tem vplivali na produkcijsko kodo. Vanje smo lahko dodajali poljubne spremembe, ki so sprožile cevovode, kar nam je omogočilo preizkus vseh ključnih točk v procesu tako za platformo A kot tudi platformo B.

S predhodnim testiranjem smo zmanjšali tveganje za napake pri dejanskem prehodu, saj so bili vsi postopki testirani v okolju, ki je posnemalo dejansko uporabo. CI/CD procesi so bili tako v trenutku prehoda pripravljeni na uporabo in integrirani v prenovljen delovni tok. To nam je omogočilo gladek prehod na Git brez zastojev v razvoju in brez potrebe po dodatnem usklajevanju po začetku dela v novem sistemu.

5 Zaključek

Migracija iz SVN na Git je bila za naše projekte več kot le tehnična posodobitev – šlo je za strateško odločitev, ki je vplivala na razvojne procese, orodja in sodelovanje v ekipi. Postopek selitve je bil zaradi zgodovine, ki je zajemala več let razvoja in veliko število revizij, tehnično zahteven. Kljub temu smo s skrbnim načrtovanjem uspeli selitev delovnega procesa izvesti z ohranitvijo celotne zgodovine projektov in postavili temelje za učinkovit razvojni proces.

V prispevku smo opisali celoten postopek selitve zgodovine – od začetnega prenosa s pomočjo orodja git-svn, urejanja avtorjev in oznak, do obsežnih korakov za rekonstrukcijo logične strukture repozitorija, ki je bila zaradi razlik med sistemoma SVN in Git pogosto neustrezno interpretirana. Pri tem smo uporabili lastne skripte, s katerimi smo poskrbeli za avtomatizacijo ponavljajočih se nalog. Migracijo zgodovine smo izvedli dovolj zgodaj v procesu, saj gre za časovno najzahtevnejši korak, ki pa ne ovira nadaljnjega razvoja v SVN-ju. To nam je omogočilo vzporedno pripravo novih CI/CD procesov, postavitev delovnega toka in testiranje na dejanskem repozitoriju, še preden je bil izveden dejanski prehod razvojnega procesa.

Z uporabo Gita smo uvedli prenovljen delovni tok, ki temelji na preizkušenih praksah. Uvedli smo izolirano delo na funkcionalnih vejah in integrirane preglede kode preko obveznih zahtev za združitev. S tem smo povečali kakovost kode, zmanjšali možnost napak in omogočili večjo odgovornost posameznikov znotraj ekipe. Poleg tega smo lahko Git z njegovo fleksibilnostjo in distribuirano naravo učinkovito izkoristili tudi za vzporedno vzdrževanje dveh aplikacijskih platform, ki sta imeli povsem ločena CI/CD procesa in razvojna poteka, ne da bi pri tem ogrožali stabilnost drug drugega.

S prehodom na Git smo vzpostavili temelje, ki omogočajo dolgoročno razširljivost in večjo agilnost ekipe. Git zaradi svoje razširjenosti in priljubljenosti omogoča lažje uvajanje sodobnih orodij in pristopov, ki se skupaj s tehnologijami nenehno razvijajo. Fleksibilnost Gita pomeni, da lahko sproti prilagajamo delovni tok glede na nove potrebe projekta in ekipe, pri tem pa ne potrebujemo spreminjati osnovnega orodja ali strukture repozitorija. Prehod se je tako izkazal kot pomembna naložba v prihodnost projektov – izboljšal je preglednost, omogočil boljšo integracijo z drugimi orodji, olajšal uvajanje novih razvijalcev ter poenotil način dela z ostalimi ekipami v podjetju. Mlajši kader, ki že pozna delo z Gitom, se po novem lažje vključuje v obstoječe procese na teh projektih, kar zmanjšuje stroške uvajanja in omogoča hitrejši začetek produktivnega dela.

S prispevkom smo delili izkušnje, ki smo jih pridobili med selitvijo na Git, glede na dosežen rezultat pa bi selitev priporočili tudi drugim ekipam, ki še vedno uporabljajo SVN ali druge starejše rešitve. Čeprav je selitev lahko zahtevna, se njen doprinos hitro obrestuje – tako skozi izboljšano kakovost razvoja kot tudi skozi večjo preglednost, avtomatizacijo in podporo sodobnim pristopom razvoja programske opreme.

Literatura

[1] git-scm.com/book/ms/v2/Getting-Started-A-Short-History-of-Git, A Short History of Git, obiskano 26. 6. 2025.

[2] git-scm.com/docs/git-svn, git-svn Documentation, obiskano 30. 6. 2025.

[3] github.com/nirvdrum/svn2git, svn2git: Ruby tool for importing existing svn projects into git., obiskano 30. 6. 2025.

[4] subgit.com, SVN to Git Migration – TMate SubGit, 30. 6. 2025.

[5] github.com/newren/git-filter-repo, git filter-repo, obiskano 30. 6. 2025.

[6] www.jetbrains.com/idea, IntelliJ IDEA – the IDE for Pro Java and Kotlin Development, obiskano 10. 7. 2025.

[7] git-scm.com/book/en/v2/Appendix-A%3A-Git-in-Other-Environments-Git-in-IntelliJ-/-PyCharm-/-WebStorm-/-PhpStorm-/-RubyMine, Git in Intellij / PyCharm / WebStorm / PhpStorm / RubyMine, obiskano 10. 7. 2025.

[8] nvie.com/posts/a-successful-git-branching-model, A successful Git branching model, obiskano 11. 7. 2025.

[9] docs.github.com/en/get-started/using-github/github-flow, GitHub flow, obiskano 11. 7. 2025.

[10] about.gitlab.com/topics/version-control/what-is-gitlab-flow, What is GitLab Flow?, obiskano 11. 7. 2025.

[11] trunkbaseddevelopment.com, Trunk Based Development, obiskano 11. 7. 2025.

Implementacija prilagodljivega ERP sistema z malo ali nič kode

Tomaž Bizjak, Gregor Kovač, Tomaž Kozlar

Bilumina d.o.o., Maribor, Slovenija
tomaz.bizjak@bilumina.com, gregor.kovac@bilumina.com, tomaz.kozlar@bilumina.com

Ob ustanovitvi podjetja smo se srečali s problematiko lažjega in hitrejšega razvoja aplikacije, kot smo ga poznali v preteklosti. V preteklosti smo razvijali aplikacije tako, da smo poslovno logiko programirali na nivoju posamezne vnosne maske. S tem načinom smo hoteli prekiniti, saj smo vedeli, da nas čaka veliko dela, poleg tega smo hoteli lansirati izdelek v nekem doglednem času. Po raziskavi smo naleteli na low-code princip razvoja programske opreme. Odločili smo se za svojo low-code rešitev, ki temelji na programskem jeziku SQL z Javo v ozadju. V članku predstavljamo kakšna orodja smo razvili za podporo low-code principu razvoja programske opreme.

Ključne besede:
low-code,
prenova rešitve,
ERP,
SQL,
Java.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.5

1 Uvod

1.1. Zgodovina

Podjetje Bilumina je nastalo leta 2011. Na začetku smo bili štirje, od tega eden, ki je zelo dobro poznal vsebino (poslovne procese podjetij, ...) in trije programerji (Java, SQL). Že od prej smo najbolj poznali trgovinski segment trga (retail), zato smo se tudi v novem podjetju odločili, da hočemo v nekem doglednem času lansirati lastno ERP (Enterprise Resource Planning) rešitev za ta segment trga, skupaj s podporo za skladišča, zaloge, različne vrste dokumentov, blagajna (POS), … Najprej smo si okvirno izračunali kakšen obseg dela nas čaka, če želimo to zahtevo (v doglednem času lansirati ERP) tudi izpolniti.

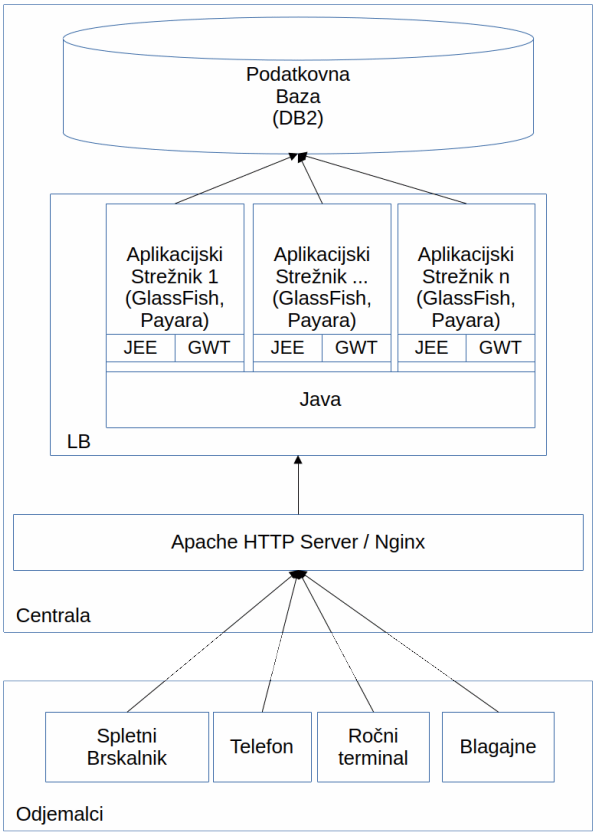
Prišli smo do naslednjih številk:

- nekaj sto vnosnih mask. Ta številka je trenutno narasla na preko tisoč.
- nekaj sto baznih objektov. Ta številka je trenutno narasla na nekaj tisoč baznih objektov (tabel, view-jev, trigger-jev, procedur)

Glede na te številke smo bili že na začetku prepričani, da s tradicionalnim razvojem ne bomo mogli zadostiti zahtevi po doglednem času lansiranja ERP-a. To je bil poglavitni razlog, da smo se odločili, da potrebujemo neko drugo rešitev, nek drug način razvoja. Po pregledu stanja na trgu, kaj sploh obstaja in kakšne te rešitve so (funkcionalnosti, strošek, ...), smo se na koncu odločili, da izdelamo lastno low-code rešitev.

1.2. Arhitektura ERP-a

Odločili smo se za naslednjo arhitekturo ERP rešitve:



Slika 1: Arhitektura ERP rešitve.

Vsi podatki so shranjeni v relacijski podatkovni bazi IBM Db2, do katere, preko aplikacijskih strežnikov, dostopajo klienti preko spletnega brskalnika (Google Chrome, ...), telefoni (Android), ročni terminali (Android) in blagajne (Windows, Java); vsak preko svoje namenske aplikacije. Po navadi odjemalci dostopajo do aplikacijskih strežnikov preko LB (load balancer) komponente, preko kater imajo eno vstopno točko, mi pa lahko za load balancer-jem dodajamo aplikacijske strežnike glede na potrebe. Če je potrebno, pred load balancer postavimo tudi spletni strežnik (Apache HTTP Server ali Nginx). To storimo v primeru, da je potrebno sistem odpreti navzven, recimo v Internet.

1.3. Osnove low-code rešitve

Glede na to, da smo vsi najbolje poznali dva programska jezika, Java in SQL, in da je jezik SQL dosti lažji za učenje kot programski jezik Java, smo se odločili, da za osnovo low-code rešitve uporabimo SQL, ki ga razširimo s funkcijami, ki jih bomo potrebovali.

Primer SQL poizvedbe na vnosni maski: SELECT IME, PRIIMEK FROM UPORABNIKI WHERE ID = :ID

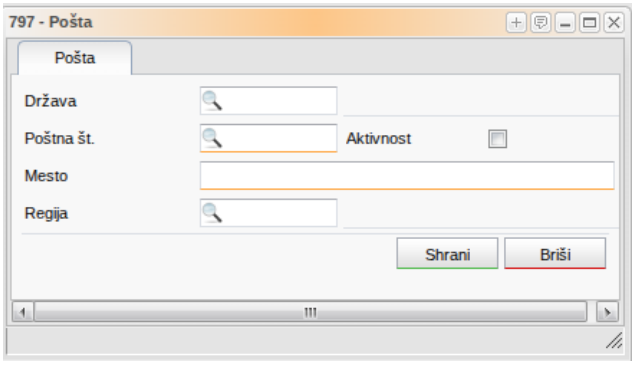
Vrednost parametra ID se uporabi iz maske, ki je to SQL poizvedbo izvedla. Kako in kaj bo predstavljeno v nadaljevanju članka.

2 Orodja

Da smo lahko začeli kreirati strukturo podatkovne baze in vnosnih mask smo morali razviti več orodij.

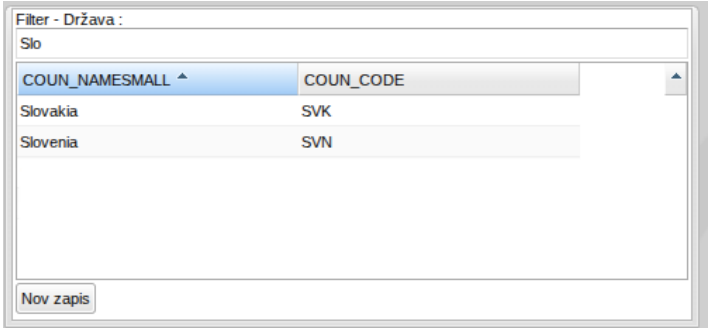
2.1. Design vnosnih mask (t. i. FormTool)

Poglejmo na primeru preprostega vnosa pošt kako kreiramo vnosne maske:



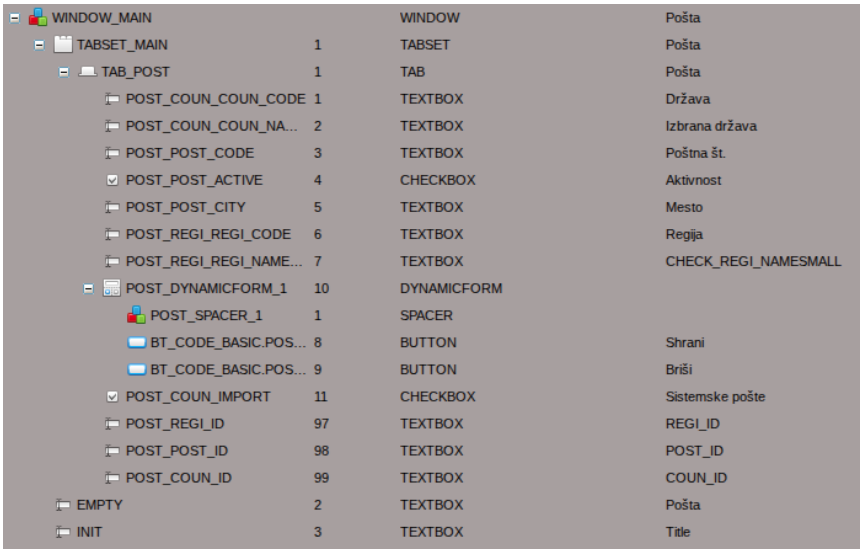
Slika 2: Vnosna maska za vnos pošte.

Prikazana maska ima nekaj vnosnih polj (Država, Poštna št., Mesto, Aktivnost, Regija). Polja z ikono lupe so iskalna polja:



Slika 3: Primer iskanja.

Orodje za urejanje vnosnih mask prikazuje vnosno masko v drevesni strukturi:



WINDOW_MAIN		WINDOW	Pošta
TABSET_MAIN	1	TABSET	Pošta
TAB_POST	1	TAB	Pošta
POST_COUN_COUN_CODE	1	TEXTBOX	Država
POST_COUN_COUN_NA...	2	TEXTBOX	Izbrana država
POST_POST_CODE	3	TEXTBOX	Poštna št.
POST_POST_ACTIVE	4	CHECKBOX	Aktivnost
POST_POST_CITY	5	TEXTBOX	Mesto
POST_REGI_REGI_CODE	6	TEXTBOX	Regija
POST_REGI_REGI_NAME...	7	TEXTBOX	CHECK_REGI_NAMESMALL
POST_DYNAMICFORM_1	10	DYNAMICFORM	
POST_SPACER_1	1	SPACER	
BT_CODE_BASIC.POS...	8	BUTTON	Shrani
BT_CODE_BASIC.POS...	9	BUTTON	Briši
POST_COUN_IMPORT	11	CHECKBOX	Sistemske pošte
POST_REGI_ID	97	TEXTBOX	REGI_ID
POST_POST_ID	98	TEXTBOX	POST_ID
POST_COUN_ID	99	TEXTBOX	COUN_ID
EMPTY	2	TEXTBOX	Pošta
INIT	3	TEXTBOX	Title

Slika 4: Drevesna struktura.

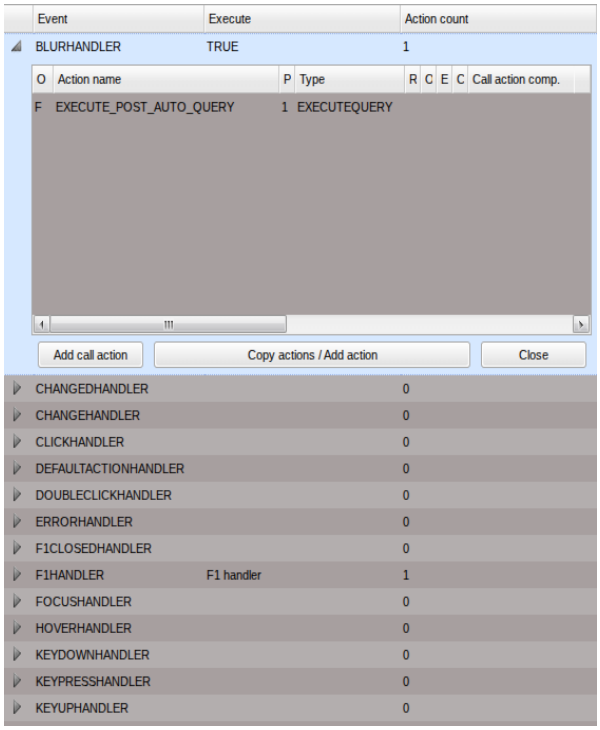
Na sliki 4 vidimo strukturo maske. Vsako polje ima svoje unikatno ime, svoj tip (TEXTBOX, CHECKBOX, BUTTON, …) in labelo (kaj uporabnik vidi ko odpre vnosno masko). Labelo se lahko prevede v primeren jezik uporabnika.
Vsako polje na vnosni maski ima mnogo lastnosti, ki jih lahko nastavimo, da dobimo ustrezen izgled in funkcionalnost vnosne maske.

O	Property name	Value	Type
C	ALIGN	LEFT	STYLE
C	ALLOWUSERVISIBLE	FALSE	STYLE
C	AUTOFIT	TRUE	STYLE
C	CELLSTYLE	DEFAULT	STYLE
C	COLSPAN	1	STYLE
C	HEIGHT	22	STYLE
C	HINTSTYLE	DEFAULT	STYLE
C	HOVERWIDTH	300	STYLE
F	POSITION	8	STYLE
C	PRINTADDWATERMARK	FALSE	STYLE
C	SHOWTITLE	FALSE	STYLE
C	TEXTALIGN	CENTER	STYLE
C	TITLEALIGN	CENTER	STYLE
C	TITLEORIENTATION	LEFT	STYLE
C	TITLESTYLE	DEFAULT	STYLE
C	TITLEVALIGN	TOP	STYLE
C	WIDTH	*	STYLE
C	WRAPTITLE	FALSE	STYLE
C	HINT		TEXT

Slika 5: Nekatere lastnosti polja.

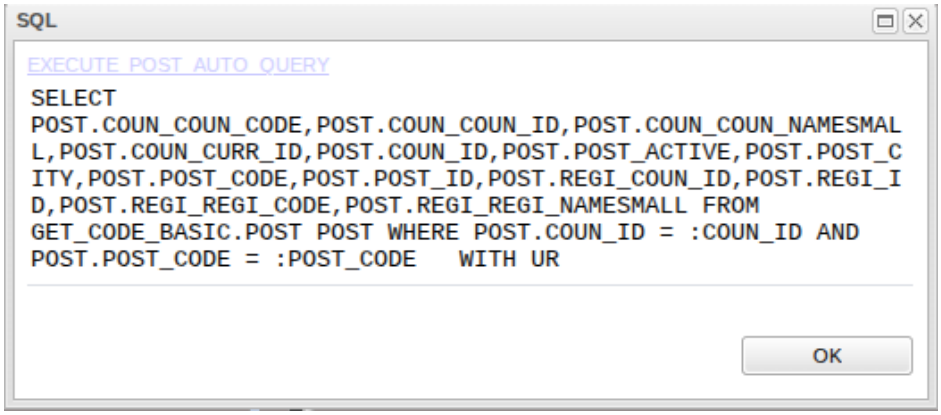
Polju lahko določimo različne skupine lastnosti:

- stilske: kje na maski se polje pojavi, kako je tekst znotraj polja poravnan, ...
- vedenjske: ali se vrednost polja prenese na aplikacijski strežnik, katero je naslednje polje na katerega se prenese vnos, ...
- vsebinske: katere akcije se zaženejo kdaj (slika 6)



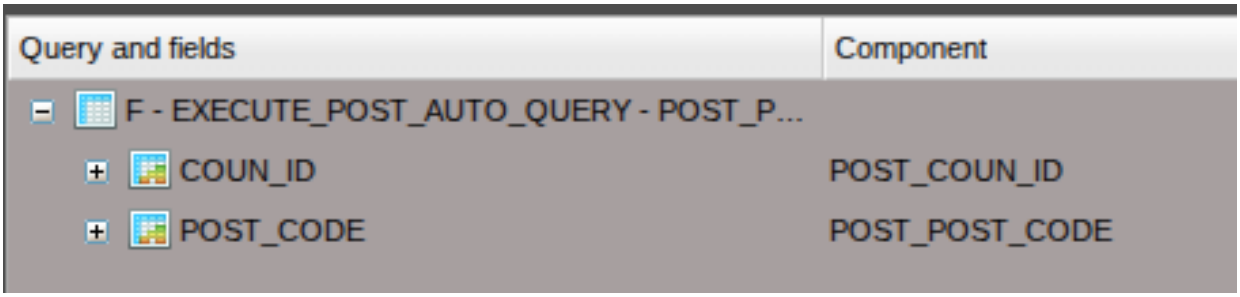
Slika 6: Različni dogodki (event-i) in povezane akcije.

Ob dogodku vnosa podatka v polje »Poštna št.« se sproži akcija EXECUTE_POST_AUTO_QUERY tipa EXECUTEQUERY. Tip EXECUTEQUERY pomeni, da se bo sprožil določen SQL (slika 7) na podatkovni bazi.



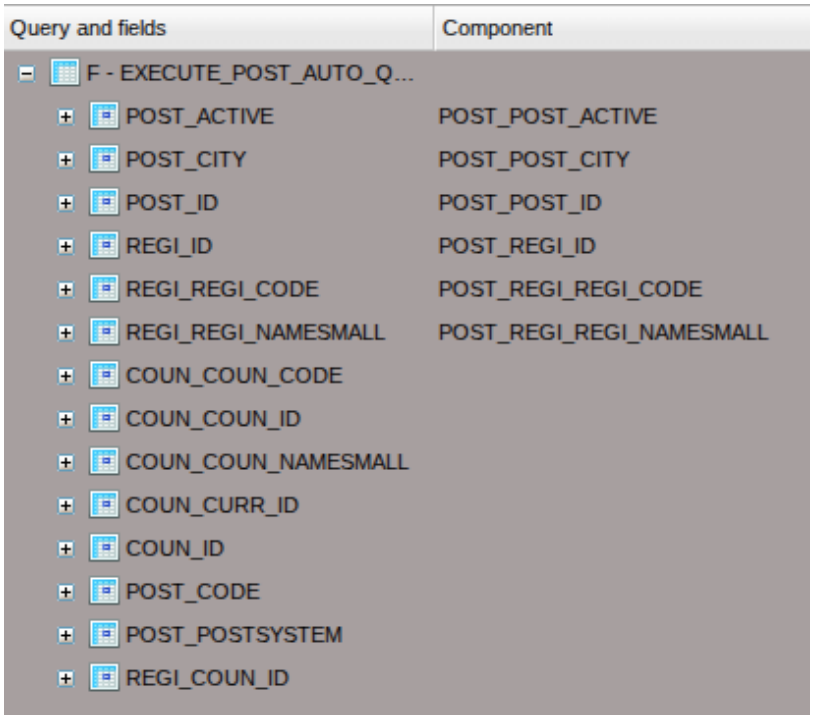
Slika 7: Primer SQL-a.

V SQL-u sta dva parametra (COUN_ID, POST_CODE). Prepoznamo ju, ker imata pred imenom znak :. Vrednost teh dveh parametrov moramo napolniti z ustrezno vrednostjo iz vnosne maske (slika 8).



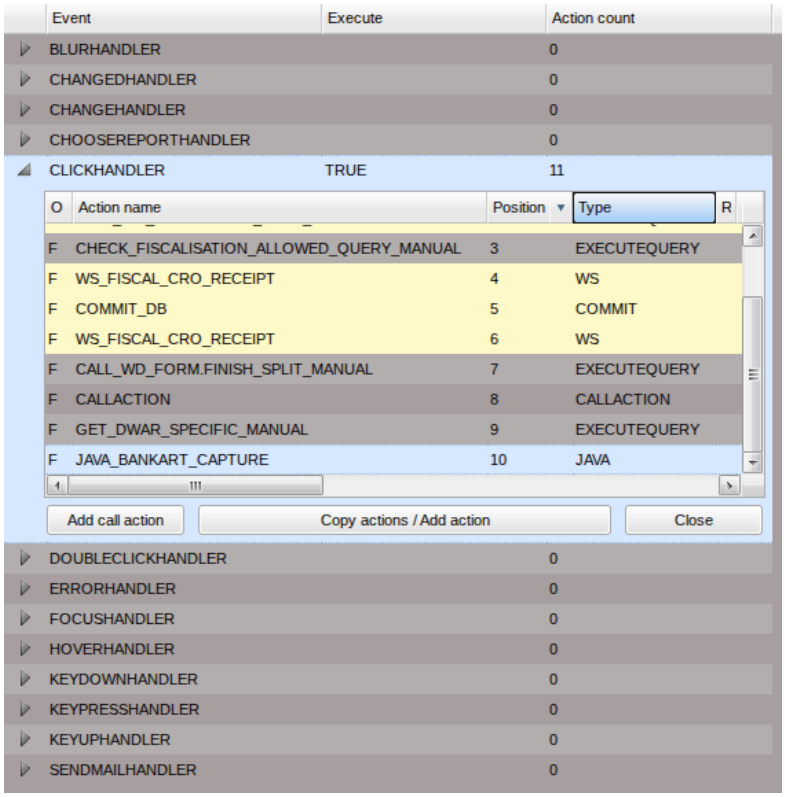
Slika 8: Povezava SQL parametrov s polji na vnosni maski.

Ko se SQL požene in pridobi rezultate iz podatkovne baze, jih mora prikazati na vnosni maski (slika 9).



Slika 9: Povezava polj iz SQL-a in polj na maski.

Poleg SQL-ov lahko akcije na poljih poženejo tudi vnaprej pripravljene akcije napisane v programskem jeziku Java (slika 10).



Slika 10: Primer gumba z večjim številom akcij.

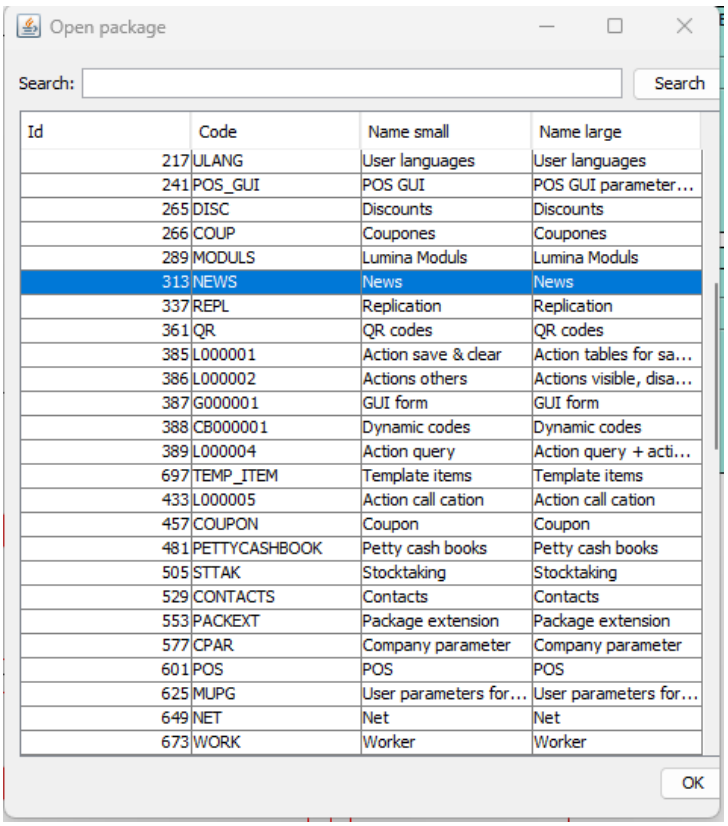
Na sliki 10 vidimo, da lahko ima posamezno polje več akcij, istega ali drugačnega tipa. Na sliki vidim več tipov akcij:

- EXECUTEQUERY: pridobi podatke iz podatkovne baze,
- WS: pridobi podatke iz spletne storitve,
- COMMIT: zaključi transakcijo,
- CALLACTION: pokliči akcijo na drugem polju,
- JAVA: pokliči vnaprej pripravljeno akcijo v programskem jeziku Java.

Pri vseh teh različnih tipih akcij povežemo parametre akcije s polji na maski in rezultat akcije s polji na maski na isti način kot je bilo prikazano pri EXECUTEQUERY akciji (slika 8).

2.2. Design baze (t. i. Linea)

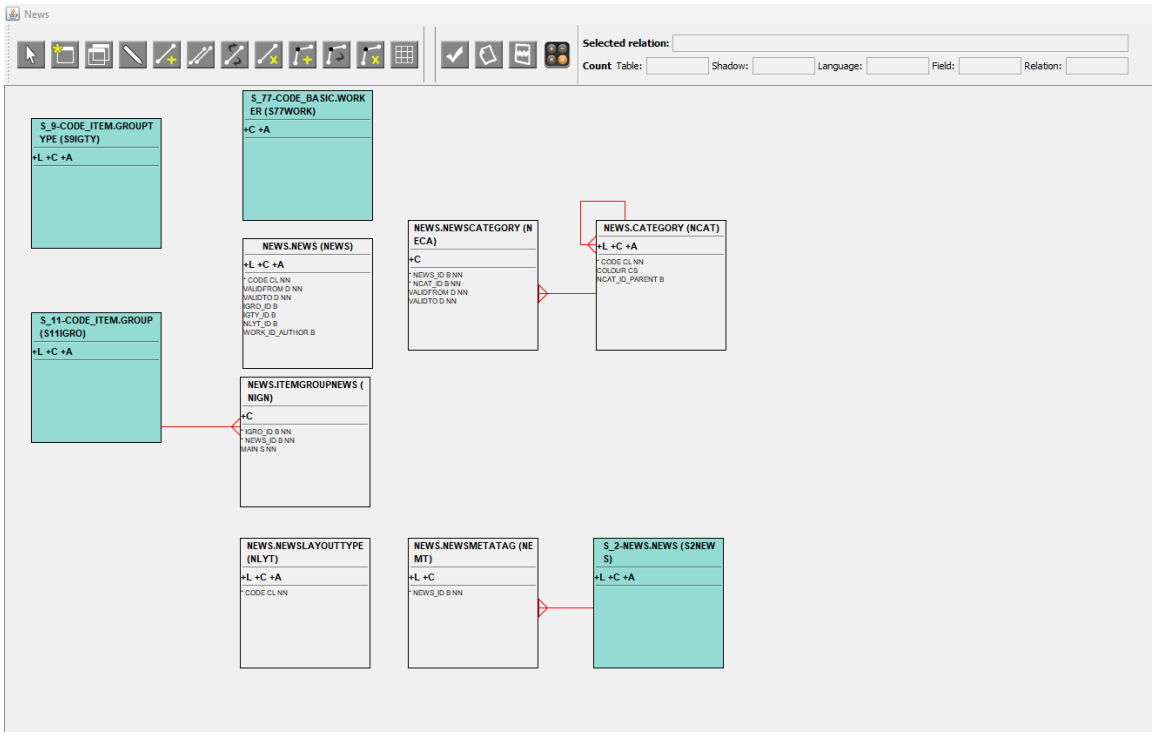
Za hitrejši razvoj strukture baze smo razvili lastno orodje. Tabele so, zaradi lažje vizualizacije, razporejene v posamezne pakete, kar prikazuje slika 11.



Id	Code	Name small	Name large
217	ULANG	User languages	User languages
241	POS_GUI	POS GUI	POS GUI parameter...
265	DISC	Discounts	Discounts
266	COUP	Coupones	Coupones
289	MODULS	Lumina Moduls	Lumina Moduls
313	NEWS	News	News
337	REPL	Replication	Replication
361	QR	QR codes	QR codes
385	L000001	Action save & clear	Action tables for sa...
386	L000002	Actions others	Actions visible, disa...
387	G000001	GUI form	GUI form
388	CB000001	Dynamic codes	Dynamic codes
389	L000004	Action query	Action query + acti...
697	TEMP_ITEM	Template items	Template items
433	L000005	Action call cation	Action call cation
457	COUPON	Coupon	Coupon
481	PETTYCASHBOOK	Petty cash books	Petty cash books
505	STTAK	Stocktaking	Stocktaking
529	CONTACTS	Contacts	Contacts
553	PACKEXT	Package extension	Package extension
577	CPAR	Company parameter	Company parameter
601	POS	POS	POS
625	MUPG	User parameters for...	User parameters for...
649	NET	Net	Net
673	WORK	Worker	Worker

Slika 11: Tabele paketa News.

Slika 12 prikazuje tabele znotraj paketa News.

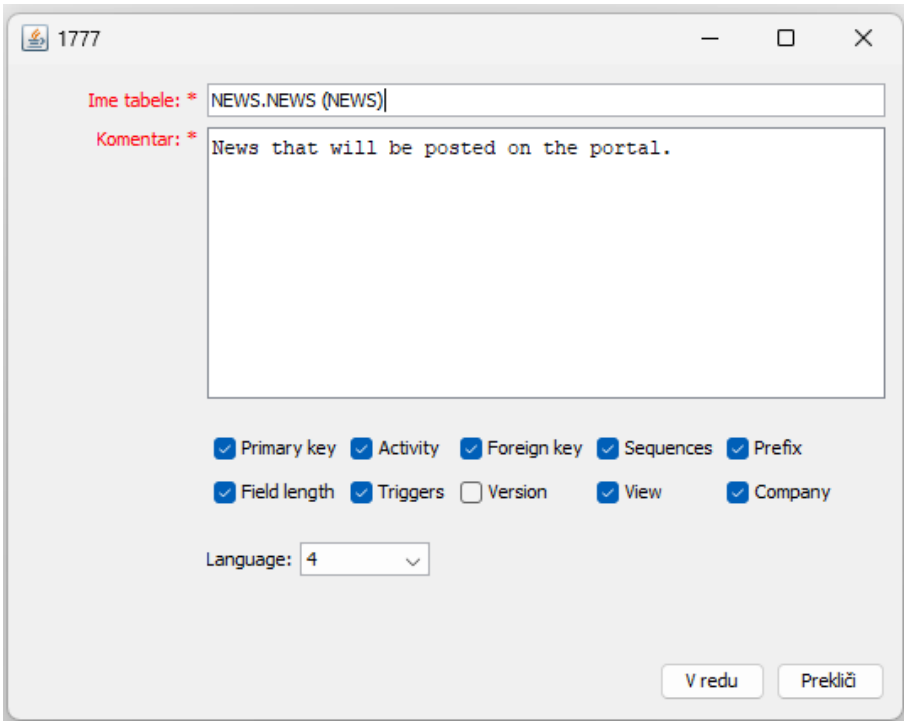


Slika 12: Tabele paketa News.

Tabele v temnejši barvi so iz drugega paketa. Orodje omogoča tudi povezovanje tabel:

- One – to – One,
- One – to – Many,
- Many – to – Many povezave rešujemo z uporabo vmesnih tabel.

Slika 13 prikazuje nastavitve posamezne tabele.

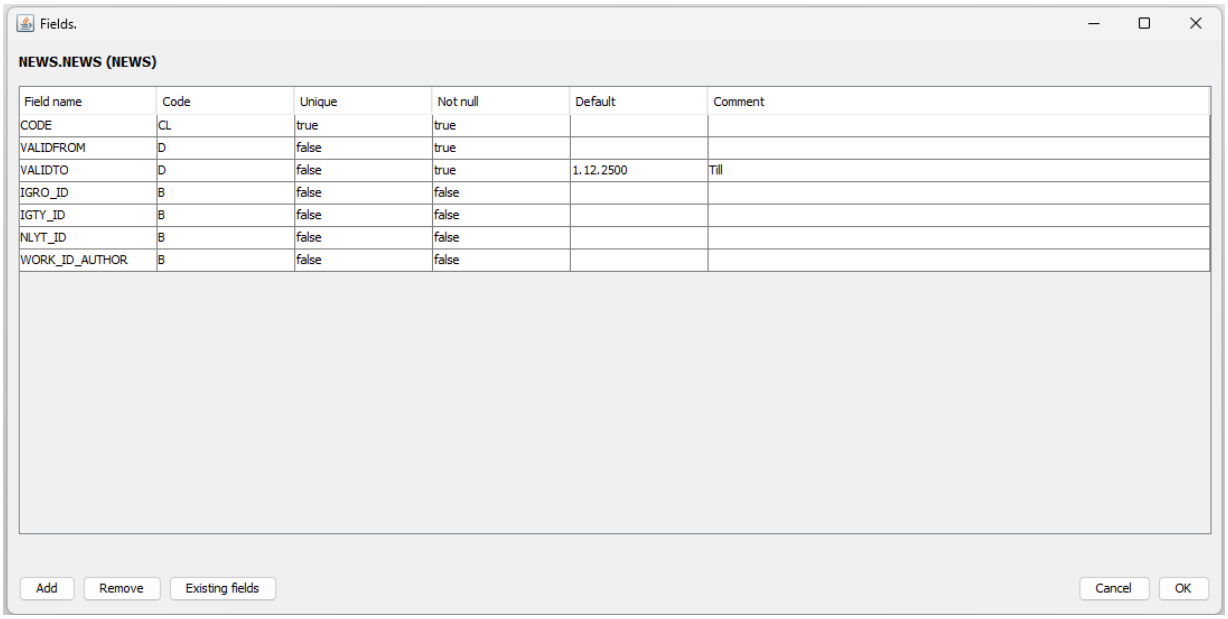


Slika 13: Nastavitve tabele.

Tabeli lahko nastavimo:

- ime tabele,
- opis tabele,
- ali ima tabela primarni ključ,
- ali se za njo kreirajo pogledi,
- ali se za njo kreira ustrezna sekvenca za vrednosti primarnega ključa,
- ali ima ta tabela posebno jezikovno tabelo,
- ali je posamezna vrstica vezana na posamezno podjetje ali je splošna za vsa podjetja.

Slika 14 prikazuje nastavitve kolon določene tabele.



Slika 14: Kolone tabele NEWS.NEWS.

Posamezna kolona ima:

- ime,
- opis,
- podatkovni tip,
- ali dovoljuje vrednost NULL,
- ali je del unikatnega indeksa,
- kakšna je privzeta vrednost.

To orodje pripravi ustrezne tabele, poglede in triggerje na podatkovni bazi, da lahko vnosna maska pravilno zapiše podatke v podatkovno bazo ali pa jih iz nje prebere. Podatek je lahko vezan na posamezno podjetje, lahko pa ima tudi jezikovni del, kot je recimo opis artikla v slovenščini, angleščini, …

3 Mnenja uporabnikov

3.1. Tomaž Bizjak

S prihodom v podjetje sem tudi sam začel ustvarjati enostavne vnosne maske. Že na začetku me je navdušilo dejstvo, da ima orodje že vnaprej implementiranih veliko splošnih gradnikov, ki jih je treba le povezati z ustreznimi tabelami in stolpci v podatkovni bazi. Ta pristop močno pospeši razvoj, saj zmanjša količino ročnega programiranja in s tem tudi možnost napak pri implementaciji. Zaradi tega je tudi uvajanje novih sodelavcev hitrejše in manj zahtevno, saj se lahko že v kratkem času vključijo v razvojne naloge.

Druga pomembna prednost orodja je enostavno prevajanje vnosnih mask, kar je še posebej priročno za stranke, ki poslujejo na več različnih trgih. S tem se bistveno poenostavi prilagajanje aplikacij lokalnim uporabnikom, kar izboljša uporabniško izkušnjo in hkrati zmanjša potrebo po dodatni tehnični podpori. Možnost hitrega preklapljanja med jeziki je pogosto tudi ena izmed prvih funkcionalnosti, ki jih opazijo naše mednarodne stranke, kar pogosto vpliva na njihovo pozitivno začetno izkušnjo.

Na splošno lahko rečem, da je orodje preprosto za uporabo, a hkrati dovolj zmogljivo, da omogoča hitro razširjanje funkcionalnosti glede na potrebe posameznega projekta. Zaradi svoje prilagodljivosti in preglednosti pa IT-kadrom omogoča, da več časa posvetijo zahtevnejšim tehničnim izzivom znotraj podjetja, namesto da bi se ukvarjali z rutinskimi opravili.

3.2. Tomaž Kozlar

Z orodjem FormTool sem začel delati, ko je omogočalo izgradnjo enostavnih šifrantov. Orodje se je z leti postopoma razvijalo in še vedno se. Sedaj že obsega možnost gradnje vedno kompleksnejših vnosnih mask, klicanja zunanjih akcij, še vedno pa se dodajajo novi gradniki, ki jih uporabljamo glede na potrebe stranke in nas. Z orodjem FormTool se lahko hitro naredi nova maska ali pa se doda novo funkcionalnost na obstoječo masko.

Prednost takega načina dela vidim v tem, da je manj možnosti za napake pri razvoju, saj gre za omejen nabor gradnikov, a hkrati omogoča veliko fleksibilnost, ko se lahko različne akcije, ki jih prožijo gumbi/polja napišejo tudi v programskem jeziku Java. Ker je orodje FormTool spletno orodje tudi ni potrebna inštalacija, prav tako pa lahko na različnih formah dela več ljudi hkrati.

Orodje Linea sem pomagal razviti tudi sam. Orodje omogoča grafični prikaz tabel v podatkovni bazi, kolon v posamezni tabeli in relacij med njimi (ER diagram). Omogoča tudi enostaven pregled in upravljanje velikih količin tabel, saj se lahko združujejo tabele v pakete. V Linea-i se nato lahko upravlja z vsakim paketom posebej, kar omogoča večjo preglednost. Prav tako je z njim enostavno pokazati in razložiti relacije ter tabele drugim zaposlenim, ki delajo na istem projektu, saj je lažje stvari predstaviti na grafičnem pogledu. Ker omogoča tudi tiskanje diagrama ter opisov tabel in posameznih polj v tabeli, je tako možno narisano tudi poslati stranki in to uporabiti kot dokumentacijo.

3.3. Dolores Sonički

FormTool je naš interni pripomoček za gradnjo mask v spletnem ERP sistemu. Maske predstavljajo uporabniške obrazce za vnos, pregled in obdelavo podatkov, ki jih zaposleni uporabljamo pri vsakodnevnih procesih. Orodje omogoča, da obrazce sestavljamo samostojno – brez zahtevnega programiranja, če imamo osnovno razumevanje strukture podatkov in delovanja ERP okolja.

Čeprav nisem iz tehničnega okolja, sem se v uporabo FormTool-a hitro vživela. Osnovno razumevanje delovanja ERP sistema mi je pri tem zelo pomagalo. Vmesnik je pregleden, gradniki pa so zasnovani tako, da se jih z nekaj uvoda da hitro osvojiti. Uporaba je logična in strukturirana, kar omogoča samostojno delo tudi tistim, ki niso programerji, a imajo nekaj vsebinskega predznanja.

Vsak gradnik ima svoje lastnosti, ki jih prilagajamo prek enostavnih nastavitev. Mnoga polja že vključujejo prednastavljene akcije, kar pohitri delo in omogoča večjo osredotočenost na funkcionalnost obrazca.

FormTool podpira kompleksne strukture – drevesne prikaze, tabele znotraj tabel, dinamične pogoje, privzete vrednosti in še mnogo več. S tem močno povečamo fleksibilnost, skrajšamo čas prilagajanja procesov ter omogočimo, da pri razvoju ERP okolja sodeluje širši krog uporabnikov z različnih področij.

4 Zaključek

Low-code pristop se je izkazal kot zanesljiva in učinkovita alternativa klasičnemu razvoju ERP sistemov. Omogoča hitro prilagajanje spremembam, večjo vključenost uporabnikov in boljšo izrabo IT virov. Takšen pristop bo v prihodnosti igral čedalje pomembnejšo vlogo pri digitalizaciji poslovanja.

Zip & Go: Samodejno nameščanje z zabojniki v produkcijo

Boštjan Praznik, Nejc Maleš

Novum-RGI Germany GmbH Podružnica Maribor, Maribor, Slovenija
bostjan.praznik@rgigroup.com, nejc.males@rgigroup.com

ZIP & GO predstavlja svež in praktičen pristop k dinamičnemu posodabljanju Node.js mikrostoritev v produkcijskem Docker okolju – brez potrebe po ustavitvi storitev ali ponovni gradnji zabojnikov. Namesto klasičnih, pogosto zapletenih CI/CD postopkov, omogoča ta rešitev preprost vnos nove funkcionalnosti z ZIP paketom, ki se samodejno razpakira znotraj zabojnika in sproži vnovični zagon aplikacije z novo vsebino. Z uporabo preizkušenih tehnologij, kot so Node.js, PM2 in Docker, rešitev ponuja visoko modularnost, zanesljivost in izjemno hitro implementacijo sprememb brez izpada storitev. Sistem je zasnovan za ekipe, ki potrebujejo učinkovito, robustno in dolgoročno vzdržno strategijo za upravljanje mikrostoritev. Članek skozi konkretne primere, konfiguracije in skripte pokaže, kako lahko z nekaj vrsticami kode in pravim arhitekturnim pristopom bistveno poenostavimo produkcijsko upravljanje aplikacij.

Ključne besede:
Docker
Node.js
PM2
Mikrostoritve
CI/CD

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.6

1 Uvod

V sodobnih informacijskih sistemih postaja potreba po hitrem, zanesljivem in nemotenem posodabljanju programske opreme vse bolj izrazita. Razvoj mikrostoritvene arhitekture, podprt z orodji za avtomatizacijo in upravljanje z zabojniki, kot so Docker, Kubernetes [4] in PM2, je močno spremenil pristop k razvoju in vzdrževanju aplikacij. Kljub tem napredkom ostaja izziv, kako v produkcijskem okolju zagotoviti dinamično in čim bolj nemoteno nadgradnjo vsebine mikrostoritev, ne da bi bilo potrebno ponovno ustvarjanje (angl. rebuild) zabojnikov.

Predstavljena rešitev z imenom "ZIP & Go"[1] omogoča prav to: hitro in varno posodabljanje z uporabo ZIP datotek, ki vsebujejo nove konfiguracije ali funkcionalne module. Ključna prednost tega pristopa je, da zabojnik teče neprekinjeno, posodobitve pa se izvajajo zgolj na nivoju vsebine preko samodejnega razpakiranja ZIP paketa in ponovnega zagona aplikacije z novo vsebino.

2 Pristop k rešitvi

Glavni cilj ZIP & Go pristopa je ločitev zabojniške infrastrukture od poslovne logike. S tem omogočimo, da se funkcionalni del sistema (npr. kalkulacijski moduli) posodablja neodvisno od zabojnika samega. Ta cilj dosežemo z naslednjimi koraki:

- ZIP datoteka vsebuje .mjs module (ESM format), ki predstavljajo poslovno logiko,
- ZIP se postavi v zunanji volumen, dostopen zabojniku,
- Skripta (watcher) nadzira ZIP datoteko in ob spremembi sproži razpakiranje,
- Aplikacija se ponovno zažene, tokrat z novo poslovno logiko.

Tako zagotovimo modularnost, hitrost in zanesljivost. Aplikacija se ne prekine dlje kot nekaj milisekund, vse spremembe pa ostanejo lokalizirane na nivoju poslovne logike.

3 Uporabljena orodja

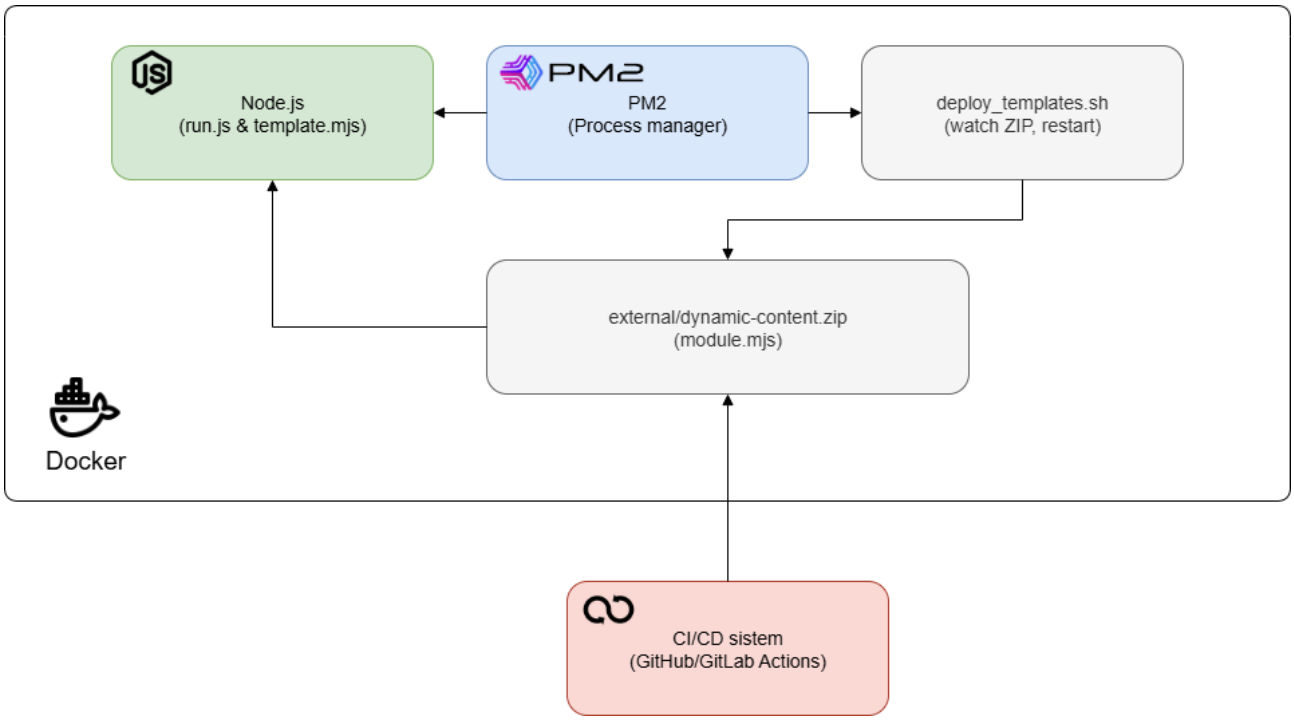
Orodja, uporabljena v opisani rešitvi:

- **Node.js:** Osnova mikrostoritve. Z uporabo ES modulov (.mjs) omogoča dinamično nalaganje logike. Poslovna pravila so definirana kot moduli, ki jih aplikacija ob vsakem zagonu uvozi.
- **PM2:** Upravitelj Node.js procesov. Omogoča nadzor, ponovni zagon in spremljanje delovanja mikrostoritve. Njegova podpora za konfiguracijo (.config.cjs) olajša upravljanje več procesov.
- **Docker:** Sistem za upravljanje z zabojniki poskrbi za izolirano okolje, v katerem teče aplikacija. ZIP datoteka je dostopna preko priklopljenega volumna, kar pomeni, da se vsebina zabojnika fizično ne spremeni.
- **Bash:** Preprosta, a učinkovita skriptna rešitev za razpakiranje vsebine ZIP in nadzor izvajanja PM2 procesov.

[1] Zip & Go je ime rešitve, kreirano in uporabljeno izključno za namene konference OTS.

4 Arhitektura

Arhitektura sistema je zasnovana modularno, z jasnim ločevanjem med infrastrukturo (Docker zabojnik, PM2, skripte) in poslovno logiko (ES moduli v ZIP datoteki). Namen je omogočiti vročo zamenjavo poslovne logike brez potrebe po rekonstrukciji zabojnika.



Slika 1: Arhitektura opisane rešitve.

4.1. CI/CD sistem

Zunanja komponenta, npr. GitHub Actions ali GitLab CI, samodejno generira ZIP datoteko, ki vsebuje .mjs module (poslovno logiko). Ko razvijalec potrdi spremembo v repozitoriju:

- se izvede testiranje,
- ustvari ZIP paket,
- prenese ZIP v zunanji volumen, dostopen zabojniku.

4.2. Docker zabojnik

Zabojnik vsebuje vnaprej pripravljeno osnovno okolje:

- mapo /opt/myapp/engine/ z glavno skripto run.js in nadzorno skripto deploy_templates.sh,
- mapo /opt/myapp/external/, kamor se namešča ZIP datoteka z novejšo poslovno logiko,
- referenčno datoteko template.mjs, ki se uporabi ob zagonu mikrostoritve.

Zabojnik je operativen ves čas – ne rekonstruira se, kar bistveno zmanjša čas uvajanja sprememb.

Drevesni prikaz strukture zabojnika:

```
/opt/myapp/
├── engine/
│   ├── run.js
│   └── deploy_templates.sh
├── external/
│   └── dynamic-content.zip
└── template.mjs
```

4.3. PM2 procesni nadzornik

V zabojniku se izvajata dva procesa:

- watcher: nadzira ZIP datoteko in ob spremembi sproži skripto za razpakiranje,
- myapp: Node.js paket, ki izvaja poslovno logiko iz .mjs modula.

PM2 omogoča avtomatiziran ponovni zagon aplikacije po razpakiranju nove logike in olajša spremljanje izpadov.

Primer konfiguracije:

```
module.exports = {
  apps: [
    {
      name: "watcher",
      script: "/opt/myapp/engine/deploy_templates.sh",
      watch: ["/opt/myapp/external/dynamic-content.zip"],
      watch_options: { followSymlinks: false, usePolling: true },
      autorestart: false,
      exec_mode: "fork"
    },
    {
      name: "myapp",
      script: "/opt/myapp/engine/run.js",
      watch: false,
      args: "8080 file:///opt/myapp/template.mjs",
    }
  ]
};
```

4.4. Bash skripta (deploy_templates.sh)

Skripta, ki jo sproži watcher, izvede naslednje:

- razpakira ZIP v ustrezno mapo (ponavadi /opt/myapp/),
- preveri vsebino (po potrebi z zgoščeno vrednostjo ali podpisom),
- ponovno zažene glavno aplikacijo z novo logiko.

Gre za preprosto rešitev, ki je hitro prenosljiva in enostavna za vzdrževanje.

Primer skripte:

```
#!/bin/bash

echo "Executing deploy_templates.sh..."
echo "ENV: $ENV"

ZIP_FILE="${BASE_DIR}/external/dynamic-content.zip"
EXTRACT_DIR="${BASE_DIR}"

# Extract ZIP, overwrite existing files
unzip -o "$ZIP_FILE" -d "$EXTRACT_DIR"

# Restart the app with PM2
pm2 restart myapp || pm2 start server.js --name myapp
```

4.5. Node.js mikrostoritev

Glavna aplikacija (run.js) prejme pot do .mjs modula kot parameter:

- node run.js 8080 file:///opt/myapp/template.mjs

V aplikaciji se .mjs modul dinamično uvozi, kar omogoča prilagoditev logike med izvajanjem brez potrebe po spremembi osnovnega sistema.

4.6. Zunanji volumen

ZIP datoteka se ne prenaša neposredno v zabojnik, temveč v zunanji volumen (npr. Kubernetes PersistentVolume), ki ga zabojnik le bere. To omogoča:

- delitev vsebine med več zabojnikih,
- zmanjšanje varnostnega obsega sprememb,
- ohranitev ločenosti med infrastrukturnim in logičnim delom sistema.

5 Težave in omejitve

Implementacija prinaša nekatere tehnične in varnostne izzive:

- **Polling obremenitev:** Neprestano preverjanje sprememb lahko obremeni sistem, a gre za najzanesljivejši način v zabojniških okoljih, kjer inotify pogosto ni na voljo.
- **ZIP struktura:** Zahteva strogo določeno strukturo (npr. imeniki, imena datotek). Napačno strukturiran ZIP lahko povzroči zlom aplikacije.
- **Dostopna dovoljenja:** Zabojnik potrebuje pravice za zapisovanje v ciljna mesta in izvajanje skript.
- **Napake ob zagonu:** Napačen .mjs modul lahko povzroči, da se aplikacija ne zažene. Potrebna je validacija pred zagonom (npr. z uporabo testnega okolja ali digitalnih podpisov).
- **Varnostni vektorji:** ZIP je lahko vektor za zlonamerno kodo. Priporočamo preverjanje vsebine pred razpakiranjem (npr. s SHA256 ali podpisanimi ZIP paketi).

6 Integracija v CI/CD

ZIP paket lahko postane artefakt v CI/CD verigi. Ob vsaki potrjeni spremembi repozitorija se izvede:

- testiranje funkcionalnosti modulov,
- generiranje ZIP paketa,
- prenos ZIP v zunanji volumen (npr. preko SFTP, SCP ali neposredno v omrežno mapo).

Primer CI/CD skripte:

```
zip_and_go:
  stage: deploy
  script:
    - npm run test
    - zip -r dynamic-content.zip ./modules
    - scp dynamic-content.zip user@host:/external/
```

7 Primer uporabe

V našem podjetju razvijamo programsko opremo za področje zavarovalništva. Zavarovalnice, s katerimi sodelujemo, imajo različne zahteve, ponujajo svoje zavarovalniške produkte in delujejo v različnih državah in geografskih okoljih. Produkte in pakete, ki jih ponujajo zavarovalnice, običajno pripravljajo zavarovalniški aktuarji, ti pa imajo omejeno tehnično znanje in dostope do produkcijskih sistemov, na katerih je nameščena programska oprema. V preteklosti smo morali vsako zahtevano spremembo podpreti v razvojnem oddelku. Da bi se temu izognili, smo kot podporo glavnemu delu programskega paketa razvili mikrostoritev za izračun zavarovalniških parametrov ter ga zapakirali v okolje, ki stranki omogoča, da ga pod posebnimi varnostnimi pogoji, lahko nadgradi sama. Tako ob spremembi tehnični ekipi ni več potrebno sestavljali nov Docker image, temveč uporabnik sam spremeni le vsebino in po potrebi tudi poslovno logiko, ogrodje in podporni deli pa ostanejo statični.

Prednosti takšnega pristopa so:

- hitra namestitev (v manj kot 1s),
- noben docker image se ne spreminja,
- razvijalci se lahko osredotočajo le na logiko,
- varen, ker se uporablja preverjanje vsebine.

8 Zaključek

ZIP & Go predstavlja sodoben in pragmatičen pristop za obvladovanje posodobitev mikrostoritev v produkcijskem okolju. Med njegove prednosti sodijo:

- rekonstrukcija zabojnika ni potrebna,
- hitro nalaganje novih vsebin,
- manjša kompleksnost pri razvoju,
- enostavna integracija v obstoječe DevOps tokove,
- prilagodljivost in razširljivost.

Čeprav obstajajo varnostna in tehnična tveganja, jih je z dobrimi praksami (npr. preverjanje vsebine ZIP, nadzor dostopov) mogoče učinkovito omiliti. Ta pristop je še posebej primeren za podjetja, ki si želijo večjo agilnost in boljšo odzivnost na spremembe poslovnih pravil brez žrtvovanja stabilnosti infrastrukture.

Literatura

[1] https://pm2.keymetrics.io/: Advanced, production process manager for Node.JS

[2] https://nodejs.org/en: Free, open-source, cross-platform JavaScript runtime environment

[3] https://www.docker.com/: Foundation for secure, intelligent development

[4] https://kubernetes.io/: Kubernetes, Production-Grade Container Orchestration

[5] https://www.flaticon.com/free-icons: slike, uporabljene v glavni sliki arhitekture (Docker created by orvipixel, Node.js created by Grand Iconic, CI/CD created by Ida Desi Mariana)

Govorno omrežje Policije/MNZ – Projekti na ključ ali lastno znanje?

Marko Koblar

Ministrstvo za notranje zadeve, Generalna policijska uprava, Ljubljana, Slovenija
marko.koblar@policija.si

Omrežje MNZ/Policije sodi med večja »zasebna« omrežja v državi. Od začetkov strojne obdelave podatkov leta 1957, se je v letih ki so sledila, vzpostavilo omrežje, ki ga danes upravlja Urad za informatiko in telekomunikacije na Generalni policijski upravi. Omrežje se glede na razpoložljive vire vsakodnevno prilagaja najrazličnejšim potrebam različnih služb policije in aktualnim tehnologijam. Pri ključnih odločitvah, je izredno pomembna tudi dolgoročna finančna vzdržnost, ki v nobenem pogledu ne sme vplivati na nivo varnosti, zanesljivosti in robustnosti celotnega sistema – saj mora omrežje delovati nemoteno tudi v primeru različnih» izrednih« razmer. Brez dvoma smo lahko na stališču, da ima policija v tem trenutku (na prenovljenem delu omrežja) enega najsodobnejših sistemov v državi, saj sledi aktualnim trendom tako v tehnološkem kot uporabniškem smislu – pa naj bo to v smislu povezljivosti z javnim telekomunikacijskim omrežjem kot načina »vpetosti« v druga omrežja.

Ključne besede:
ITSP - Informacijsko telekomunikacijski sistem policije,
lasten razvoj,
sistemska integracija,
IKT,
varnostno-kritični sistemi.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.7

1 Uvod

Dobro poznavanja procesov, potreb različnih operativnih služb policije ter tehničnega okolja (ITSP – informacijsko telekomunikacijski sistem policije), omogoča zaposlenim na Uradu za informatiko in telekomunikacije razumevanje realnih potreb končnih uporabnikov, kar se na podlagi končnemu uporabniku največkrat skritih tehničnih rešitev - na koncu pokaže v obliki najrazličnejših rezultatov (npr. boljši uporabniški izkušnji, uporabnik lahko lažje, bolje ali hitreje opravi svoje naloge, dosežena je višja robustnost sistema, povečana varnost, nezanemarljivi so tudi prihranki na nivoju zaupanih proračunskih sredstev, …).

Brez dvoma vse navedeno omogoča tudi uspešne sistemske integracije na različnih nivojih oziroma lasten razvoj kompleksnejših IKT (Informacijsko-komunikacijska tehnologija) rešitev. Pri pripravi rešitev so vodilo različni dejavniki – realne potrebe končnih uporabnikov, razmere na trgu telekomunikacijski storitev, optimizacija razpoložljivih finančnih sredstev, zagotavljanje varnega in zanesljivega delovanja, možnost hitrega prilagajanja spremembam, …

V besedilu je predstavljen kratek pregled zgodovine razvoja govornega omrežja MNZ/policije, s fokusom na izvedenih aktivnostih v zadnjih letih, a hkrati tudi s pogledom v prihodnost.

2 Nove tehnologije, nove možnosti

Pri načrtovanju vseh predvidenih posegov so bile vodilo zahteve omrežja in želene oziroma potrebne funkcionalnosti končnih uporabnikov. Glede na nove tehnološke priložnosti, so bile izvedene različne aktivnosti v smislu spremenjene arhitekture, spremenjenem konceptu registrofonskega / snemalnega sistema, kjer je potrebno in mogoče ter smiselno uporabljeni virtualizaciji, lastnem razvoju klicnih centrov za sprejem klicev na interventno številko policije 113 oziroma enotno evropsko številko za prijavo pogrešanih otrok 116000. Pri vpeljavi sprememb je bilo v veliko pomoč dobro poznavanje in obvladovanje možnosti, ki jih ponujajo posamezni gradniki - pa tudi dejstvo, da danes večina sodobnih IKT rešitev, temelji na uporabi odprtih protokolov, programskih vmesnikov API, IP protokola ter »standardne« strojne opreme, katere funkcionalnost je odvisna od programske opreme (op. za razliko od preteklih desetletij, ko so bile tovrstne rešitve zasnovane pogosto na dražji in namenski strojni opremi). Glede na velikost ITSP, so bili na segmentih kjer je to smiselno in potrebne predvideni tudi mehanizmi, ki ji poznamo iz javnih telekomunikacijskih omrežij.

3 Izhodišča

Pri načrtovanju vseh predvidenih posegov so bile vodilo zahteve omrežja in želene oziroma potrebne funkcionalnosti končnih uporabnikov. Glede na nove tehnološke priložnosti, so bile opravljene različne aktivnosti v smislu spremenjene arhitekture, upoštevaje kar se da racionalno porabo proračunski sredstev, v luči zagotavljanja vseh zahtev povezanih z zanesljivim in varnim delovanjem omrežja. Odločitev glede morebitnega lastnega razvoja (npr. posamezna področja, obseg, dolgoročna vzdržnost posamezne rešitve), je bila neposredno povezana tudi z upoštevanjem razpoložljivih človeških virov (obseg predvidenih aktivnost, potrebna znanja in kompetence).

4 Izvedba

Po pripravljenih načrtih glede posameznih rešitev, je sledil nakup ustrezne strojne opreme. V veliko pomoč so bili pri razvoju lastnih rešitev tudi različni »projekti« s področja prosto dostopnih odprtokodnih rešitev oziroma orodij za potreben (do)razvoj. Na tem mestu izpostavljamo le eno od njih – Asterisk (https://www.asterisk.org/). Gre za odprtokodno ogrodje, ki je namenjeno gradnji komunikacijskih aplikacij. S pomočjo Asteriska lahko običajni

računalnik spremenimo v zmogljiv govorni strežnik, ki mu lahko dodamo tudi druge storitve (npr. funkcionalnosti klicnega centra). Ne glede na vsa dejstva ter morebitne pomisleke glede tovrstnega pristopa, so se tovrstne rešitve pokazale kot učinkovite – tako glede ustreznosti v izvedenem pilotskem projektu, kot na podlagi analize opravljenih testiranj. Nenazadnje tudi v smislu finančnega in potrebnega časovnega vložka, možnostih (do)razvoja potrebnih funkcionalnosti, fleksibilnosti izbora terminalne opreme (saj vsaj načelno ni več potrebne nikakršne »vezave« na dobavitelja telekomunikacijskega sistema).

Med kompleksnejše faze sodita brez dvoma lastna rešitev za klicne centre, v okviru katerih na nivoju »regije« policija sprejema klice na številko 113 ter številko 116000 oziroma projekt prenove registrofonskega sistema. V primeru prenove registrofonskega sistema lasten razvoj ni bil smiseln (glede na predviden finančni vložek za že preverjen izdelek oziroma glede na potrebne vložke v druge vire). Pripravljena arhitektura oziroma rešitev v sodelovanju z dobaviteljem sistema, je omogočila potrebne optimizacije v smislu vseh potrebnih virov.

5 Kako ?

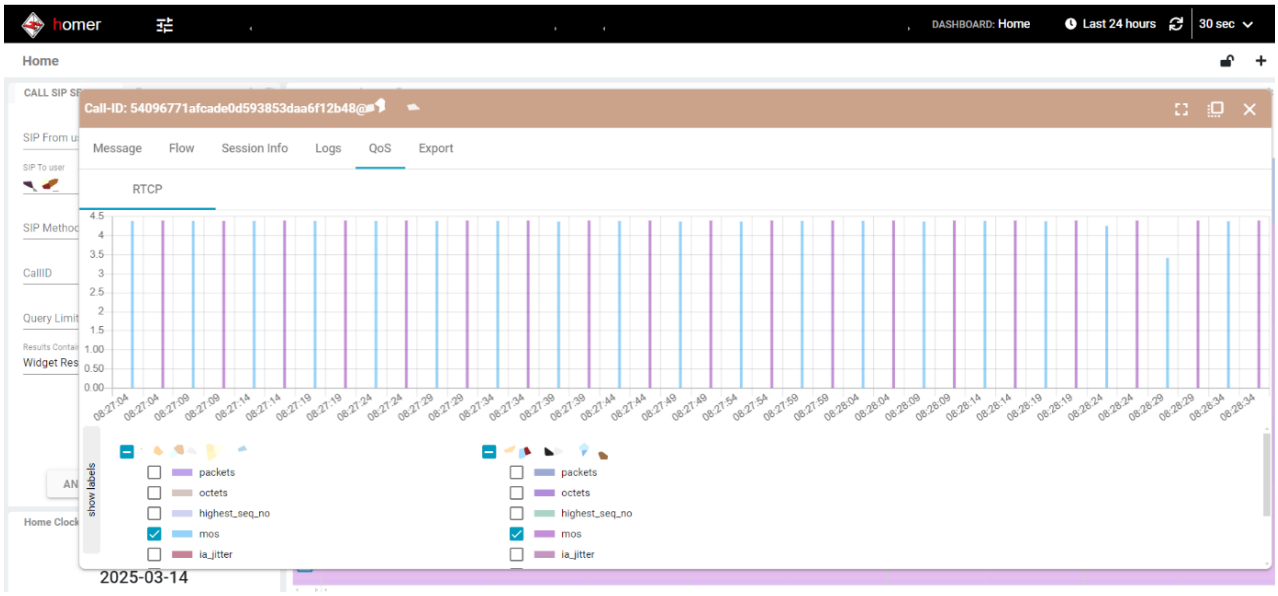
Brez dvoma se sama po sebi zastavljajo določena vprašanja, kot na primer – Kako je to mogoče ob vsesplošnem pomanjkanju ustreznih kadrov ? Na kakšen način so bili izvedeni posegi ? Kdo je zagotovil potrebno podporo projektu oziroma posameznim fazam ? Ali so bile tehnološke spremembe upravičene v finančnem smislu (investicija, mesečni stroški) ? Ali tehnološke spremembe ne pomenijo višje stopnje tveganja v smislu robustnosti oziroma zanesljivosti delovanja? Kaj vsi posegi pomenijo v smislu dolgoročnega vzdrževanja sistema ? Kakšne so koristi za končnega uporabnika ? Nenazadnje – kakšno tveganje pomeni uporaba odprtokodne rešitve v smislu dolgoročne podpore ….

Spisek morebitnih vprašanj še zdaleč ni popoln. V nekaj stavkih je tudi nemogoče odgovoriti na vsa vprašanja. Na marsikatero vprašanje pa je mogoče dogovoriti »na način«, da so bile ideje o drugačnem konceptu reševanja problematike predstavljene s strani oseb, ki vsakodnevno skrbijo za nemoteno delovanje omrežja (in ga zato tudi najbolje poznajo) - tako, da je bilo pravzaprav ključno vprašanje … Zakaj tega ne bi storili sami ?

6 Korak za korakom …

Dejstvo je, da je govorno omrežje tako veliko, da sprememb ni bilo mogoče vpeljati v enem koraku. Glede na naravo sprememb, so se izvajale posamezne faze – najprej na nivoju centralne infrastrukture govornih strežnikov (vključno s pripravo rešitev za spremljajoče storitve), zagotovitvijo ustreznih in centraliziranih povezav za SIP dostop do javnega telekomunikacijskega omrežja ter potrebnih posegov na delih »podatkovnega« omrežja. Sledile so spremembe na »lokalnem« oziroma uporabniškem nivoju (na preko 110 lokacijah je bil izvede izklop obstoječega govornega sistema na nivoju posamezne policijske enote, nadomestitev terminalne opreme, ukinjanje lokalne povezave z javnim telekomunikacijskim omrežjem, ki jo je takoj nadomestila povezava preko SIP govornega dostopa). V praksi je prehod na nov del omrežja pomenil za končnega uporabnika le krajši izpad povezan z zamenjavo terminalne opreme. Po opravljeni zamenjavi, je uporabnik uporabljal osnovne storitve na praktično enak način kot do sedaj, hkrati pa je pridobil določene nove funkcionalnosti (npr. dostop do centralnega imenika, možnost naprednejšega upravljanja telefona preko aplikacije).

Za nemoteno delovanje govornega omrežja je potrebno to omrežje ustrezno upravljati in nadzorovati na način, da se število morebitnih napak kar se da zmanjša, hkrati pa je primeru okvare mogoče to pravočasno zaznati in v najkrajšem času odpraviti. V pred leti vzpostavljen nadzorni sistem (na osnovi Nagios - https://www.nagios.org), smo vključili praktično vse naprave za katere smo ocenili, da je to smiselno (npr. govorni strežniki, GSM vmesniki, SBC-ji oziroma mejni krmilniki sej, …). Za lažje analizo preteklih dogodkov je bil vzpostavljen tudi sistem zajemanja SIP signalizacijo (preko HEP/EEP protokola) za vse lokacije po državi, na centralnem strežniku Homer (https://github.com/sipcapture/homer).



Slika 1: Centralizirano zbiranje signalizacijskih sporočil omogoča naprednejšo analizo.

Na ta način pridobljeni podatki omogočajo tudi naprednejšo analizo posameznega klica (praktično v realnem času) na nivoju uporabnika, ki je opozoril na neustrezno delovanje storitve. Preko tega strežnika je mogoče preveriti tudi različne parametre (npr. MOS, izguba paketov, podrhtavanje, …), ki vplivajo na kakovost govorne zveze. S tem namenom so bila v večini primerov uporabljena standardna orodja oziroma razvita orodja, kot pomoč tehničnemu osebju (npr. elektronski »delilnik«, sonde za zajemanje TCP/IP prometa na osnovi kartičnega računalnika Raspberry Pi). V večini primerov gre za lasten razvoj oziroma integracije prosto dostopnih orodij s področja odprtokodnih rešitev.

7 Rezultati

V nezanemarljivem delu omrežja so že danes uporabljeni tudi mehanizmi , ki smo jih vajeni iz omrežjih telekomunikacijskih operaterjev (npr. centralizirano upravljanje z uporabniškimi napravami / provisioning /, s pomočjo katerega lahko med drugim poskrbimo za centralizirano posodabljanje programske opreme končnih naprav oziroma nastavitev terminalne opreme). Razvoj programske opreme, ki omogoča prilagajanje uporabniški storitve končnemu uporabniku (npr. aplikacija TPsplet, aplikacija za mobilni telefon) pa je v večini primerov plod lastnega znanja oziroma razvoja, upoštevajoč sodobne koncepte pri razvoju le te…

Slika 2: S pomočjo uporabe dostopnih vmesnikov API je bila razvita aplikacija s pomočjo katere lahko uporabnik sam določa dostopne parametre telefonskega aparata.

Zgrajen sistem omogoča hitro prilagajanje potrebam specifičnim službam policije (npr. Specialna enota policije) na področju države oziroma podpori enotam v tujini (npr. Center za policijsko sodelovanje). Dobro poznavanje omrežja in opreme, je omogočil tudi hiter odziv v času epidemije povezane s COVID-19, ko so bile brez kakršnihkoli dodatnih finančnih sredstev s pomočjo prerazporeditve obstoječe opreme, praktično čez noč zagotovljene nujno potrebne storitve glede na nove okoliščine (npr. konferenčni strežniki, oddaljen storitveni center / HelpDesk).

Na podlagi že izvedenih aktivnosti je mogoče ugotoviti sledeče:

- Skupni stroški za prenovo govornega omrežja skupaj s prenovo registrofonskega sistema znašajo okrog 490.000 € (z vključenim DDV), kar je bistveno manj, glede na dosedanje izkušnje v primerih, ko je bila izvedba realizirana preko pogodbenih partnerjev. Upoštevati je potrebno tudi dejstvo, da bi bilo ta znesek (na področju dela strojne opreme) mogoče do določene mere dodatno znižati. V tej fazi smo se namreč odločili za okolje, ki omogoča tako uporabo virtualizacije kot možnosti namestitve in uporabe fizičnih vmesnikov.

- Glede na predvideno prenovo na regionalnem nivoju oziroma državnem nivoju , naj bi bil celoten znesek za prenovo celotnega govornega omrežja policije nižji od 750.000-780.00 € (v tem znesku, so vključena tudi že porabljena sredstva).

- Vse ključne komponente so podvojene in večkratno povezane v omrežje.

- Zagotovljene so naprednejše možnosti rezervnih poti - v smislu povezovanja z javnim telekomunikacijskim omrežjem in znatni finančni prihranki na nivoju mesečnih stroškov.

- Rezultat prenove je govorno omrežje, ki v smislu dobave opreme ne pomeni neposredne odvisnosti od posameznega proizvajalca.

- Razpoložljivi odprti vmesniki omogočajo možnosti dodatnih povezav, integracij oziroma lasnega (do)razvoja.

- Uporabljene so rešitve, ki so preverjene in po potrebi omogočajo tudi »nadgradnjo« v smislu najema višjega nivoja tehnične podpore (najem storitve).

- V primerih, ko so potrebne komercialne licence (z lastnimi viri pa so bile izvedene integracije), govorni sistemi omogočajo polno funkcionalnost, licence pa niso časovno omejene (op. ni uporabljen najemni model). Prav tako ni nobenih omejitev glede največjega števila uporabnikov na posameznem govornem strežniku.

- Vzpostavljeno je bilo testno okolje, ki ima popolnoma enake funkcionalnosti kot produkcijsko.

- Uporabnik je pridobil določene možnosti pri funkcionalnostih, hkrati pa do določene mere razbremenil tehnično osebje.

Kljub vsem pozitivnim učinkom pa je potrebno upoštevati tudi:

- Dejstvo, da so bile ključne aktivnosti izvedene v okviru relativno ozke skupine strokovnjakov. Potrebno se je zavedati problematike povezane z omejenimi viri glede razvoja oziroma v smislu morebitne odsotnosti – tudi kot na primer - kaj lahko razumemo kot potencialno »kritični dogodek« (v smislu morebitne istočasne izgube ali daljše odsotnosti več oseb). S tem namenom je vzpostavljeno okolje za izmenjavo potrebnih informacij (portal). Kljub vsemu pa je stopnja sistemske integracije takšna, da bi bil prenos storitve na zunanjega ponudnika, vsaj za določeno obdobje najverjetneje otežen.

- Da niso bile izvedene vse tehnično mogoče »integracije« - saj niso potrebne niti smiselne (npr. konferenčni strežnik BigBlueButton, strežnik za izmenjavo sporočil – IM).

- Izpostavljene so različne dileme glede uporabljenih konceptov (npr. tipa terminalne opreme, nujnih oziroma ne nujnih vložkov v obstoječ sistem, možnost neposredne povezave z operaterjem kot alternativne možnosti za GSM vmesnike).

- Da lahko na določenih področjih. do določene mere zunanji dejavniki vplivajo na razvoj. Ključnega pomena bo upoštevanje primerov dobrih praks (tako lastnih kot tujih), ki se bodo uporabljeni pri nadaljnjih korakih posodobitve govornega omrežja na regionalnem oziroma državnem nivoju. Kljub enovitemu omrežju (ITSP) so uporabniške zahteve glede na naravo dela, na posameznih organizacijskih nivoji policije različne.

- Prav tako bo potreben razmislek glede morebitnih sprememb/nadgradenj in obsega na področju IKT rešitev zaradi zunanjih dejavnikov.

8 Načrti

Glede na predvidene dobave opreme, je za obdobje 2025/2026 načrtovana nadgradnja sistemov na regionalnem in državnem nivoju. V neposredni povezavi z nadgradnjo govornega omrežja, je tudi predvidena posodobitev klicnih centrov za sprejem klicev na številko 113 oziroma številko 116000. Glede na potrebe operativnih služb, pričakovanja uporabnikov in spremembe na področju zakonodaje, pri nadgradnji ne bo šlo le za nadomestitev in posodobitev telekomunikacijske opreme.

Prihajajo klicni centri nove generacije, ki bodo poleg govorne komunikacije omogočale tudi sprejem drugih komunikacijskih »kanalov« (npr. RTT- Real-time text, video, …). Upoštevaje velika pričakovanja povezana z novo generacijo klicnih centrov, so se bile že izvedle tudi določene aktivnosti v povezavi z orodji za umetno inteligenco (npr. področje uporabe velikih jezikovnih modelov, detekcija govora, priprava transkripcij). Upoštevaje naravo organa v sestavi MNZ, se je potrebno zavedati, da gre za specifičnega uporabnika, ki ne more uporabljati storitev različnih ponudnikov na spletu preko različnih programskih vmesnikov. Vzpostavitev potrebnih funkcionalnosti s pomočjo orodij umetne inteligence v lastnem omrežju pa glede na pričakovanja in potrebe, zahteva relativno visoke vložke v različne vire (strokovnjake, strojno opremo, čas …) - vključno z delovanjem in vzdrževanjem tovrstnih rešitev.

Arhitektura podatkovne platforme za podporo naprednih BI rešitev

Jani Kajzovar, Klemen Drev, Uroš Trstenjak, Aljaž Mislovič, Miha Pavlinek

Databox Inc., Ptuj, Slovenija
jani.kajzovar@databox.com, klemen.drev@databox.com, uros.trstenjak@databox.com, aljaz@databox.com, miha.pavlinek@databox.com

Predstavljamo modularno arhitekturo podatkovnega jezera, ki omogoča izvajanje analitičnih poizvedb neposredno na Parquet datotekah, shranjenih v objektni shrambi (npr. S3) z uporabo vgrajenega SQL-pogona DuckDB. Sistem vsebuje zajem in pretvorbo surovih podatkov iz heterogenih virov v stolpčni format Parquet, njihovo shranjevanje v objektni shrambi ter dinamično poizvedovanje znotraj mikroservisov. Ključni prispevek je neposredna integracija rezultatov poizvedb v poslovnoanalitična orodja brez potrebe po klasičnih podatkovnih bazah, kar omogoča hiter dostop do podatkov in horizontalno skalabilnost. Rešitev je univerzalna in primerna za različne domene, vključno z IoT, telemetrijo in spletno analitiko.

Ključne besede:
podatkovno jezero,
objektna shramba,
Parquet,
DuckDB,
mikroservisna arhitektura,
poslovna analitika.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.8

1 Uvod

V času, ko količina razpoložljivih podatkov eksponentno narašča, podjetja iščejo arhitekturne pristope, ki omogočajo hitrejši dostop do podatkov, zmanjšanje stroškov infrastrukture in večjo prilagodljivost pri analitiki. Klasična podatkovna skladišča s centralizirano obdelavo pogosto ne zadoščajo več – ne le zaradi omejitev pri skalabilnosti, temveč tudi zaradi:

- visoke kompleksnosti upravljanja gruče strežnikov in vzdrževanja,
- težav pri horizontalnem razširjanju (elastičnosti),
- togosti sheme, ki otežuje evolucijo in migracije,
- slabše zmogljivosti zaradi tesne povezanosti med shranjevanjem in obdelovanjem.

Pojmi, kot so "data lake" in "in-process analytics", odražajo premik k bolj modularnim, porazdeljenim rešitvam, ki podatke obdelujejo tam, kjer so shranjeni – in samo takrat, ko je to potrebno.

Oblikovanje arhitektur, kjer lahko aplikacije izvajajo poizvedbe neposredno na podatkih v objektni shrambi, se je izkazalo kot obetaven pristop za številne primere rabe. Pojav orodij, kot sta DuckDB in Apache Arrow, je omogočil razvoj lahkih analitičnih rešitev, ki se ne zanašajo več na centralne baze, temveč na vgrajene SQL pogone, ki tečejo v kontekstu aplikacijskih servisov. Tak pristop je še posebej primeren za mikroservisne arhitekture, kjer posamezni deli sistema potrebujejo lasten vpogled v podatke, brez globalne koordinacije ali skupne podatkovne plasti.

Takšno arhitekturo smo v podjetju Databox zasnovali za potrebe produkta »Datasets«, ki uporabnikom omogoča poglobljeno analitiko na osnovi surovih podatkov, ki so združeni v podatkovne nabore (ang. datasets).

2 Zahteve in načela zasnove

Motiv za vpeljavo nove platforme izhaja iz produktne zahteve za rešitev "Datasets", kjer uporabniki na preprost način iz heterogenih virov pridobivajo surove podatke, ki so organizirani v podatkovne nabore (datasets). Ti so uporabnikom prikazani v obliki tabel, nad katerimi je omogočeno filtriranje, sortiranje, skrivanje/premikanje stolpcev, dodajanje novih kalkuliranih stolpcev ipd. Te modificirane nabore je kasneje mogoče združevati ter nad njimi pripravljati poslovne metrike, ki so namenjene vizualizaciji ali nadaljnji analizi.
Za potrebe jasnejše strukture smo podatkovne nabore razdelili na tri logične komponente: izvorne nabore (ang. source datasets), ki predstavljajo originalne, neobdelane podatke; modificirane nabore (modified datasets), kjer poteka sled modifikacij in transformacij; ter združene nabore (ang. merged datasets), ki združujejo vse izvorne ter druge združene nabore v nov nabor za poizvedovanje.

Druge pomembnejše zahteve:

- vsak nabor ima lastno podatkovno shemo z možnostjo prilagajanja,
- promptno odzivanje ob nalaganju izvornih naborov, modifikacijah, združevanju in pripravi metrik,
- podpora transformacijam ter sprotnim kalkulacijam na osnovi formul,
- v naborih je mogoče shranjevati večje količine podatkov (več milijonov vrstic),
- nizki obratovalni stroški.

Skladno s podanimi zahtevami, smo arhitekturo za podatkovno platformo zasnovali okoli štirih ključnih načel, ki omogočajo visoko učinkovitost, skalabilnost in nizke operativne stroške:

- **Ločenost shrambe in izvajanja** (ang. separation of storage and compute) Skladiščenje podatkov in izvajanje analitične logike sta v sistemu strogo ločena. Vsi podatki so shranjeni v enotnem formatu znotraj objektne shrambe, medtem ko se analitične poizvedbe izvajajo po potrebi v aplikacijskih servisih, z ločenim poizvedbenim pogonom. Ta zasnova omogoča neodvisno skaliranje

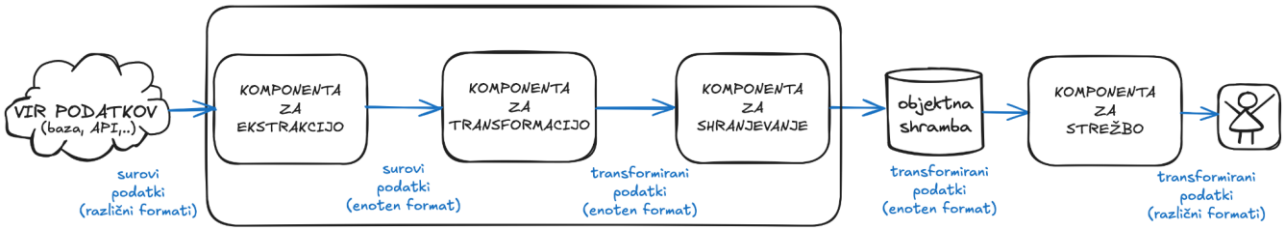
posameznih slojev in preprečuje tesno povezanost med podatkovno in obdelovalno komponento, kot je to pogosto pri klasičnih podatkovnih bazah.

- **Modularnost in zamenljivost komponent:** Vsi gradniki sistema – od zajema podatkov, transformacij, shranjevanja, pa do poizvedb – so zasnovani kot neodvisne, zamenljive enote. Tako lahko posamezne komponente zamenjamo ali nadgradimo brez večjih posegov v preostanek sistema. To omogoča hitrejše iteracije, testiranje alternativnih pristopov (npr. zamenjava objektne shambe z drugo tehnologijo) in lažje prilagajanje spremembam v oblačni infrastrukturi.
- **Stateless pristop v mikroservisnem okolju:** Obdelava podatkov poteka znotraj aplikacijskih mikroservisov, ki so po zasnovi "stateless". Vsaka poizvedba se izvede neodvisno, z uporabo začasnega konteksta. To pomeni, da mikroservisi ne hranijo stanja med zahtevki, kar omogoča enostavno razporejanje, samodejno skaliranje in odpornost na napake.
- **Horizontalna skalabilnost z minimalnim upravljanjem:** Ker je vsak mikroservis popolnoma neodvisna enota, z zmožnostjo analitične poizvedb, lahko sistem enostavno skaliramo horizontalno. To pomeni, da lahko avtomatsko prilagajamo število izvajalnih enot glede na obremenitev, na primer število zahtevkov ali dolžino vrste v sistemu za razpošiljanje sporočil. Ta pristop omogoča dinamično prilagajanje zmogljivosti – na primer ob povečanem številu zahtev za vizualizacijo, izvoze podatkov ali periodične transformacije – brez vpliva na druge dele sistema. Vsak izvajalna enota obdeluje svojo nalogo z izoliranim kontekstom, kar omogoča skoraj linearno skaliranje brez potrebe po deljenem stanju.

3 Arhitektura

3.1. Visokonivojska predstavitev

Za lažje razumevanje zasnove predstavljamo visokonivojski diagram arhitekture za pripravo izvornih naborov (ang. source datasets), ki predstavljajo originalne, neobdelane podatke. Diagram prikazuje ključne komponente sistema in pretok podatkov med njimi. Fokus je na logičnem zaporedju faz: od uvoza surovih podatkov do njihove transformacije, shranjevanja in produktne uporabe



Slika 1: Visokonivojska predstavitev arhitekture.

3.2. Tok podatkov

Arhitektura sistema sledi pretočnemu modelu, kjer so vse faze – od vnosa podatkov do strežbe rezultatov – ločene, modularne in sledljive. Le te smo v nadaljevanju podrobneje opisali:

Vnos podatkov iz zunanjih virov

Podatke v sistem dostavlja interni ETL mehanizem, ki skrbi za zajem iz zunanjih API-jev, relacijskih baz in drugih virov. Ta komponenta za ekstrakcijo podatkov predstavlja vhodno točko v podatkovno jezero. Pomembno je, da sistem vse podatke prevzame in jih pretvori v enoten stolpčen format in s tem skrije podrobnosti posameznega vira podatkov. To omogoča enoten podatkovni model in takojšnjo integracijo z nadaljnjimi komponentami.

Transformacija in tipizacija

Prejete podatke sistem najprej obdela z nizom transformacij, ki vključujejo:

- pretvorbo vrednosti (npr. normalizacija enot),
- tipizacijo stolpcev glede na predpisano shemo (npr. int64, timestamp, decimal),
- odstranjevanje ali preimenovanje stolpcev za boljšo razumljivost,

Transformacije se izvajajo znotraj komponente za transformacijo. Sistem ob tem spremlja spremembe sheme – če se vhodna shema razlikuje od pričakovane, se aktivira mehanizem za validacijo, ki zazna neskladja in po potrebi sproži opozorilo ali ustavi obdelavo. Ta pristop omogoča kontrolirano prilagajanje na evolucijo podatkov skozi čas.

Shranjevanje in verzioniranje v objektni shrambi

Pripravljen izvorni nabor podatkov v poenotenem formatu in normalizirani vsebini nato shranimo v objektno shrambo. Dodatno ta komponenta skrbi za verzioniranje. Verzioniranje omogoča obnovljivost preteklih stanj, podporo za "time-travel" razhroščevanje in enostavno testiranje novih transformacij brez izgube zgodovine. Poleg izvornih (source) naborov podatkov sistem podpira tudi shranjevanje modificirane (modified) in združene (merged) različice, ki služijo kot predelana ali kombinirana nadgradnja izvornih podatkov. Ostale različice imajo ločen ETL tok podatkov, ki pa kot vhod uporabijo predhodno shranjen nabor podatkov iz objektne shrambe.

Strežba rezultatov

Shranjene nabore podatkov iz objektne shrambe ponujamo preko različnih funkcionalnosti produktne rešitve "Datasets". Rezultati poizvedb so uporabnikom dostopni preko različnih kanalov:

- Vpogled v podatke: kot JSON odgovor prek REST API-ja,
- Izvoz podatkov: kot CSV ali Parquet export v namenske S3 exports mape,
- Vizualizacija: kot interaktivni prikaz v BI orodju, kjer API služi kot query backend.

Servis samodejno optimizira poizvedbe glede na kontekst (npr. paginacija, agregacije, filtri) in zagotavlja odzivne čase, primerno za interaktivno raziskovanje podatkov.

4 Ključne tehnologije

Izbira ključnih tehnologij, ki sestavljajo hrbtenico podatkovne arhitekture, vključujejo stolpčni format Parquet ter analitični pogon DuckDB. Poseben poudarek je na učinkovitem shranjevanju in obdelavi podatkov v podatkovnem jezeru, kjer te tehnologije omogočajo visoko zmogljivost tudi pri obdelavi večjih količin podatkov. Opisane so prednosti za analitiko ter praktične možnosti neposrednega branja, pisanja in procesiranja podatkovnih nizov.

4.1. Uporaba stolpčnega formata Parquet

Pri zasnovi arhitekture podatkovnega jezera smo se odločili za uporabo odprtokodnega stolpčnega formata Apache Parquet kot osnovnega formata za shranjevanje podatkov. Parquet je bil razvit posebej za potrebe analitičnih sistemov, kjer je branje podatkov pogosto omejeno na posamezne stolpce ali podmnožice tabel, ne pa

celotne vrstice. V takih primerih stolpčni formati nudijo bistveno boljšo učinkovitost glede hitrosti in porabe virov v primerjavi z vrstičnimi formati (npr. CSV, JSON).

Struktura in shema

Parquet temelji na strogo definirani shemi, ki opisuje strukturo podatkov v datoteki (stolpce, podatkovne tipe, gnezdenje, ponovitve). Ta shema je shranjena skupaj s podatki v datoteki Parquet, kar omogoča podatkovne vire z vgrajeno shemo in poenostavi izvajanje poizvedb brez zunanjih metapodatkov. Shema je definirana v formatu Apache Thrift, kar omogoča interoperabilnost s številnimi orodji in jeziki (npr. Spark, Pandas, DuckDB, Hive) [1]. Uporaba statične sheme pomeni tudi, da lahko poizvedbeni pogoni, kot je DuckDB, že pred začetkom branja podatkov optimizirajo poizvedbe - katere stolpce bodo brali in kako bodo interpretirali podatkovne tipe (npr. INT64, FLOAT, BOOLEAN, TIMESTAMP).

Kompresija in kodiranje

Parquet vključuje napredne možnosti kompresiranja in kodiranja, kot so:

- Stiskanje na ravni stolpcev (ang. Column-level compression): Vsak stolpec se stisne neodvisno, kar omogoča boljše razmerje kompresije, saj imajo podatki znotraj stolpcev pogosto podoben tip in porazdelitev.
- Kodirne sheme (ang. Encoding schemes): Uporablja učinkovite sheme, kot so "run-length encoding", "dictionary encoding" in "bit-packing", kar zmanjša velikost brez izgub.
  Podpora več algoritmom: Parquet omogoča uporabo različnih kompresijskih algoritmov (npr. Snappy, Gzip, Brotli, ZSTD), odvisno od primera uporabe in kompromisa med hitrostjo in kompresijo [1].

Kompresija v Parquetu ni le orodje za zmanjševanje velikosti, temveč ima neposreden vpliv na hitrost poizvedb – manjši podatkovni bloki pomenijo manj vhodno/izhodnih operacij in hitrejše prenose preko omrežja (kar je še posebej pomembno pri obdelavi podatkov, shranjenih v objektni shrambi).

Prednosti za analitiko v oblaku

Uporaba Parquet formata prinaša več ključnih prednosti za obdelavo podatkov v porazdeljenih in oblačnih okoljih:

- Selektivno branje (ang. predicate pushdown): Parquet omogoča, da poizvedbeni pogoni preberejo samo relevantne dele podatkov – tako po stolpcih kot po vrsticah (z uporabo statistik v metapodatkih) [2].
- Kompaktnost: Zaradi učinkovite kompresije so Parquet datoteke pogosto 5–10× manjše od enakovrednih CSV ali JSON datotek.
- Optimizacija prenosa iz objektne shrambe: Manjša količina podatkov pomeni nižje stroške prenosa iz shrambe ter manjšo obremenitev omrežja in procesorja.
- Podpora za evolucijo sheme: Parquet podpira določeno stopnjo fleksibilnosti pri spreminjanju sheme (npr. dodajanje/odstranjevanje stolpcev), kar olajša razvoj v hitro spreminjajočem se podatkovnem okolju.

4.2. DuckDB kot vgrajen SQL pogon za obdelavo Parquet podatkov

DuckDB je sodoben, stolpčno zasnovan relacijski pogon, optimiziran za analitične poizvedbe in zasnovan za "in-process" uporabo. Deluje kot vgrajena komponenta znotraj aplikacijskega procesa, brez potrebe po zunanjem strežniku, omrežnih povezavah ali kompleksni infrastrukturi. Zaradi tega je posebej primeren za mikroservisne in oblačne arhitekture, kjer želimo analitične zmožnosti vključiti neposredno v aplikacijsko logiko.

Neposredno branje in pisanje Parquet datotek

DuckDB podpira nativno branje in zapisovanje Parquet datotek – tako lokalno kot prek objektne shrambe, kot je Amazon S3 [3]. Poizvedbe se lahko izvajajo neposredno nad datotekami v oblaku brez vnaprejšnjega uvoza, kar omogoča query-on-read arhitekturo brez replikacije podatkov.
Primer branja Parquet iz S3:

```
SELECT *
FROM read_parquet('s3://my-bucket/data/events.parquet')
WHERE event_type = 'click';
```

Primer zapisovanja Parquet nazaj na S3:

```
COPY (
  SELECT * FROM session_events WHERE duration > 30
)
TO 's3://my-bucket/filtered/long_sessions.parquet'
(FORMAT PARQUET, COMPRESSION ZSTD);
```

Obdelava podatkov, večjih od pomnilnika (ang. spill to disk)

Čeprav DuckDB deluje "in-process" in primarno v pomnilniku, vključuje mehanizem za obdelavo naborov podatkov, ki presegajo razpoložljiv RAM. S pomočjo funkcionalnosti "spill to disk" DuckDB samodejno shranjuje vmesne rezultate na disk, kadar zmanjka pomnilnika med izvajanjem poizvedbe. To omogoča robustno obdelavo velikih Parquet datotek brez ročnega upravljanja z delitvijo podatkov ali ustvarjanjem podatkovnih delov (chunking) [4].

Privzeto se začasne datoteke ustvarjajo v sistemski začasni mapi, vendar lahko pot in velikost diskovnih medpomnilnikov prilagodimo z uporabo konfiguracijskih nastavitev DuckDB.

Visoka zmogljivost zaradi optimizirane izvedbe

DuckDB izvaja poizvedbe s t.i. vektoriziranim izvajanjem (ang. vectorized execution), kjer se podatki obdelujejo v blokih (vektorjih) – običajno 1024 vrstic naenkrat. To omogoča boljši izkoristek procesorskega predpomnilnika in SIMD ukazov v primerjavi z pristopi branja vrstico po vrstico (ang. row-at-a-time). Poleg tega izvaja številne optimizacije, kot so:

- odlaganje branja vrednosti do trenutka, ko so res potrebne (ang. late materialization) [6],
- filtriranje vrstic že med branjem iz Parquet datotek (ang. predicate pushdown),
- uporaba metapodatkov za optimizacijo načrta poizvedbe (ang. statistics-aware query planning),
- branje zgolj potrebnih stolpcev (ang. column pruning).

Kot rezultat DuckDB pogosto dosega hitrosti primerljive ali boljše od distribuiranih rešitev (npr. Spark, Presto) pri manjših do srednje velikih naborov podatkov, brez potrebe po grozdu strežnikov ali zapletenem usklajevanju.

Podpora za standardni SQL in napredne poizvedbe

DuckDB podpira obsežen podnabor SQL standarda: CTE-je (WITH izrazi), okenske funkcije (window functions), agregacije, GROUP BY ROLLUP, PIVOT, date/time funkcije, pod-poizvedbe, UNION, INTERSECT, EXCEPT, JOIN vseh vrst in še več [7]. To pomeni, da lahko kompleksne analitične poizvedbe, ki bi sicer zahtevale več korakov ali zunanja orodja, izvedemo v enem koraku - neposredno nad Parquet datotekami.
To zmanjša potrebo po dodatni obdelavi v aplikacijski kodi in omogoča bolj deklarativen slog razvoja.

Zaradi kombinacije lastnosti – "in-process" izvedbe, podpore za Parquet, neposredne integracije z objektno shrambo ter možnosti obdelave večjih naborov podatkov brez izpada – je DuckDB izjemno primeren za sodobne podatkovne arhitekture, kjer je potrebna hitra, fleksibilna in nizkocenovna analitika brez tradicionalnih omejitev centraliziranih podatkovnih skladišč, opisanih v uvodu.

5 Implementacija

Implementacija sistema temelji na načelih vgradljivosti, pretočnosti in odprtosti. Tehnologiji iz prejšnjega poglavja – DuckDB in Parquet – sta v središču zasnove, ki smo jo kombinirali z objektno shrambo kot trajnim slojem. Skupaj tvorijo osnovo pretočnega in modularnega podatkovnega sistema, prilagojenega sodobnim analitičnim potrebam.

5.1. Ključne komponente

Arhitekturo sestavlja pet ključnih komponent, ki skupaj tvorijo skalabilen, modularen in nizkocenoven podatkovni sistem, zasnovan za analitične delovne obremenitve v oblaku.

Parquet datoteke
Prej opisan Parquet je naš privzeti podatkovni format zaradi svoje stolpčne zasnove, učinkovite stiskanja, podpore za kompleksne tipe in možnost selektivnega branja. Vsaka datoteka vsebuje tudi samo-pojasnjevalno shemo, kar omogoča dinamično analizo brez zunanjih metapodatkov.

Amazon S3 objektna shramba
Amazon S3 predstavlja trajno plast za shranjevanje vseh naborov podatkov. Podatki so zapisani v Parquet formatu in organizirani po naborih, verzijah in vlogah (izvorni, predelani, združeni). Objektna shramba omogoča nizke stroške, visoko razpoložljivost in neposreden dostop prek HTTP(S), kar jo naredi idealno za podatkovna jezera.

DuckDB
DuckDB deluje kot vgrajen SQL pogon (embedded engine) znotraj vsakega mikroservisa. Poizvedbe se izvajajo neposredno nad Parquet datotekami na S3, brez potrebe po predhodnem nalaganju podatkov ali vzdrževanju ločene baze. Zaradi podpore za "spill to disk" in vektoriziranega izvajanja DuckDB omogoča zmogljivo obdelavo tudi večjih naborov podatkov v omejenih kontekstih. Kombinacija teh treh tehnologij tvori jedro naše aplikacije.

Dataset Engine / API servis
"Dataset Engine" je aplikacijski sloj, ki povezuje vse ostale komponente. Njegove naloge vključujejo pridobivanje podatkov iz zunanjih virov, pretvorbo v Parquet, transformiranje in normalizacijo podatkov, zapisovanje v S3, izvajanje poizvedb prek DuckDB ter vračanje rezultatov uporabnikom ali BI orodjem preko API vmesnikov. Ta komponenta predstavlja centralni zaledni servis naše rešitve "Datasets".

Kubernetes

Kubernetes služi kot temeljna infrastruktura za orkestracijo in izvajanje sistema. Vsi mikroservisi, vključno z instancami, ki uporabljajo DuckDB, tečejo kot podi (osnovne enote za izvajanje enega ali več povezanih kontejnerjev) znotraj Kubernetes okolja. To omogoča:

- avtomatsko skaliranje števila izvajalcev glede na obremenitev (npr. velikost čakalne vrste zahtevkov),
- dinamično upravljanje virov (CPU, pomnilnik, disk za začasne datoteke),
- visoko razpoložljivost z mehanizmi za ponovni zagon neuspešnih komponent.

S tem Kubernetes ni zgolj infrastrukturna plast, ampak aktivno podpira elastičnost, robustnost in učinkovitost celotne platforme.

5.2. Organizacija S3 prostora

Za učinkovito, pregledno in razširljivo upravljanje s podatki uporabljamo dosledno strukturirano organizacijo objektne shrambe v Amazon S3. S tem omogočamo enostavno upravljanje različic, arhiviranje, izvoze in obdelavo naborov podatkov na podlagi njihovega tipa in izvora.

Strukturirane mape

Organizacija S3 prostora temelji na jasni razdelitvi glede na tip nabora podatkov. Vsaka vrsta nabora ima svojo lastno korensko mapo, kar omogoča enostavno ločevanje, upravljanje in iskanje po posameznih tipih podatkov:

- sources/ – izvorni nabori podatkov, kot jih posreduje ETL sistem opisan v poglavju Arhitektura,
- modified/ – modificirani nabori podatkov so transformirane in obogatene različice izvornih podatkov (npr. z dodatnimi izračunanimi stolpci, pretvorbo podatkovnih tipov, filtriranjem),
- merged/ – rezultat združevanja dveh ali več naborov podatkov tvori združen nabor,
- exports/ – začasne datoteke, ustvarjene za zahtevke uporabniških izvozov keteregakoli nabora podatkov (npr. v format CSV ali Parquet).

Primer tipične poti do datoteke:

```
s3://databox-dev-datasets/sources/{datasetId}/v{timestamp}_{uuid}.parquet
s3://databox-dev-datasets/sources/{datasetId}/v{timestamp}_{uuid}.json
```

Verzioniranje in identifikacija

Vsak nabor podatkov je verzioniran z uporabo časovnega žiga v{timestamp} (npr. v202407210845), kar omogoča sledenje zgodovini in ponovljivost obdelav. Poleg verzije vsak zapis vsebuje tudi UUID, ki služi kot globalni identifikator posamezne datoteke. S tem lahko isti nabor podatkov obstaja v več različicah, vsaka z lastno vsebino in enolično identifikacijo.

Dodatno se za potrebe optimizacije shema vsake Parquet datoteke hrani tudi ločeno v .json obliki pod istim imenom, kar omogoča samostojen dostop do metapodatkov brez potrebe po branju vsebine parquet datoteke.

Arhiviranje in življenjski cikel

Za optimizacijo stroškov shranjevanja uporabljamo S3 "lifecycle policies", ki samodejno arhivirajo stare verzije. Po npr. enem mesecu se stare verzije datotek izbrišejo. Operativno s tem zagotavljamo:

- nadzorovano porabo prostora,
- samodejno čiščenje starih različic,
- ločevanje aktivnih in arhiviranih podatkov brez ročnega posega.

5.3. Mikroservisna integracija DuckDB

Ena izmed osrednjih značilnosti arhitekture je uporaba DuckDB kot vgrajenega (embedded) SQL pogona znotraj posameznih mikroservisov. Ta integracija omogoča, da se podatkovna poizvedba izvede neposredno na S3 v okviru aplikacije, brez potrebe po vmesnih bazah ali ločenih odložiščih podatkov.

Za branje podatkov z objektne shrambe (S3) mikroservisi dinamično nastavijo S3 parametre, ki vključujejo dostopne ključe, regijo in ciljno končno točko (endpoint). Po tem je mogoče brati Parquet datoteke neposredno prek "read_parquet" ali "parquet_scan".

Začasne tabele in podpora za začasne datoteke (ang. spill to disk)

DuckDB omogoča ustvarjanje začasnih tabel (CREATE TEMP TABLE), ki se pogosto uporabljajo za pripravo vmesnih rezultatov, združevanje podatkov iz več virov ali zaporedno izvajanje več transformacij.

```
CREATE TEMP TABLE filtered AS (
  SELECT * FROM parquet_scan('s3://...') WHERE active = true
);
```

V primeru, ko količina podatkov preseže razpoložljiv pomnilnik, DuckDB samodejno aktivira mehanizem "spill to disk", pri čemer vmesne rezultate shranjuje v začasno mapo.

Konfigurabilnost in parametri za nadzor porabe virov

Arhitektura je zasnovana tako, da omogoča enostavno prilagajanje različnim operativnim okoljem in obremenitvam, hkrati pa podpira postopno širitev funkcionalnosti z minimalnim posegom v obstoječi sistem. Ta lastnost je ključna za dolgoročno vzdržnost in prilagodljivost platforme.

Vsak mikroservis, ki izvaja DuckDB poizvedbe, ima možnost dinamične konfiguracije virov, kar omogoča fino uravnavanje delovanja glede na zahteve okolja ali naloge. Tipične nastavitve vključujejo:

```
SET memory_limit='4GB';             -- maksimalna količina RAM-a
SET threads=4;                      -- število sočasnih niti za izvajanje
SET max_temp_directory_size='20GB'; -- meja za uporabo diska
SET temp_directory='/tmp/duckdb';   -- pot do začasnih datotek
```

V kombinaciji z možnostjo konfiguracije DuckDB parametrov na nivoju aplikacije ali okolja (npr. prek .env ali Kubernetes ConfigMap) to omogoča centralizirano upravljanje zmogljivosti brez sprememb v kodi.

Podpora za različna okolja

Celoten sistem podpira več okolij (npr. produkcija, razvoj, testiranje) z uporabo okoljsko pogojenih konfiguracij. Parametri, kot so dostopni ključi, končne točke podatkovne shrambe in poti do izhodnih map, se določijo prek okoljskih spremenljivk, kar omogoča:

- izolacijo med okolji,
- lokalni razvoj brez povezave do oblaka (npr. v testnih okoljih namesto S3 uporabljamo LocalStack),
- konsistentno izvajanje testov z uporabo nadzorovanih virov.

6 Zaključek

Z razvojem lastne arhitekture za obdelavo naborov podatkov neposredno na objektni shrambi smo v praksi potrdili, da je možno zgraditi zmogljiv analitični sistem tudi brez uporabe tradicionalnih podatkovnih skladišč ali kompleksnih distribuiranih rešitev. Ključni element uspeha je bila odločitev za arhitekturo, ki združuje: stolpčno shranjevanje (Parquet), vgrajeno obdelavo (DuckDB) in poenostavljeno operativno plast v Kubernetes okolju.

Med razvojem in uvedbo smo se osredotočili na tri kritične cilje:

operativno učinkovitost (minimalna kompleksnost upravljanja),
odpornost na spremembe shem (razvoj sistema za zaznavanje in prilagoditev),
hitro odzivnost v aplikacijah, kjer poizvedbe pogosto izhajajo iz konteksta uporabnika.

Arhitektura se je izkazala kot stabilna tudi pri večjih količinah podatkov, zlasti zaradi možnosti obdelave, ki presega pomnilnik (spill to disk), in zaradi sposobnosti dinamičnega skaliranja prek Kubernetes. Hkrati nam modularna zasnova omogoča postopno širitev sistema brez potrebe po nadgradnji celotne infrastrukture.

Izkušnje, pridobljene pri gradnji tega sistema, kažejo, da lahko tudi relativno majhna in osredotočena rešitev, če je pravilno zasnovana, pokrije večino sodobnih analitičnih potreb – od verzioniranja, transformacij, dinamičnih poizvedb pa vse do integracije z uporabniškimi aplikacijami.

Literatura

[1] Apache Parquet Format Specification. GitHub, https://github.com/apache/parquet-format, obiskano 24.7.2025

[2] Apache Arrow documentation. https://arrow.apache.org/docs/python/parquet.html#filters-predicate-pushdown, obiskano 24.7.2025

[3] DuckDB httpfs extension. https://duckdb.org/docs/stable/core_extensions/httpfs/overview, obiskano 24.7.2025

[4] DuckDB Memory tuning guide. https://duckdb.org/docs/stable/guides/performance/how_to_tune_workloads.html#larger-than-memory-workloads-out-of-core-processing, obiskano 24.7.2025

[5] DuckDB Execution format. https://duckdb.org/docs/stable/internals/vector.html, obiskano 24.7.2025

[6] DuckDB blog. https://motherduck.com/blog/announcing-duckdb-13-on-motherduck-cdw/, obiskano 24.7.2025

[7] DuckDB SQL documentation. https://duckdb.org/docs/stable/sql/introduction, obiskano 24.7.2025

Izkušnje uporabe podatkovnih zbirk časovnih vrst v Elektro Ljubljana

Tomaž Dolenc, Daniela Maksimović

Elektro Ljubljana, Ljubljana, Slovenija
tomaz.dolenc@elektro-ljubljana.si, daniela.maksimovic@elektro-ljubljana.si

V zadnjem desetletju smo priča izjemnemu razvoju tehnologij interneta stvari (Internet of Things – IoT). Uporaba IoT naprav in pametnih števcev je v celotnem energetskem sektorju (in širše) povzročila ekspanzijo podatkov, ki se jih lahko pridobiva v skoraj realnem času iz različnih merilnih naprav, ki so na voljo in so instalirani neposredno pri končnih odjemalcih in/ali na samem omrežju oziroma v našem primeru na distribucijskem energetskem omrežju DO, ki je v upravljanju na Elektro Ljubljana. Infrastrukture za zbiranje, prenos, shranjevanje in obdelavo ogromnih količin podatkov se spreminja z uvedbo novih tehnologij na področju vele podatkov in sledi tehnološkim trendov. Na Elektro Ljubljana smo primerjali različne odprtokodne podatkovne zbirke časovnih vrst in primerjali njihove lastnosti. V tem članku bomo predstavili naše izkušnje pri zbiranju, prenosu, shranjevanju in obdelavi ogromnih količin podatkov iz merilnih naprav v našem omrežju. S pomočjo primerjalnih testiranj smo pokazali, da je ClickHouse ena najhitrejših analitičnih baz podatkov na trgu, pri čemer se je v realnih primerih uporabe skozi leta pokazalo znatno izboljšanje zmogljivosti pri izvajanju tipičnih poizvedb.

Ključne besede:
časovne serije,
meritve napetosti in energij,
Clickhouse,
TimescaleDB,
Elektro Ljubljana.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.9

1 Uvod

V zadnjih letih je tehnološki napredek na področju interneta stvari (Internet of Things – IoT) bistveno spremenil način delovanja sodobnih elektroenergetskih sistemov. Sodobni elektroenergetski sistemi se vse bolj naslanjajo na digitalne tehnologije za zbiranje, prenos in obdelavo podatkov, pri čemer igrajo ključno vlogo pametni števci in druge IoT naprave, ki so nameščene bodisi pri končnih odjemalcih bodisi neposredno v omrežju [1,2].

Uvedba teh tehnologij je povzročila eksponentno rast količine operativnih podatkov, ki jih je mogoče zbirati v skoraj realnem času. Ti podatki vključujejo informacije o trenutni porabi električne energije, napetostnih profilih, kakovosti oskrbe in drugih ključnih parametrih, ki so bistveni za učinkovito in zanesljivo delovanje distribucijskega omrežja [3]. V primeru distribucijskega omrežja (DO), ki je v upravljanju podjetja Elektro Ljubljana, to pomeni celovito spremljanje stanja omrežja, identifikacijo motenj, optimizacijo obratovanja ter boljše načrtovanje investicij in vzdrževalnih del.

Povezovanje merilnih naprav v pametna omrežja (angl. smart grids) omogoča prehod iz reaktivnega v proaktivno upravljanje, saj operaterji omrežij pridobijo možnost sprotnega prilagajanja obratovalnih parametrov glede na zaznane razmere v omrežju [4]. Tako se zmanjšujejo izgube v omrežju, izboljšuje kakovost oskrbe in povečuje energetska učinkovitost, kar je ključno v kontekstu prehoda na nizkoogljično družbo ter vključevanja obnovljivih virov energije.

Z vidika informacijsko-komunikacijskih tehnologij (IKT) je izziv predvsem v zagotavljanju ustrezne infrastrukture za zbiranje, prenos, shranjevanje in obdelavo ogromnih količin podatkov, kar zahteva uporabo naprednih metod strojnega učenja, podatkovnega rudarjenja ter orodij za analitiko velikih podatkov (big data) [5]. IoT naprave so tako postale ključni element digitalne transformacije energetskega sektorja in predstavljajo osnovo za nadaljnji razvoj inteligentnih in odzivnih elektroenergetskih sistemov.

2 Baze časovnih serij na Elektro Ljubljana

Zadnjih 10 let na trgu postajajo vse bolj popularne time-series podatkovne baze, kot ne-relacijske podatkovne baze ali celo kot kombinacija in združitev relacijskih in ne-relacijskih baz pri čemer so na voljo vse funkcionalnosti od obeh baz in omogočajo za uporabnika dostop do podatkov na že ustaljen standarden način. V ta namen smo preverili delovanje in performanse ter primerjali med seboj timeseries baze ClickHouse in TimescaleDB [6].

2.1. Clickhouse baza časovnih serij

Clickhouse baza je odprtokodna OLAP podatkovna baza, zasnovana za visoko zmogljivo analitiko nad podatkovnimi nabori v petabajtnem obsegu z visokimi hitrostmi vnosa podatkov. Njen sloj za shranjevanje združuje podatkovni format, ki temelji na tradicionalnih drevesih z zlivanjem dnevniških struktur (LSM – log-structured merge) z inovativnimi tehnikami za neprekinjeno preoblikovanje (npr. agregacija, arhiviranje) zgodovinskih podatkov v ozadju. Poizvedbe so zapisane v priročnem SQL dialektu in jih obdeluje najsodobnejši vektorizirani mehanizem za izvajanje poizvedb, z možnostjo kompilacije kode.

Z arhitekturnega vidika baze podatkov sestavljata (vsaj) sloj za shranjevanje podatkov in sloj za obdelavo poizvedb. Medtem ko je sloj za shranjevanje odgovoren za shranjevanje, nalaganje in vzdrževanje podatkov v tabelah, je sloj za obdelavo poizvedb zadolžen za izvajanje uporabniških poizvedb. V primerjavi z drugimi bazami podatkov ClickHouse uvaja inovacije v obeh slojih, kar omogoča izjemno hitre vstavljanja podatkov (inserts) ter poizvedbe (Select queries).

Clickhouse je posebej prilagojena za delo z velepodatki. Ima številne prilagoditve, na primer lahko uporabljamo standardni SQL, lahko pa uporabimo tudi funkcije podatkovne zbirke, ki pohitrijo izvajanje SQL-a. S stališča

skrbništva podatkovne zbirke je upravljanje poenostavljeno, saj sama podatkovna zbirka skrbi za particioniranje tabel, stiskanje podatkov v tabelah in pa za samodejno odstranitev podatkov, ko le-ti niso več potrebni. Podatkovna zbirka omogoča številne prilagoditve in optimiranja, kar je opisano v nadaljevanju.

ClickHouse agresivno uporablja tehnike obrezovanja (pruning), da se izogne obdelavi nepomembnih podatkov med izvajanjem poizvedb. Druge sisteme za upravljanje podatkov je mogoče vključiti na ravni funkcij tabel, mehanizmov tabel ali celo na ravni mehanizmov celotnih podatkovnih baz.

Na zmogljivost baze podatkov poleg usmerjenosti podatkov vpliva še mnogo drugih dejavnikov. V nadaljevanju bomo podrobneje razložili, zakaj je ClickHouse tako hiter, zlasti v primerjavi z drugimi stolpično usmerjenimi podatkovnimi bazami.

2.2. TimescaleDB

TimescaleDB je razširitev baze PostgreSQL, z enostavno namestitvijo in poznanim načinom uporabe. Je odprtokodna baza podatkov, ki uporablja standardni poizvedovalni jezik SQL. Izkorišča številne prednosti PostgreSQL, kot so zanesljivost, varnost in povezljivost s široko paleto orodij drugih proizvajalcev (številni različni programski jeziki, za katera so na voljo gonilniki, ; številne podprte platforme, Linux, Windows, OS X, AIX, Solaris, itd.). Izkaže se kot izjemno zanesljiva in učinkovita rešitev za poizvedovanje nad ogromnimi časovnimi vrstami, hkrati pa jo je moč porazdeliti med več strežnikov in zagotoviti hitro poizvedovanje [6].

TimescaleDB je bila najprej naša prva izbira časovne baze. Največjo prednost smo videli v tem da se podatki samodejno razdelijo po času in prostoru (čas + druga dimenzija npr. naprava, uporabnik). Zato smo razdelili podatke po času in po različnih IoT naprav (števci, ki merijo električno energijo ter druge merilno krmilne naprave. Medtem ko so podatki znotraj same baze razporejeni optimalno v t.i. Hipersegmentirane tabele (Hypertables) uporabnik tega ne opazi (ne rabi se prilagoditi) dela kot s "navadno" tabelo in piše navadne SQL poizvedbe, ki imajo odlično zmogljivost za časovne poizvedbe. Prav tako tudi insert podatkov je dokaj hiter in učinkovit v koliko se pravilno nastavijo in uporabljajo optimizirani indeksi za časovno razvrščene podatke.

3 Karakteristične operacije z podatkovnimi zbirkami časovnih vrst

3.1. Polnjenje podatkovne zbirke

Operacija polnjenje podatkovne baze je ena osnovnih operacij, ki jo izvajamo ob uporabi podatkovnih zbirk podatkov. V našem primeru podatki so specifični sami po sebe ker niso navadni relacijsko povezani podatki ali transakcijski podatki tem več so podatki časovnih vrst oz. podatki so v zaporedju, ki so zabeležene ali opazovane v določenih časovnih intervalih. Vsaka podatkovna točka ima časovni žig (timestamp), ki označuje, kdaj je bila izmerjena.

Pomembno je razumevanje arhitekture procesa nastajanja in obdelave podatkov v podatkovni zbirki. To posebej velja ko so podatki časovne serije in si med seboj sledijo. Dodatno narava podatkov v našem primeru ko gre za meritve električne energije (meritve energije in napetosti) iz IoT naprav je taka, da se jih prepiše/dopolni vsak dan.

Proces polnjenja podatkovne zbirke se prilagodi izvornemu sistemu (na Elektro Ljubljana ostajata 2 glavna vira merilnih podatkov) ter velja za standarden način prenosa podatkov med sistemi (ETL). Vzpostavljen ETL proces vsebuje tudi napredne validacije podatkov ter analize podatkov v času prenašanja ('on the fly') do, pod pogojem, da pri tem nista motena pretok podatkov oziroma ni zmanjšana hitrost samega pretoka. Dodatno si pri tem želimo uporabiti čim več naprednih funkcij, ki se nanašajo na časovnih vrst in omogočajo dopolnitev manjkajočih podatkov z vnaprej določenimi vrednostmi (z uporabo pravil za nadomeščanje, ki jih bomo dobili iz vsebinskih poznavalcev/uporabnikov izvornega sistema, na podlagi statistični analiz podatkov iz prejšnjih nekaj let ter na podlagi algoritmov za strojno učenje, ki se nanašajo na dopolnitev manjkajočih podatkov).

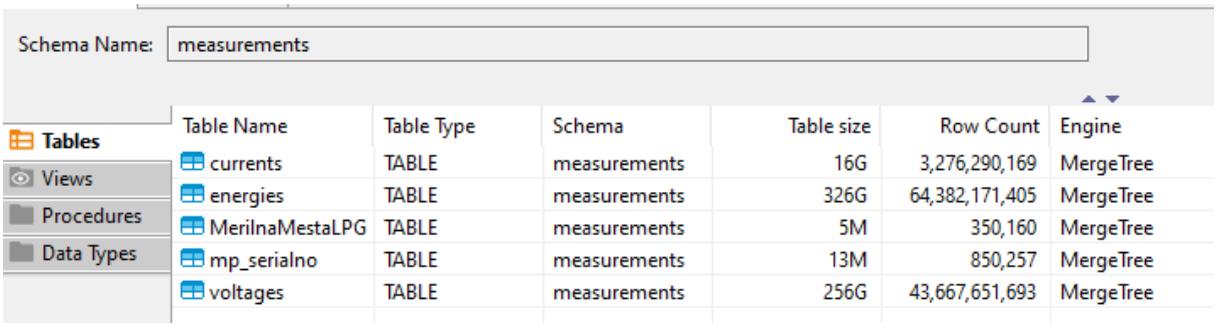


Table Name	Table Type	Schema	Table size	Row Count	Engine
currents	TABLE	measurements	16G	3,276,290,169	MergeTree
energies	TABLE	measurements	326G	64,382,171,405	MergeTree
MerilnaMestaLPG	TABLE	measurements	5M	350,160	MergeTree
mp_serialno	TABLE	measurements	13M	850,257	MergeTree
voltages	TABLE	measurements	256G	43,667,651,693	MergeTree

Slika 1: Velikost ClichHouse baze.

Na Sliki 1 za ilustracijo pokažemo velikost in organiziranost baze časovnih serij na določen dan na Elektro Ljubljana. Velikost baze se spreminja oz. narašča vsak dan. Podatki so grupirani po tipih meritev, ki jih IoT naprave beležijo na terenu.

V našem primeru ko smo delali inicialne load podatkov v obeh bazah, ki imajo enako sistemsko arhitekturo je bil ClickHouse tri krat hitrejši v eno letnem zapisu podatkov meritev energije in napetosti za vsa (cca. 350 000) merilna mesta na Elektro Ljubljana.

Glede celotne zasedenosti prostora na disku tega ranga podatkov je bistvena prednost ponovno bila tudi na strani ClickHouse-a.

3.2. Obremenitev podatkovne zbirke

Tudi če imamo zelo zmogljiv računalnik, ne moremo s preprosto akcijo zmeriti in/ali pospešiti odzivnost podatkovne zbirke. Hitrost oziroma odzivnost podatkovne zbirke je omejena z drugimi dejavniki, kot so hitrost shranjevanja, omrežna pasovna širina ali zasnova same podatkovne zbirke. Če se pričakuje, da se bo odzivnost podatkovne zbirke povečala za n-krat, potem je potreba temeljita analiza vsega kar se dogaja v podatkovni zbirki. V primeru relacijskih baz te že ponujajo svoja lastna orodja s pomočjo katerih se lahko spremlja in analizira obremenitev podatkovnih zbirk. To žal ne velja za bazah časovnih serij in to je analiza prepuščena končnemu uporabniku ter odvisna od njegovih scenarijev uporabe podatkov in načina pisana SQL poizvedb. Pri uporabi smo se omejili in dva »use case« in zanj posledično načrtovali in optimizirali vse operacije nad podatki: poizvedovanje vseh meritev za posamezno merilno mesto ter poizvedovanje veh meritev za manjšo/večjo skupino merilnih mest združeno. Agregacije podatkov in/ali agregatne operacije nad podatkih niso naši najbolj pogosti scenarije uporabe.

3.3. Odstranitve podatkov

Zaradi prostorske omejitve moramo poskrbeti za brisanje/uničenje podatkov in zato je bilo potrebno dobro razmisliti, kako se bodo podatki ustrezno odstranili iz podatkovne zbirke. Sprva smo načrtovali, da bi ročno odstranjevali podatke po časovnih obdobjih, ko jih ne bomo rabili več ampak v izogib ročnemu delu smo vzpostavili avtomatski mehanizem, ki sem briše podatke, ko so te starejše od določenega datuma.

3.4. Sprememba podatkov

Zaradi izvajanja sprememb podatkov, kar odstopa od običajnih praks v okviru časovnih vrst, nismo uspeli najti ustreznih spletnih virov za uspešno izvedbo procesa sprememb (update) podatkov. Zato smo bili primorani izvesti testiranja delovanja podatkovne zbirke, da bi prepoznali ustrezne, neoptimalne in neprimerne funkcionalnosti. Proces spreminjanja podatkov smo začeli tako, da smo najprej pobrisali podatke, ki smo jih morali spremeniti in jih ponovno vstavili v podatkovno zbirko. Ta metoda je dokaj hitra ker jo omejimo, na časovni okvir v katerega se

zgodijo spremembe vhodnih podatkov in sicer 40 dni za nazaj. Metoda, ni optimalna ne tehnično optimizirana ampak v našem primeru zadošča glede načina uporabe podatkovne zbirke in je v skladu z izvornem sistemu na Elektro Ljubljana, ki je vir za naše podatke časovnih vrst.

3.5. Poizvedovanje in število povezav do podatkovne zbirke

Uporabe podatkovne zbirke je razvidna preko uporabe poizvedb v specifičnem časovnem okvirju. Število povezav je odvisno od število posameznih uporabnikov oz. poizvedb, ki se izvedejo teh od njihove kompleksnosti in trajanja.

Največkrat poizvedovanje se zgodi po potrebi in sicer tako da ga sproži uporabnik sam (on demand oz. ad-hoc). Običajno je govora o poizvedovanju večkrat na dan za eno specifično merilno mesto ali manjše/večje skupine merilnih mest, ki imajo določene skupne lastnosti. Pri večjih na primer pri letnih oz. večletnih poizvedb na majhno skupino merilnih mest običajno gre za poizvedovanje, ki je specifično/optimizirano za časovne vrste.

Na primer ko poizvedujemo meritve za število dni (1000) in skupino merilnih mest MM (100) in izračunamo število podatkov, ki jih pri povpraševanju lahko dobimo v matriki merilnih podatkov (glede na 15 min frekvenco podatkov tokov in energije):

$$(1000 \times 96 \times 100) \times 4 + (1000 \times 120 \times 100) \times 3 = 100.000 \times (4 \times 96 + 120 \times 3) = 74.400.000$$

kar pomeni poizvedba po 74,4 milijonov podatkov. Tako poizvedovanje v klasičnih relacijskih baz običajno traja neprimerno dolgo.

Testi so ponovno pokazali večjo prednost na strani ClikHouse v povprečju je čas trajanje poizvedb bil za 3.5 krat hitrejši na ClikHouse kot na TimescaleDB za enak obseg podatkov, ki so v bazah shranjene v podobni strukturi, ki izkorišča posamezne bazne karakteristike.

V primeru ko hočemo pisati kompleksne poizvedbe, ki grupirajo podatke in izvajajo agregat operacije nad njimi pa hitro TimescaleDB pridobi v prednosti. Prednost je večja takoj ko se začne z uporabo specifičnih funkcij za TimescaleDB kot so time_bucket, last, first v kombinaciji z relacijskimi funkcijami.

3.6. Zaklepanja na podatkovni zbirki

Pri večjih spremembah podatkov v podatkovni zbirki se lahko hitro zgodi, da prihaja do zaklepanja objektov podatkovne zbirke. V standardnih relacijskih bazah se to pogosto zgodi, na primer če se posamezni SQL-i izvajajo več kot nekaj milisekund. Dlje traja posamezen SQL, večja je verjetnost, da bo prišlo do medsebojnega zaklepanja objektov na podatkovni zbirki.

ClickHouse nima vidno na uporabnika zaklepanje. Uporablja asinhrone operacije i notranje mehanizme za konkurente dostope.

TimescaleDB ima to prednost nad PostgreSQL da uporablja hypertable, ki so narejeni da izvajajo particionairanje in s tem v osnovi preprečijo zaklepanje.

3.7. Particioniranje

Največja tabela v našem okolju je velika 800 GB (130 bilijon zapisov), kar predstavlja podatke energij za vseh merilnih mest na Elektro Ljubljana za obdobje pet pol. Vsak dan v podatkovni zbirki nastane dodatne 150 milijonov zapisov. ClickHouse uporablja MergeTree engine in s tem zagotavlja particioniranje. To je proces, kjer logično tabelo razdelimo na več fizičnih tabel. S stališča aplikacije obstaja ena tabela, fizično pa jih je lahko več

tisoč. Pomembno je, da pravilno nastavimo »particioniranje« tabele, ki se pri podatkih časovne vrste praviloma nastavi na časovno značko nastanka dogodka.

TimescaleDB nujno rabi, da niso ti podatki v eni sami veliki tabeli, ker bi SQL-i delovali občutno prepočasi ter nujno rabi dodatne mehanizme. Zato se za vsako tabelo ustvari indeks za pohitritev iskanja podatkov in ta indeks ne sme biti prevelik, da bi deloval optimalno. Posamezna TimescaleDB particionirana tabela je pomembno da ni kljub temu prevelika. Potrebno je izračunati priporočene velikosti particionirane tabele glede na frekvence polnjenja podatkov. Uporabili smo razmejitev tako, da so za vsak dan podatki v svoji particijski tabeli in na podlagi tipa meritev (energija ali napetost).

3.8. Indeksiranje

Z vidika uporabnika TimescaleDB izpostavlja objekte, ki izgledajo kot klasične tabele, imenujejo pa se hipertabele (ang. hypertables). Pravzaprav so te abstrakcije ali virtualni pogled na številne manjše tabele, ki vsebujejo podatke, imenovane kosi (ang. chunks). Kosi so ustvarjeni z razdelitvijo podatkov hipertabele na eno ali več dimenzij. Vse hipertabele so razdeljene glede na časovni interval, dodatno pa jih lahko delimo še z izbiro ključa, npr. ID naprave, lokacija, ID uporabnika, itd. Temu rečemo porazdelitev glede na čas in prostor (ang. Partitioning across time and space).

ClickHouse uporabljajo malo drugačen pristop baziran na mehanizmov MergeTree, ki so bili zasnovani in optimizirani za obdelavo ogromnih količin podatkov. Podatki se hitro zapisujejo v tabelo po chunkih, pri čemer se v ozadju uporabljajo pravila za združevanje chunk (delov). V ClickHouseu ima vsak chunk svoj primarni indeks. Ko se chunk izdružijo, se združijo tudi primarni indeksi združenih chunkov. Pri zelo velikem obsegu, za katerega je zasnovan ClickHouse, je ključnega pomena velika učinkovitost diska in pomnilnika. Zato namesto indeksiranja vsake vrstice ima primarni indeks za del en indeksni vnos (znan kot »oznaka«) na skupino vrstic (imenovano »granula«) – ta tehnika se imenuje redko indeksiranje.

TimescaleDB pridobi znatno z uporabo indeksov medtem ko (presenetljivo) ClickHouse zgubi pri performansah po dodajanju indeksov.

4 Zaključek

Uporaba IoT naprav in pametnih števcev je v celotnem energetskem sektorju (in širše) povzročila ekspanzijo podatkov, ki se jih lahko pridobiva v skoraj realnem času iz različnih merilnih naprav, ki so na voljo in so instalirani neposredno pri končnih odjemalcih in/ali na samem omrežju oziroma v našem primeru na distribucijskem energetskem omrežju DO, ki je v upravljanju na Elektro Ljubljana.

S stališča informacijsko-komunikacijskih tehnologij (IKT) je izziv predvsem v zagotavljanju ustrezne infrastrukture za zbiranje, prenos, shranjevanje in obdelavo ogromnih količin podatkov, kar zahteva uporabo naprednih metod strojnega učenja, podatkovnega rudarjenja ter orodij za analitiko velikih podatkov (big data) [5]. S tem namenom smo na Elektro Ljubljana preizkusili načrtovanje in uporabo odprtokodnih baz časovnih serij in jih primerjali med sabo. V predstavljam članku smo povzeli naše izkušnje ter poudarili prednosti in slabosti uporabe in umeščenosti v naši IKT podatkovni in sistemski infrastrukturi. Rezultat preizkusa je v prednosti uporabe ClickHouse baze.

IoT naprave še naprej bodo naš ključni element podatkovni in sistemski infrastrukturi. Njihov razvoj in razvoj tehnologij, ki jih podpirajo spreminja in nadgrajuje digitalne transformacije celotnega energetskega sektorja in predstavljajo osnovo za nadaljnji razvoj inteligentnih in odzivnih elektroenergetskih sistemov. Na Elektro Ljubljana sledimo tem razvojem in novitete vpeljujemo v naše procese in postopke dela na vse področjih.

Literatura

[1] Gungor, V. C., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C., & Hancke, G. P. (2011). Smart grid technologies: Communication technologies and standards. IEEE Transactions on Industrial Informatics, 7(4), 529–539.

[2] Mohassel, R. R., Fung, A., Mohammadi, F., & Raahemifar, K. (2014). A survey on advanced metering infrastructure. International Journal of Electrical Power & Energy Systems, 63, 473–484.

[3] Amin, S. M., & Wollenberg, B. F. (2005). Toward a smart grid: Power delivery for the 21st century. IEEE Power and Energy Magazine, 3(5), 34–41.

[4] Fang, X., Misra, S., Xue, G., & Yang, D. (2012). Smart grid – The new and improved power grid: A survey. IEEE Communications Surveys & Tutorials, 14(4), 944–980.

[5] Zhou, K., Fu, C., & Yang, S. (2016). Big data driven smart energy management: From big data to big insights. Renewable and Sustainable Energy Reviews, 56, 215–225.

[6] NOVAK, Matic, 2019, Uporaba tehnologije NoSQL za učinkovitejše izračunavanje regulatornih poročil v zavarovalništvu [na spletu]. 2019. [Dostopano 25 oktober 2022]. Pridobljeno https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=110061

Poslovna inteligenca v podporo odločanju na primeru upravnih postopkov

Paula Kolenko[1], Alenka Krebs[1], Karmen Kern Pipan[1], Gabriela Weiss Živič[1], Mirko Stopar[2], Iztok Peroša[2]

[1] Ministrstvo za digitalno preobrazbo Republike Slovenije, Ljubljana, Slovenija
paula.kolenko@gov.si, alenka.krebs@gov.si, karmen.kern-pipan@gov.si, gabriela.weiss-zivic@gov.si

[2] Ministrstvo za javno upravo Republike Slovenije, Ljubljana, Slovenija
Mirko.stopar@gov.si, iztok.perosa@gov.si

Prispevek prikazuje izkušnje in dobre prakse poslovno inteligenčnega sistema Skrinja na primeru upravnih postopkov, ki jih izvajajo Upravne enote v Sloveniji. Sistem Skrinja je horizontalna platforma za organe državne uprave, ki vključuje podatkovno skladišče s poslovno inteligenčnim sistemom in omogoča analitikom analitičen vpogled v podatke na določenem področju in pridobivanje informacij v realnem času, odločevalcem pa nudi informacije o kazalnikih uspeha za odločanje na podlagi podatkov (Data-driven decision making). Podatkovni vir Upravni postopki Upravnih enot uporablja vodstvo Ministrstva za javno upravo (MJU) in načelniki Upravnih enot z vodji notranjih organizacijskih enot. Podatki se dnevno osvežujejo iz dokumentnega sistema Krpan v sistem Skrinja, zato sistem Skrinja na enem mestu nudi uporabnikom vpogled v sveže podatke, ki jih lahko uporabijo za različne potrebe, pripravo statistik, analiz in poročil za svoje deležnike in v podporo odločanju. Z uporabo sistema Skrinja se je povečala tudi transparentnost ter poenostavil, avtomatiziral in optimiziral proces poročanja in obdelave podatkov na upravnih enotah.

Ključne besede:

poslovna inteligenca,
podatkovna analitika,
horizontalna platforma,
odločanje,
transparentnost.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.10

1 Uvod

Sistem Skrinja – poslovna inteligenca je horizontalna platforma za organe državne uprave, ki vključuje podatkovno skladišče in poslovno analitiko in deluje na Državni informacijski infrastrukturi. Platforma je bila razvita na Ministrstvu za digitalno preobrazbo v letu 2020, ko se je začela vpeljava organov v sistem in s tem krepitev analitične organizacijske kulture kot vir zaupanja vrednih statističnih podatkov, potrebnih za proces odločanja.

Z uvedbo poslovno inteligenčnega sistema Skrinja v delovanje posameznega organa, sistem Skrinja uporabnikom omogoči, da podatke spremljajo in analizirajo na enem mestu in v realnem času, vodstvu pa omogoča dnevni vpogled v podatke ključnih kazalnikov in nudi podporo odločanju na podlagi podatkov.

2 Sistem Skrinja

2.1. Sistem Skrinja kot horizontalna platforma za organe državne uprave

V državni upravi imamo veliko količino podatkov v različnih podatkovnih zbirkah, ki pa med seboj niso povezane in zato je tudi poročanje oteženo. Upravljalci podatkov različnih podatkovnih zbirk v državni upravi – analitiki, imajo veliko dela s tem, da podatke zberejo iz različnih virov, jih analizirajo in sestavijo vsa potrebna poročila. To je časovno zelo potratno, saj se analitiki ukvarjajo v glavnem z zbiranjem in s čiščenjem podatkov z namenom priprave potrebnih analitičnih poročil. S sistemom Skrinja smo uporabnikom želeli omogočiti avtomatizacijo zajema in čiščenja podatkov in vpogled v podatke v realnem času, da bi se lahko bolj posvetili analizam in interpretaciji podatkov. Na ta način smo omogočili tudi transparenten vpogled v podatke kot tudi boljšo osnovo za odločanje na podlagi podatkov (Data-driven decision making).

V sistem Skrinja iz izvorne aplikacije prenesemo le tiste podatke, ki dajejo odgovor na analitična vprašanja, ki jih upravljalec podatkovnega vira pri tem potrebuje [1]. Do teh odgovorov pridemo z analitičnimi tehnikami, ki jih orodja poslovne inteligence omogočajo. Po ureditvi strukturiranih podatkov v področnem podatkovnem skladišču (analitičnem zvezdnem modelu oziroma shemi z opredeljenimi merami, dimenzijami in dejstvi) in pripravi multidimenzionalnih podatkovnih modelov (kock), ki omogočajo hitro obdelavo in prikaz vizualiziranih podatkov, upravljalec podatkovnega vira preveri pravilnost rezultatov v sistemu Skrinja in jih interpretira. Sistem Skrinja podatkov ne spreminja, temveč jih z uporabo različnih analiz strukturirano prikazuje kot informacije. Vizualizacija je pri tem močno orodje.

Z razvojem sistema Skrinje kot horizontalne platforme za organe državne uprave smo organom omogočili, da nepovezani podatkovni viri z novim znanjem postanejo bolj pregledne in celoviteje dostopne informacije za odločanje tako, da iz različnih virov na relativno enostaven način uporabniki pridobijo kakovostne informacije za boljše odločanje.

2.2. Razvoj sistema

Prve korake k razvoju sistema Skrinja smo naredili na Ministrstvu za digitalno preobrazbo, prej Ministrstvu za javno upravo, ko smo pristopili k razvoju pilotnega projekta »Big data pilot«, katerega izkušnje smo uporabili pri vzpostavitvi »Koncepta poslovne inteligence« za razvoj sistema Skrinja. V nadaljevanju smo koncept predstavili Informacijskem pooblaščencu, saj gre tudi za obdelavo osebnih podatkov. Izdelali smo Oceno učinkov za obdelavo osebnih podatkov in pridobili pozitivno mnenje Informacijskega pooblaščenca za obdelavo osebnih – psevdonimiziranih podatkov, s čimer smo lahko začeli s postavitvijo ustrezne infrastrukture. V letu 2020 smo v sistem uvedli prvi podatkovni vir Plače v javnem sektorju (preko 180.000 javnih uslužbencev, preko 2000 proračunskih uporabnikov, preko 750 vrst izplačil), v letu 2021 pa Oddana javna naročila v Republiki Sloveniji (letno presega 5 MLD EUR oz. preko 11 % BDP). Od leta 2023 dalje so v sistem vključeni podatki Upravnih

postopkov Upravnih enot (1 MIO dokumentov letno), zanimanje za vključitev v sistem Skrinja v organih državne uprave pa se povečuje.

2.3. Vidiki poslovne inteligence

Pri razvoju poslovne analitike se srečujemo s tremi vidiki: podatkovni, analitični in človeški vidik. BI ni zbiranje podatkov, ampak avtomatizirana obdelava podatkov za podporo odločanju, s čimer želimo uporabnikom ponuditi okolje, v katerem bodo lahko na uporabniškem nivoju pregledovali podatke, pa tudi razvijali lastne analize. Vsi trije vidiki se močno prepletajo skozi vse faze razvoja.

Podatkovni vidik: v državni upravi so podatki shranjeni v več podatkovnih zbirkah, podatke je za potrebne poslovne analitike potrebno ustrezno pripraviti in oblikovati podatkovno skladišče, ki služi kot vir, nad katerim izvajamo analitiko. Velik izziv je kakovost podatkov, zato je potrebno pri vsakem viru sprejeti ukrepe, ki kakovost povečujejo in ukrepe za čiščenje podatkov. Pri teh postopkih sodelujejo tisti, ki podatke poznajo in jih zbirajo in tako lahko odpravijo napake na samem izvoru.

Analitični vidik: Za pripravo analitičnih poročil, ki bodo tudi preprosta za uporabo, je potrebno zajeti čim širši obseg potreb s podatki, uporabniška izkušnja mora biti vrhunska, intuitivna. Tehnologija mora poskrbeti za pripravo kakovostnih podatkov, analitiki pa se morajo ukvarjati predvsem z interpretacijo podatkov. Prav tako je potrebno poskrbeti za verodostojnost rezultatov analiz, kar pa dosežemo predvsem s temeljitim testiranjem pri poznavalcih vsebine podatkov. Izkušnje pravijo, da je nemogoče vnaprej zajeti prav vse potrebe, zato je v sistemu Skrinja, poleg avtomatiziranih poročil, na voljo tudi okolje za ad hoc analitiko.

Človeški vidik: V procesu odločanja in razvoja sistema so vključeni odločevalci, analitiki, podatkovni arhitekti in poznavalci oz. skrbniki zbirk podatkov. Vloga odločevalcev je, da razumejo pomen podatkovnega skladišča in znajo uporabljati analitska orodja do te mere, da lahko kakovostno izražajo zahteve. Znanja analitikov se sicer po organih razlikujejo, vendar je ključno, da analitik pozna podatke in obvlada koncept analize in vizualizacije, ki jih lahko kar najbolje predstavi podatkovnem arhitektu in razvijalcu poročil za ustrezno pripravo podatkovnega vira, modeliranja podatkov, priprave in nadgradnje podatkovnih skladišč, priprave analitičnih modelov in vizualizacij.

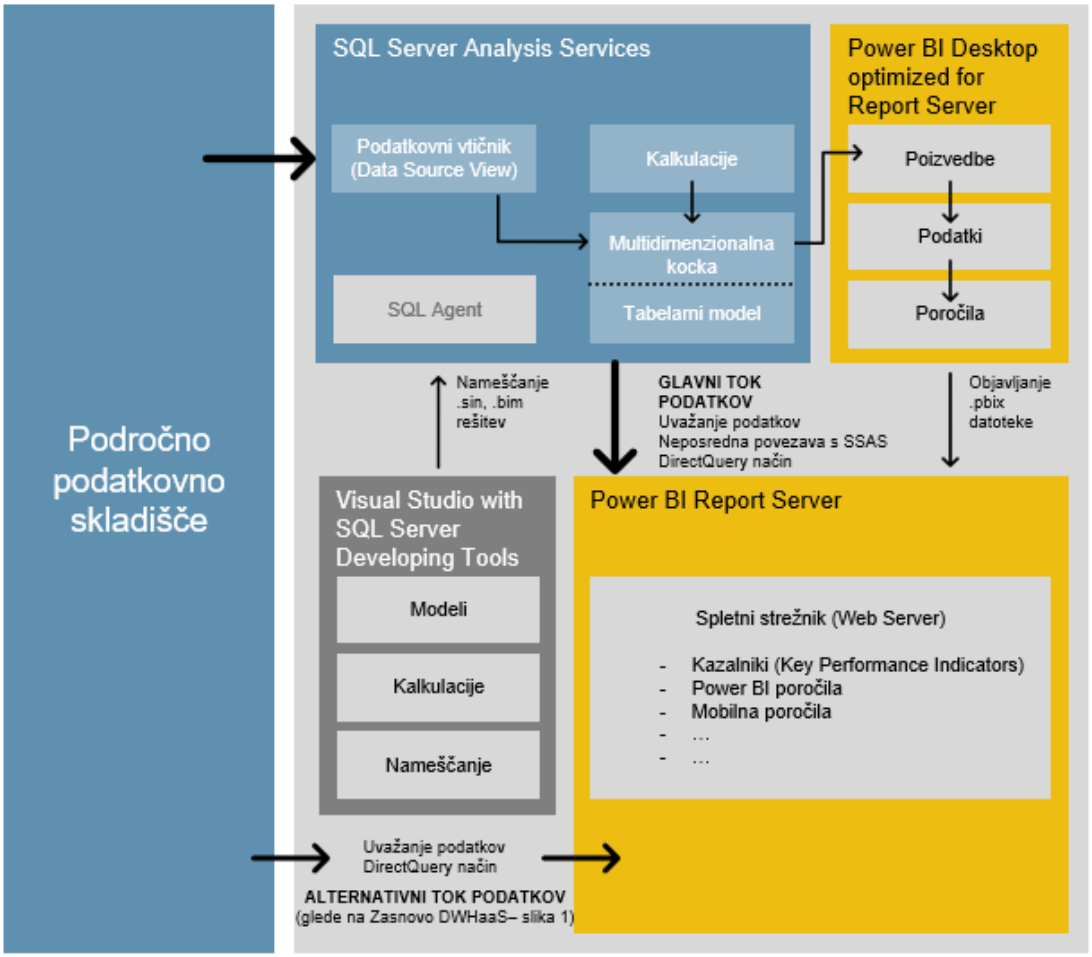
2.4. Logična arhitektura sistema Skrinja

Distribucijsko okolje podatkovnega vira

Upravljalci podatkov nekega vsebinskega področja zbirajo, urejajo in obdelujejo podatke v svojem produkcijskem okolju oz. aplikaciji. Podatki so v strukturah, za katere skrbi znan upravljalec in pokrivajo neko vsebinsko področje. Za namen izvoza podatkov v okolje Skrinja morajo upravljalci posameznih virov organizirati svoja distribucijska (ločena) okolja, kamor odložijo podatke za namen prenosa v sistem Skrinja. V okviru zajema uporabniških zahtev ob uvedbi podatkovnega vira v sistem Skrinja se določi nabor podatkov, ki je potreben za izdelavo analitičnih poročil. Upravljalec podatkov (aplikacije) te podatke prenese v distribucijsko okolje, kjer so na voljo sistemu Skrinja, da te podatke prevzame. V primeru, da vir vsebuje osebne podatke, mora le-te ustrezno zakriti oz. psevdonimizirati in takšne odložiti v distribucijsko okolje.

Podatkovno skladišče

Vsakemu viru v okviru sistema Skrinja je dodeljeno ločeno vsebinsko področno podatkovno skladišče (Data Mart). Vanj se (preko prehodnega okolja) pretakajo podatki iz distribucijskega okolja posameznega podatkovnega vira z uporabo avtomatiziranega ETL postopka (Extract, Transform, Load). Podatki se pretakajo na osnovi dogovorjenih pravil, ki so določena ob postavitvi podatkovnega modela in se tako ustrezno preverijo in transformirajo. Za vsak podatkovni vir je določen urnik dnevnega prenosa podatkov.



Slika 1: Logična arhitektura sistema Skrinja. [1]

Multidimenzionalni podatkovni model

Vsakemu področnemu podatkovnemu skladišču je dodeljeno ločeno BI okolje za razvoj in uporabo poročil. Za poslovno analitiko v sistemu uporabljamo analitični podatkovni strežnik Analysis Services (v nadaljevanju SSAS). Ta zagotavlja izvedbo poslovnih semantičnih multidimenzionalnih podatkovnih modelov za prikaz nadzornih plošč in poročil.

Podatkovni model je vmesna shramba podatkov, ki je poleg razdelitve na dejstva in dimenzije še dodatno obdelana tako, da so odzivi na želje uporabnikov čim krajši. Tako so na primer že izračunane izvedene vrednosti ter delni seštevki glede na hierarhije dimenzij in se pri pregledovanju takoj prikažejo. Uporabniki - napredni analitiki lahko direktno dostopajo do multidimenzionalnega podatkovnega modela in s tem možnost samostojnega oblikovanja novih poizvedb in priprave poročil.

Nadzorne plošče in poročila

Končna poročila z vizualizacijami so prikazana na nadzornih ploščah v okviru Power BI Report Server (PBRS), ki je strežnik z vključenim spletnim portalom in omogoča spletno prikazovanje in upravljanje poročil, nadzornih in vodstvenih plošč s kazalniki uspeha (KPI). Do izdelanih poročil, plošč in KPI-jev lahko upravljalec podatkovnega vira dostopa v spletnem brskalniku in na mobilni napravi in odobri dostop tudi drugim uporabnikom.

Skupne dimenzije - šifranti

V okviru sistema Skrinja je ločeno podatkovno skladišče Skupne dimenzije - šifranti, za katerega skrbi MDP kot upravljalec skupnih dimenzij. Skupne dimenzije vsebujejo javno dostopne in splošno veljavne šifrante, ki se vedno pridobivajo iz izvornega, primarnega vira. Namenjene so uporabi s strani več področnih podatkovnih skladišč in ne vsebujejo osebnih podatkov.

Urnik osveževanja podatkov

V sistem Skrinja so predvideni dnevni, tedenski in letni prenosi podatkov iz primarnih podatkovnih virov, odvisno od potreb podatkovnega vira, ter avtomatsko osveževanje podatkovnih kock. O urniku prenosa (Data scheduler) se s podatkovnim virom dogovorimo pred inicialnim polnjenjem področnega podatkovnega skladišča.

2.5. Uvedba podatkovnega vira v sistem Skrinja

Proces uvajanja novega vira v sistem Skrinja vključuje več pomembnih mejnikov. Uvedbo vira v sistem vodi MDP, kot upravljalec sistema Skrinja in pri tem vodi, razvija in koordinira delovanje, nadgradnje in vzdrževanje sistema, koordinira medsebojne dogovore z vsemi nalogami, pristojnostmi in odgovornostmi vseh vključenih deležnikov, razvija in podpira metodologijo in tehnično uvedbo, upravlja proces uvedbe in delovanja za uporabnika, skrbi za vodenje, tehnično podporo, prenos znanja in izobraževanje uporabnikov, skrbi za pripravo in implementacijo sistemskih tehničnih pravil, vzdržuje in upravlja skupne šifrante.

Skozi celoten proces uvedbe vira v sistem Skrinja je ključnega pomena aktivno sodelovanje vsebinskih sodelavcev oz. upravljalcev podatkovnega vira (lastnikov podatkov) z ekipo sistema Skrinja. Sodelovanje se začne z zajemom zahtev uporabnikov, postavljanjem poslovnih pravil in izračunov, ki so skladni z metodologijo vira, čiščenja podatkov (v izvorni aplikaciji), testiranja pravilnosti podatkov in končne vsebinske potrditve rešitve za produkcijsko uporabo. Hitrost uvedbe podatkovnega vira v sistem je tako odvisen predvsem od aktivnosti in sodelovanja uporabnikov na organu.

Ocena učinkov v zvezi z varstvom osebnih podatkov

Za sistem Skrinja smo na MDP izdelali splošno Oceno učinkov v zvezi z varstvom osebnih podatkov (DPIA), v kateri je bila kot pogoj za vključitev virov v storitev Skrinja določena obveznost priprave DPIA za vsak podatkovni vir, kjer se obdelujejo osebni podatki. S tem je bil verificiran koncept sistema Skrinja.

Ob vključitvi vsakega naslednjega podatkovnega vira v sistem Skrinja je potrebno najprej preveriti, ali slednji vsebuje osebne podatke in skladno s tem pripraviti dokument »Ocena učinkov«.

Zajem uporabniških zahtev

Uvedbo podatkovnega vira v sistem Skrinja začnemo z nizom sestankov (intervjujev) za zajem uporabniških zahtev in oblikovanje dokumenta »Zajem uporabniških zahtev za podatkovni vir…«. Dokument pripravljajo vsebinski upravljalci podatkovnega vira ob podpori upravljalcev sistema Skrinja in arhitektov sistema. V okviru priprave dokumenta potekajo intervjuji z vsebinskimi upravljalci podatkovnega vira, ki jih vodi in izvaja arhitekt sistema Skrinja.

Razvoj – podatkovno skladišče in BI

V tej fazi poteka razvoj z zunanjimi strokovnjaki - podatkovnimi arhitekti, strokovnjaki za razvoj podatkovnega skladišča, razvijalci in vzdrževalci ETL procesov, strokovnjaki za razvoj poslovne analitike in poslovnimi analitiki za poročila, nadzorne plošče in vizualizacijo. Rezultat razvoja so BI nadzorne plošče, postavljene v predprodukcijskem okolju in na voljo uporabnikom za testiranje in validacijo podatkov.

Testiranje in validacija podatkov

Testiranje poteka v testnem in uvajalnem ali predprodukcijskem okolju. Po uspešno izvedenem testiranju v testnem okolju (tehnično testiranje), se testiranje nadaljuje na uvajalnem ali predprodukcijskem okolju (vsebinsko testiranje).

Uporabnik (upravljalec podatkovnega vira) testira vsebino – pravilnost pripravljenih izračunov, tehnična ekipa sistema Skrinja (upravljalec sistema) pa dostope, uporabniške pravice, odzivnost in obremenitve. Vsebinsko testiranje v omrežju HKOM poteka z uporabo AD uporabniškega računa in na realnih podatkih.

Usposabljanja

Del uvedbe novega podatkovnega vira v sistem je tudi osnovno usposabljanje za ključne uporabnike novo uvedenega podatkovnega vira v sistem Skrinja in v nadaljevanju tudi obdobna uvajanja za nove uporabnike. Uporabniki se usposobijo za samostojno uporabo sistema Skrinja, predvsem vizualizacij standardiziranih poročil in vodstvene plošče in uporabo funkcionalnosti (filtri, puščice, izvozi, sortiranja, dodatne funkcionalnosti, ki so značilne za posamezen podatkovni vir).

2.6. Nadaljnji razvoj sistema Skrinja

V sistem Skrinja so vključeni štirje podatkovni viri: Plače v javnem sektorju, Oddana javna naročila v Republiki Sloveniji, Skupne dimenzije in Upravni postopki Upravnih enot. Pri vseh virih v produkciji potekajo nadgradnje, saj so se tekom razvoja pokazale dodatne zahteve s strani uporabnikov – upravljalcev podatkovnih virov, kar kaže na to, da so uporabniki prepoznali moč analitike tako za pripravo podatkov, analiz in poročil, kot za odločanje na podlagi podatkov.

V razvoju in načrtovani za produkcijsko delovanje so novi podatkovni viri: Centralna kadrovska evidenca organov državne uprave Ministrstva za javno upravo, Inšpekcijski postopki Tržnega inšpektorata Republike Slovenije, Inšpekcijski postopki Zdravstvenega inšpektorata Republike Slovenije, podatki Službe Vlade za obnovo po poplavah in plazovih, podatki Skupne kmetijske politike Ministrstva za kmetijstvo, gozdarstvo in prehrano, podatki javnih razpisov in pozivov Ministrstva za kulturo in podatki Službe Vlade za zakonodajo. Za vključitev v sistem Skrinja želimo pritegniti predvsem podatkovne vire s podatki, ki so pomembni za pridobivanje ključnih informacij za učinkovito vodenje države.

Sistem Skrinja bomo nadgradili z uvedbo algoritmičnih orodij in napovedne analitike, predvidoma nad podatki Oddanih javnih naročil v Republiki Sloveniji in s podatki Skupne kmetijske politike Ministrstva za kmetijstvo, gozdarstvo in prehrano.

3 Primer dobre prakse Upravni postopki Upravnih enot

3.1. Namen vključitve v sistem Skrinja

Na Ministrstvu za javno upravo (MJU) smo se, skladno s sodobnimi smernicami poslovanja, odločili za pospešen razvoj podatkovne analitike v podporo odločanju na podlagi podatkov za področje upravnih enot, tako za Službo za upravne enote (SUE) kot za odločevalce in vodstvo upravnih enot (načelniki). S tem smo želeli izboljšati in optimizirati sisteme upravljanja s podatki, poročanja in odločanja na vseh ravneh ministrstva.

Pred vključitvijo podatkovnega vira Upravni postopki Upravnih enot (UPP) v sistem Skrinja, so bila vodstvu MJU na voljo letna poročila o izvajanju upravnih postopkov na upravnih enotah le enkrat letno za preteklo (zaključeno) leto. Na podlagi takšnih poročil je bilo sprejemanje sprotnih odločitev za optimizacijo obremenjenosti upravnih enot močno oteženo. Tako smo se dolgo soočali z razdrobljenostjo podatkov, ki so bili shranjeni v različnih, med seboj nepovezanih podatkovnih bazah, pri tem pa je prihajalo do neučinkovitega poročanja, povečanega administrativnega bremena in posledično oteženega odločanja in šibkejšega strateškega načrtovanja.

Odgovor na te izzive smo prepoznali v vključitvi v sistem Skrinja, ki na enem mestu združuje podatke, jih povezuje in pretvarja v uporabno znanje. Z vključitvijo podatkovnega vira UPP v sistem Skrinja je dosežen namen hitrega poročanja in izdelave analitike na realnih in svežih podatkih in posledično boljše in učinkovitejše odločitve o porabi javnih sredstev in razporejanju dela na podlagi relevantnih, kvalitetnih podatkov. To pomeni tudi hitrejši odziv na razna kritična stanja, odpravljanje anomalij in napak pri odločanju, ter učinkovitejšo pripravo informacij. S tem bo v prihodnje izraba časa zaposlenih optimalnejša, povečalo se bo zaupanje v podatke in posledično bodo mogoče morebitne izboljšave zakonodaje in sistema.

3.2. Podatki iz dokumentnega sistema Krpan

Podatkovni vir UPP obsega podatke o zadevah, dokumentih in dogodkih, ki jih rešujejo upravne enote in so vezani na reševanje zadev po Zakonu o splošnem upravnem postopku (ZUP) in zadev drugih upravnih nalog (DUN) in jih uslužbenci na upravnih enotah zbirajo v dokumentnem sistemu Krpan.

Informacijski sistem Krpan je centralna spletna aplikacija, nameščena na Državni informacijski infrastrukturi in je poenotena informacijska rešitev za zaposlene v organih državne uprave. Podpira evidentiranje in vodenje splošnih in upravnih zadev ter dokumentnih seznamov, nudi podporo delu z e-računi in ostalimi finančno računovodskimi dokumenti.

Iz dokumentnega sistema Krpan se podatki, preko vmesnega oz. distribucijskega okolja, dnevno prenašajo v sistem Skrinja. Prenašajo se le tisti podatki, ki so bili v procesu zajema uporabniških zahtev za uvedbo podatkovnega vira Upravni postopki Upravnih enot prepoznani kot potrebni za prikaz poročil in analiz za dnevno spremljanje, analiziranje in poročanje oz. odgovore na najpomembnejša vprašanja. Odločevalcem, analitikom in drugim uporabnikom so tako dnevno na voljo sveži in realni podatki (podatki preteklega dne) za pripravo analiz in poročil ter pregled stanja reševanja zadev. Podatkovni vir ne vsebuje podatkov o zaposlenih na upravnih enotah in ne vsebuje osebnih podatkov.

3.3. Izkušnje v procesu razvoja

Vsebinski nosilec podatkovnega vira UPP je Služba za upravne enote (SUE) Ministrstva za javno upravo, ki je tudi glavni vsebinski uporabnik in sogovornik razvijalcem za vključitev podatkovnega vira v sistem Skrinja.

Prvi in zelo pomemben korak uvedbe vira v sistem je zajem zahtev uporabnikov, ki ga vodi zunanji izvajalec za razvoj sistema Skrinja in poteka v obliki delavnic z vsebinskimi nosilci, kjer se razjasnijo potrebe uporabnikov in opredeli vsebina in vizualizacije za končni rezultat – standardizirana poročila. Kot osnovo smo v SUE pripravili 10 različnih poročil, ki jih kot analitiki največkrat uporabljamo pri svojem delu in so podlaga za analitiko. Pripravili smo tudi vprašanja, na katera odgovarjamo različnim deležnikov (odločevalci, novinarji, javnost). Rezultat te faze je bil izdelan dokument s popisom vseh zahtev, podatkov, atributov in ključnih kazalnikov ter poslovnih pravil, kar je podlaga za nadaljnje delo razvijalcev – za izdelavo podatkovnega in analitičnega modela.

Sledila je faza razvoja podatkovnega in analitičnega modela, kjer smo razvijalcem sprotno odgovarjali na dodatna zastavljena vprašanja, da so lahko pripravili modele in servise za zajem podatkov v sistem Skrinja. Po ustreznem prenosu podatkov iz aplikacije Krpan v podatkovno skladišče sistema Skrinja, je bil razvit analitični model (SSAS) in pripravljene nadzorne plošče z vizualizacijami.

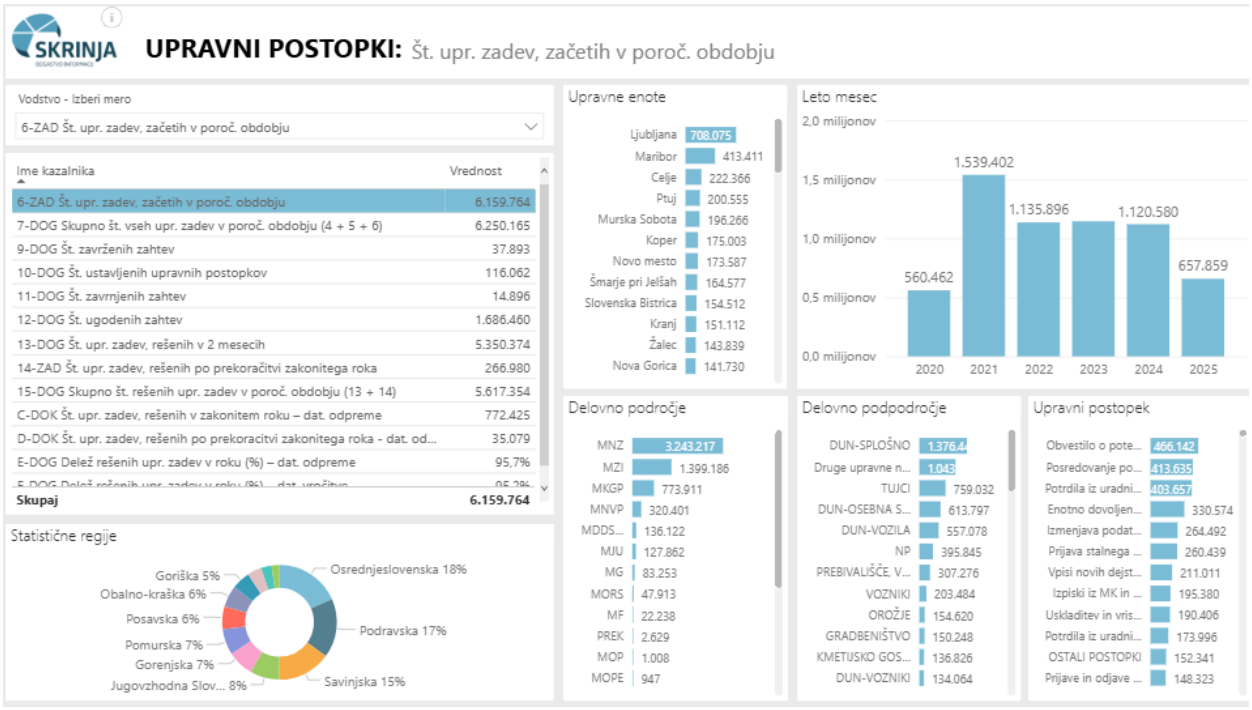
Po tehnični postavitvi sistema, smo začeli z vsebinskim testiranjem in pri tem vključili tudi skupino načelnikov upravnih enot, saj ti najbolj poznajo podatke in vsebino poročil. Pokazalo se je, da je vsebinsko testiranje velik izziv, ki vključuje tudi čiščenje podatkov in popravke podatkov na izvoru – v dokumentnem sistemu Krpan. Sistem Skrinja namreč podatke le prikazuje in analizira, vse anomalije v podatkih, pa je potrebno popraviti na izvoru. V nekaj primerih smo postavili tudi nova pravila vnosa v dokumentni sistem Krpan, s tem pa izboljšali kvaliteto podatkov.

3.4. Primer uporabe sistema Skrinja

V juliju 2024, po uspešni validaciji podatkov, smo dodelili dostop do podatkov vsem načelnikom upravnih enot in vodjem notranjih organizacijskih enot, ki so se udeležili več usposabljanj za uporabo sistema in delom z orodjem Power BI. Danes vir UPP uporablja že preko 250 uporabnikov.

Dostop do podatkovnega vira UPP v sistemu Skrinja imajo uporabniki, ki jih odobrimo v SUE kot upravljalci podatkovnega vira. V sistemu je predvidenih več nivojev uporabnikov: vodje, napredni analitiki in analitiki.

Za najvišje vodstvo MJU je predviden dostop z vlogo »vodstvo«, ki dnevno pregleduje ključne kazalnike v okviru vodstvene plošče vira, uporabniki v vlogi »analitik« pregledujejo tudi podrobna standardizirana poročila, uporabniki v vlogi »napredni analitik« pa imajo dostop do analitičnega modela, kjer si lahko izdelajo tudi ad hoc poročila za primere, ki niso vnaprej predvideni.



Slika 2: Nadzorna vodstvena plošča Upravni postopki UE.

3.5. Dobra praksa v podporo odločanju

Podatki so v Skrinji na voljo v enotnem podatkovnem repozitoriju – podatkovnem skladišču in se iz podatkovnega sistema Krpan dnevno prenašajo v sistem Skrinja, kjer so uporabnikom na voljo preko nadzornih plošč v spletnem brskalniku. Nadzorne plošče prikazujejo poročila z vizualizacijami, kompleksni podatki so prikazani z enostavnimi vizualizacijami, ki so uporabnikom na voljo na klik. Napredni uporabniki analitiki pa si lahko s pomočjo teh podatkov lahko izdelajo interaktivna in mobilna poročila in what-if analize (možnost simulacije kompleksnejših scenarijev).

V obdobju enega leta, kar smo vključeni v sistem Skrinja, smo prepoznali konkretne učinke, ki se kažejo v poenostavljenem poročanju (ad hoc analize brez potrebe po tehnični podpori), večjemu prihranku časa in kadrovskih virov, manj ročnega vnašanja in posledično manj podvajanja podatkov in manjša možnost napak. V SUE lahko s pomočjo spremljanja teh kazalnikov pravočasno ukrepamo in prispevamo k natančnejšemu in bolj strateškemu načrtovanju kadrovskih potreb in porabe proračunskih sredstev.

3.6. Nadgradnja z vključitvijo novih podatkovnih baz

Podatkovni vir UPP v okviru sistema Skrinja je bil predan v redno delo uporabnikom v juliju 2024, ko je upravljavec vira Služba za upravne enote MJU potrdil pravilnost podatkov (validacija podatkov). V tem okviru so zajeti podatki iz dokumentnega sistema Krpan, začela pa se je že nadgradnja sistema za vključitev podatkov iz podatkovnih baz Ministrstva za notranje zadeve (MNZ) in Ministrstva za infrastrukturo (MZI) [2]. Tako bodo v podatkovnem viru UPP na enem mestu in v realnem času prikazani podatki o vseh upravnih postopkih, ki jih vodijo upravne enote.

4 Zaključek

Na MDP smo razvili enotno horizontalno platformo za državne organe, ki vključuje podatkovno skladišče s poslovno inteligenčnim sistemom in omogoča analitičen vpogled v podatke na določenem področju in pridobivanje informacij v realnem času. Gre za način pred pripravljenih poslovnih kazalnikov (nadzornih plošč, poročil), ki dnevno odražajo stanje podatkov za vse skupine uporabnikov od odločevalcev, do naprednih analitikov in končnih uporabnikov [3]. S tem se racionalizira poslovanje in zaposlenim omogoča, da se ukvarjajo z dejanskimi analizami podatkov in ne z njihovo pripravo. S tem pa se prvenstveno izboljšuje in optimizira sistem upravljanja s podatki, sistem poročanja in proces odločanja. V preteklih letih smo v produkcijsko delovanje in uporabo uvedli štiri podatkovne vire. Trenutno je v razvoju več podatkovnih virov, ki jih upravljajo različni državni organi, razvoj pa bomo nadaljevali z uvedbo napovedne analitike z uporabo algoritmičnih orodij.

Tudi z uporabo sistema na Upravnih enotah, se je izjemno povečala transparentnost ter poenostavil, avtomatiziral in optimiziral proces poročanja in obdelave podatkov za 58 upravnih enot, ki je prej potekal na ne avtomatiziran in zamuden način. Obenem se je zmanjšala obremenitev zaposlenih z rutinskim poročanjem ter omogočil takojšen dostop do podatkov ter hitra analitična obdelava le-teh.

Skrinja ni le zbirka podatkov ampak je digitalna opora državne uprave. Omogoča pametnejše odločanje, boljše javne storitve in večjo digitalizacijo.

Literatura

[1] Uvedba novega vira v sistem Skrinja,
https://nio.gov.si/products/skrinja%2B20%2Bsistem%2Bposlovne%2Banalitike?release=2.0, pridobljeno 1. 8. 2025

[2] Skrinja UPP Navodila za uporabo,
https://nio.gov.si/products/skrinja%2B20%2Bsistem%2Bposlovne%2Banalitike?release=2.0, pridobljeno 1. 8. 2025

[3] Splošni pogoji sistema Skrinja,
https://nio.gov.si/products/skrinja%2B20%2Bsistem%2Bposlovne%2Banalitike?release=2.0, pridobljeno 1. 8. 2025

DSX Engine: Decentraliziran povezovalnik podatkovnih prostorov

Martin Ferenec, Teo Lah, Muhamed Turkanović

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

Članek obravnava izzive suverene in varne med-organizacijske izmenjave podatkov, ki jih rešujejo podatkovni prostori. Temelji na evropski strategiji za podatke in standardu International Data Spaces Association, ki predvideva uporabo povezovalnikov kot ključnih tehničnih gradnikov. DSX Engine nadgrajuje referenčno implementacijo Eclipse Dataspace Components z mehanizmi za decentralizirano zaupanje: decentralizirane identitete, pametne pogodbe in podporo trgovanja podatkov s pomočjo žetonov ERC-20. S tem odpravlja osrednje točke odpovedi, avtomatizira sklenitev podatkovnih pogodb ter omogoča sledljiv, kriptografsko preverljiv ekosistem za deljenje in trgovanje podatkov. Funkcionalnost DSX Engine je potrjena v pilotnem podatkovnem prostoru DIH AGRIFOOD Data Space, ki povezuje kmetijske, okolijske in logistične podatke različnih deležnikov, hkrati pa ohranja njihovo suverenost. Članek prispeva metodološke in tehnične smernice za nadaljnji razvoj podatkovnih prostorov v industriji in javnem sektorju.

Ključne besede:

podatkovni prostori,
povezovalniki podatkovnih prostorov,
Eclipse Dataspace Components,
DSX Engine,
decentralizacija,
Web 3.0,
tehnologije veriženja blokov,
pametne pogodbe,
podatkovna suverenost,
varna, zaupanja vredna izmenjava podatkov.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.11

1 Uvod

V zadnjem desetletju smo bili priča premiku od paradigme masovnih podatkov k paradigmi zaupanja vrednih podatkov. Medtem ko so se prejšnje rešitve osredotočale predvsem na količino in hitrost obdelave podatkov, današnje pobude vse bolj poudarjajo kakovost, integriteto, sledljivost in predvsem zaupanje v podatke in njihove vire. Ta sprememba je tesno povezana z razvojem decentraliziranih tehnologij, ki odpravljajo odvisnost od enega ponudnika infrastrukture ter uvajajo preverljive, kriptografsko zaščitene mehanizme za upravljanje dostopa in uporabe podatkov. V sodobnem podatkovno vodenem gospodarstvu postajajo podatkovni prostori (angl. data spaces) ključni za varno in zanesljivo deljenje podatkov med organizacijami. Evropska unija je v okviru svoje strategije za podatke prepoznala potrebo po oblikovanju *skupnih evropskih podatkovnih prostorov* v različnih sektorjih, ki bodo omogočali nadzorovano izmenjavo podatkov ob zagotavljanju podatkovne suverenosti deležnikov [1]. Klasični pristopi k deljenju podatkov, kot so centralizirana skladišča ali ad-hoc izmenjava pogosto ne zagotavljajo zadostnega nadzora nad tem, kdo in pod kakšnimi pogoji dostopa do podatkov. Prav tako je velikokrat nerazrešeno finančno vprašanje oz. finančna kompenzacije. Podatkovni prostori rešujejo te izzive z uvedbo federativnega modela – podatki ostajajo pri viru, deljenje pa poteka preko standardiziranih vmesnikov in pogodb, kjer imajo lastniki podatkov ves čas nadzor nad uporabo svojih podatkov [2]. V takem okolju osrednjo tehnološko vlogo igrajo povezovalniki podatkovnih prostorov, ki predstavljajo prehode (ang. gateway) med različnimi deležniki ter omogočajo povezovanje različnih informacijskih sistemov v podatkovni prostor ob uveljavljanju pravil uporabe in varnosti [3]. V nadaljevanju članka najprej predstavimo koncept podatkovnih prostorov ter okvir, ki ga je razvilo Mednarodno združenje za podatkovne prostore (IDSA), nato pa se osredotočimo na **Eclipse Dataspace Components (EDC)** kot referenčno implementacijo povezovalnika podatkovnih prostorov. Osrednji del prispevka je posvečen naši rešitvi **DSX Engine**, ki EDC nadgrajuje z vrsto decentralizacijskih mehanizmov in novimi funkcionalnostmi. Članek zaključujemo s predstavitvijo **pilotne implementacije v podatkovnem prostoru DIH AGRIFOOD Data Space (DADS)** in razpravo o izzivih, pridobljenih izkušnjah ter možnih prihodnjih nadgradnjah.

2 Podatkovni prostori

2.1. Potreba po podatkovnih prostorih in pobude Evropske komisije

Obseg podatkov, ki jih ustvarjajo podjetja in druge organizacije, eksponentno narašča, s tem pa tudi potencial za ustvarjanje dodane vrednosti z njihovo delitvijo in analiziranjem. V praksi izmenjavo podatkov onemogočajo številne ovire: pravna in varnostna tveganja, pomanjkanje zaupanja, tehnološka nezdružljivost ter strah pred izgubo nadzora nad podatki. Podatkovni prostori so se pojavili kot koncept, ki naj bi naslovil te izzive in omogočil zaupanja vredno med-organizacijsko izmenjavo podatkov ob ohranjanju suverenosti vseh vpletenih deležnikov [3]. Ključna ideja je, da udeleženci sami nadzorujejo svoje podatke in pogoje deljenja: vsak udeleženec odloča, s kom bo podatke delil in pod katerimi vnaprej določenimi pogoji uporabe, identiteta in pravice vsakega udeleženca pa so nadzorovane [3]. Tako podatkovni prostori zagotavljajo, da se podatki lahko uporabijo za nove storitve in inovacije, ne da bi lastniki izgubili nadzor nad njihovo uporabo.

Evropska komisija je prepoznala strateški pomen takšnega pristopa in v evropski strategiji za podatke začrtala vzpostavitev skupnega evropskega podatkovnega prostora, ki bo okrepil konkurenčnost in podatkovno suverenost Evrope [4]. V praksi to pomeni vzpostavitev več tematskih podatkovnih prostorov (npr. industrijski, zdravstveni, kmetijski, finančni, energetski itd.), kjer bodo deležniki posameznih sektorjev lahko učinkoviteje izmenjevali vsebinsko in domensko povezane podatke pod skupnimi pravili. Za podporo temu cilju je EU sprejela tudi regulatorne ukrepe, kot sta *Akt o upravljanju podatkov* (angl. Data Governance Act - DGA) in *Akt o podatkih* (angl. Data Act), ki vzpostavljata pravila za podatkovne posrednike, deljenje podatkov med podjetji in zagotavljanje zaupanja [4]. Te pobude poudarjajo vlogo nevtralnih posrednikov in standardov pri izmenjavi podatkov. V okviru

industrije sta nastala tudi pomembna projekta, International Data Spaces Association (IDSA) in Gaia-X, ki s tehničnimi standardi in referenčnimi arhitekturami podpirata vizijo decentraliziranih podatkovnih ekosistemov na evropski ravni.

2.2. Standardizacija podatkovnih prostorov – IDSA: sloji, komponente, procesi

Za uresničitev podatkovnih prostorov je ključno skupno razumevanje arhitekture in komponent. Mednarodno združenje za podatkovne prostore (IDSA) je že od leta 2015 vodilno pri definiranju takšne arhitekture. Rezultat tega je *Referenčni arhitekturni model* (angl. Reference Architecture Model – IDSA RAM), ki podaja več nivojski opis gradnikov podatkovnega prostora. Uvaja pet slojev arhitekture [5]:

Poslovni sloj

Opredeljuje poslovne vloge udeležencev v podatkovnem prostoru ter njihove medsebojne interakcije in poslovne modele. Te vloge so ponudnik podatkov, potrošnik podatkov, posrednik, skrbnik podatkovnega prostora itd. Ta sloj zagotavlja okvir, znotraj katerega lahko nastajajo nove digitalne storitve in poslovni modeli na osnovi podatkov.

Funkcionalni sloj

Določa potrebne funkcionalnosti in zahteve sistema za zagotovitev zaupanja, varnosti in suverenosti podatkov. Sem sodijo mehanizmi za upravljanje identitet in dostopa, certificiranje udeležencev, posredovanje metapodatkov (ang. broker), uveljavljanje politik uporabe podatkov ter zanesljivo komunikacijo (Slika 1) [6]. Opredeljene so tudi podatkovne aplikacije (Data Apps) za izvedbo obdelav nad podatki in koncept klirinške podatkov (ang. Clearing House), ki omogoča monetizacijo podatkov z vključenimi mehanizmi nadzora in obračunavanja. Naslavlja šest vidikov sistema: zaupanje, varnost in podatkovna suverenost, ekosistem podatkov, standardizirana interoperabilnost, podatkovne aplikacije ter podatkovne tržnice.

Slika 1: IDSA RAM funkcijski sloj. [5]

Informacijski sloj

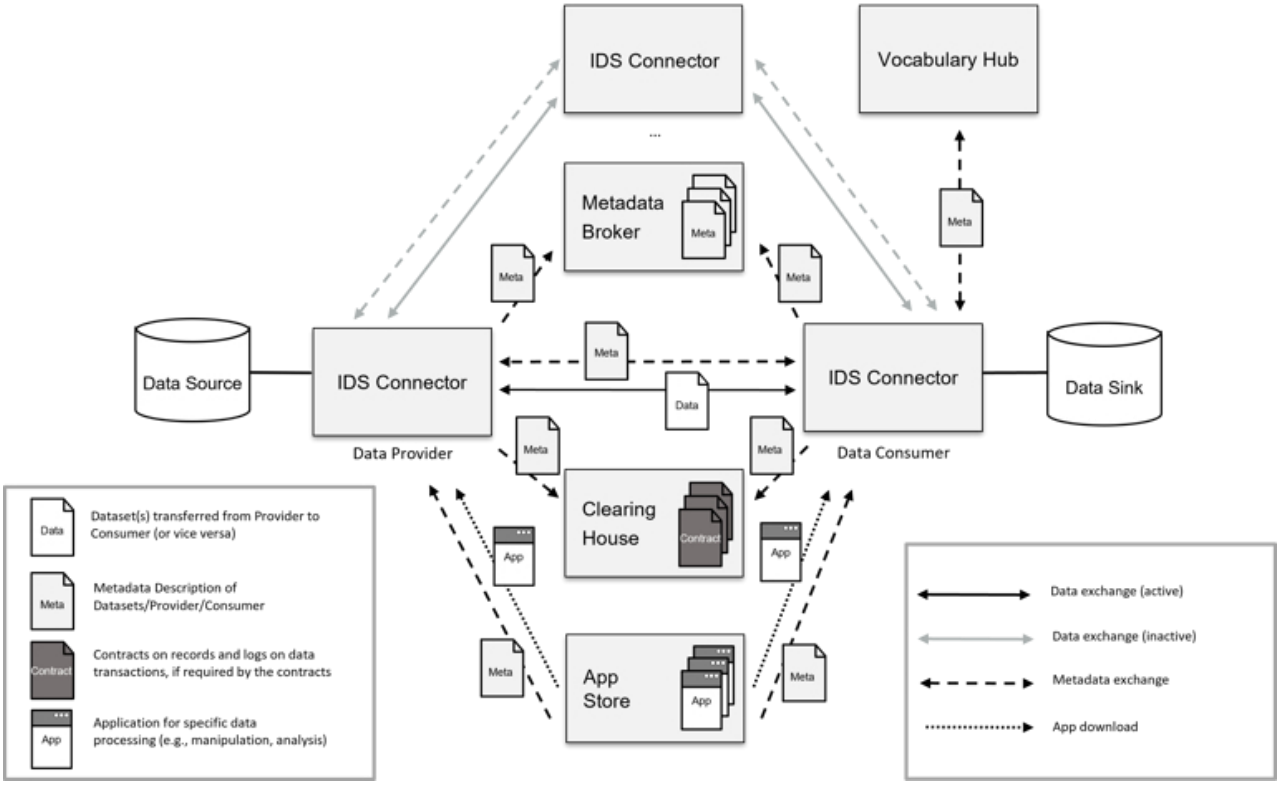
Ureja semantični model podatkov in standardizirane podatkovne formate. Da lahko različni deležniki učinkovito izmenjujejo podatke, morajo uporabljati skupne pojme in ontologije – informacijski sloj tako definira, kako so podatki opisani in predstavljeni, vključno z metapodatki (npr. opis ponudbe podatkov, pogojev uporabe itd.).

Procesni sloj

Podaja dinamični vidik delovanja podatkovnega prostora – opisuje osnovne procese in poteke dela, kot so priključitev novega udeleženca (angl. onboarding), objava ponudbe podatkov, pogajanje in sklenitev podatkovne pogodbe, izmenjava podatkov ter izključitev udeleženca [6]. Ta sloj opisuje zaporedje interakcij med komponentami, kot so povezovalnik, posrednik in ponudnik identitet, v posameznem scenariju znotraj podatkovnega prostora.

Sistemski sloj

Opisuje tehnično infrastrukturo in arhitekturo komponent (Slika 2). Določa, katere tehnične komponente sestavljajo podatkovni prostor (npr. **povezovalniki**, posredniki, ponudniki identitet, beleženje transakcij, hranjenje digitalnih potrdil) in kako te komponente med seboj komunicirajo, vključno s protokoli in vmesniki (API-ji). Ta sloj ponuja načrt za implementacijo podatkovnega prostora v obliki programske opreme in storitev [6].



Slika 2: IDSA RAM sistemski sloj – arhitektura. [5]

Čez vse sloje referenčnega modela se kot prečne teme obravnavajo še varnost, zasebnost, zaupanje in upravljanje (angl. governance) [6]. Poseben poudarek v International Data Spaces (IDS) je na certificiranju – ključne komponente, predvsem povezovalniki in udeleženci morajo pridobiti certifikate zaupanja, kar zagotavlja, da vsi v podatkovnem prostoru spoštujejo skupna pravila in varnostne zahteve [6]. IDS je tako vzpostavil celovit okvir, ki naslavlja tako organizacijske kot tehnične vidike podatkovnih prostorov.

Pomembno je, da vsi udeleženci spoštujejo enotne protokole in semantiko – le tako lahko dosežemo interoperabilnost med različnimi implementacijami podatkovnih prostorov. IDSA aktivno vzdržuje te specifikacije (trenutna različica referenčnega modela je 4.0) in jih usklajuje z drugimi pobudami (npr. Gaia-X), da se prepreči razdrobljenost standardov.

IDSA knjiga pravil

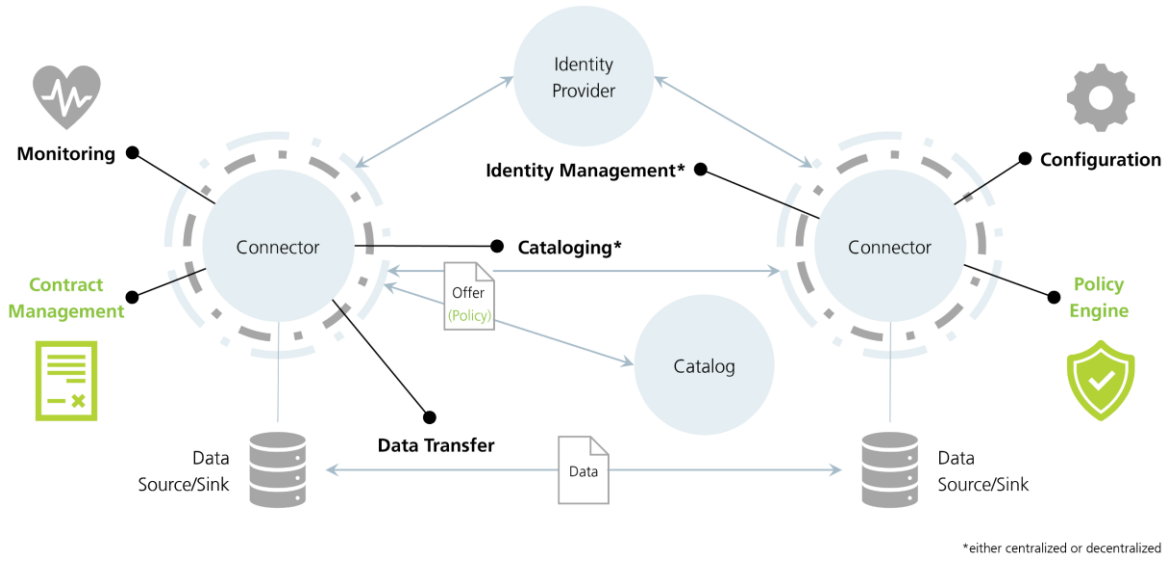
Predstavlja osrednji normativni dokument za standardizacijo v podatkovnih prostorih, saj določa enotna pravila za vključevanje udeležencev, certificiranje povezovalnikov in uveljavljanje politik uporabe. V njem so natančno opredeljene poslovne vloge (ponudnik, potrošnik, posrednik, upravljalec), minimalne varnostne zahteve, postopki revizije ter merila za izdajanje zaupanja vrednih poverilnic (angl. Verifiable Credentials - VC). Pravilnik (ang. rulebook) vzpostavlja tudi mehanizem reševanja sporov in sankcij v primeru neskladnosti, kar omogoča dosledno uveljavljanje suverenosti podatkov in medsebojne interoperabilnosti [7]. S tem služi kot obvezen referenčni okvir, ki ga morajo spoštovati vsi člani podatkovnega prostora IDS vključno z njihovimi tehničnimi implementacijami, kot so EDC ali DSX Engine.

3 Eclipse Dataspace Components (EDC)

Z namenom pospešitve razvoja podatkovnih prostorov je skupina industrijskih partnerjev (npr. v okviru pobude Catena-X v avtomobilski industriji) sprožila odprtokodni projekt pod okriljem Eclipse Foundation, imenovan Eclipse Dataspace Components (EDC). EDC je ena izmed referenčnih implementacij osnovnih komponent povezovalnika, združljivega s standardi IDS in prilagojenega evropskim pobudam za podatkovne prostore. Gre za ogrodje (angl. framework), ki organizacijam omogoča enostavnejšo vzpostavitev lastnega povezovalnika in vključitev v podatkovni prostor, kjer lahko varno delijo podatke z ostalimi udeleženci. Vsaka organizacija namesti svoj prilagojen EDC povezovalnik, ki deluje kot končna točka za deljenje podatkov in kot čuvaj, ki zagotavlja upoštevanje korporativnih politik in predpisov o suverenosti podatkov [1].

Prvi cilj projekta EDC je bil implementirati koncept IDS povezovalnika v praksi ter hkrati le tega utemeljiti na konceptih decentralizacije (Slika 3). Zgrajen je tako, da lahko komunicira z ostalimi osnovnimi storitvami v podatkovnem prostoru, kot so ponudniki identitet za preverjanje identitet udeležencev, posredniki (ang. Brokerji) za iskanje podatkovnih ponudb ali klirinška hiša (ang. Clearing House) za beleženje dogodkov, skladno z IDS referenčnim modelom [1]. S tem EDC lahko prevzame vlogo osnovne infrastrukture podatkovnega prostora. Projekt se je ustanovil leta 2021 in hitro pritegnil pozornost različnih domen – od industrije 4.0, pametne mobilnosti, do finančnega in kmetijskega sektorja – skratka povsod, kjer se pojavi potreba po zaupanja vredni izmenjavi podatkov med partnerji.

3.1. Arhitektura in komponente EDC



Slika 3: Arhitektura EDC. [8]

Arhitektura EDC sledi načelom modularnosti, razširljivosti in robustnosti. Sestavljena je iz več komponent oziroma modulov, od katerih vsak opravlja določeno funkcijo v procesu deljenja podatkov [1]. Ključni elementi arhitekture EDC so [8]: (1) krmilna ravnina (ang. control plane), (2) podatkovna ravnina (ang. data plane), (3) upravljalnik identitet (ang. identity hub), (4) izdajatelj poverilnic oziroma člen zaupanja, (5) mehanizem za upravljanje politik (ang. policy engine) in drugi moduli za upravljanje ter revizijo.

Krmilna ravnina je ključni del povezovalnika, ki skrbi za upravljanje in pogajanje glede dostopa do podatkov. Tu potekajo procesi, kot so registracija podatkovnih ponudb v katalog, iskanje podatkov, pogajanje med ponudnikom in potrošnikom ter sklenitev pogodbe o podatkih. Krmilna ravnina vsebuje ustrezne module za izvedbo poizvedb, usklajevanje izmenjave, preverjanje in uveljavljanje politik uporabe, spremljanje in revizijo transakcij. Pomembno je poudariti, da EDC uporablja asinhron način komunikacije med komponentami in ne uvaja nobene centralne baze podatkov, kar izboljšuje razpršenost in odpravlja eno točko odpovedi [1]. Preko nadzornika organizacije prav tako upravljajo in definirajo svoje kataloge podatkov, politike uporabe (angl. policy), definicije pogodb (angl. contract defnition) in dostopajo do lokalno predpomnjenega kataloga podatkovnega prostora, ki ga povezovalnik gradi s pomočjo pajka (ang. crawler) komponente, ki se nahaja v EDC Federated Catalog rešitvi.

Pajek je komponenta EDC, ki periodično razpošilja poizvedbe po katalogih vsem udeležencem podatkovnega prostora iz lokalnega seznama udeležencev. Nadzornik posameznega udeleženca ob zahtevi vrne prilagojen podatkovni katalog, ki vsebuje le ponudbe podatkov s katerimi politikami dostopa (angl. access policy) je povezovalnik, ki poizveduje po katalogu, skladen. Pajek nato shrani kataloge udeležencev v svoj lokalni podatkovni katalog, ki predstavlja katalog podatkovnega prostora. Glede na število udeležencev podatkovnega prostora, krmilna ravnina samodejno prilagodi, koliko pajkov je razposlanih sočasno, za namene pohitritve posodobitev lokalnega kataloga podatkov [8] [9].

Podatkovna ravnina je ločen del, ki skrbi za dejanski prenos podatkov od ponudnika do potrošnika. Ko je v krmilni ravnini sklenjena pogodba in dodeljena avtorizacija, podatkovna ravnina vzpostavi neposredno komunikacijo (npr. prenos datotek, pretok podatkov preko API) med povezovalnikoma. EDC privzeto zagotavlja revizijsko sled za prenos podatkov – vse transakcije se beležijo v dnevnik (angl. audit log), ki ga lahko preverita obe strani, vsaka pri svojem povezovalniku. Za povečanje varnosti EDC hrani občutljive informacije (npr. ključe, žetone za dostop) v varovanem trezorju (Hashicorp Vault) in uporablja začasne poverilnice, da minimizira možnosti zlorabe [1]. Vsak nadzornik lahko ima registriranih več podatkovnih posrednikov. S tem zagotovimo horizontalno skalabilnost, kar omogoča večji pretok podatkov.

Upravljalnik identitet je ločena komponenta EDC, odgovorna za shranjevanje, upravljanje in izmenjavo zaupanja vrednih poverilnic, upravljanje decentraliziranih identifikatorjev, certifikatov in ostalih z varnostjo ter identiteto povezanih zadev med udeleženci podatkovnega prostora. Izvaja naslednje funkcije [8]:

- Shranjuje in upravlja poverilnice, ki jih izdajajo zaupanja vredni izdajatelji poverilnic;
- Na podlagi poverilnic, generira preverljive predstavitve (angl. Verifiable Presentations - VP), ki omogočajo predstavitve poverilnic drugim udeležencem, pri čemer preverja in upravlja dostop na osnovi pravil. S tem pripomore pri ohranjanju popolnega nadzora nad vsebino, obsegom in kontekstom razkritih informacij;
- Upravlja DID organizacije;
- Upravlja s kriptografskimi pari ključev, pri čemer poskrbi tudi za njihovo rotacijo;

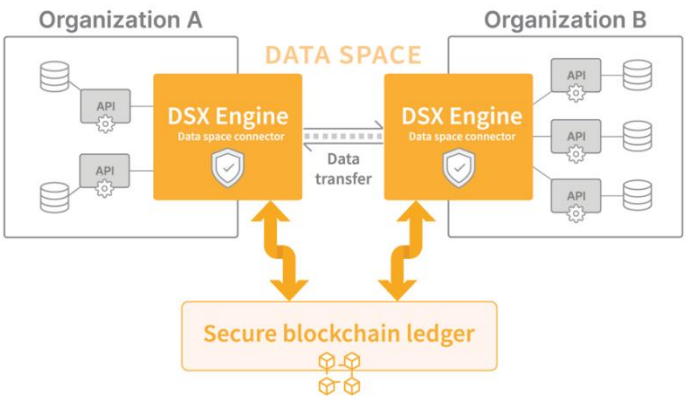
Deluje kot decentralizirana denarnica identitet, ki omogoča varno in nadzorovano izmenjavo verodostojnih trditev (angl. claim), s čimer podpira dinamično overjanje in avtorizacijo udeležencev v skladu z Eclipse Decentralized Claims Protocol (DCP) specifikacijo [8].

Izdajatelj poverilnic v arhitekturi EDC predstavlja člen zaupanja, katerega naloga je ustvarjanje, podpisovanje in posredovanje zaupanja vrednih poverilnic udeležencem podatkovnega prostora. Izdaja poverilnic je definirana v specifikaciji Decentralized Claims Protocol (DCP) kot asinhron proces, v katerem upravljalnik identitet povezovalnika pošlje zahtevo za poverilnico na vnaprej znan končni naslov izdajatelja, ki mu zaupa. Ta zahteva vključuje samopodpisan identitetni žeton (angl. identity token), ki vsebuje tudi dostopni žeton (angl. access token),

s katerim lahko izdajatelj po postopku odobritve dostavi podpisano poverilnico nazaj v upravljalnik identitet pošiljatelja [8] [10].
Postopek odobritve lahko vključuje preverjanje dodatnih verodostojnih trditev pošiljatelja. Na primer, izdajatelj lahko pred izdajo zahteva dokazilo, da prejemnik že razpolaga z drugo zanesljivo poverilnico kot je dokaz o članstvu v podatkovnem prostoru. Za identifikacijo udeležencev DCP uporablja decentralizirane identifikatorje (DID). EDC podpira DID z metodo did:web, ki omogoča, da organizacije določijo svojo identiteto prek HTTPS infrastrukture. DID je uporabljen pri podpisovanju poverilnic in verifikaciji ter overjanju izvora zahtevkov v podatkovnem prostoru [8] [10].
EDC ima vgrajen **mehanizem za izvrševanje politik uporabe podatkov** (ang. policy engine), ki omogoča, da ponudnik ob objavi podatkovne ponudbe določi pogoje (npr. kdo lahko dostopa, časovna omejitev dostopa, namen uporabe itd.). Te politike se prenesejo do povezovalnika potrošnika ob sklenitvi pogodbe. Povezovalnika na obeh straneh nato avtomatizirano uveljavljata ta pravila – npr. blokirata dostop, če pogoji niso izpolnjeni, ali omejita število dostopov. S tem je zagotovljeno spoštovanje načel rabe v celotni življenjski dobi podatkov, tudi po prenosu [1].
Kot del nadzornika, EDC ponuja **module za spremljanje poteka izmenjave in revizijsko sled**. Vsako dejanje – od zahtevka za podatke, pogodbe, do prenosa – je mogoče zabeležiti, kar omogoča poznejše preverjanje skladnosti, obračunavanje porabe podatkov in reševanje sporov. Centralizirane podatkovne shrambe v EDC ni. Namesto tega vsaka organizacija hrani svoje podatke na svoji infrastrukturi, EDC pa poskrbi, da se ti podatki na zanesljiv in varen način pretakajo do pooblaščenih potrošnikov.
Čeprav EDC omogoča modularno razširitev in interoperabilnost skladno z IDS specifikacijami, v svoji osnovni obliki **ni v celoti decentraliziran**. Tipične implementacije se pri registraciji udeležencev in beleženju dogodkov še vedno opirajo na osrednje komponente ali zaupanja vredne posrednike. To pomeni, da v primeru izpada ali kompromitacije teh centralnih točk lahko pride do motenj ali celo onemogočanja delovanja podatkovnega prostora.
Ena od prednosti EDC je njegova razširljivost – zasnovan je kot ogrodje, ki ga je mogoče prilagajati specifičnim potrebam. Ima podporo za vtičnike (plug-in točke preko SPI, angl. Service Provider Interface) za integracijo različnih sistemov za avtentikacijo in avtorizacijo, podporo različnim protokolom prenosa podatkov, povezovanje z različnimi katalogi in storitvami. Skupnost razvijalcev EDC je do danes dodala že več komponent, npr. modul za decentralizirane identitete (angl. decentralized identity) in integracijo z denarnicami za potrebe Gaia-X [11], ter načrtuje nadaljnje izboljšave v smeri skladnosti z Gaia-X specifikacijami (npr. infrastruktura za preverjanje udeležencev in storitev). EDC tako predstavlja temeljni gradnik, ki sicer omogoča minimalno delujoč podatkovni prostor, vendar na osnovi številnih statičnih spremenljivk in modulov, vendar nam kot ogrodje omogoča, da gradimo prilagojene rešitve povezovalnikov – ena takšnih nadgradenj osnovne arhitekture EDC je naš DSX Engine.

4 DSX Engine

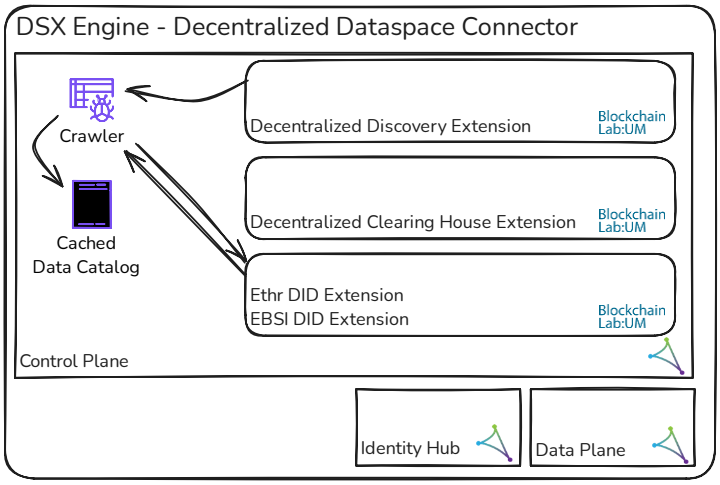
DSX Engine je nastal z namenom razširitve in nadgradnje obstoječih povezovalnikov podatkovnih prostorov, za potrebe še bolj decentraliziranega, zaupanja polnega in funkcionalno bogatega podatkovnega prostora. Osnovna arhitektura podatkovnih prostorov, kot jo opredeljuje IDSA in implementira EDC, predstavlja odlično izhodišče, vendar so se v praksi pokazale potrebe po dodatnih funkcionalnostih, zaradi katerih je bila razvita rešitev DSX Engine. Slika 4 prikazuje konceptualno osnovo povezovalnika DSX Engine.



Slika 4: Konceptualni prikaz podatkovnega prostora z DSX Engine povezovalniki.

4.1. Implementirane razširitve

Z namenom doseganja polno delujočega, dinamičnega in samostojnega povezovalnika, ki bo izpolnjeval dodatno zastavljene kriterije decentralizacije, smo obstoječi EDC nadgradili s tremi osrednjimi razširitvami in drugimi podpornimi funkcionalnostmi, ki jih v nadaljevanju podrobneje opišemo. Slika 5 prikazuje visokonivojsko arhitekturo DSX Engine, kjer so izpostavljeni osrednje implementirane razširitve.



Slika 5: Arhitektura DSX Engine.

Razširitev decentralizirane poizvedbe

Z namenom podpreti popolnoma decentraliziran način povezovanja med povezovalniki, smo razvili t. i. razširitev decentralizirane poizvedbe (ang. Decentralized Discovery Extension - DDE), ki omogoča dinamično registracijo in odkrivanje udeležencev (tj. drugih povezovalnikov) v podatkovnem prostoru, ki temelji na infrastrukturi verige blokov. DDE je implementiran kot EDC razširitev, ki vzpostavi neposreden most med EDC povezovalnikom in pametno pogodbo v okolju Ethereum. Pametna pogodba, ki smo jo implementirali in objavili na verigo blokov, deluje kot javni in preverljiv register udeležencev.

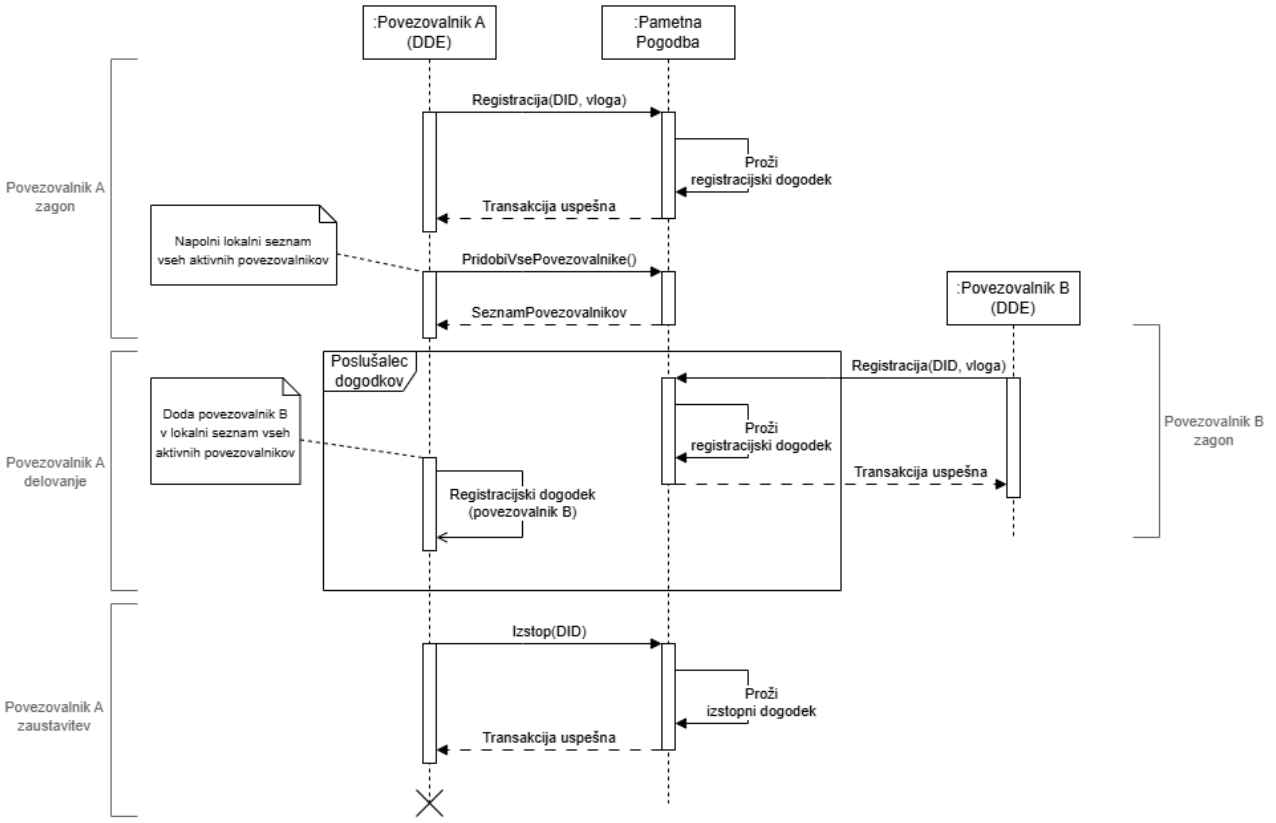
DDE v povezovalniku dinamično gradi lokalni seznam vseh aktivnih ponudnikov v podatkovnem prostoru. Ponudniki so povezovalniki (udeleženci), ki imajo vlogo nastavljeno na *ponudnik*. Organizacija pa ima možnost ročno dodati na seznam tudi udeleženca, ki ni ponudnik. EDC komponenta pajek izvede iskanje podatkovnih katalogov pri odkritih udeležencih (povezovalnikih) in jih shrani v lokalni predpomnjen seznam ter s tem omogoči poizvedovanje in procesiranje vsebine katalogov [8].

Povezovalnik se ob vzpostavitvi samodejno registrira v pametni pogodbi, pri čemer objavi svoj decentraliziran identifikator (DID) in vlogo ali je ponudnik ali potrošnik. Vloga potrošnik sporoči ostalim udeležencem, da ta povezovalnik ne nudi podatkov, kar pomeni, da ga ostali udeleženci ne bodo samodejno poizvedovali po katalogu. S tem se zmanjša število nepotrebnih klicev med udeleženci podatkovnega prostora. **Pametna pogodba ob registraciji povezovalnika sproži dogodek, ki ga ostali povezovalniki zaznajo**.

Po zagonu DDE pridobi vse udeležence in nato aktivira poslušalca dogodkov, ki posluša dogodke v pametni pogodbi (vstop ali izstop udeleženca). Tako samodejno periodično posodablja seznam ponudnikov. Vsak povezovalnik tako upravlja svojo lokalno, a usklajeno kopijo seznama vseh aktivnih ponudnikov, brez potrebe po centraliziranem imeniku. Proces je predstavljen na sliki 6.

DSX Engine vključuje izvorno podporo za izvajanje pametnih pogodb v **Ethereum** omrežjih. To pomeni, da lahko deluje tako v **javnih** omrežjih (npr. Ethereum mainnet, testneti) kot tudi v **zasebnih ali konzorcijskih** EVM-kompatibilnih omrežjih. Ta fleksibilnost omogoča prilagoditev različnim regulativnim in varnostnim zahtevam, obenem pa odpira možnost integracije z obstoječimi rešitvami in orodji verig blokov.

Vse decentralizirane razširitve DSX Engine lahko uporabljajo katerokoli Ethereum-kompatibilno infrastrukturo kot temelj za izvajanje pametnih pogodb, kar omogoča prenosljivost in ponovljivo uvajanje v različnih okoljih.



Slika 6: Diagram zaporedja razširitve DDE.

DDE uporablja decentralizirane identifikatorje (DID), kjer je identiteta povezovalnika zgrajena v skladu s specifikacijo W3C DID [12]. DSX Engine trenutno podpira DID:ethr, v razvoju pa je DID:ebsi. DID:ethr je osrednji identifikator, saj se za odkrivanje uporablja omrežje in veriga blokov Ethereum. DID je možno razrešiti (angl. resolve) in pridobiti DID dokument z atributi, ki med drugim vključujejo *verificationMethod*, ki vsebuje javni kriptografski ključ povezovalnika in *service* polje, kjer je definiran javni naslov (URL). Ta naslov definira, kje je izpostavljen povezovalnikov aplikacijsko programski vmesnik (API) do protokola podatkovnega prostora (Dataspace Protocol - DSP) [13]. Vsak povezovalnik ima lasten par kriptografskih ključev (javni in zasebni ključ), s katerimi podpisuje transakcije na verigi blokov ter omogoči varno in preverljivo komunikacijo med samimi

povezovalniki. Vsaka organizacija skrbi za lastno identiteto s čimer se zagotavlja avtentikacija in avtorizacija povezovalnikov brez centralne avtoritete.
Razširitev zagotavlja:

- Popolno decentralizacijo, brez potrebe po centralnem katalogu udeležencev, avtoriteti ali posredniku;
- Avtomatizirano integracijo, saj se udeleženci v podatkovni prostor vključijo zgolj z registracijo DID-a in vloge, ostali povezovalniki ga nato samostojno odkrijejo preko tehnologije veriženja blokov;
- Samostojnost udeležencev, ki sami skrbijo za svojo prisotnost v omrežju;
- Samoorganizacijo podatkovnih prostorov na osnovi pametnih pogodb;
- Interoperabilnost med povezovalniki;
- Usklajevanje udeležencev brez medsebojnega zaupanja (angl. trustless coordination).

Razširitev decentralizirane klirinške hiše

Namen razširitve decentralizirane klirinške hiše (ang. Decentralized Clearing House – DCH) je vzpostavitev zanesljivega, transparentnega, neizpodbitnega in decentraliziranega mehanizma za beleženje dogodkov, ki nastajajo znotraj podatkovnega prostora, saj je to ključna funkcionalnost za vse nadaljnje korake in funkcionalnosti, povezane s podatkovno ekonomijo, ki jo podpiramo s tokenizacijo. Povezovalniki beležijo vse svoje dogodke obeh ravnin (nadzorniška in podatkovna) na pametno pogodbo, ki smo jo implementirali in objavili na Ethereum verigi blokov in ima vlogo decentralizirane, transparentne, nespremenljive in zaupanja vredne shrambe dogodkov. DCH razširitev temelji na specifikaciji IDS Clearing House [14]. Vsak povezovalnik ima svoj DCH, ki spremlja relevantne interne EDC dogodke in jih pošilja na verigo blokov. Ti podatki se lahko na primer uporabijo za obračunavanje porabe podatkov ali sledljivost transakcij med dvema povezovalnikoma.
Primeri internih EDC dogodkov so:

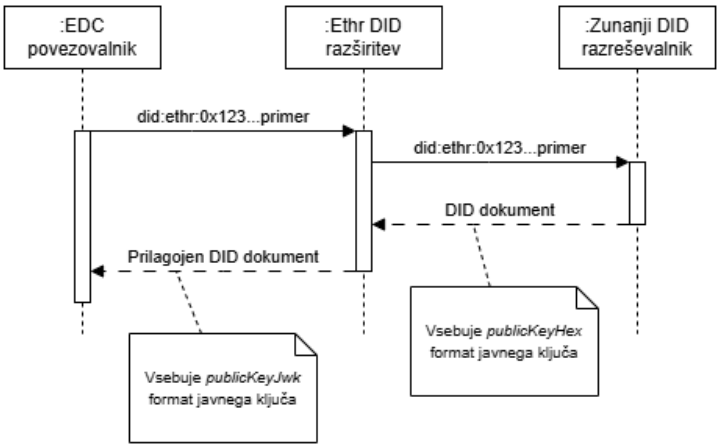
- Ponudbe podatkov (angl. asset): ustvarjanje, posodabljanje, brisanje
- Definicije politik (angl. policy definition): ustvarjanje, posodabljanje, brisanje
- Definicije pogodb (angl. contract definition): ustvarjanje, posodabljanje, brisanje
- Pogajanje pogodb (angl. contract negotiation): sprememba stanja (zahtevano, ponujeno, sprejeto, dogovorjeno, preverjeno, prekinjeno, zaključeno) [13]
- Procesi prenosa podatkov (angl. transfer process): sprememba stanja (zahtevano, začeto, suspendirano, zaključeno, prekinjeno) [13]

Ob zaznavi dogodka razširitev izlušči relevantne metapodatke (npr. ID ponudbe podatkov, ID pogajanja itd.) in jih trajno objavi na pametno pogodbo.

Razširitev Ethr DID

Zaradi potrebe po uporabi decentraliziranih identifikatorjev (ang. decentralized identifiers - DID) temelječih na infrastrukturi Ethereum, smo razvili Ethr DID razširitev, ki omogoča podporo za DID metodo »did:ethr«. Osnovni EDC podpira le metodo did:web [8].
Ob zagonu EDC povezovalnika se razširitev interno registrira kot razreševalnik za *did:ethr* in s tem povezovalniku omogoči obravnavo vseh DID identifikatorjev z metodo »did:ethr«.
Ko udeleženec *A* želi začeti komunikacijo z drugim že odkritim udeležencem *B* znotraj podatkovnega prostora, EDC povezovalnik pridobi DID in v primeru, da uporablja metodo »did:ethr« uporabi Ethr DID razširitev za pridobitev spletnega naslova (URL), kjer se nahaja DSP API povezovalnika udeleženca *B* in njegov verifikacijski javni ključ. Razširitev uporabi poljubni ustrezni zunanji DID razreševalnik (angl. DID resolver), ki vrne DID dokument, ki med drugim mora vsebovati vsaj eno verifikacijsko metodo z atributom »publicKeyHex« in URL do DSP API povezovalnika (Slika 7).
EDC trenutno podpira le »publicKeyJwk« ali »publicKeyMultibase« kot tipe verifikacijskih metod kot jih definira W3C DID [12]. Ker »did:ethr« običajno uporablja »publicKeyHex« format, Ethr DID razširitev izvede

transformacijo iz tega formata v publicKeyJwk format. »publicKeyJwk« je javni kriptografski ključ zapisan v JSON Web Key (JWK) formatu, medtem ko je »publicKeyHex« zapisan v šestnajstiškem formatu [12].
Podobno kot Ethr DID razširitev pripravljamo tudi EBSI DID razširitev, ki bo omogočala podporo »did:ebsi« metode decentraliziranih identifikatorjev, in sicer javni DID:ebsi, kjer bi se DID dokument shranjeval v javnem registru DID na infrastrukturi EBSI (European Blockchain Service Infrastructure). To smatramo ključnega pomena, saj je EBSI primarno namenjena vpeljavi decentraliziranih tehnologij v javni sektor Evropske unije in širše, kar posledično omogoča tudi lažjo integracijo in vpeljavo povezovalnika podatkovnega prostora, kot je DSX Engine v javni sektor.



Slika 7: Diagram zaporedja delovanja razširitve DID:ethr.

Razširitev registracije

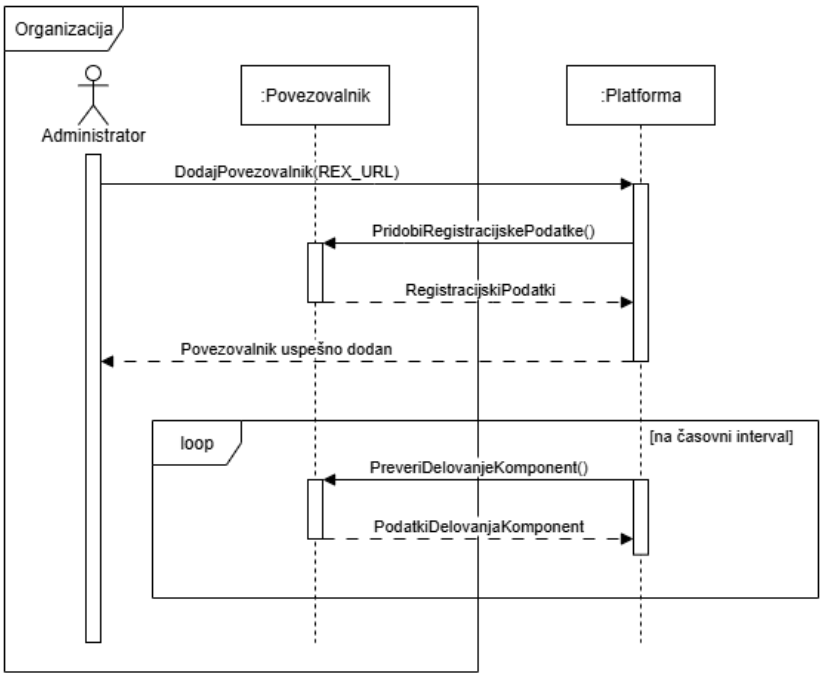
Namen razširitev registracije (ang. Registration Extension - REX) je omogočiti izpostavitev osnovnih informacij o povezovalniku, z namenom preprostejše povezave z nadzornimi ploščami, oziroma platformami podatkovnega prostora. Platforme nudijo uporabniški grafični vmesnik preko katerega organizacija lahko upravlja svoj povezovalnik, pregleduje podatkovni katalog, sklepa pogodbe in ostale povezane zadeve.
Osnovna namestitev EDC povezovalnika ne ponuja mehanizma za razkrivanje podatkov o povezovalniku, kot so identiteta udeleženca, naslovi API-jev ter stanje delovanja ali napak internih komponent (Izsek kode 1). To predstavlja oviro za hitro, enostavno in potencialno avtomatizirano vzpostavitev novega povezovalnika ter sprotno spremljanje delovanja.
REX izpostavi naslednje informacije o povezovalniku:

- Osnovne podatke (DID, vlogo);
- URL naslove, kjer se nahajajo API-ji določenih komponent, kot sta API za upravljanje povezovalnika in API za upravljanje podatkovnega kataloga;
- Stanje delovanja komponent (primeri stanj: zagnano, pripravljeno, delujoče);
- Podatke in stanje vseh podatkovnih posrednikov (Data Plane).

REX pridobiva povezovalnikove interne informacije o delovanju in jih izpostavi preko REST strežnika. Glavni končni točki sta */getRegistrationData*, ki vrne osnovne informacije o povezovalniku in */getStatusInfo*, ki prikaže trenutno stanje sistema. Končne točke so zavarovane z avtentikacijo na podlagi Bearer žetonov oziroma ključev (angl. API key). Proces delovanja REX je prikazan na sliki 8.

```
{
  "participantId": "did:ethr:0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC",
  "isProvider": true,
  "managementUrl": "http://10.5.0.10:39193/api/management",
  "catalogUrl": "http://10.5.0.10:38181/api/catalog"
}
```

Izsek kode 1: Primer odgovora Registration Extension komponente za osnovne metapodatke povezovalnika



Slika 8: Diagram zaporedja uporabe komponente REX

Politike po meri

Za podporo decentraliziranih podatkovnih prostorov, ki temeljijo na zaupanju med udeleženci, smo razvili razširitev in podporo politikam po meri, ki omogočajo vrednotenje politik na osnovi poverilnic v skladu z Decentralized Claims Protocol (DCP), ki ga implementira EDC [8]. Razširitev omogoča registracijo in uporabo politik po meri v podatkovnem prostoru.

Eden izmed ključnih ciljev je zagotoviti mehanizem, ki omogoča preverjanje članstva povezovalnikov v podatkovnem prostoru na decentraliziran način – ena izmed politik po meri. Ta politika je podprta na način, da ko povezovalnik želi ovrednotiti člansko poverilnico drugega povezovalnika izmed vseh ustrezno podpisanih poverilnic, ki mu jih je drugi povezovalnik poslal, filtrira/pridobi le poverilnico, ki je tipa članska poverilnica. Tukaj govorimo o formatu preverljive poverilnice (ang. Verifiable Credential - VC), ki je skladna s standardom VC po W3C. Iz te poverilnice prebere podatke, kot so status udeleženca, čas začetka in konca veljavnosti članstva. Te podatke preveri in kot rezultat vrne ali je poverilnica veljavna ali neveljavna.

Podpora ostalih politik po meri je implementirana po enakem principu.

4.2. Delovanje v praksi

Na visoki ravni DSX Engine deluje podobno kot standardni IDS povezovalnik: vsaka organizacija, ki želi sodelovati v podatkovnem prostoru, poganja instanco DSX Engine, preko katere ponuja svoje podatke ali dostopa do podatkov drugih.

Spodaj opisani potek ponazarja avtomatiziran, decentraliziran in zaupanja vreden način izmenjave podatkov v podatkovnemu prostoru, ki ga omogoča DSX Engine. Ključna razlika v primerjavi z drugimi povezovalniki in s klasičnim pristopom je v tem, da brez ročnih ali centraliziranih korakov (preverjanje certifikatov, podpis pogodbe,

beleženje transakcije, izvedba plačila) prevzame tehnologija – bodisi znotraj samega povezovalnika, bodisi v obliki pametnih pogodb na infrastrukturi tehnologij veriženja blokov. S tem se znižuje možnost napak, zmanjšuje potreba po posrednikih in dokumentih zunaj sistema, predvsem pa se povečuje zaupanje, saj vsak korak za seboj pusti kriptografsko preverljiv dokaz. DSX Engine je s svojim delovanjem primer vzpostavitve decentraliziranega podatkovnega povezovalnika, ki združuje najboljše prakse standardov IDS/EDC ter nove pristope, ki jih omogoča razvoj tehnologije.

Pridružitev podatkovnemu prostoru

Da se organizacije lahko pridružijo podatkovnemu prostoru, morajo pri sebi vzpostaviti povezovalnik DSX Engine in ga registrirati kot udeleženca v podatkovni prostor, pri tem pa potrebujejo veljavne digitalno podpisane poverilnice (VC) udeležbe, ki jih izdajajo skrbniki podatkovnega prostora.

Pred prvo pridružitvijo v podatkovni prostor morajo organizacije od skrbnikov podatkovnega prostora pridobiti svojo zaupanja vredno poverilnico formata VC, ki izkazuje članstvo organizacije v podatkovnem prostoru. Skrbniki podatkovnega prostora organizacije preverijo in jim odobrijo ali zavrnejo članstvo. Skrbniki podpisane poverilnice o članstvu, ki so zaupanja vredne vrnejo organizacijam, ki si jih shranijo v svoje upravljalnike identitet. Ena organizacija ima lahko eno veljavno poverilnico o članstvu. Potreba po takšnih pridružitvenih poverilnicah izhaja iz ideje podatkovnih prostorov in IDSA smernic, kjer ima vsak manjši ali večji podatkovni prostor svoja pravila igra in delovanja, ki jih je potrebno na organizacijski in tehničen način sprejeti, preden se omogoči dotičnemu povezovalniku povezovati z vsemi ostalimi.

Ker ponudbe podatkov v podatkovnem prostoru lahko zahtevajo skladnost z različnimi politikami, si organizacije lahko ustvarjajo mnoge poverilnice, v katere zapisujejo, s čim so skladne. Primeri skladnosti so: da organizacije podatke hranijo le znotraj Evropske unije, imajo določen certifikat (npr. ISO/IEC/IEEE 90003:2018), se strinjajo z nečim in ostale skladnosti, ki jih definirajo poslovne potrebe organizacije. Poverilnice morajo biti skladne z oblikovanjem, ki ga definira podatkovni prostor. Organizacije ustvarijo svoje poverilnice, ki so lastnoročno podpisane in niso zaupanja vredne, dokler jih ne podpiše član zaupanja v podatkovnem prostoru s poljubnimi trditvami (angl. claim), ki so tehnično podprte znotraj podatkovnega prostora (npr. organizacija je znotraj EU). Po tem, ko jih podpiše član zaupanja, si organizacije shranijo poverilnice v svoje upravljalnike identitet. Z ustrezno podpisanimi veljavnimi zaupanja vrednimi poverilnicami povezovalniki organizacij lahko izkazujejo skladnost z različnimi politikami ostalim udeležencem.

Po uspešni namestitvi in registraciji v podatkovni prostor DSX Engine povezovalniki samodejno z uporabo razširitve DDE periodično odkrivajo povezovalnike v podatkovnem prostoru. S tem si gradijo lokalne sezname vseh aktivnih ponudnikov podatkovnega prostora, katerega uporablja EDC komponenta pajek.

Objava podatkov v podatkovni katalog

Organizacije, ki želijo objaviti podatke v podatkovni prostor z namenom delitve z ostalimi udeleženci, morajo objaviti eno ali več ponudb (angl. asset) v svoj podatkovni katalog. Ponudbe so metapodatki dejanskih podatkov, ki jih želijo objaviti. Metapodatke v podatkovnem katalogu si lahko predstavljamo kot prikazan izdelek v katalogu neke trgovine. Opisuje kaj ta izdelek je, kje ga lahko kupimo/pridobimo in s čim moramo biti skladni (npr. nakup alkohola možen le polnoletnim osebam). Ko stranka izbere izdelek, ki ga želi, gre v ustrezno trgovino, izvede transakcijo in šele nato pridobi dejanski izdelek. Po podobnem principu deluje tudi podatkovni katalog, ki hrani le ponudbe podatkov, ki med drugim podajo tudi informacije potrebne za pridobitev dejanskih podatkov.

Objave posameznih ponudb v podatkovni katalog so sestavljene iz treh delov:

Politik (politike dostopa in pogodbene politike);

Ponudb (angl. Assets);

Povezave politik s ponudbami – definicije pogodb (angl. Contract Definitions).

Prvi del, potreben za objavo ponudb, so ustvarjene politike. Vsaka politika lahko definira tri različne kategorije zahtev [15]:

- Dovoljenja (angl. permissions) – kaj se lahko dela s podatki (npr. dovoljena je uporaba za strojno učenje);
- Prepovedi (angl. prohibitions) – kaj se ne sme delati s podatki (npr. ni dovoljena komercialna uporaba);
- Obveznosti (angl. obligations) – dolžnosti ravnanja s podatki (npr. skladnost z GDPR).

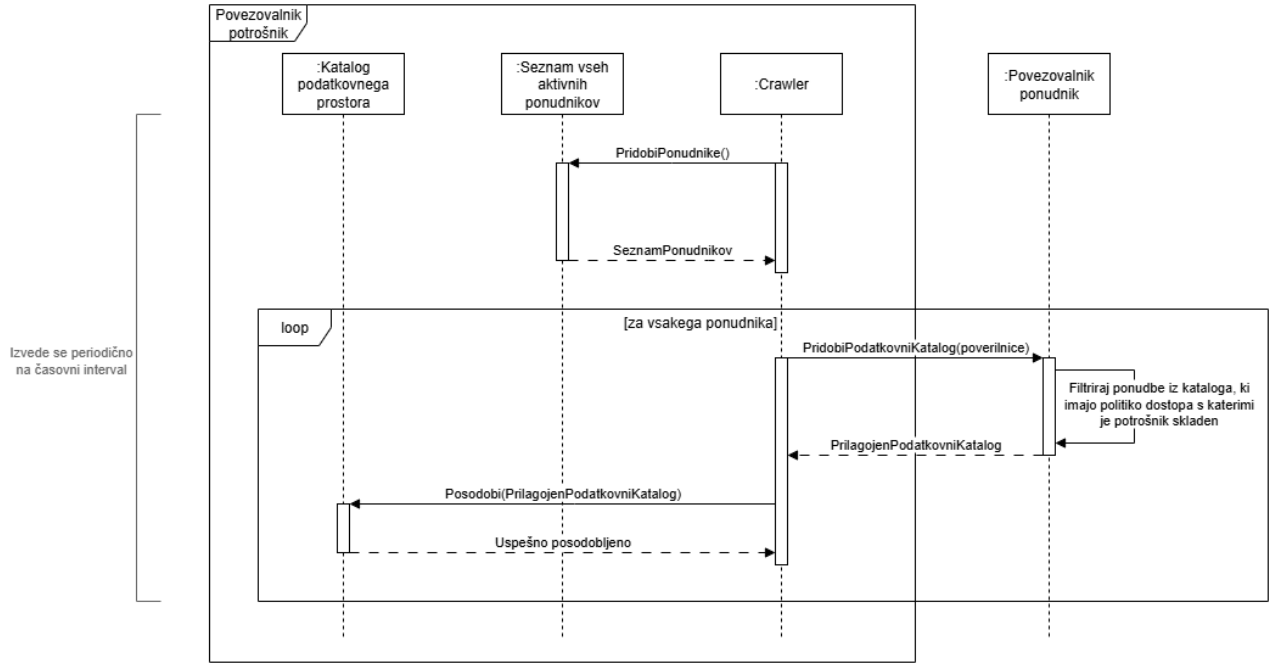
V politikah se lahko definirajo zahteve, kot so časovne omejitve (npr. dostop od 1.1.2025 naprej), lokacijske omejitve (npr. potrošnik je v Evropski uniji), omejitve shranjevanja in obdelave podatkov (npr. podatki ne smejo zapustiti Evropske Unije), cenik (npr. vsaka poizvedba stane 1 EUR ali DSX žeton), pripadnost (npr. potrošnik pripada določeni organizaciji), certifikati (npr. potrošnik mora imeti določene certifikate), namen uporabe (npr. dovoljena uporaba le za raziskovalne namene).
V teoriji so zahteve v posameznih politikah lahko karkoli, potrebno je le, da podatkovni prostor delovanje takšnih zahtev implementira. Organizacije lahko definirajo dve vrsti politik: politiko dostopa (angl. access policy) in pogodbeno politiko (angl. contract policy).
V politikah dostopa se definirajo zahteve, s katerimi morajo udeleženci podatkovnega prostora biti skladni, da jim povezovalniki ponudnikov prikažejo oz. pošljejo določene ponudbe.
Pogodbene politike definirajo zahteve s katerimi se potrošniki morajo strinjati oz. biti skladni, da bodo lahko pridobili dostop do dejanskih podatkov, ki jih želijo od ponudnikov. Politike so zapisane v jeziku ODRL (Open Digital Rights Language), ki je W3C priporočilo za izražanje pravic uporabe in omejitev za digitalne vsebine [15]. V EDC se ODRL uporablja za definiranje tako politik dostopa kot pogodbenih politik v JSON-LD formatu [8]. Organizacije lahko imajo več politik dostopa in pogodbenih politik, ki se kasneje vežejo na posamezne ponudbe.
Drugi del objave ponudb so ponudbe (angl. asset), ki jih ustvarijo organizacije za dejanske podatke, ki jih želijo deliti v podatkovnem prostoru. Ponudbam je potrebno definirati metapodatke kot so osnovna URL pot (angl. base URL) in dodatne poljubne atribute (npr. opis). Osnovna URL pot vodi do vira dejanskih podatkov znotraj organizacije, to je na primer končna točka REST strežnika znotraj organizacije. Ta pot se uporablja samo v podatkovni ravnini (ang. data plane) in ni nikoli razkrita ostalim udeležencem v podatkovnem prostoru. Poljubni atributi imajo dve kategorije, lahko so zasebni, kar pomeni, da jih vidi samo lastnik povezovalnika, ali pa javni, kar pomeni, da so vidni v podatkovnih katalogih udeležencev.
Tretji del, potreben za objavo ponudb, so definicije pogodb (angl. contract definitions), kjer se definirajo vzorci pogodb, ki se navezujejo na politiko dostopa, pogodbeno politiko in ponudbo. Več definicij pogodb z morebitnimi različnimi ponudbami se lahko navezuje na isto politiko, kar omogoča, da imajo organizacije nekaj politik, ki jih uporabljajo pri ponudbah. Definicija pogodbe v EDC je objekt *Criterion* v JSON-LD formatu [16]. Šele ob uspešnih definicijah pogodb so ponudbe, oziroma definicije pogodb, ki se navezujejo na ponudbe objavljene v podatkovni katalog povezovalnikov, kjer jih lahko odkrijejo ostali udeleženci podatkovnega prostora. Vsak povezovalnik ima en javno dostopni podatkovni katalog.

Pridobivanje podatkov

Po uspešni pridružitvi povezovalnikov v podatkovni prostor, lahko organizacije začnejo pridobivati dejanske podatke, ki jih ponujajo ostali udeleženci. Proces pridobitve podatkov je sestavljen iz naslednjih korakov [17] in prikazan na sliki 9:
Samodejno pridobivanje podatkovnih katalogov;
Pogajanje pogodbe;
Vzpostavitev procesa prenosa;
Prenos podatkov.

Prvi korak je pridobitev podatkovnih katalogov od odkritih udeležencev. EDC komponenta pajek na povezovalnikih potrošnikov, ki na podlagi lokalnega seznama vseh aktivnih ponudnikov podatkovnega prostora od vsakega ponudnika pridobi podatkovni katalog. Ko pajek od ponudnikov zahteva podatkovne kataloge,

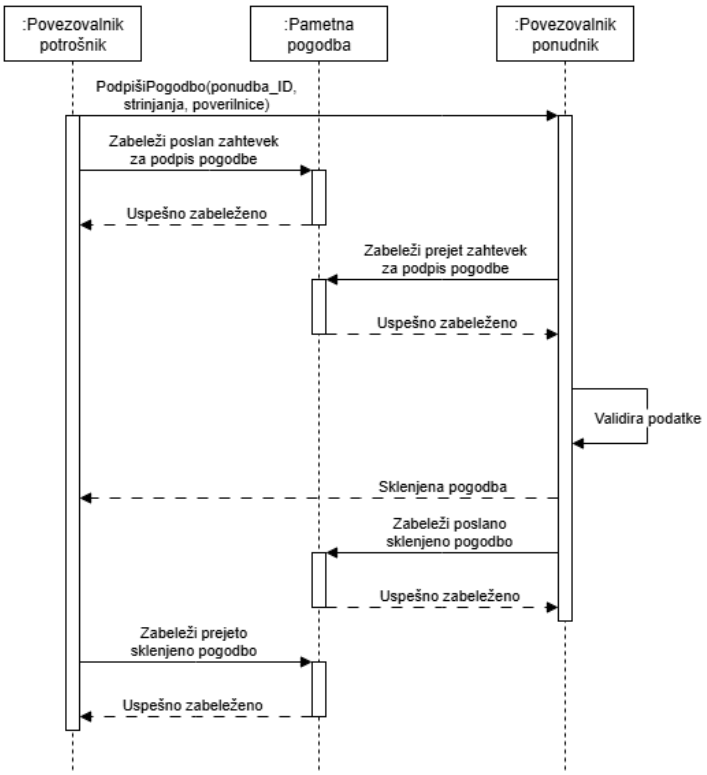
ponudniki vrnejo prilagojene podatkovne kataloge le s ponudbami, s katerimi politikami dostopa so potrošniki skladni. Ostalih ponudb posamezni potrošniki ne dobijo. Tako vsak DSX Engine povezovalnik periodično na nastavljene časovne intervale gradi lokalni predpomnjen katalog podatkovnega prostora. Glede na število aktivnih ponudnikov se dinamično paralelno sproži več instanc komponente pajek, za učinkovitejše in hitrejše posodobitve lokalnega kataloga podatkov.



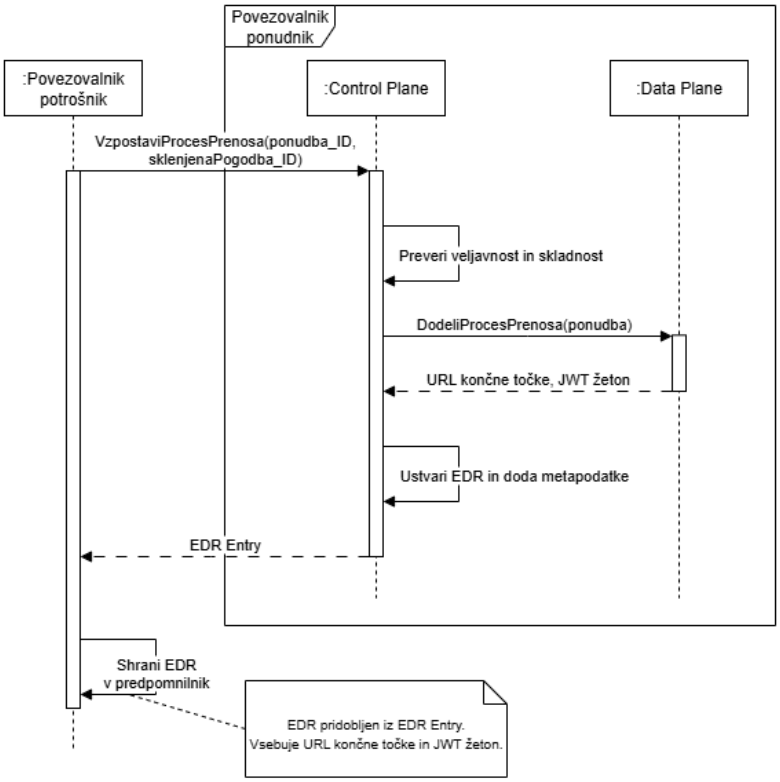
Slika 9: Posplošen diagram zaporedij samodejnega pridobivanja podatkovnih katalogov.

Drugi korak je pogajanje pogodb, med ponudniki in potrošniki. Posamezna pogodba se navezuje na natanko dva povezovalnika (eden v vlogi ponudnika, drugi v vlogi potrošnika). Ob izbiri ponudbe, potrošnik pošlje ponudniku zahtevek za podpis pogodbe. V tem zahtevku definira na katero ponudbo podatkov se navezuje in s katerimi politikami se strinja oz. je v skladu - to so dovoljenja, prepovedi, obveznosti. S tem potrošnik ponudniku posreduje tudi potrebne poverilnice. Ponudnik primerja pogodbeno politiko v definiciji pogodbe, ki se navezuje na ponudbo s politiko in poverilnicami iz potrošnikovega zahtevka. Ponudnik pričakuje, da se potrošnik strinja z vsemi zahtevami, ki so v pogodbeni politiki in prav tako preveri veljavnost in ustreznost podpisa pogodbe s strani izdajatelja poverilnic ter vsebino potrošnikovih poverilnic. V kolikor so poverilnice veljavne in vsebinsko ustrezne ter se je potrošnik strinjal z vsemi zahtevami v pogodbeni politiki, je pogodba med ponudnikom in potrošnikom uspešno sklenjena. Ta postopek se imenuje pogajanje pogodbe (angl. Contract negotiation). Pogajanja kot tudi sklenjene pogodbe (angl. Contract Agreement) razširitev DCH v povezovalnikih tako ponudnikov kot potrošnikov sproti beleži na verigo blokov. Vsak povezovalnik beleži svoje dogodke na verigo blokov in s tem zagotovi transparentnost in neizpodbitnost transakcij. Slika 10 prikazuje zaporedje sklepanja pogodbe.

Tretji korak pridobivanja podatkov je vzpostavitev procesa prenosa med določenim ponudnikom in potrošnikom. Po sklenjeni pogodbi, potrošnik ponudniku pošlje zahtevek za vzpostavitev procesa prenosa, kjer navede ponudbo in sklenjeno pogodbo na katero se navezuje. Ko ponudnik prejme zahtevek in ga uspešno preveri, dodeli proces prenosa enemu izmed razpoložljivih podatkovnih ravnin (ang. Data Plane) znotraj organizacije, preko katerega bo možno od zunaj dostopati do dejanskih podatkov. Ob uspešno vzpostavljenem procesu prenosa ponudnik ustvari Endpoint Data Reference (EDR), ki vsebuje informacije kje se nahajajo podatki (vključuje URL naslov do končne točke na podatkovnem posredniku ponudnika in JWT avtorizacijski žeton). EDR se vključno z dodatnimi metapodatki zapakira v zapis EDR Entry in kot odgovor pošlje potrošniku. Pridobljen zapis si potrošnik shrani lokalno. V kolikor JWT žetonu poteče veljavnost, potrošnik lahko ponovno ponudniku pošlje zahtevek in pridobi nov JWT žeton. Proces je predstavljen na sliki 11.



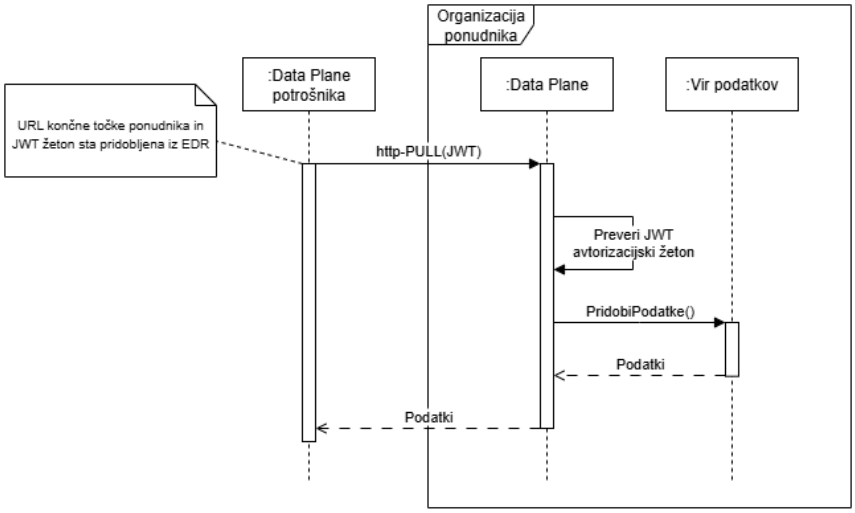
Slika 10: Posplošen diagram zaporedij pogajanja pogodbe.



Slika 11: Posplošen diagram zaporedij vzpostavitve procesa prenosa.

Četrti, zadnji korak pridobivanja podatkov v podatkovnem prostoru je prenos podatkov od ponudnika do določenega potrošnika. Ko potrošnik iz EDR razbere URL naslov končne točke podatkovnega posrednika in

avtorizacijski žeton JWT za dostop, lahko na prejet URL naslov pošlje zahtevek GET, kjer v glavi HTTP zahtevka doda atribut *Authorization*, katerega vrednost nastavi na prejet žeton JWT. Podatkovna ravnina povezovalnika ponudnika prejme zahtevek, preveri avtorizacijski žeton in po uspešni avtorizaciji pridobi dejanske podatke iz podatkovnega vira znotraj organizacije (npr. iz REST strežnika ali podatkovne baze, kjer so dejanski podatki shranjeni) ter jih posreduje potrošniku. Ta tip pridobivanja podatkov se imenuje Consumer-PULL in v zgoraj opisanem primeru poteka preko protokola HTTPS. Podprt je tudi Provider-PUSH, kjer ob vzpostavitvi procesa pridobitve podatkov podamo URL do našega ponora podatkov, na katerega bo ponudnik samodejno naložil podatke na način, kot je na primer pretakanje v realnem času (angl. streaming) ali pa periodično odlaganje podatkov. Slika 12 predstavlja proces prenosa podatkov.



Slika 12: Diagram zaporedij prenosa podatkov.

4.3. Tokenizacija

Koncept

Ena izmed osnovnih vodil podatkovnih prostorov je omogočanje t. i. **podatkovne ekonomije**. Znotraj podatkovnega prostora lahko ponudniki podatke ne le ponudijo temveč prodajajo potrošnikom. DSX Engine omogoča monetizacijo podatkov znotraj podatkovnega prostora z uporabo pametne pogodbe za DSX žetone in pametne pogodbe za izvajanje plačil na tehnologiji veriženja blokov.

Podrobnosti kako deluje tokenizacija znotraj podatkovnega prostora so odvisne od implementacije in odločitev podatkovnega prostora.

DSX žeton (ERC-20)

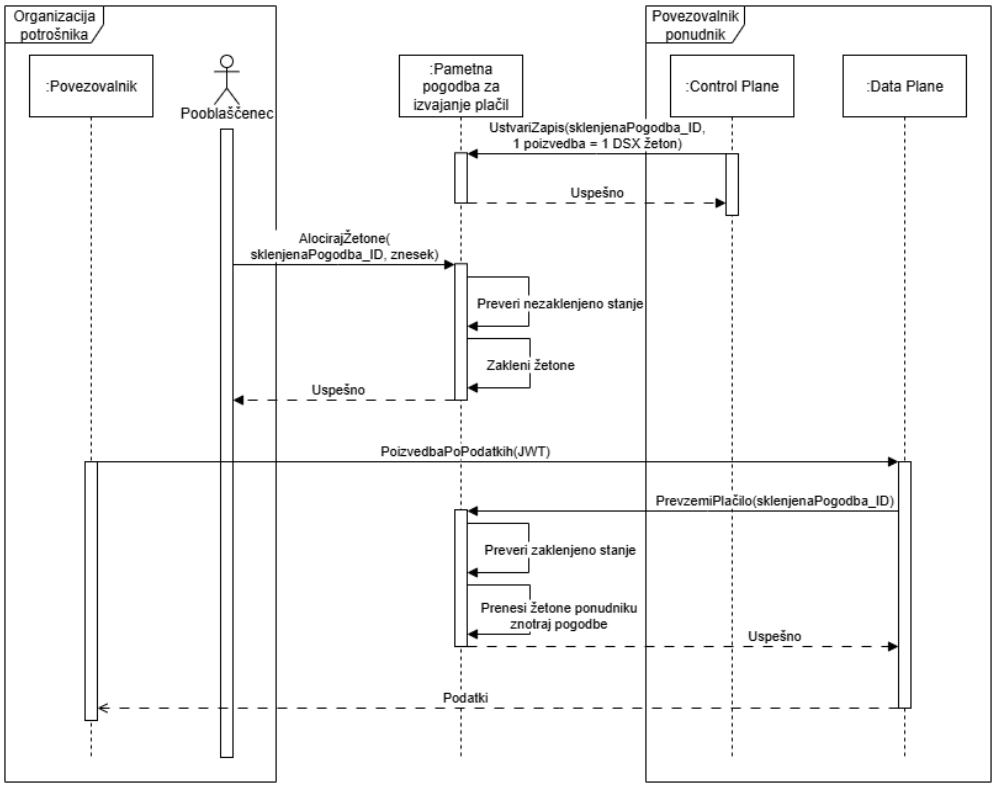
DSX žeton (angl. DSX token) temelji na pametni pogodbi, ki temelji na standardu zamenljivih žetonov OpenZeppelin ERC-20 na verigi blokov Ethereum [18]. DSX žetoni delujejo kot digitalna valuta za namene prodaje podatkov med ponudniki in potrošniki znotraj podatkovnega prostora. S tem zagotovimo transparentnost in ne-zanikanje transakcij.

Skrbniki podatkovnega prostora opredelijo, kako bodo DSX žetone delili oz., kako jih bodo predajali/prodajali udeležencem podatkovnega prostora. V spodaj opisanem primeru udeleženci podatkovnega prostora lahko DSX žetone zamenjajo z žetoni ETH in obratno, po tečaju, ki ga določi podatkovni prostor.

Ob objavi podatkov v podatkovni katalog ponudniki v pogodbeni politiki (angl. Contract Policy) pod obveznosti (angl. obligation) določijo med drugimi zahtevami tudi način obračunavanja podatkov. Spodaj je opisan primer obračunavanja na poizvedbo, na primer 1 DSX žeton na poizvedbo.

Potrošniki, ki želijo dostopati do plačljivih podatkov v podatkovnem prostoru, morajo na pametno pogodbo za izvajanje plačil naložiti žetone DSX (npr. 3 DSX žetone).

Pri pridobivanju plačljivih podatkov povezovalniki potrošnikov in ponudnikov sklenejo pogodbe (angl. Contract Agreement), nakar povezovalniki ponudnikov samodejno v pametno pogodbi za izvajanje plačil kličejo funkcijo, ki ustvari zapis na podlagi katerega potekajo plačila.
Vsak zapis se navezuje na natanko eno sklenjeno pogodbo, kjer je zabeležen tudi način obračunavanja (npr. 1 poizvedba = 1 žeton DSX).
Potrošniki morajo pred poizvedbo za podatke v pametni pogodbi klicati funkcijo s katero dovolijo, da se poljuben znesek izmed vseh žetonov, ki so jih naložili na pametno pogodbo, alocira za plačila določene sklenjene pogodbe za podatke. Potrošniki lahko kadarkoli ta sredstva tudi odklenejo.
Ko povezovalniki potrošnikov pošljejo zahtevke po podatkih, podatkovna ravnina (ang. Data Plane) ponudnikov v pametni pogodbi kličejo funkcijo za prevzem plačila. Funkcija v pametni pogodbi preveri, ali je v zaklenjenem znesku, ki se navezuje na določeno sklenjeno pogodbo, dovolj žetonov. V kolikor je žetonov dovolj, se ustrezna količina žetonov, glede na način obračunavanja, prenese ustreznemu ponudniku znotraj pogodbe. Podatkovni posredniki ponudnikov nato posredujejo podatke potrošnikom. Proces je predstavljen na sliki 13.



Slika 13: Diagram zaporedij trgovanja podatkov.

Udeleženci podatkovnega prostora lahko nezaklenjen znesek, ki ga imajo na pametni pogodbi zamenjajo v ETH žetone po tečaju, ki ga je določil podatkovni prostor (npr. 1 DSX žeton = 0,0005 ETH žetona).
V prihodnosti bo možno nastaviti tudi druge načine obračunavanja podatkov. Na primer 1 DSX žeton za 1 GB prenešenih podatkov, 10 DSX žetonov za en mesec poizvedovanja in ostale načine, ki jih potrebujejo ponudniki za njihovo poslovanje.

4.4. Primerjava

Za boljše razumevanje razlike med osnovno implementacijo EDC in našo nadgradnjo DSX Engine, v nadaljevanju podajamo primerjalni pregled ključnih funkcionalnosti in pristopov. Primerjava jasno pokaže, kako DSX Engine gradi na temeljnih zmožnostih EDC in jih nadgrajuje z dodatnimi mehanizmi decentralizacije, odpornosti in podpore novim poslovnim modelom.

Tabela 1: Primerjava med osnovno implementacijo EDC in DSX Engine.

Lastnost / Funkcionalnost	EDC (osnovna implementacija)	DSX Engine (nadgradnja EDC)
Registracija udeležencev	Centralizirana ali preko posrednika	Decentralizirana registracija preko pametnih pogodb (DDE)
Odkrivanje udeležencev	Federativno, odvisno od vnaprej znanih vozlišč	Dinamično, bazirano na pametnih pogodbah, avtomatsko odkrivanje
Upravljanje identitet	Podpora did:web	Podpora DID:ethr, načrtovana DID:ebsi; decentralizirana denarnica identitet
Beleženje dogodkov / revizijska sled	Lokalno pri vsakem udeležencu	Decentralizirana klirinška hiša (DCH) na verigi blokov
Podpora pametnim pogodbam	Ni vgrajene neposredne podpore	Neposredna podpora pametnim pogodbam na platformi Ethereum
Tokenizacija in plačila	Ni privzete podpore	DSX ERC-20 žeton in pametne pogodbe za obračunavanje
Politični mehanizmi	Standardne politike EDC	Politike po meri na osnovi preverljivih poverilnic (VC)

5 Pilotiranje

Kot pilotni primer implementacije konceptov podatkovnih prostorov in uporabe povezovalnika DSX Engine izpostavljamo DIH AGRIFOOD Data Space (DADS) – podatkovni prostor za agroživilski sektor. DADS je pobuda Digitalnega inovacijskega stičišča za kmetijstvo in prehrano (DIH Agrifood) v Sloveniji, ki vzpostavlja federativno platformo za deljenje podatkov v agroživilstvu. Po definiciji gre za "federalno organizirano in upravljano platformo za deljenje podatkov iz agriživilskega sektorja, kjer se podatki delijo na osnovi vnaprej definiranih pogojev uporabe" [19]. Ključni cilj DADS je povezati različne deležnike v agroživilskem ekosistemu – od kmetijskih gospodarstev, zadrug, svetovalnih služb, živilskopredelovalnih podjetij do javnih ustanov – in jim omogočiti nadzorovano izmenjavo podatkov v skupno korist (npr. za optimizacijo pridelave, sledljivost hrane, okoljsko spremljanje itd.).

DADS je zgrajen skladno z evropskimi smernicami za podatkovne prostore: ima upravljavski model (governance), utemeljen na etičnih načelih in evropskih regulativah (npr. Data Act, Evropska strategija za podatke), uporablja standarde, ki jih definira IDSA, ter sledi načelom, opredeljenim v okviru projekta OPEN DEI (Open Digital Ecosystems, ki je pripravil oblikovalska načela za podatkovne prostore) [20]. Tehnično gledano DADS vključuje vse osnovne gradnike podatkovnega prostora: identitetni sistem, katalog ponudb, povezovalnike (DSX Engine) pri udeležencih, modul za upravljanje soglasij in politik uporabe ter mehanizme za beleženje in sledljivost. DIH Agrifood kot iniciator projekta nastopa tudi kot koordinator podatkovnega prostora – skrbi za vzdrževanje pravil (t. i. knjiga pravil podatkovnega prostora) in nudi podporo udeležencem pri priključitvi. Pomembno je izpostaviti, da DADS ni centralizirana baza podatkov: podatki ostajajo pri posameznih ponudnikih podatkov (npr. pri kmetih, podjetjih ali institucijah), med tem ko DADS poskrbi za infrastrukturo, da ti podatki najdejo pot do porabnikov podatkov pod znanimi in nadzorovanimi pogoji [21].

V okviru DADS je uporabljen povezovalnik DSX Engine. Platforma DADS se ponaša s tem, da ne gre zgolj za deljenje statičnih naborov podatkov, temveč omogoča povezovanje do živih podatkovnih virov (npr. realno časovni podatki preko API-jev). To je bilo demonstrirano, ko je DADS uspešno povezal podatke Mestne občine Murska Sobota (MOMS). MOMS je preko svojega povezovalnika, ki teče na njeni infrastrukturi, registriraa REST API storitve z okolijskimi in GIS podatki v katalog DADS. Prav taka pa lahko potrošniki podatkov le te v realnem času uporabljajo preko DADS ob priključitvi v podatkovni prostor in uporabi povezovalnika DSX Engine [21].
V pilotnem primeru DADS uporabljamo konzorcijsko omrežje EduCTX, ki temelji na Hyperledger Besu. Omrežje je že vzpostavljeno v okviru DIH Agrifood in služi kot zaupanja vredna platforma za izvajanje pametnih pogodb, beleženje dogodkov in upravljanje identitet. Prednost te izbire je, da so vsi tehnični in organizacijski procesi za upravljanje konzorcijskega omrežja verige blokov že vpeljani, kar je skrajšalo čas implementacije in zmanjšalo stroške pilotiranja.
DIH Agrifood prav tako zagotavlja centralni portal, kjer najprej vsaka organizacija registrira oziroma poveže svoj DSX Engine povezovalnik. Nato ga preko tega portala upravlja. Povezovalnik vsake organizacije skrbi za pridobitev kataloga celotnega podatkovnega prostora, omogoča objavo podatkov organizacije v podatkovni prostor, določitev pravil in pogojev dostopa do podatkov, ter sklepanje pogodb za pridobitev podatkov z drugimi udeleženci podatkovnega prostora DADS, ki uporabljajo DSX Engine povezovalnik.
Primer DADS ponazarja vrednost podatkovnih prostorov v praksi. Kmetijski podatki, ki so bili prej razpršeni in težko dostopni, so zdaj na voljo vsem udeležencem, ki izpolnjujejo pogoje uporabe, hkrati pa kmetje kot lastniki podatkov ohranijo suverenost – sami odločajo, katere podatke delijo in pod kakšnimi pogoji. Implementacija DSX Engine povezovalnikov v DADS je pokazala, da je tehnologija dovolj zrela za realne primere uporabe, hkrati pa je razkrila tudi nekatera praktična vprašanja. Med njimi so pomembna interoperabilnost (povezovanje z drugimi podatkovnimi prostori – npr. v prihodnosti povezava DADS z evropskim podatkovnim prostorom za kmetijstvo), uporaba standardne semantike (v DADS so se morali dogovoriti o skupnih podatkovnih shemah za npr. podatke o pridelkih, kar kaže na potrebo po mednarodnih standardih) in pravno zagotavljanje skladnosti (pogodbe, sklenjene preko sistema, je treba uskladiti z obstoječim pravnim okvirom varstva podatkov in obligacij).
DADS sodeluje tudi v mednarodnih projektih (Horizon Europe projekti, kot so DIVINE, ZeroW, ipd. [21]), kjer deluje kot primer dobre prakse pri postavljanju podatkovnih prostorov. Izkušnje, pridobljene z DADS, se vračajo v nadaljnji razvoj DSX Engine – pilotiranje je namreč razkrilo, katere funkcionalnosti so najbolj koristne in katere je treba še izboljšati. Tako je DADS pomemben korak na poti od koncepta do polnopravne operativne platforme za podatkovno ekonomijo v kmetijstvu.

6 Zaključek

V tem članku smo predstavili DSX Engine kot primer decentraliziranega povezovalnika podatkovnih prostorov, ki nadgrajuje referenčno odprtokodno ogrodje Eclipse Dataspace Components z mehanizmi za decentralizirano zaupanje in avtomatizirano sklepanje pogodb. Pokazali smo, kako uvedba pametnih pogodb, decentraliziranih identitet in tokenizacije omogoča odpravo osrednjih točk odpovedi ter povečuje stopnjo zaupanja in varnosti v med-organizacijski izmenjavi podatkov. Predstavljene razširitve, kot so DDE, DCH in podpora za decentralizirane identifikatorje omogočajo popolnoma decentralizirano registracijo in odkrivanje udeležencev, neizpodbitno revizijsko sled ter interoperabilnost med heterogenimi sistemi in organizacijami.
Praktično vrednost in zrelost DSX Engine smo prikazali na primeru implementacije v pilotnem podatkovnem prostoru DIH AGRIFOOD Data Space (DADS), kjer rešitev omogoča kmetijskim in okoljskim deležnikom varno, nadzorovano in sledljivo izmenjavo podatkov ob ohranjanju njihove suverenosti.
V prihodnje nameravamo nadaljevati z razvojem DSX Engine v smeri večje skladnosti z evropskimi specifikacijami, kot so Gaia-X in EBSI DID, razširiti podporo semantičnim standardom za interoperabilnost med sektorji, omogočiti več načinov prenosa podatkov in trgovanja s podatki ter izboljšati uporabniško izkušnjo upravljanja in nadzora nad podatkovnimi povezovalniki. Prav tako bomo na podlagi izkušenj iz pilotnih

implementacij naslavljali odprta vprašanja pravne skladnosti in standardizacije pogodb ter prispevali k vzpostavitvi zaupanja vredne podatkovne ekonomije v evropskem in mednarodnem prostoru.

Literatura

[1] M. Spiekermann, „Eclipse Dataspace Connector: Trusted Data Sharing With Sovereignty," 20. oktober 2021. [Elektronski]. Dostopno na: https://newsroom.eclipse.org/eclipse-newsletter/2021/october/eclipse-dataspace-connector-trusted-data-sharing-sovereignty. [Poskus dostopa 26. maj 2025].

[2] T. Dam, L. D. Klausner, S. Neumaier in T. Priebe, „A Survey of Dataspace Connector Implementations," 9. januar 2024. [Elektronski]. Dostopno na: https://arxiv.org/pdf/2309.11282. [Poskus dostopa 26. maj 2025].

[3] N. Kliček, M. Šestak in M. Turkanović, „Inovacije v arhitekturah podatkovnih prostorov," september 2024. [Elektronski]. Dostopno na: https://plus.cobiss.net/cobiss/si/sl/bib/ktfmb/207213827. [Poskus dostopa 26. maj 2025].

[4] European Commision, „A European strategy for data," 9. april 2025. [Elektronski]. Dostopno na: https://digital-strategy.ec.europa.eu/en/policies/strategy-data. [Poskus dostopa 26. maj 2025].

[5] International Data Spaces Association, „IDS-RAM 4," [Elektronski]. Dostopno na: https://docs.internationaldataspaces.org/ids-knowledgebase/ids-ram-4. [Poskus dostopa 26. maj 2025].

[6] I. Chulani, „Understanding the IDS Reference Architecture Model," 2. oktober 2024. [Elektronski]. Dostopno na: https://internationaldataspaces.org/understanding-the-idsa-reference-architecture-model. [Poskus dostopa 26. maj 2025].

[7] International Data Spaces Association, „IDSA Rulebook," [Elektronski]. Dostopno na: https://docs.internationaldataspaces.org/ids-knowledgebase/idsa-rulebook. [Poskus dostopa 26. maj 2025].

[8] Eclipse Foundation, „EDC Documentation," [Elektronski]. Dostopno na: https://eclipse-edc.github.io/documentation/. [Poskus dostopa 26. maj 2025].

[9] J. Marino, „The Eclipse Dataspace Connector Architecture and Principles," 31. januar 2022. [Elektronski]. Dostopno na: https://raw.githubusercontent.com/eclipse-edc/Collateral/main/Events/Conferences/2022-01%20EDC%20Conference/2022-01-31%20EDC%20-%20Architecture%20and%20Concepts.pdf. [Poskus dostopa 30. junij 2025].

[10] A. Bertagnolli, P. Koen, M. Kollenstart, P. Latzelsperger, J. Marino, J. Pampus, R. Rajani, E. Risa, M. Ruland, B. Scholtes, N. Schulte, M. Spiekermann, S. Steinbuss, A. Weiß in H. Yildiz, „Eclipse Decentralized Claims Protocol," Eclipse Foundation, 19. junij 2025. [Elektronski]. Dostopno na: https://eclipse-dataspace-dcp.github.io/decentralized-claims-protocol/v1.0-RC4. [Poskus dostopa 3. julij 2025].

[11] Gaia-X, „Building a Dataspace: Technical Overview," april 2023. [Elektronski]. Dostopno na: https://www.gaia-x.at/wp-content/uploads/2023/04/WhitepaperGaiaX.pdf. [Poskus dostopa 26. maj 2025].

[12] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele in C. Allen, „Decentralized Identifiers (DIDs) v1.0," World Wide Web Consortium (W3C), 19. julij 2022. [Elektronski]. Dostopno na: https://www.w3.org/TR/did-1.0. [Poskus dostopa 20. junij 2025].

[13] P. Koen, M. Kollenstart, J. Marino, J. Pampus, A. Turkmayali, S. Steinbuss in A. Weiß, „Dataspace Protocol 2025-1-RC2," Eclipse, 19. junij 2025. [Elektronski]. Dostopno na: https://eclipse-dataspace-protocol-base.github.io/DataspaceProtocol/2025-1-RC2/. [Poskus dostopa 24. junij 2025].

[14] D. i. H. Bastiaansen, G. Bramm, J. Ceballos, M. Gall in M. Kollenstart, „Specification: IDS Clearing House," International Data Spaces Association, 1. april 2020. [Elektronski]. Dostopno na: https://internationaldataspaces.org/wp-content/uploads/dlm_uploads/IDSA-White-Paper-Specification-IDS-Clearing-House-.pdf. [Poskus dostopa 23. junij 2025].

[15] W3C, „ODRL Information Model 2.2," World Wide Web Consortium (W3C), 15. februar 2018. [Elektronski]. Dostopno na: https://www.w3.org/TR/odrl-model/. [Poskus dostopa 26. junij 2025].

[16] Eclipse Foundation, „EDC Developer's Handbook," 29. marec 2024. [Elektronski]. Dostopno na: https://github.com/eclipse-edc/docs/blob/main/developer/handbook.md. [Poskus dostopa 26. junij 2025].

[17] Eclipse Foundation, „EDC transfer samples,“ 25. junij 2025. [Elektronski]. Dostopno na: https://github.com/eclipse-edc/Samples/tree/main/transfer. [Poskus dostopa 27. junij 2025].

[18] OpenZeppelin, „ERC-20,“ OpenZeppelin, [Elektronski]. Dostopno na: https://docs.openzeppelin.com/contracts/5.x/erc20. [Poskus dostopa 4. julij 2025].

[19] DIH AGRIFOOD, „DIH AGRIFOOD Data Space (DADS) PowerPoint,“ 18. januar 2024. [Elektronski]. Dostopno na: https://www.gov.si/assets/ministrstva/MDP/Dokumenti/Podatkovni-prostori/DIH-Agrifood.pdf. [Poskus dostopa 26. maj 2025].

[20] DIH AGRIFOOD, „DIH AGRIFOOD Data Space (DADS),“ 2025. [Elektronski]. Dostopno na: https://dih-agrifood.com/dih-agrifood-data-space. [Poskus dostopa 26. maj 2025].

[21] M. Bunderla, „DADS Milestone - MOMS Collaboration,“ december 2024. [Elektronski]. Dostopno na: https://www.linkedin.com/posts/miran-bunderla-a11a47b5_important-announcement-from-the-dih-agrifood-activity-7262825514682269696-bd8L. [Poskus dostopa 26. maj 2025].

Varna shramba uporabniških overitvenih podatkov v jedru 5G

Miha Rihtaršič, Uroš Brovč, Urban Zaletel

Kontron d.o.o, Kranj, Slovenija
miha.rihtarsic@kontron.com, uros.brovc@kontron.com, urban.zaletel@kontron.com

Varna hramba, obdelava in administracija občutljivih podatkov je zanimiv izziv, ki ima lahko različne rešitve glede na način obdelave, uporabniški namen ter stopnjo zahtevane varnosti. V 5G sistemih varnostna zahteva predpisuje zaščito občutljivih overitvenih podatkov med hranjenjem ter njihovo varno obdelavo med uporabo. V članku predstavljamo tehnologijo zaupnega procesiranja Intel® SGX, ki omogoča splošno obdelavo podatkov z zaščito pred okolico. Opisujemo način, kako z uporabo SGX zagotovimo močno varnost overitvenih podatkov in kako je potrebno prilagoditi administrativne naloge, da sovpadajo s tem načinom delovanja. Tak pristop je še posebej primeren za uporabo v mobilnih postavitvah 5G jeder, kjer obstaja večja nevarnost fizičnega dostopa do sistema — na primer pri jedru, nameščenem v vojaškem vozilu ali nahrbtniku za hitro taktično postavitev omrežja na terenu. S tem pristopom lahko tudi takšne postavitve z visoko izpostavljenostjo zadostijo strogim varnostnim zahtevam sodobnih 5G omrežij.

Ključne besede:
zaupno procesiranje,
5G,
overitev,
šifriranje,
varnost.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.12

1 Uvod

Sistem 5G jedro za delovanje potrebuje neprekinjen dostop do baze naročniških overitvenih podatkov, ki so tipično občutljive narave. Izpostavljenost teh podatkov — bodisi zaradi varnostne ranljivosti, bodisi zaradi zlonamernih dejanj — lahko vodi do resnih posledic, glede na obseg, kritičnost in javno vidnost 5G infrastrukture.

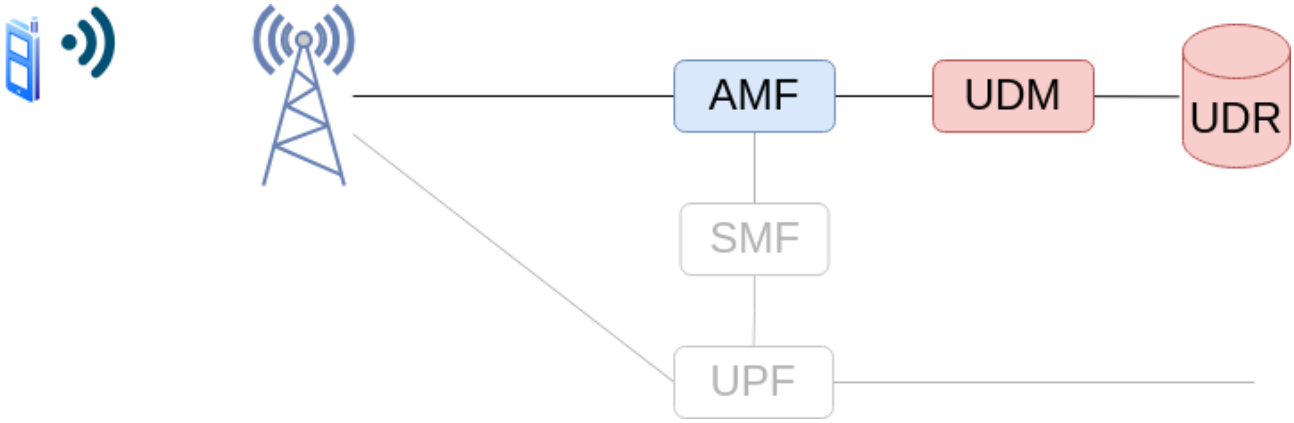
Posebej izpostavljen primer je uporaba prenosnih 5G jeder, ki delujejo zunaj klasičnih podatkovnih centrov – na primer jedro vgrajeno v vojaško vozilo ali nahrbtnik, namenjeno hitri postavitvi zasebnega omrežja v kriznih situacijah. V takšnih okoljih so naprave pogosto fizično dostopne napadalcem, kar še dodatno poveča tveganje za kompromitacijo zaupnih podatkov.

S pomočjo Intel® Software Guard Extensions in podobnimi tehnologijami smo razvili pristop, kjer lahko samo ena komponenta dostopa do nešifrirane verzije overitvenih podatkov in jim dodatno utrdili varnost pred več tipi napadov, zlasti pred napadalci s fizičnim dostopom do sistema.

2 Overitev v 5G

Vsaka uporabniška naprava (UE) mora 5G jedru dokazati verodostojnost svoje identitete, preden od 5G sistema zahteva omrežne storitve. Ta postopek overitve temelji na tem, da si USIM modul znotraj naprave in 5G jedro delita skupno skrivnost. Te skrivnosti hranita pri sebi in jih ne pošiljata neobdelane. 5G jedro jo uporabi pri ustvarjanju kriptografskega izziva, ki ga je mogoče rešiti samo lastnikom te iste skrivnosti. Izziv pošlje UE-ju, ki pa je praviloma edina naprava poleg 5G jedra, ki ima to skrivnost. Če reši izziv in pošlje pravilni odgovor, ga 5G sistem smatra kot dokaz identitete. Hkrati je en izmed rezultatov tega izziva simetričen ključ, ki si ga po overitvi lastita samo 5G omrežje in UE, in ga uporabljata za šifriranje nadaljnje komunikacije [1].

2.1. Sistem 5G



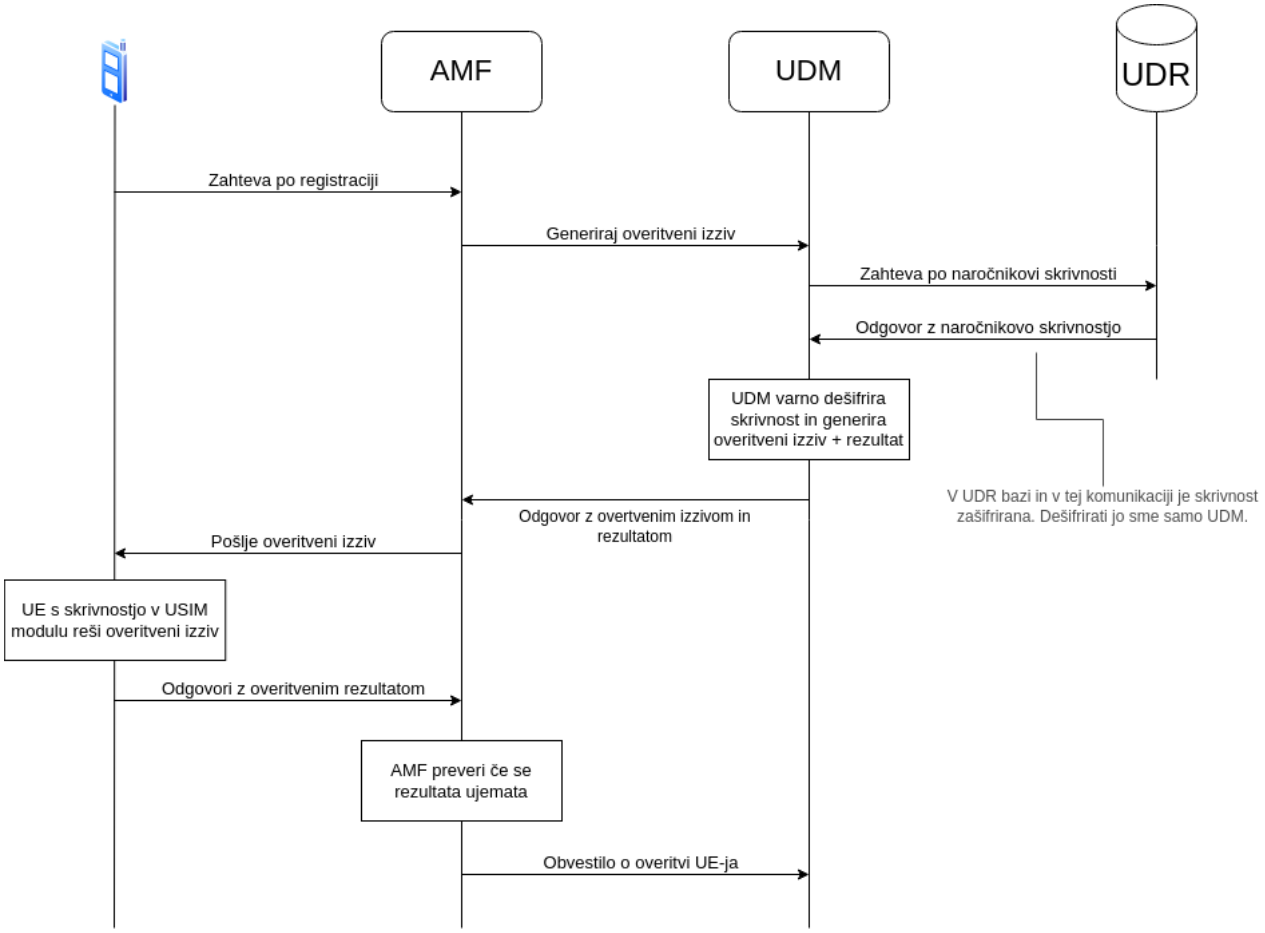
Slika 1: 5G omrežje.

5G sistem je sestavljen iz več elementov, kot je prikazano na sliki 1. Na levi sta ponazorjena UE in gNB antena, na desni so pa prikazani elementi 5G jedra. Ti tipično delujejo kot strežniki in med seboj komunicirajo preko mrežnih protokolov, največkrat preko HTTP/2. Osredotočamo se zlasti na AMF, UDM in UDR, ki sodelujejo pri overitvi UE. AMF se smatra kot vstopna točka v 5G jedro. UE-ji preko njega pridobivajo dostop do omrežnih storitev. UDM je pa med drugim zadolžen za ustvarjanje overitvenih izzivov iz podatkov, ki jih pridobi od UDR-ja. AUSF-ja, ki overitveni izziv priredi v interesu ohranjanja UE-jeve zasebnosti in overitven izziv tudi preveri, tukaj posebej ne izpostavljamo. Izziv (RAND) ter ostali avtentikacijski podatki se izračunajo v okviru komponente

ARPF, ki je del UDM. Namen poenostavitve je, da se osredotočimo na vidike, povezane z upravljanjem in zaščito ključev, kar spada v domeno UDM, medtem ko je AUSF v tem kontekstu obravnavan posredno.

Poleg AMF, UDM in UDR so v 5G jedru prisotne še ostale omrežne funkcije, ki omogočajo razne druge storitve. Kot primer sta na sliki 1 narisana SMF in UPF, ki overjenim UE-jem na AMF-jevo zahtevo nudita omrežno storitev.

2.2. Potek overitve



Slika 2: Potek overitve.

Na sliki 2 je na visokem nivoju prikazano, kako poteka overitev UE-ja v 5G omrežje [2]. Ob registracijski zahtevi UE-ja AMF pošlje UDM-ju zahtevo za pridobitev overitvenega izziva za tega uporabnika. UDM od UDR-ja pridobi zašifrirano skrivnost tega UE-ja, jo dešifrira ter na njeni podlagi ustvari izziv in pripadajoči pričakovani rezultat. Oboje posreduje AMF-ju, ki nato UE-ju pošlje le izziv. UE uporabi svojo lokalno skrivnost, shranjeno v USIM-modulu, za izračun odgovora na izziv in ga pošlje AMF-ju. AMF preveri, ali se prejeti odgovor ujema s pričakovanim rezultatom, ki ga je prejel od UDM-ja. Če se vrednosti ujemajo, to potrjuje, da UE razpolaga s pravilno skrivnostjo, kar pomeni, da je uspešno overjen.

5G sistem je fleksibilen in omogoča tudi gostovanje. V tem primeru UE komunicira z AMF-jem iz gostujoče mreže, AMF pa z UDM-jem iz UE-jeve domače mreže. Edino ta UDM ima namreč dostop do UE-jevih podatkov.

2.3. Varnost overitvenih podatkov

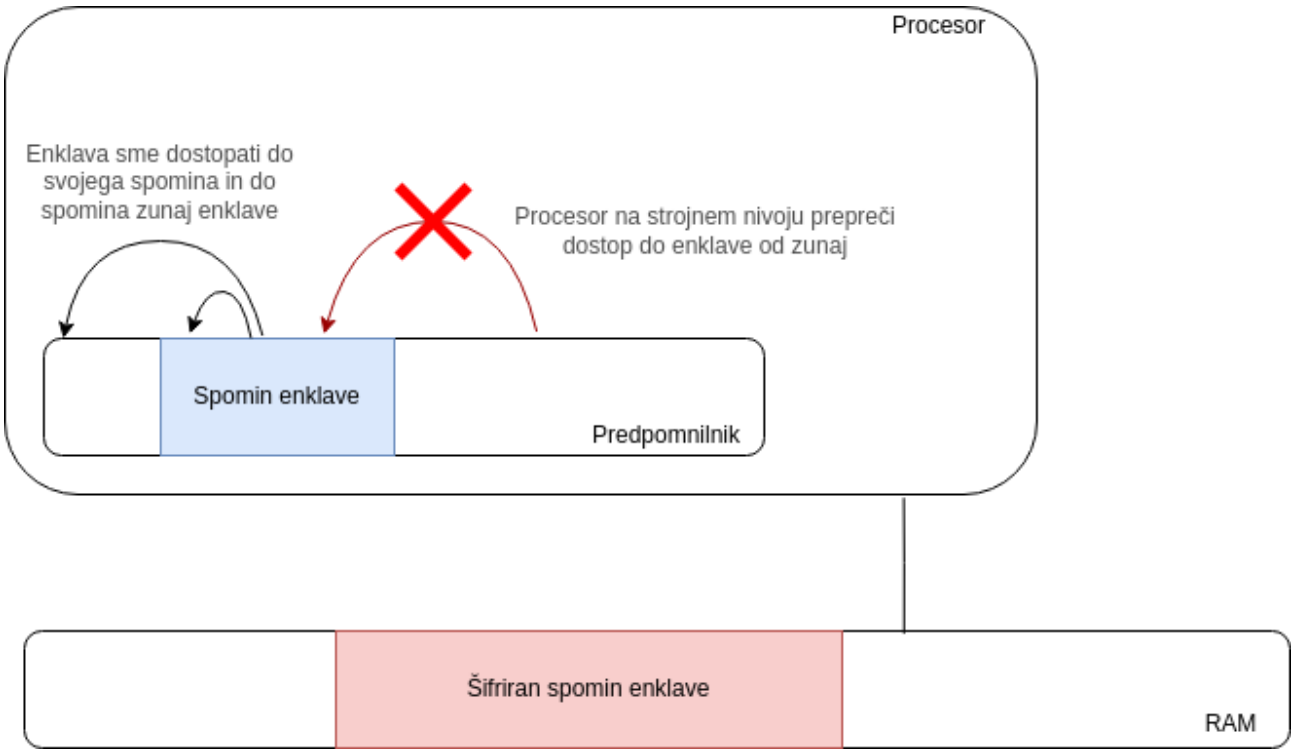
UDR-jeva baza UE-jevih skrivnosti je zašifrirana tako, da lahko podatke dešifrira samo UDM. Zato UDR-jeva baza sama po sebi ni občutljiva. UDM iz dešifrirane verzije skrivnosti ustvari overitveni izziv in rezultat, nato pa skrivnost izbriše. Velja tudi, da se iz izziva in rezultata ne da rekonstruirati skrivnosti, zato tudi nista sama po sebi občutljiva podatka. Najbolj občutljiv podatek je dešifrirana skrivnost, s katero ima pa opravka samo UDM, zato se osredotočamo zlasti na varno delovanje UDM-ja.

3 Predstavitev Intel® Software Guard Extensions (SGX)

Intel® Software Guard Extensions (SGX) je funkcionalnost na nekaterih Intelovih procesorskih modelih, ki omogoča zaščiteno zaupno procesiranje. Z njo definiramo enklave, to so območja računalniškega spomina, ki so na nivoju strojne opreme zaščitene pred okolico, vključno z napadalcem s fizičnim dostopom in ostalo programsko opremo na istem procesorju [3].

3.1. Enklave

Navaden proces v sodelovanju z operacijskim sistemom definira enklavo. Vanjo vpiše programsko kodo in podatke, ki so potrebni za varno procesiranje. Nato s SGX ukazom inicializira enklavo, po tem pa lahko vanjo večkrat vstopi in izstopi, kar pomeni, da začne poganjati kodo znotraj enklave.



Slika 3: Vizualizacija enklave.

Po inicializaciji je enklava zaščitena pred branjem in pisanjem iz okolice, kot je ponazorjeno na sliki 3. Procesor prepreči posege ostalim procesom na strojnem nivoju. S tem je izboljšana zaščita pred programskimi vdori v katerikoli proces zunaj enklave. Pomembno pa je tudi, da je v veliki meri poseg preprečen tudi napadalcem s fizičnim dostopom do sistema. Spomin v enklavi nikoli ne zapusti procesorja nešifriran. Preden se podatki zapišejo v RAM, se znotraj procesorja zašifrirajo in njihova celovitost se zaščiti. Napadalec z dostopom do fizičnega RAM-

a ali vodil med RAM-om in procesorjem ne more brati podatkov. In tudi če podatke skuša prirediti, CPU to zanesljivo zazna kot okvaro celovitosti in enklavo podre.

Če želi napadalec s fizičnim dostopom brati ali prirejati podatke, mora posegati v sam procesor. Zaradi majhnosti vezji na procesorju se to smatra kot zelo drag poseg [5]. Cenejši posegi so destruktivni in jih ni težko zaznati, ker mora napadalec za dalj časa vzeti procesor v analizo, preden ga prebere in nadomesti z delujočo kopijo.

3.2. Funkcionalnosti enklave

Koda, ki se izvaja znotraj enklave, je skoraj popolnoma poljubna in ima samo nekaj specifičnih omejitev zaradi tehničnih razlogov [4]. Dodatni stroški vstopov v enklavo in poganjanja enklav so tudi majhni. To nam omogoča fleksibilnost pri načrtovanju varnostnih sistemov s pomočjo SGX. Na primer, programiramo lahko poljubne algoritme za šifriranje in izpeljavo simetričnih ali asimetričnih ključev, operiramo lahko s poljubnim številom ključev glede na podatke in ustvarjamo kriptografske izzive po poljubnih standardih.

Poleg tega nam SGX v okviru enklave omogoča razne zanimive dodatne funkcionalnosti. Pomemben primer je ustvarjanje tako imenovanega pečatnega ključa. Znotraj enklave koda s SGX ukazom pridobi pečatni ključ, za katerega SGX zagotavlja, da ga je možno ustvariti samo v tej specifični enklavi na tem specifičnem procesorju. Z njim lahko enklava zašifrira občutljive podatke ter jih zapiše na nezaščiteno mesto zunaj enklave, na primer na disk z namenom, da po ponovnem zagonu sistema podatke prebere ter dešifrira s pečatnim ključem.

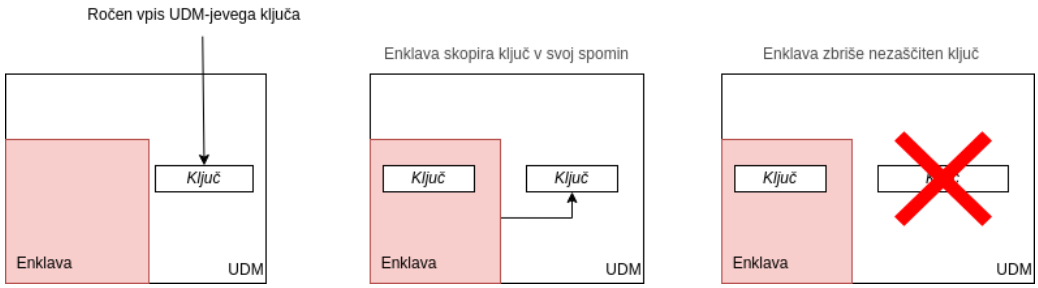
4 Zaščita overitvenih podatkov s pomočjo SGX

5G jedro hrani overitvene podatke svojih UE-jev v UDR-jevi naročniški bazi, ki mora biti ves čas na voljo za neprekinjeno zagotavljanje storitev. Baza vsebuje naročnikove skrivnosti, ki so eden najbolj občutljivih podatkov v 5G sistemu. Napadalec, ki mu uspe pridobiti te podatke, lahko namreč pooseblja tuje UE-je in s tem pride do nepooblaščenih storitev in podatkov ali povzroča različne vrste škode drugim UE-jem in celotnemu omrežju. Zato imamo v interesu čim močnejšo zaščito teh podatkov.

5G standard predpisuje, da so skrivnosti v UDR-ju šifrirani in da ima UDM možnost te skrivnosti na varen način dešifrirati. S tem je UDR-jeva baza overitvenih podatkov zaščitena pred branjem. Bazo lahko tudi brez problema varnostno kopiramo in po želji repliciramo.

4.1. Integracije SGX na UDM

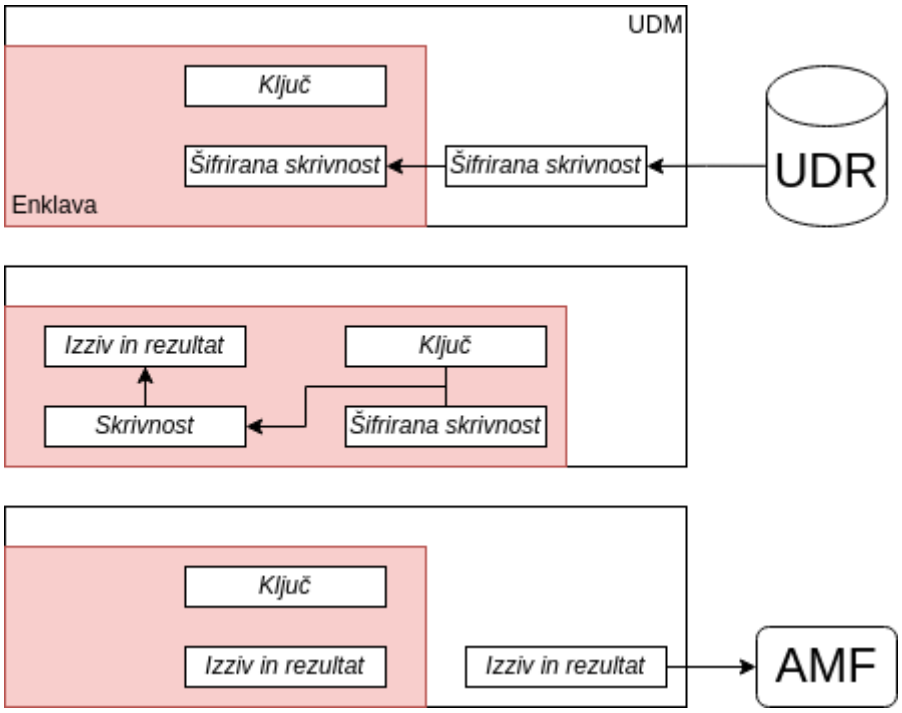
Naš pristop deluje tako, da najprej ustvarimo simetričen ključ, ki ga kličemo UDM-jev ključ. Pri prvem zagonu UDM-ja ga na roke nezaščiteno vnesemo v UDM aplikacijo. Postopek je orisan na sliki 4. UDM nato zažene enklavo, ki ključ skopira v svoj zaščiten spomin, njegov nezaščiten original pa zbriše iz spomina. Predpostavljamo, da imamo v tem časovnem oknu, ko je ključ še nezaščiten v UDM-jevem spominu, dovolj nadzora nad sistemom, da napadalec ne more prebrati ključa. Na primer, to prvo oskrbo izvajamo z izklopljenim mrežnim kablom.



Slika 4: Zagon UDM-ja.

UDM-jeva enklava nato zapečati simetričen ključ, kar pomeni, da ga zašifrira s svojim pečatnim ključem, ter ga shrani na disk. SGX zagotavlja, da je UDM-jeva enklava edina, ki pozna pečatni ključ, zato je z njim zašifriran simetričen ključ varen pred okolico. Če se strežnik ponovno zažene, ali če UDM ponovno zaženemo, na primer zaradi posodobitve, potem ponovni ročni simetričnega ključa ni potreben. UDM po ponovnem zagonu samodejno prebere zapečateni ključ iz diska, enklava ga pa odpečati.

Kot je prikazano na sliki 5, pošlje UDR za vsako overitev UDM-ju šifrirano skrivnost. UDM-jeva enklava jo prebere v zaščiten spomin in jo s simetričnim ključem dešifrira, nato pa iz nje izračuna overitveni izziv. Le tega skopira v spomin izven zaščitenega območja, da ga nato UDM pošlje naprej AMF-ju. Na tak način dešifrirane skrivnosti nikoli ne zapustijo zaščitenega spomina enklave.



Slika 5: UDM dešifrira in uporabi UE-jevo skrivnost.

Ta pristop ima tudi to prednost, da nam bistveno ne oteži replikacije UDM-jev za namen visoke razpoložljivosti. Z enakim postopkom zaženemo UDM na več strežnikih in enklava bo na vsakem v disk zapisala zapečaten simetričen ključ. Avtomatizacija nato poljubno zaganja in ugaša replike UDM-ja na teh strežnikih brez potrebe ročnega vnosa simetričnega ključa.

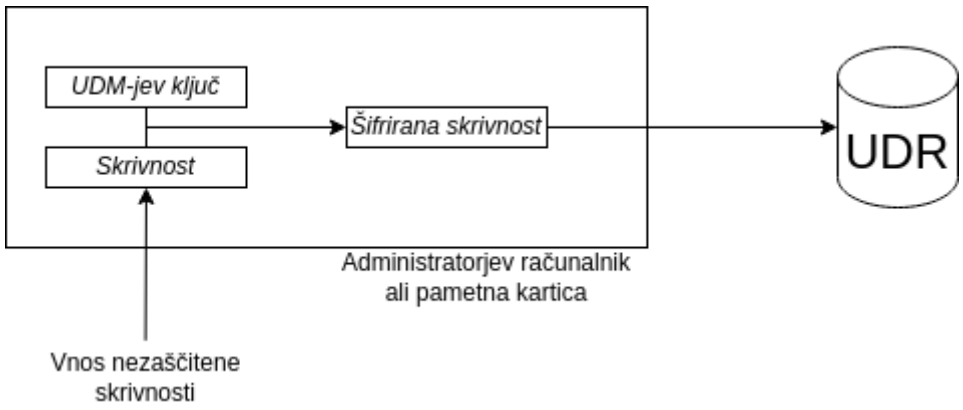
4.2. Varnostna kopija simetričnega ključa

Varnostna kopija UDR-jeve baze same ni dovolj, ker je šifrirana in bi ob izgubi UDM-jevega simetričnega ključa izgubili možnost dešifriranja podatkov. Zato je potrebno varnostno kopirati tudi simetričen ključ, za kar tipično uporabimo eno izmed shem delitve skrivnosti [6]. Tako ima več lastnikov pri sebi shranjene koščke ključa na tak način, da potrebujemo podatke od določene podmnožice lastnikov za rekonstruiranje ključa.

4.3. Vnos novih podatkov v UDR bazo

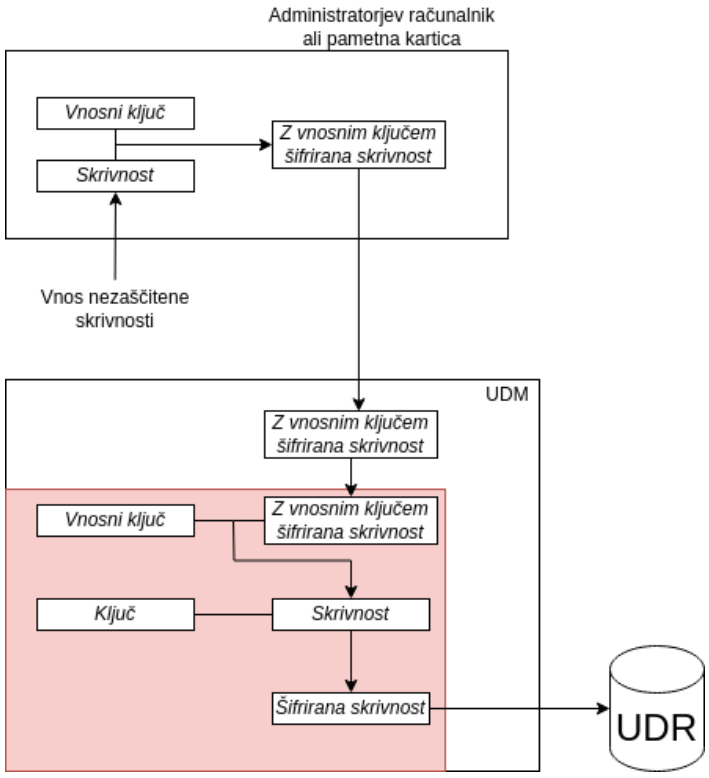
Za vnos novih naročnikov v UDR bazo potrebuje administrator UDM-jev simetričen ključ, da šifrira naročnikove skrivnosti pred vpisom. Postopek je prikazan na sliki 6. Za varnejše upravljanje ima možnost, da podatke šifrira na pametni kartici ali podobnem HSM-ju, ki zagotavlja varno shrambo ključa. V primeru delitve skrivnosti med več

lastnikov je pa za vnos novega naročnika potrebna prisotnost več lastnikov hkrati, kar na račun oteženi priročnosti poveča varnost.



Slika 6: Vnos skrivnosti novega naročnika v UDR-jevo bazo.

Druga možnost je uporaba dodatnega simetričnega ključa, imenovanega vnosni ključ, ki se uporablja izključno za namen vnosa novih naročnikov. Tega hranijo administratorji, ki želijo v UDR bazo vnašati nove naročnike. UDM-jevi enklavi ob zagonu poleg UDM-jevega simetričnega ključa priskrbimo še vnosni ključ. Na sliki 7 je ponazorjeno, kako administrator z vnosnim ključem zašifrira skrivnost novega naročnika in jo pošlje UDM-ju, po možnosti na daljavo.



Slika 7: Vnos skrivnosti novega naročnika preko vnosnega ključa.

UDM-jeva enklava to skrivnost z vnosnim ključem dešifrira in nazaj zašifrira z UDM-jevim simetričnim ključem, nato pa jo vpiše v UDR-jevo bazo. Enklava mora biti programirana tako, da vnosni ključ uporabi samo za namene šifriranja in nikoli dešifriranja. Zagotoviti moramo tudi, da se operacije dešifriranja kriptografsko ne da zlorabiti za pomoč pri šifriranju. Na primer, v shemi AES-CTR se IV nikoli ne sme ponoviti [7]. Na tak način zagotovimo, da vnosni ključ ne more omogočiti dostopa do skrivnosti že vpisanih naročnikov in je zato manj občutljiv od UDM-

jevega ključa. To je kar zaželena lastnost, saj se vnosni ključ tipično potrebuje bolj pogosto od UDM-jevega ključa, ki je potreben zgolj za postavitev UDM-ja na nov strežnik. Omogočena je tudi fleksibilnejša razdelitev nalog med administratorji, kateri bodo hranili UDM-jev ključ in kako, ter kateri bodo hranili vnosni ključ.

5 Alternativne rešitve za zaščito overitvenih skrivnosti

Čeprav Intel® SGX ponuja učinkovito rešitev za zaščito overitvenih skrivnosti znotraj procesorskih enklav, obstajajo tudi druge arhitekture, ki omogočajo primerljivo varnost z drugačnimi pristopi.

5.1. Uporaba strojnih varnostnih modulov (HSM)

Strojni varnostni moduli (HSM – Hardware Security Modules) so namensko grajene naprave, zasnovane za varno upravljanje s kriptografskimi ključi in občutljivimi operacijami. Podobno kot SGX omogočajo tudi HSM-ji, da se skrivnosti obdelujejo v izoliranem, strojno zaščitenem okolju. V kontekstu 5G overitve lahko arhitektura deluje tako, da:

- UDM podatke iz UDR-ja posreduje HSM-ju;
- HSM izvede dešifriranje skrivnosti in generacijo overitvenega izziva;
- izziv se preko UDM-ja posreduje AMF-ju, brez da bi dešifrirane skrivnosti zapustile HSM.

HSM pa med tem zagotavlja varnost simetričnega ključa in skrivnosti po podobnem principu kot opisana SGX enklava v poglavju 4.1. HSM-ji zagotavljajo trajno zaščito ključa, možnost varnostnega revizorstva in so primerni za okolja, kjer je zanesljivost preverjene strojne opreme ključna (npr. bančništvo, vojska, kritične infrastrukture).

5.2. Programske rešitve z uporabo sistemov za upravljanje tajnosti

Poleg rešitev, ki temeljijo na strojni izolaciji (npr. SGX ali HSM), lahko podobno arhitekturo za zaščito overitvenih skrivnosti uresničimo tudi povsem programsko z uporabo sistemov za upravljanje tajnosti. Za okolja, kjer strojna zaščita ni izvedljiva ali ekonomsko upravičena, so primerna orodja za programsko zaščito skrivnosti, kot je HashiCorp Vault.

Vault sistem omogoča, da se skrivnosti hranijo centralizirano in šifrirano, dostop pa je omogočen le avtoriziranim komponentam preko varnostnih politik (RBAC). Dodatno Vault podpira tudi uporabo t. i. "transit engine", s katerim se kriptografske operacije, kot je generacija overitvenih izzivov, izvajajo neposredno v Vaultu, ne da bi bila skrivnost kadarkoli izpostavljena zunanjemu okolju.

V 5G arhitekturi lahko tako UDM prevzame vlogo posredovalca: pridobi podatke od UDR, jih posreduje Vaultu, ta izvede potrebne operacije in vrne rezultat (npr. overitveni izziv), ki ga UDM nato posreduje naprej proti AMF. Ta pristop je z vidika integracije zelo fleksibilen in primeren za okolja, kjer fizična strojna izolacija ni mogoča ali je neučinkovita in se uporablja npr. za oblačna in porazdeljena okolja.

5.3. Primerjava pristopov

Lastnost	Intel SGX	HSM	SW
Tip zaščite	Procesorska enklava	Namenska strojna oprema	Programska rešitev
Fizična varnost	Srednja–visoka	Zelo visoka	Odvisno od okolja
Integracijska zahtevnost	Nizka–srednja	Visoka	Nizka
Primernost za mobilna 5G jedra	Da	Omejena	Da
Razširljivost	Srednja	Omejena	Visoka
Stroškovna učinkovitost	Srednja	Nizka	Visoka

6 Nadaljnja raziskava - SGX-ova oddaljena atestacija

SGX omogoča tako imenovano oddaljeno atestacijo [8]. To je proces, s katerim enklava nekemu drugemu sistemu dokaže, da vsebuje avtentično kodo in podate, in da se resnično izvaja na SGX zaščiteni strojni opremi. Omogoča tudi delitev podatka med oddaljenim sistemom in enklavo z zagotovitvijo, da je ta podatek res enak in da ni nihče mogel posegati vanj in ga spremeniti.

To je precej močno orodje, ki odpira možnosti načrtovanja zanimivih novih arhitektur ali za izboljšavo obstoječe rešitve. Na primer, začetni vnos UDM-jevega in vnosnega ključa bi lahko elegantneje potekal preko varnega komunikacijskega kanala med oddaljenim sistemom in UDM-jevo enklavo, vzpostavljenega s pomočjo atestacije.

7 Zaključek

Overitev v 5G sistemu zahteva varno upravljanje z občutljivimi uporabniškimi podatki. Tehnologija Intel SGX nudi možnost varnega obdelovanja zaupnih podatkov in nam omogoča implementacijo sistema, ki ustreza tej 5G varnostni zahtevi. Predstavljeni pristop uporabi SGX tako, da noben občutljiv podatek nikoli ne zapusti zaščitenega območja za katerega je zagotovljena močna varnost pred zunanjimi napadalci. Hkrati načrtuje način, kako upravljati z varnostnimi kopijami podatkov, ter kako zaščititi postopek vpisa novih uporabniških podatkov v bazo znotraj 5G sistema.

Poleg opisanega pristopa so predstavljene tudi alternative za upravljanje z občutljivimi podatki, skupaj z njihovimi ključnimi značilnostmi, prednostmi in slabostmi. Končna izbira rešitve je odvisna predvsem od konkretnega primera uporabe ter stopnje zahtevanih varnostnih mehanizmov.

Literatura

[1] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirović, Ralf Sasse, Vincent Stettler, "A Formal Analysis of 5G Authentication", arXiv:1806.10360.

[2] 3rd Generation Partnership Project; Technical Specification Group, "Security architecture and procedures for 5G system.", TS 33.501, v17.5.0.

[3] https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/linux-overview.html, Intel® SGX SDK for Linux* OS, obiskano 10. 7. 2025.

[4] Intel®, Intel® Software Guard Extensions Programming Reference, 2014.

[5] Victor Costan, Srinivas Devadas, "Intel SGX Explained", Cryptology {ePrint} Archive, Paper 2016/086, 2016, https://eprint.iacr.org/2016/086.

[6] https://en.wikipedia.org/wiki/Secret_sharing, Secret sharing – Wikipedia, obiskano 2. 4. 2025.

[7] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#CTR, Block cipher mode of operation – Wikipedia, obiskano 22. 7. 2025.

[8] https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html, Attestation Services for Intel® Software Guard Extensions, obiskano 22. 7. 2025.

Od Modre, Vijolične in Rdeče do Črne

Matej Perhavec

Sportradar d.o.o., Ljubljana, Slovenija
m.perhavec@sportradar.com

V sodobnem digitalnem okolju organizacije letno namenjajo velik del svojih sredstev za krepitev kibernetsko-varnostnih zmogljivosti, predvsem z namenom zaščite pred oddaljenimi, pogosto anonimnimi napadalci. Ob tem se zastavlja pomembno vprašanje: koliko pozornosti je pri tem namenjeno fizični varnosti? Ali obstoječe organizacijske skupine znotraj kibernetske varnosti, kot so modra, rdeča in vijolična ekipa, res zadoščajo za učinkovito obvladovanje varnostnih tveganj ali pa vse večja izpostavljenost fizičnim grožnjam narekuje potrebo po dodatni, specializirani skupini? V prispevku smo predstavili že uveljavljene varnostne ekipe na področju kibernetske varnosti (modro, rdečo in vijolično), osredotočili pa smo se na še ne tako prepoznavno a vseeno ključno črno ekipo, katere osrednja naloga je izvajanje fizičnih penetracijskih testov in varnostnih pregledov. Opisali smo njene glavne aktivnosti in metodologijo, umestili njeno vlogo v kontekstu nove evropske direktive CER in poudarili sodelovanje črne ekipe z drugimi varnostnimi skupinami. Predstavili smo tudi konkretne primere fizičnih ranljivosti z digitalno komponento, ki dodatno potrjujejo pomen vključevanja črne ekipe v kibernetsko-varnostno strategijo organizacij.

Ključne besede:
modra ekipa,
rdeča ekipa,
vijolična ekipa,
črna ekipa,
fizično penetracijsko testiranje,
fizični varnostni pregledi,
fizična varnost.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.13

1 Uvod

V sodobni kibernetski varnosti se za opis ključnih vlog in pristopov pogosto uporablja barvna klasifikacija namenskih skupin, kot so modra ekipa (angl. blue team), rdeča ekipa (angl. red team), vijolična ekipa (angl. purple team) in črna ekipa (angl. black team), ki označujejo specializirane organizacijske enote z različnimi odgovornostmi za zaščito in testiranje informacijskih sistemov.

Izraza Blue Team in Red Team izvirata iz vojaške prakse, kjer so skupine za simulacijo bojev zasedale vloge branilcev in napadalcev, da bi izboljšale taktiko, pripravljenost in odzivnost na sovražne sile. Tradicionalno je bila rdeča barva simbol nasprotnikov oziroma napadalcev, medtem ko je modra barva predstavljala zavezniške oziroma obrambne enote. Ta simbolika se je kasneje prenesla v informacijsko varnost, kjer rdeča ekipa prevzema vlogo simuliranih napadalcev, medtem ko modra ekipa deluje kot obrambna enota, ki zaznava in preprečuje grožnje [1].

Z naraščajočo kompleksnostjo kibernetskih napadov in potrebo po bolj učinkovitem sodelovanju med napadalskimi in obrambnimi ekipami se je uveljavil koncept vijolične ekipe. Vijolična ekipa združuje znanja in veščine rdeče in modre ekipe v sodelovanje, ki omogoča hitrejše odkrivanje varnostnih pomanjkljivosti in neposredno izboljšanje varnostnih ukrepov. Ta pristop se je uveljavil kot odgovor na potrebe po agilnosti in stalnem prilagajanju varnostnih sistemov glede na nove tehnike napadalcev.

Črna ekipa, ki se osredotoča na fizično penetracijsko testiranje in fizično varnost organizacij, izvira iz varnostnih in vojaških praks testiranja fizičnih varnostnih ukrepov, kot so kontrole dostopa, varnostno osebje in postopki za zaščito kritične infrastrukture. S prihodom digitalizacije in integracije fizičnih in kibernetskih varnostnih elementov se je pomen črne ekipe okrepil kot ključni del celostne varnostne strategije. Pristopi črne ekipe so danes nepogrešljivi pri prepoznavanju ranljivosti, ki izhajajo iz fizičnega dostopa do sistemov, kar pogosto predstavlja začetek kibernetskih incidentov.

Ti štirje koncepti skupaj tvorijo celosten okvir za varnostne prakse, ki vključujejo simulacijo napadov, obrambno odzivnost, sodelovanje ekip in fizično varnost, s ciljem povečanja odpornosti organizacij na sodobne varnostne grožnje.

2 Modra ekipa

Modra ekipa predstavlja osrednji obrambni mehanizem organizacije, katerega ključna naloga je zagotavljanje celovite zaščite informacijskih sistemov, omrežij in podatkov pred različnimi vrstami kibernetskih groženj. Njeno delovanje temelji na stalnem spremljanju varnostnih dogodkov, zaznavanju anomalij, analizi groženj ter izvajanju tehničnih in organizacijskih ukrepov za preprečevanje, odkrivanje in odzivanje na varnostne incidente.

Ključno operativno okolje za delovanje modre ekipe predstavlja VOC[2] (angl. SOC[3]), ki je zadolžen za zbiranje in analizo dnevniških zapisov, opozoril in dogodkov iz različnih virov, vključno z IDS[4]/IPS[5] sistemi, požarnimi zidovi, SIEM[6] in EDR[7]/XDR[8] rešitvami. VOC tako omogoča modri ekipi pravočasen vpogled v stanje varnosti informacijskega okolja in koordiniran odziv na zaznane grožnje.

[2] VOC - Varnostni Operativni Center
[3] SOC - Security Operations Center
[4] IDS - Intrusion Detection System
[5] IPS - Intrusion Prevention System
[6] SIEM - Security Information and Event Management
[7] EDR - Endpoint Detection and Response
[8] XDR - Extended Detection and Response

Pomemben del funkcionalnosti modre ekipe predstavlja tudi odziv na incidente (angl. incident response), ki je natančno opredeljen v okviru smernic NIST SP 800-61r3 [2] in vključuje štiri faze:

- pripravo,
- zaznavo in analizo,
- zajezitev, odpravo in obnovitev,
- zaključne aktivnosti z analizo po incidentu.

Strukturiran odziv na incidente skrajša čas zaznave in odprave (MTTD[9]/MTTR[10]) ter posledično prispeva k zmanjšanju vpliva varnostnih dogodkov.

Modra ekipa prav tako izvaja lov na grožnje (angl. threat hunting), ki predstavlja proaktiven in hipotezno usmerjen (angl. hypothesis-driven) proces iskanja naprednih, pogosto prikritih groženj znotraj informacijskega okolja organizacije. Ta proces temelji na analizah vedenjskih vzorcev, zgodovinskih podatkov in groženj iz zunanjih virov (npr. uporaba MITRE ATT&CK ogrodja) [3]. Lov na grožnje presega pasivne obrambne ukrepe; omogoča identifikacijo napadalcev, ki so se morda že infiltrirali v sistem, a še niso povzročili zaznanih škodljivih posledic.

Poleg omenjenih osrednjih nalog modra ekipa izvaja tudi upravljanje ranljivosti (angl. vulnerability management), ki obsega redno izvajanje ocen ranljivosti (angl. vulnerability assessments) z uporabo avtomatiziranih orodij kot so skenerji ranljivosti (angl. vulnerability scanner), ter vzpostavitev procesov za odpravo kritičnih pomanjkljivosti s pomočjo vpeljave varnostnih popravkov (angl. patch management). Ti postopki omogočajo zmanjšanje napadalne površine ter preprečevanje izkoriščanja znanih ranljivosti. Modra ekipa aktivno sodeluje tudi pri načrtovanju odziva na incidente in vpeljevanju kriznih načrtov.

Kot ključni obrambni člen v celostnem okviru kibernetske varnosti modra ekipa omogoča proaktivno obvladovanje varnostnih dogodkov, zgodnje odkrivanje napadalcev in učinkovito zmanjševanje posledic incidentov. Njeno delovanje neposredno prispeva k stabilnosti poslovanja in zrelosti celotne varnostne strategije organizacije.

3 Rdeča ekipa

Rdeča ekipa predstavlja specializirano varnostno skupino, katere ključne naloge vključujejo penetracijsko testiranje, simulacije napadov, izvajanje socialno-inženirskih kampanj in celovitih red teaming aktivnosti z namenom preverjanja učinkovitosti obrambnih zmogljivosti organizacije.

Penetracijsko testiranje temelji na odkrivanju in izkoriščanju odkritih ranljivosti z namenom preverjanja odpornosti sistemov, aplikacij in omrežij ter se izvaja v skladu s smernicami NIST SP 800-115 [4] in metodologijami, kot so PTES[11], OWASP[12] Testing Guide in OSSTMM[13]. Razlika med penetracijskim testiranjem in red teaming aktivnostmi je v ciljih in pristopu: medtem ko je pri penetracijskem testiranju cilj odkriti čim več ranljivosti, se pri red teaming aktivnostih rdeča ekipa osredotoča na neopažen doseg specifičnih ciljev, kot so kompromitacija domenskega krmilnika, dostop do zaupnih podatkov ali dolgotrajen in prikrit dostop do interne infrastrukture.

Poleg specifičnih ciljev, red teaming aktivnosti celovito in realistično preverjajo odpornost organizacije na napade, pri čemer posnemajo taktike, tehnike in postopke (angl. TTP[14]) dejanskih napadalcev. Zajemajo širok spekter simuliranih napadov, vključno s socialnim inženiringom, fizičnim vdorom in preizkusom operativne varnosti.

[9] MTTD - Mean Time To Detect
[10] MTTR - Mean Time To Repair
[11] PTES - Penetration Testing Execution Standard
[12] OWASP - Open Worldwide Application Security Project
[13] OSSTMM - Open Source Security Testing Methodology Manual
[14] TTP - Tactics, Techniques and Procedures

Takšen pristop omogoča realistično simulacijo napadov naprednih trajnih groženj (angl. APT[15]), katere cilj je ocena dejanske pripravljenosti modre ekipe, zlasti varnostno operativnega centra in njene odzivnosti na incidente.

Posebnost red teaming aktivnosti je vključevanje fizičnih vektorjev napada za doseg zadanih kibernetskih ciljev, kar predstavlja pomembno komponento celostnega testiranja. V skladu z metodologijo OSSTMM Chapter 8 - Physical Security Testing [5] in smernicami ISO/IEC 27001 [6] in 27002 [7] ter NIST SP 800-53r5 [8] o fizični varnosti, se v sklopu red teaming aktivnosti preverja varnost fizičnih dostopov, nadzor in segmentacija dostopov znotraj organizacije in ozaveščenost zaposlenih. Primeri fizičnih testov vključujejo poskuse neopaznega vstopa v stavbo (npr. tailgating), namestitev nepooblaščenih naprav (npr. keylogger), socialni inženiring (npr. uporaba lažne identitete vzdrževalca) ali izkoriščanje slabih fizičnih praks (npr. nezaščiten dostop do strežniške sobe).

Rdeča ekipa z realističnim, nadzorovanim in etičnim testiranjem tako ne le razkriva tehnične pomanjkljivosti, temveč preverja tudi odpornost na človeške napake, socialni inženiring in fizično varnost. Njihovo delovanje predstavlja eno najpomembnejših orodij za organizacije, ki si prizadevajo doseči visoko stopnjo varnostne zrelosti in operativne pripravljenosti.

4 Vijolična ekipa

Vijolična ekipa predstavlja metodološki in organizacijski pristop k izboljšanju kibernetske varnosti z aktivnim povezovanjem dveh tradicionalno ločenih skupin - rdeče ekipe (simuliranje napadov) in modre ekipe (obramba pred napadi). Namesto ločenega delovanja, skupen pristop omogoča usklajeno sodelovanje med obema ekipama z namenom neposredne izmenjave znanja, testiranja zaznavnih zmožnosti in sprotnega prilagajanja obrambnih mehanizmov glede na dejanske tehnike napadalcev.

Ključna značilnost takšnega sodelovanja je sprotno pridobivanje povratnih informacij: ko rdeča ekipa izvede specifičen napad ali tehniko (npr. lateral movement ali privilege escalation), modra ekipa sočasno spremlja sposobnost zaznave dogodkov ter nato skupaj z rdečo ekipo analizira vrzeli v zaznavi, odzivu ali eskalaciji. Takšen pristop pogosto temelji na vzorcu »assume breach«, ki predpostavlja, da se je napadalcu že uspelo infiltrirati v sistem ali omrežje in preverja, kako globoko lahko napadalec prodre znotraj organizacije. Vse to poteka v varnem in nadzorovanem okolju, s ciljem izboljšati detekcijo, notranjo segmentacijo in odzivnih zmogljivosti.

Pri delu vijolične ekipe se uporabljajo tako ročna kot avtomatizirana ogrodja za simulacije napadov. Med ročnimi ogrodji izstopata ogrodji Atomic Red Team [9] in Caldera [10], ki omogočata izvajanje posameznih taktik, tehnik in postopkov z namenom preverjanja zaznavnih zmožnosti obrambnih orodij. Hkrati se v okviru aktivnosti uporabljajo tudi avtomatizirana orodja za simulacijo vdorov in napadov (npr. BAS[16]), ki omogočajo ponovljivost, standardizacijo ter kontinuirano validacijo učinkovitosti obrambnih mehanizmov.

Vpeljava vijolične ekipe v varnostno arhitekturo omogoča hitrejšo izpopolnjevanje varnostnih mehanizmov, ciljno usmerjeno zaznavanje in boljšo usklajenost med varnostnimi in poslovnimi cilji. Organizacije, ki integrirajo tak pristop v svojo varnostno strategijo, dosegajo višjo stopnjo zrelosti, saj ne odkrivajo zgolj pomanjkljivosti, temveč jih tudi aktivno odpravljajo na način, ki je prilagojen njihovemu okolju.

[15] APT - Advanced Persistent Threat
[16] BAS - Breach and Attack Simulation

5 Črna ekipa

V pogovoru o varnosti in zaščiti organizacij pogosto prevladuje digitalna varnost, vendar fizična varnost ostaja ključen, a pogosto spregledan steber celostne zaščite organizacije. Napadalci ne potrebujejo zmeraj napredne zlonamerne programske opreme ali 0-day ranljivosti, včasih je dovolj fizični dostop do neustrezno zavarovane strežniške sobe ali do sejne sobe kjer vodstven kader sprejema interne odločitve, da nepooblaščene osebe pridejo do ključnih informacij. Tu nastopi črna ekipa - specializirana varnostna enota, katere glavna naloga je preizkušanje in izboljševanje fizične varnosti organizacije.

Črna ekipa se osredotoča na fizično varnost organizacije in njen preplet z digitalno varnostjo. Ključne naloge črne ekipe so fizično penetracijsko testiranje, simulacija vdorov ter socialni inženiring, s čimer organizaciji pomaga odkriti varnostne pomanjkljivosti, povezane s fizičnim dostopom do informacijskih sistemov, kritične infrastrukture in drugih ključnih virov organizacije. Gre za nadzorovane in etične simulacije napadov, ki vključujejo preizkušanje varnostnih sistemov, kot so sistemi kontrole dostopa, alarmni sistemi, kamere, fizične pomanjkljivosti ter ozaveščenost in odzivnost zaposlenih [5].

5.1. Aktivnosti črne ekipe

Delovanje črne ekipe temelji na mednarodno priznanih standardih in smernicah, kot so OSSTMM Chapter 8 - Physical Security Testing [5] in priporočilih Evropske agencije za kibernetsko varnost (ENISA) glede fizične varnosti [11,12]. Poleg izvajanja fizičnih penetracijskih testov in simulacij napadov, črna ekipa zaradi svojega specifičnega nabora znanj in izkušenj s področja fizične in kibernetske varnosti, prav tako izvaja in sodeluje pri izvedbi fizičnih varnostnih in revizijskih pregledov.

Fizično penetracijsko testiranje

Fizično penetracijsko testiranje, je ključnega pomena za odkrivanje ranljivosti, ki jih tehnološki varnostni sistemi ne zaznajo. Primeri simulacij resničnih fizičnih groženj vključujejo: poskuse nepooblaščenega vstopa v stavbe, infiltracijo skozi varnostne preglede, krajo in kloniranje dostopnih kartic, namestitev nepooblaščenih naprav (npr. keylogger, rogue access point) in pridobitev fizičnega dostopa do ključnih organizacijskih prostorov in virov (npr. strežnikov, omrežne opreme, zaupnih dokumentov).

Tako kot pri digitalnem penetracijskem testiranju tudi fizično penetracijsko testiranje poteka po jasno določenih fazah:

- **Priprava (angl. Pre-engagement)** - pred začetkom aktivnosti je ključno izvesti usklajevalna srečanja z naročnikom in:
    - opredeliti obseg testiranja (angl. scope),
    - določiti jasen cilj (npr. dostop do strežniške sobe, dostop do direktorjeve pisarne, pridobitev zaupnih dokumentov),
    - dogovoriti se o pravilih sodelovanja (angl. rules of engagement) (npr. brez povzročanja škode, časovni okvir testiranja, obveščanje o kritičnih ranljivostih),
    - vzpostaviti varnostne mehanizme, vključno z dokumentom o dovoljenju za testiranje (angl. get out of jail free card), ki ščiti člane ekipe v primeru pravnih ali operativnih zapletov.

- **Izvidništvo (angl. Reconnaissance)** - izvidništvo predstavlja temelj za nadaljnje faze testiranja. Cilj je zbrati čim več informacij o ciljnem objektu brez aktivnega poseganja v okolje. Začne se z uporabo OSINT[17] tehnik, pri čemer se analizirajo:
    - satelitske slike,
    - javno dostopni podatki o objektu (npr. lastništvo, tlorisi, varnostni protokoli),
    - fizična postavitev zgradbe (npr. ograje, vhodi in izhodi, varnostne kamere),
    - objave organizacije in zaposlenih na socialnih omrežjih.

  Na podlagi zbranih informacij črna ekipa načrtuje taktiko in izbere najbolj primerno vrsto izvidništva glede na oddaljenost in zahtevnost dostopa, oddaljeno izvidništvo (angl. long range reconnaissance) ali izvidništvo od blizu (angl. short range reconnaissance) s ciljem pridobiti informacije o:
    - rutini zaposlenih (prihodi in odhodi),
    - pravilih oblačenja (angl. dress code),
    - izgledu dostopnih kartic,
    - lokaciji senzorjev, kamer in drugih zunanjih varnostnih elementov.

  Zaključna faza izvidništva je notranje izvidništvo (angl. embedded reconnaissance), kjer ekipa poskuša pridobiti informacije o notranjosti objekta in zaposlenih z diskretnim, pogosto nevsiljivim vstopom v objekt pri čemer je cilj:
    - identificirati notranje varnostne mehanizme,
    - opazovanje vedenjskih vzorcev zaposlenih (receptorke/ji, tajnice/ki, varnostno osebje),
    - locirati elektronsko nadzorovane prehode,
    - preveriti prisotnost alarmov, notranjih senzorjev, kamer in drugih varnostnih elementov,
    - pridobiti informacije ter vzpostaviti poznanstva s tehnikami socialnega inženiringa.
- **Izraba ranljivosti (angl. Exploitation)** - za izvidništvom sledi faza izrabe identificiranih ranljivosti, kjer črna ekipa na podlagi zbranih informacij aktivno izkoristi odkrite pomanjkljivosti za vdor v ciljni objekt. Namen te faze je pridobiti nepooblaščen dostop do notranjih prostorov in sistemov z izrabo odkritih pomanjkljivosti, na primer:
    - neustrezno zavarovane točke dostopa,
    - uporaba dostopnih kartic, ki jih je možno klonirati,
    - pomanjkljivi postopki preverjanja identitete,
    - ranljive točke v sistemih nadzora dostopa.

  Izraba lahko vključuje uporabo različnih tehnik, kot so:
    - kloniranje dostopnih kartic,
    - manipulacija čitalcev kartic,
    - fizični dostop preko neprimerno varovanih vhodov (npr. preskakovanje preko ali plazenje pod vrati, tailgating, lockpicking),

[17] OSINT - Open Source Intelligence

  - uporaba lažnih identitet ter socialni inženiring za izkoriščanje zaupanja zaposlenih in prehod varnostnih kontrol (npr. novo zaposleni, dostava, pozabljena dostopna kartica).

- **Pridobitev dostopa (angl. Gaining Access)** - po uspešno izvedeni izrabi ranljivosti, sledi pridobitev dostopa v ciljni objekt ali okolje. Poudarek je na neopaženem in nadzorovanem vstopu, ki omogoča izvajanje nadaljnjih aktivnosti brez sprožitve varnostnih mehanizmov. V tej fazi se izvedejo ključni koraki, kot so zbiranje občutljivih informacij, dostop do nezavarovanih naprav ali strežnikov, nameščanje zlonamerne strojne opreme (npr. keylogger, LAN Turtle, Rubber Ducky) ter, če je v obsegu testiranja, vzpostavitev dolgoročne prisotnosti (angl. persistence). Namen te faze ni zgolj vstop, temveč zavzetje ključne pozicije znotraj okolja za nadaljnjo eskalacijo ali preverjanje odpornosti organizacije.
- **Poročanje (angl. Reporting)** - zaključna faza penetracijskega testiranja je priprava podrobnega poročila, ki povzema ugotovitve, izvedene tehnike in odkrite pomanjkljivosti. Poročilo vsebuje oceno tveganj, vplivov na varnost ter priporočila za odpravo ugotovljenih ranljivosti. Pomembno je, da poročilo vključuje tudi vse možne poti dostopa, ki so bile odkrite med fazo izvidništva, čeprav je bila med fazo izrabe ranljivosti preizkušena le ena izmed njih. Ostale poti, ki niso bile neposredno testirane, se v poročilu posebej izpostavijo, da se lahko naročnik sam odloči za njihovo nadaljnjo preizkušnjo ali pa je na njih še posebej pozoren pri izboljševanju varnostnih ukrepov.

  Pomemben del te faze je tudi predstavitev rezultatov naročniku, kjer se razložijo ključni problemi, možni napadi in predlagane izboljšave. Namen poročanja je omogočiti naročniku boljše razumevanje ranljivosti ter zagotoviti izhodišče za izboljšanje varnostnih ukrepov ter zmanjšanje tveganj v prihodnosti.

Fizični varnostni pregledi

Črna ekipa poleg fizičnih penetracijskih testov pogosto izvaja tudi strukturirane fizične varnostne preglede, pri katerih za razliko od penetracijskih testov, člani ekipe ne ohranjajo prikritosti, temveč se lahko odkrito gibljejo po prostorih organizacije ter sistematično prepoznavajo in preverjajo varnostne pomanjkljivosti. V primerjavi s penetracijskim testom, kjer je za dosego cilja testirana le ena izmed vseh odkritih pomanjkljivosti, so pri fizičnem varnostnem pregledu testirane vse odkrite pomanjkljivosti, saj kot že prej omenjeno, člani ekipe ne rabijo paziti na neopaznost. Med fizičnim pregledom se ocenjujejo različni varnostni elementi, kot so:

- analiza sistemov kontrole dostopa (npr. vrsta in možnost kloniranja dostopnih kartic, uporabljeni protokoli, možnost zaobitve vgrajenih sistemov),
- pregled politik za obiskovalce in dobavitelje (npr. vpis v evidenco, določanje spremljevalca, omejen dostop),
- ocenjevanje učinkovitosti fizičnih ovir, alarmov in nadzorne opreme (npr. postavitev kamer in senzorjev, mrtve točke, zaobitev ovir, možnost onesposobitve alarmnih naprav),
- simulacije evakuacij in kriznega odziva,
- stanje in učinkovitost ključavnic in vrat (npr. možnost vdiranja v ključavnice, uporaba orodij za odpiranje vrat od znotraj, zdrs zatiča ključavnice),
- vidnost in odzivnost varnostnega osebja,
- označevanje in nadzorovanje nevarnih ali omejenih območij.

Večina organizacij praviloma na začetku še ni pripravljenih na izvedbo celovitega fizičnega penetracijskega testa, saj pogosto nimajo jasnega pregleda nad lastno fizično varnostno izpostavljenostjo. Zaradi pomanjkanja evidence osnovnih varnostnih kontrol in neustrezne ocene tveganj, cilji testiranja pogosto niso jasno opredeljeni. Nejasno zastavljeni cilji so lahko tudi posledica širših organizacijskih izzivov, kot so pomanjkanje zrelosti varnostne

strategije, odsotnost sodelovanja med ključnimi oddelki (npr. fizično varovanje, IT in poslovno vodstvo), slabo razumevanje metodologije testiranja ali izvajanje testov zgolj zaradi zunanjih zahtev oziroma skladnosti z regulativo in ne zaradi dejanske notranje varnostne potrebe.

Priporočljivo je, da organizacija najprej izvede sistematičen fizični varnostni pregled, v okviru katerega se identificira in dokumentira večje število obstoječih varnostnih pomanjkljivosti. Ta korak omogoča organizaciji, da se seznani s ključnimi tveganji ter pripravi ustrezne ukrepe za odpravo ranljivosti. Šele po implementaciji korektivnih ukrepov je smiselna izvedba fizičnega penetracijskega testa, katerega cilj je preveriti učinkovitost uvedenih izboljšav in odkriti morebitne preostale ranljivosti.

5.2. Vloga fizičnega penetracijskega testiranja v evropski direktivi CER

Evropska direktiva o odpornosti kritičnih subjektov (**CER**[18]**)**, ki je začela veljati decembra 2022 in je morala biti do oktobra 2024 implementirana v zakonodaje držav članic Evropske unije, uvaja obvezne varnostne ukrepe za zaščito **kritične infrastrukture** pred fizičnimi in hibridnimi grožnjami. Med ključnimi novostmi direktive je tudi okrepljen poudarek na **celovitem upravljanju fizične varnosti**, kjer pomembno vlogo dobivajo **realistične simulacije napadov**, vključno z **rednim izvajanjem fizičnih penetracijskih testov**.

Fizični penetracijski testi, kot jih predvideva CER, morajo preverjati odpornost organizacije na napade, ki izvirajo iz fizičnega dostopa do objektov, vključno z nepooblaščenim vdorom, socialnim inženiringom, zlorabo dostopnih točk ter fizičnim izklopom ključne infrastrukture. V kontekstu aktivnosti **črne ekipe** kjer napadalci nimajo vnaprejšnjih informacij ali začrtane poti, takšni testi predstavljajo verodostojno simulacijo realnega tveganja, s katerim se lahko soočajo energetski, prometni, zdravstveni in drugi kritični sektorji.

Zahteve CER tako organizacijam ne nalagajo zgolj formalnega ocenjevanja varnosti, temveč jih zavezujejo k **aktivnemu preverjanju odpornosti** z uporabo metod, ki vključujejo **tako napovedljive kot nepredvidljive napade**, kar predstavlja pomemben korak k bolj proaktivni obrambi. Aktivnosti črne ekipe, v tem okviru, postajajo ena izmed ključnih praks za dokazovanje skladnosti in dejanske pripravljenosti organizacije na fizične grožnje.

5.3. Primeri fizičnih pomanjkljivosti z digitalnimi elementi

Fizični dostop napadalcem omogoča izkoriščanje fizičnih ranljivosti, človeških napak ter neposredno manipulacijo s strojno in omrežno opremo, ki lahko vodi v digitalno kompromitacijo. Gre za napade, kjer napadalec pridobi fizični dostop do naprav, omrežne infrastrukture ali okolja, v katerem lahko nato sproži digitalne vektorje napada, pogosto popolnoma mimo klasičnih varnostnih kontrol. V nadaljevanju predstavljamo primere takšnih napadov, ki vključujejo digitalne komponente in so posledica pomanjkljive zaščite fizične infrastrukture, neustreznih praks ali tehnično zastarelih rešitev. Ti primeri izpostavljajo potrebo po tesnem prepletu fizične in informacijske varnosti znotraj organizacijske varnostne strategije.

Kloniranje dostopnih kartic

Kloniranje dostopnih kartic je ena izmed najpogostejših in najučinkovitejših tehnik fizičnega napada. Številne organizacije še zmeraj uporabljajo starejše RFID[19] tehnologije (npr. 125 kHz HID Prox ali 13.5 MHz MIFARE Classic), ki ne vključujejo osnovne kriptografske zaščite in jih je možno klonirati z uporabo enostavno dostopnih naprav, kot sta Proxmark3 ali Flipper Zero. Napadalec lahko zabeleži signal kartice mimoidočega zaposlenega,

[18] CER - **Critical Entities Resilience**
[19] RFID - Radio-frequency identification

pogosto brez njegove vednosti in brez fizičnega stika, ter ustvari popoln klon, s katerim nato pridobi fizični dostop do zaščitenih območij.

V nasprotju s tem naprednejše kartične tehnologije, kot je MIFARE DESFire EV2/EV3, uporabljajo sodobne kriptografske mehanizme (npr. AES-128), kar bistveno otežuje oziroma onemogoča kloniranje brez dostopa do skrivnih ključev. Žal so takšne rešitve v praksi še zmeraj redkost, bodisi zaradi stroškov bodisi zaradi pomanjkanja ozaveščenosti o tveganjih.

Priporočljiva je opustitev uporabe zastarelih RFID kartic brez kriptografske zaščite ter prehod na varnejše tehnologije, kot so MIFARE DESFire EV2/EV3 ali druge kartice s podporo za AES šifriranje, avtentično dvosmerno komunikacijo ter centralizirano upravljanje ključev. Takšna nadgradnja bistveno zmanjša verjetnost kloniranja in zagotavlja višjo stopnjo zaščite fizičnega dostopa, ki je pogosto prva linija obrambe pred usmerjenimi napadi.

Manipulacija čitalcev kartic

Kljub uporabi varnejših kartičnih tehnologij, še ne pomeni, da je sistem kontrole dostopa varen kot celota. Eden izmed pogosto spregledanih napadov vključuje fizično manipulacijo čitalcev kartic, pri čemer napadalec izpostavi ali prestreže komunikacijo med čitalcem in krmilnikom. Večina starejših sistemov uporablja Wiegand protokol, ki je nekriptiran, enosmeren in ranljiv za pasivno prisluškovanje, ponavljanje prenosa (angl. replay attack) ali celo manipulacijo podatkov.

Napadalec lahko s fizičnim dostopom do ožičenja čitalca (npr. zunaj stavbe ali v slabo varovanem hodniku) priključi napravo, ki beleži ali posreduje podatke o avtorizaciji. V določenih primerih je mogoče s pomočjo preprostega napada reproducirati signal in tako simulirati legitimno kartico brez njene prisotnosti.

Priporočljiva je uporaba kriptografsko zaščitenih protokolov, kot so OSDP[20] s podporo za AES šifriranje in dvosmerno preverjanje pristnosti. Poleg tega je potrebno zagotoviti fizično zaščito komunikacijskih poti med čitalci in krmilniki (npr. uporaba oklopljenih vodnikov ali zaznava fizične manipulacije čitalcev kartic) ter spremljanje nenavadnih dogodkov v sistemu dostopa.

Prekomerne pravice dostopa

Pogosta, a večkrat spregledana ranljivost je dodeljevanje preširokih dostopnih pravic znotraj sistemov kontrole dostopa. V številnih organizacijah imajo zaposleni ali pogodbeni delavci dostop do večjega števila prostorov, kot jih dejansko potrebujejo za svoje delo. Napadalec, ki pridobi klonirano kartico takšne osebe, lahko brez omejitev vstopi v občutljive prostore, kot so strežniške sobe, arhivi ali tehnični prostori, kjer ima neposreden dostop do informacijskih sistemov, omrežne opreme ali dokumentacije.

Poleg očitnega povečanja varnostnega tveganja takšna ureditev otežuje tudi revizijsko sledenje in odziv v primeru incidenta, saj dostop ni omejen glede na dejanske potrebe uporabnika.

Priporočljivo je dosledno uveljavljanje načela najmanjšega potrebnega dostopa (angl. least access) v okviru sistemov kontrole dostopa. Vsaka dostopna pravica naj bo dodeljena na podlagi konkretnih delovnih nalog in preverjena ob spremembi zaposlitve, vloge ali lokacije dela. Na ta način morebitni nepooblaščeni fizični vdori ne ogrozijo celotne organizacije, temveč ostanejo omejeni na najmanjši možni obseg, kar bistveno zmanjša varnostno tveganje.

[20] OSDP - Open Supervised Device Protocol

Nenadzorovani in odklenjeni računalniki

Ena pogostejših napak zaposlenih je puščanje odklenjenih računalnikov brez nadzora. V času kratke odsotnosti uporabnika, lahko napadalec z dostopom do delovne postaje priključi zlonamerno napravo (npr. Rubber Ducky ali Bash Bunny), ki je ob priklopu v računalnik prepoznana kot vhodna periferna naprava (tipkovnica, miška, ipd.) in samodejno izvede preddefinirane sistemske ukaze ter skripte brez uporabnikove interakcije. S tem lahko napadalec že v nekaj sekundah pridobi nadzor nad sistemom.

Priporočljivo je dosledno uveljavljanje politike samodejnega zaklepanja zaslona po krajšem obdobju neaktivnosti (npr. 3-5 minut) ter ozaveščanje zaposlenih, da ob vsaki odsotnosti zaklenejo delovno postajo. Poleg tega je priporočljiva omejitev uporabe administrativnih privilegijev pri rednem delu in kontrola priklopa perifernih naprav.

Varnostne ranljivosti videonadzornih sistemov

Videonadzorni sistemi (CCTV[21]) predstavljajo ključen steber fizične varnosti, a obenem tudi pogosto spregledano točko napada, če niso ustrezno fizično in logično zaščiteni. Ena izmed najpogostejših ranljivosti je nezaščiteno ožičenje, ki je pogosto lahko dostopno (npr. nad spuščenimi stropi ali ob fasadah), kar omogoča fizične posege npr. prekinitve, preusmeritve ali celo vključitev zlonamernih naprav. Prav tako lahko IP-kamere, povezane na slabo segmentirano omrežje, služijo kot izhodišče za vstop v IT okolje organizacije.

Druga kritična pomanjkljivost je neoptimalna postavitev kamer, ki ustvarja slepe točke, kjer se lahko napadalec giblje ali izvaja dejavnosti brez nadzora. V praksi se črne ekipe pogosto poslužujejo vstopov skozi servisne vhode, kletne garaže ali izkoriščanja mrtvih kotov ob kamerah, ki niso ustrezno usmerjene ali se prekrivajo s slepimi območji drugih kamer. Poleg tega so kamere pogosto nameščene na dosegljivih mestih brez zaščite proti fizičnim posegom, kar omogoča njihovo obračanje, prekrivanje ali mehansko poškodbo, brez sprožitve alarma.

Za učinkovit videonadzor je potrebno več kot le prisotnost kamer. Priporočljiva je:

- uporaba oklopljenih in skritih vodnikov,
- aktiviranje zaznave fizičnih posegov,
- ločena omrežna segmentacija IP-kamer,
- redni pregledi in testiranja vizualnega pokritja za odpravo slepih točk,
- vključitev videonadzornih sistemov v redne fizične penetracijske teste.

Kamera, ki je fizično ali logično ranljiva, ne le izgubi svojo varovalno funkcijo, temveč lahko postane tudi vektor napada zoper organizacijo samo.

6 Sodelovanje med ekipami

Med aktivnostmi črne in rdeče ekipe lahko potegnemo določene vzporednice. Tako pri digitalnem penetracijskem testiranju kot pri fizičnem varnostnem pregledu je cilj široko zastavljen: identificirati čim več ranljivosti v obravnavanem sistemu ali okolju. Gre za celovit pregled, katerega namen je zagotoviti temeljito razumevanje varnostne izpostavljenosti organizacije.

Za razliko od tega pa imata fizično penetracijsko testiranje in red teaming aktivnosti jasno definirane cilje. Ti vključujejo preverjanje možnosti doseganja konkretnega cilja, kot je na primer nepooblaščen dostop do strežniške sobe ali dostop do zaupnih dokumentov. V tem kontekstu izvajalci testiranja običajno ne preverjajo drugih

[21] CCTV - Closed-circuit television

priložnostnih ranljivosti, kot so denimo prost dostop do direktorjeve pisarne ali dostop do varnostnih kopij občutljivih sistemov - razen če so takšni cilji vnaprej opredeljeni v obsegu testa.

Najuspešnejši napadi pogosto združujejo prepletenost aktivnosti rdeče in črne ekipe. V simulacijah lahko črna ekipa najprej izvede fizični vdor, nato pa iz notranjega omrežja sproži kibernetski napad kjer rdeča ekipa nadaljuje z identifikacijo in izkoriščanjem digitalnih pomanjkljivosti. Takšna integracija je ključna za testiranje realne odpornosti organizacije na kompleksne napade in omogoča realno presojo, ali obrambni sistemi (modra ekipa) zaznajo tako kompleksne grožnje.

Po končanih simulacijah lahko vijolična ekipa deluje kot posrednik in optimizator, ki povezuje opažanja črne, rdeče in modre ekipe ter pomaga prenesti ugotovitve v varnostne izboljšave.

7 Zaključek

Čeprav so v središču sodobnih varnostnih razprav večinoma digitalni napadi, fizična varnost ostaja ključna, a pogosto spregledana komponenta celostne kibernetske obrambe. Črna ekipa ponuja redko priložnost za etično in realistično preverjanje fizičnih obrambnih mehanizmov, ki jih digitalne analize ne morejo zajeti. Takšne simulacije razkrivajo pomanjkljivosti, ki jih standardni tehnični pregledi pogosto spregledajo, zlasti na ravni človeškega faktorja, organizacijske discipline in fizične dostopnosti.

Z vključevanjem črne ekipe v redne varnostne preglede lahko organizacije:

- Okrepijo celostno odpornost na napade vseh vrst.
- Prepoznajo in ublažijo ranljivosti, povezane s socialnim inženiringom, notranjimi grožnjami in fizičnim dostopom.
- Spodbujajo sodelovanje med IT-jem, fizičnim varovanjem in poslovnim vodstvom.
- Uskladijo varnostno prakso z aktualnimi regulatornimi zahtevami.

Še posebej pomembno je to za sektorje z visokim tveganjem kot so finančne ustanove, energetika, zdravstvo, obrambni sistemi in kritična infrastruktura, kjer je lahko že ena sama spregledana pomanjkljivost usodna.

Poleg tega je z uveljavitvijo evropske direktive CER skupaj z direktivo NIS2, postalo zakonsko obvezno, da organizacije, ki sodijo med kritične subjekte, izvajajo ukrepe za zaščito tudi pred fizičnimi grožnjami. To vključuje testiranje ranljivosti na fizični ravni, kar črna ekipa omogoča na varen, strukturiran in neinvaziven način. Neupoštevanje teh zahtev pa ne pomeni le povečane izpostavljenosti tveganjem, temveč tudi regulatorne in finančne posledice.

Celostna kibernetska varnost se ne konča pri požarnih zidovih in geslih - njen resnični preizkus se zgodi, ko nekdo fizično prestopi prag. In prav zato postaja vloga črne ekipe nepogrešljiv del sodobnega varnostnega ekosistema.

Literatura

[1] Stephen Northcutt, Lenny Zeltser, Scott Winters, Karen Kent Federick, Ronald W. Ritchey, »Inside Network Perimeter Security«, New Riders Publishing, 2023.

[2] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r3.pdf, NIST SP 800-61r3 - Incident Response Recommendations and Considerations for Cybersecurity Risk Management, obiskano 9.6.2025.

[3] https://attack.mitre.org, MITRE ATT&CK Framework - Adversarial Tactics, Techniques, and Common Knowledge, obiskano 9.6.2025.

[4] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf, NIST SP 800-115 - Technical Guide to Information Security Testing and Assessment, obiskano 10.6.2025.

[5] https://www.isecom.org/OSSTMM.3.pdf, OSSTMM (ISECOM), »Chapter 8 - Physical Security Testing«, obiskano 10.6.2025.

[6] https://www.iso.org/standard/27001, ISO/IEC 27001:2022, obiskano 10.6.2025.

[7] https://www.iso.org/standard/75652.html, ISO/IEC 27002:2022, obiskano 10.6.2025.

[8] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf, NIST SP 800-53r5 - Security and Privacy Controls for Information Systems and Organizations, obiskano 14.6.2025.

[9] https://www.atomicredteam.io, Atomic Red Team, obiskano 19.6.2025.

[10] https://caldera.mitre.org, MITRE Caldera, obiskano 19.6.2025.

[11] https://www.enisa.europa.eu/sites/default/files/2024-11/Implementation%20guidance%20on%20security%20measures_FOR%20PUBLIC%20CONSULTATION.pdf, ENISA Implementing Guidance - Environmental And Physical Security, obiskano 21.7.2025.

[12] https://www.enisa.europa.eu/sites/default/files/publications/ENISA%20-%20Guideline%20on%20Security%20Measures%20under%20the%20EECC-%204th%20edition.pdf, ENISA Guideline On Security Measures Under The EECC - D3: Security of systems and facilities, obiskano 21.7.2025.

Pot k uvedbi Evropske denarnice za digitalno identiteto

Davorka Šel, Aleš Pelan, Alenka Žužek Nemec

Ministrstvo za digitalno preobrazbo Republike Slovenije, Ljubljana, Slovenija
davorka.sel@gov,si, ales.pelan@gov.si, alenka.zuzek@gov.si

Prizadevanje za vzpostavitev evropskega okvira za digitalno identiteto je leta 2024 doprineslo do pomembnega koraka naprej, saj je 20. maja 2024 začela veljati novela EU uredbe št. 910/2014 za e-identifikacijo in storitve zaupanja, ki jo poznamo tudi kot Uredba eIDAS 2.0. Ta vzpostavlja pravno podlago za uvedbo evropske denarnice za digitalno identiteto po vsej EU. Z denarnico bodo uporabniki lahko tudi varno pridobili, shranili in delili svoje pomembne dokumente, npr. o izobrazbi in licencah, pooblastila za zastopanje pravnih oseb, finančne podatke in podatke o družbah, ter elektronsko podpisovali oz. v primeru denarnic za podjetja elektronsko žigosali dokumente. Da bi dosegli interoperabilnost med denarnicami, izdanimi s strani držav članic, so v izvedbenih aktih k Uredbi eIDAS 2.0 določeni standardi za evropsko denarnico, ki jih morajo upoštevati vse implementacije denarnic po državah, pravila za certificiranje denarnic in sporočanje Evropski komisiji. Skupne zahteve za denarnico se pripravljajo v okviru Arhitekturnega in referenčnega okvirja (ARF), poleg tega pa Evropska komisija pripravlja tudi referenčno implementacijo denarnice. V prispevku so podrobneje predstavljene nekatere visokonivojske zahteve ARF, ki se nanašajo na področje zasebnosti, še posebej uporaba metod ničelno spoznalnih dokazov (angl. Zero knowledge Proof) za zagotavljanje zasebnosti v ekosistemu denarnic.

Ključne besede:

elektronska identiteta,

eIDAS 2.0,

evropska denarnica za digitalno identiteto,

ničelno spoznalni dokaz,

javne storitve.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.14

1 Uvod

Evropska komisija si že vrsto let prizadeva za vzpostavitev enotnega evropskega okvira za digitalno identiteto. Ključni preboj na tem področju predstavlja posodobitev Uredbe (EU) št. 910/2014 [1], znana kot eIDAS 2.0 [2], ki med drugim uvaja Evropsko denarnico za digitalno identiteto (EUDIW).

Uporabniki bodo lahko s pomočjo EUDIW hranili in delili različna dokazila o identiteti, kot so elektronski osebni dokumenti, potrdila o izobrazbi, zdravstvene podatke in drugo. Denarnica bo omogočala tudi elektronsko podpisovanje in avtentikacijo z visoko ravnjo zaupanja. Ključen pogoj za uspeh EUDIW je zagotavljanje visoke ravni varnosti in zasebnosti, zlasti v kontekstu izmenjave podatkov med državami članicami, uporabniki in različnimi ponudniki storitev.

Namen EUDIW ni le tehničen, saj gre za temeljni gradnik digitalne identitete posameznika, ki mu omogoča, da v različnih državah EU dostopa do storitev na način, ki je varen, nadzorovan in zagotavlja zasebnost. S tem se udejanja tudi eden izmed temeljnih ciljev evropskega notranjega digitalnega trga: odprava ovir za čezmejno digitalno poslovanje in komunikacijo.

2 Evropska denarnica za digitalno identiteto

EUDIW je osebna digitalna aplikacija s pripadajočimi zalednimi sistemi, ki državljanom Evropske unije omogoča, da varno in enostavno dostopajo do različnih digitalnih storitev v javnem in zasebnem sektorju po vsej EU. V skladu z Uredbo eIDAS 2.0 denarnica deluje kot standardizirano, certificirano in interoperabilno sredstvo, v katerem lahko posameznik shranjuje, upravlja in deli svojo elektronsko identiteto ter digitalne dokumente oz. digitalna potrdila o atributih – kot so dokazila o izobrazbi, licencah, poklicnih pooblastilih, bančnih računih ali pravnih zastopanjih. Denarnica uporabniku omogoča elektronsko podpisovanje dokumentov ter dokazovanje identitete in pravic na način, ki zagotavlja visoko raven varnosti in zasebnosti. Ključna prednost EUDIW je, da je zasnovana z načelom "najmanjše nujne informacije", kar pomeni, da lahko uporabnik sam odloča, katere podatke bo delil in v kakšnem obsegu. Ta pa je pogosto brez razkritja celotne vsebine potrdila. Funkcionalnost temelji na uporabi naprednih kriptografskih tehnik, kot je metoda ničelno spoznalnih dokazov (angl. Zero-Knowledge Proof oz. ZKP), ki omogoča preverjanje trditev brez razkritja podatkov. Poleg tega EUDIW temelji na skupnih standardih, ki zagotavljajo medsebojno delovanje denarnic različnih držav članic in skladnost z zakonodajo o varstvu osebnih podatkov. S tem postavlja temelje za zanesljivo, zasebnosti prijazno in vključujočo digitalno družbo prihodnosti.

2.1. Pravni okvir

Uredba eIDAS 2.0 [2], ki je bila sprejeta maja 2024, predstavlja prelomnico v razvoju digitalne identitete v Evropski uniji. Nadgrajuje in razširja izhodišča prvotne uredbe eIDAS [1] iz leta 2014, ki je vzpostavila temelje za čezmejno priznavanje sredstev elektronske identifikacije in storitev zaupanja. Novela uredbe uvaja ambiciozen cilj: vzpostavitev univerzalno dostopne, varne in zasebnosti prijazne evropske digitalne denarnice, ki jo bodo lahko uporabljali vsi prebivalci in podjetja v državah članicah EU. Uredba uvaja pravico vsakega državljana do brezplačne denarnice. Ta denarnica bo omogočala ne le preverjanje identitete, temveč tudi upravljanje z elektronskimi potrdili o atributih, uporabo za elektronsko podpisovanje in nadzorovano deljenje osebnih podatkov. Uredba določa, da mora biti denarnica zagotovljena brezplačno za fizične osebe in da mora biti interoperabilna s sistemi drugih držav EU. Poleg tehničnih zahtev uvaja tudi stroge pogoje glede varnosti, upravljanja s podatki in vloge deležnikov. Uredba daje poseben pomen uporabniški izkušnji, zato predvideva tudi minimalne standarde za dostopnost, prenosljivost in nadzor nad podatki. Uredba vzpostavlja tudi okvir za nadzor, poročanje in certifikacijo storitev in tehnologij EUDIW, s čimer želi zagotoviti trajnostno, zaupanja vredno in tehnološko odprto digitalno okolje.

eIDAS 2.0 torej ni le pravna podlaga, ampak tudi jasna razvojna vizija digitalne Evrope, ki temelji na zaupanju, zasebnosti in tehnični odličnosti.

Pravni okvir vključuje tudi stroge določbe o varstvu osebnih podatkov, kjer se eIDAS 2.0 neposredno navezuje na Splošno uredbo o varstvu podatkov (GDPR) [3]. Vsak obdelovalec ali izdajatelj podatkov mora ravnati v skladu z načelom minimizacije podatkov, zakonitosti in preglednosti obdelave.

Na podlagi uredbe so bili sprejeti tudi izvedbeni akti, ki opredeljujejo tehnične standarde, varnostne protokole, certifikacijske postopke in načine preverjanja skladnosti, ki služijo kot temelj za konkretno implementacijo digitalne denarnice. Izvedbeni akti se naslanjajo na Arhitekturni in referenčni okvir, ki podaja zahteve za implementacijo EUDIW. Z uveljavitvijo prvega paketa izvedbenih aktov, ki določa pravila za osnovne funkcionalnosti denarnic in njihovo certificiranje. je začel veljati tudi zakonski rok za uvedbo denarnic s strani držav članic, ki je 24. december 2026. Drugi paket izvedbenih aktov, poveznih z denarnico, naslavlja , , seznam certificiranih evropskih denarnic za digitalno identiteto, odzivanje na kršitve varnosti evropskih denarnic, registracijo na denarnice zanašajočih se strank in čezmejno ujemanje identitete fizičnih oseb je bil objavljen 6. maja 2025, razen akta z zahtevami za ponudnike elektronskih potrdil o atributih. Objava tega akta se pričakuje do septembra 2025.

Države članice pa morajo na podlagi Uredbe eIDAS 2.0 in spremljajočih izvedbenih aktov tudi opredeliti obveznosti v svoji nacionalni zakonodaji, da omogočijo uvedbo EUDIW.

2.2. Arhitektura in referenčni okvir (ARF)

Evropska komisija je v pomoč državam pri implementaciji EUDIW objavila Arhitekturni in referenčni okvir (ARF) [4], ki določa visokonivojske zahteve za vse implementacije v okviru ekosistema EUDIW.

ARF predstavlja osrednji tehnični dokument Evropske komisije, katerega namen je usmerjati razvoj, uvedbo in interoperabilnost EUDIW. Evropska komisija ga oblikuje v sodelovanju z organi za standardizacijo, industrijskimi konzorciji in predstavniki držav članic. ARF sicer ni pravno zavezujoč dokument, a je zaradi svoje povezave z eIDAS 2.0 in izvedbenimi akti, na katere vpliva, de facto standard, katerega uporaba je za države članice nujna za zagotovitev interoperabilnosti in enotne uporabniške izkušnje. ARF vključuje sistemski pogled na vse ključne komponente in vloge v ekosistemu elektronskih denarnic, vključno z EUDIW, izdajatelji identitet in atributov, ponudniki storitev ter varnostnimi komponentami, kot so varni kriptografski moduli. Poleg funkcionalnih zahtev definira tudi tehnične protokole, specifikacije vmesnikov, vrste in formate potrdil, načine preverjanja in prenosa podatkov, ter priporočene kriptografske algoritme.

Dokument je strukturiran tako, da omogoča prilagodljivo implementacijo, ki upošteva različne nacionalne pristope in obstoječe tehnične zmogljivosti, hkrati pa zagotavlja interoperabilnost na evropski ravni. ARF se osredotoča tudi na življenjski cikel EUDIW, od inicializacije do ponovne uporabe in preklica, ter vključuje napredne koncepte, kot so mehanizmi zaščite pred sledenjem uporabnikov. Pomemben del ARF predstavljajo tudi tokovi informacij med deležniki, ki so jasno opisani skozi komunikacijske scenarije in uporabniške tokove, vključno z vlogami ponudnika identitete, ponudnika potrdil o atributih in drugih zanašajočih strank.

Zaradi nenehnega tehnološkega razvoja se ARF redno posodablja, kar omogoča vključevanje novih rešitev in postopno nadgrajevanje obstoječih sistemov.

Ključno sporočilo ARF je, da tehnična interoperabilnost ni dosegljiva le s spoštovanjem minimalnih zahtev, temveč z aktivnim sodelovanjem držav članic, standardizacijskih teles in industrije, pri čemer mora biti vedno v ospredju uporabnik in varovanje njegove zasebnosti.

2.3. Referenčna implementacija in standardizacija

Evropska komisija poleg ARF aktivno razvija tudi referenčno implementacijo EUDIW. Njena glavna naloga je podpreti države članice pri razvoju lastnih nacionalnih rešitev, zagotoviti minimalne funkcionalnosti, preveriti skladnost s tehničnimi zahtevami in olajšati doseganje čezmejne interoperabilnosti. Referenčna implementacija ni obvezna za uporabo, vendar služi kot osnovni vzorec oziroma dokaz koncepta, ki upošteva vse bistvene komponente ARF. S tem omogoča testiranje in integracijo z različnimi deležniki v ekosistemu, kot so ponudniki identitet, izdajatelji potrdil o atributih in ponudniki storitev.

Ob tem imajo standardi v ekosistemu evropske denarnice za digitalno identiteto ključno vlogo pri zagotavljanju skladnosti, varnosti in interoperabilnosti. Brez skupno dogovorjenih specifikacij bi vsak ponudnik in vsaka država lahko razvijala rešitve, ki bi bile medsebojno nezdružljive, kar bi neposredno ogrozilo uspešnost EUDIW. Zato Evropska komisija v tesnem sodelovanju z evropskimi in mednarodnimi standardizacijskimi telesi (npr. ETSI, CEN, ISO, W3C) pripravlja nabor tehničnih standardov, ki vključujejo definicije struktur elektronskih potrdil, varnostne protokole, kriptografske algoritme, formate potrdil o atributih in komunikacijske protokole med komponentami sistema.

Posebno pomembni so standardi, ki omogočajo selektivno razkritje podatkov, zaščito pred sledljivostjo ter uporabo varnih kriptografskih aplikacij in naprav. Prav tako so določeni tudi varnostni profili za mobilne naprave in zahteve glede certificiranja programske ter strojne opreme. Ključno pri tem je, da vsi standardi niso le tehnične narave, ampak zajemajo tudi organizacijske in upravljavske vidike, kot so postopki izdaje in preklica potrdil, beleženje revizijskih sledi ter mehanizmi za obvladovanje incidentov. Referenčna implementacija tako služi kot laboratorij za uveljavljanje teh standardov v praksi, kar je bistven korak za uspešno implementacijo nacionalnih denarnic. Le s skupno uporabo odprtih, preverljivih in široko sprejetih standardov bo mogoče zagotoviti, da bo EUDIW resnično delovala kot zanesljivo, varno in univerzalno orodje za digitalno identiteto med državami članicami.

2.4. Ekosistem evropske denarnice za digitalno identiteto

Ekosistem EUDIW je kompleksen in večslojen sistem, v katerem sodelujejo številni akterji, vsak z jasno določeno vlogo in odgovornostmi. Arhitektura sistema EUDIW temelji na modularnem in decentraliziranem modelu, ki omogoča interoperabilno delovanje različnih nacionalnih rešitev znotraj enotnega evropskega okvira.

Slika 1 prikazuje poenostavljeno arhitekturo ekosistema EUDIW s ključnimi deležniki in tokovi podatkov med njimi. V jedru ekosistema je komponenta same denarnice, ki jo uporabnik uporablja za hranjenje osebnih podatkov, elektronskih potrdil o atributih in za izvajanje elektronskih transakcij, kot so prijava v spletne storitve, dokazovanje lastnosti ali elektronsko podpisovanje dokumentov. Denarnico izdaja zaupanja vreden *ponudnik denarnice* v skladu s pravnim okvirjem iz poglavja 2.1.

Temeljni akter v ekosistemu je tako imenovani ponudnik identitet (ang. Person Identification Data Provider, v nadaljevanju *ponudnik PID*), ki je običajno državni organ in skrbi za izdajo osnovne digitalne identitete uporabniku. Ta identiteta vključuje podatke, kot so ime, priimek, datum rojstva, kraj rojstva in državljanstvo, ki tvorijo osnovo za preverjanje posameznika. Poleg ponudnika PID v ekosistemu sodelujejo tudi ponudniki elektronskih dokumentov oz. *ponudniki elektronskih potrdil o atributih* (angl. Attestation Providers), kot so državni organi, izobraževalne institucije, poklicne zbornice, delodajalci ali drugi organi, ki potrjujejo dejstva o uporabniku, npr. izobrazbo, licenco, pravico do zastopanja podjetja ipd. Medtem ko elektronska potrdila za podpisa izdaja *ponudnik kvalificiranih potrdil za elektronski podpis*. V podporo celotnemu ekosistemu so vzpostavljeni tudi varnostne komponente, kot so varne kriptografske naprave (angl. wallet secure cysptographic device - WSCD), ki zagotavljajo, da so zasebni ključi in občutljivi podatki zaščiteni na ustrezni ravni, ter orodja za upravljanje življenjskega cikla potrdil, njihovo preklicevanje in ponovno izdajo.

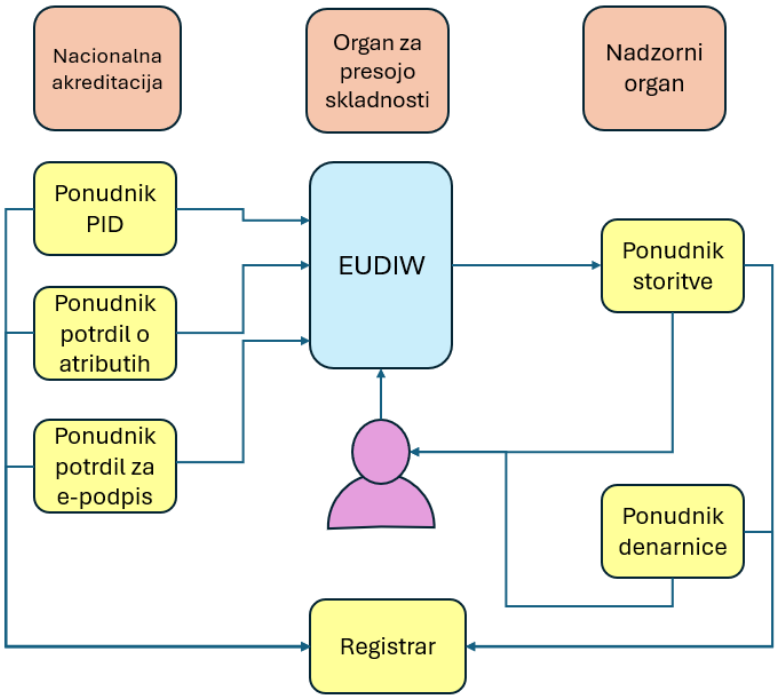
Na drugi strani so *ponudniki storitev* (angl. Relying Parties), ki dostopajo do uporabnikovih podatkov preko standardiziranih protokolov in vmesnikov. Vsa komunikacija med temi akterji poteka prek varnih protokolov, ki vključujejo identifikacijo, preverjanje pristnosti, prenos selektivno razkritih podatkov ter uporabo podpisov.

Pomembno vlogo igra tudi *registrar*. Registrar je odgovoren za preverjanje identitete zanašajočih strank in potrditve seznama atributov, do katerega lahko dostopajo, preden izda potrdilo zanašajoči stranki (angl. access certificate), da se lahko vključi v ekosistem.

Na najvišji ravni nadzora in zagotavljanja skladnosti nad sistemom so *slovenska akreditacija, nacionalni nadzorni organ, upravljavec certifikacijske sheme, organ za presojo skladnosti*. Organ za presojo skladnosti skrbi za presojo skladnosti in varnosti EUDIW v ekosistemu. Ta organ izvaja certifikacijo EUDIW na podlagi nacionalne certifikacijske sheme s čimer zagotavlja, da so tehnične rešitve skladne z zahtevami eIDAS 2.0, ARF in relevantnimi varnostnimi standardi ter potrjuje skladnost komponent (denarnic, WSCD modulov, …) s tehničnimi in varnostnimi zahtevami, opredeljenimi v nacionalni shemi. Poleg tega arhitektura vključuje tudi upravljavske mehanizme, kot so beleženje transakcij, upravljanje soglasij uporabnikov, sistemi za obvladovanje incidentov ter mehanizmi za preklic in ponovno izdajo potrdil.

Vzpostavljena je hierarhija zaupanja, kjer nacionalni nadzorni organi nadzirajo delovanje ponudnikov v državi, Evropska komisija pa vzdržuje nadzor nad usklajenostjo celotnega sistema. Sistem mora podpirati tudi različne uporabe – od spletnih prijav in podpisovanja do uporabe v fizični prisotnosti (angl. proximity use), pri čemer mora vsak podatek biti predstavljen v kriptografsko zaščiteni, preverljivi in zanesljivi obliki.

V arhitekturnem smislu gre torej za porazdeljen sistem, v katerem so vsi akterji med seboj povezani preko standardiziranih tehničnih protokolov in se zanašajo na enoten varnostni model. Poudarek na decentralizaciji, interoperabilnosti in varstvu podatkov uporabnika predstavlja temeljno razliko v primerjavi z dosedanjimi centraliziranimi sistemi identitet, kar EUDIW postavlja kot inovativno, zaupanja vredno in trajnostno rešitev v evropskem digitalnem prostoru.



Slika 1: Poenostavljena arhitektura ekosistema EUDIW.

2.5. Izzivi interoperabilnosti

Vzpostavitev ekosistema EUDIW temelji na ideji čezmejne uporabnosti in enotne uporabniške izkušnje, ne glede na državo članico, v kateri je bila denarnica izdana. Vendar pa to zahteva izjemno visoko stopnjo tehnične in organizacijske usklajenosti, kar predstavlja enega največjih izzivov v trenutni fazi uvajanja. Interoperabilnost pomeni, da sistemi različnih držav medsebojno razumejo podatkovne strukture, protokole, varnostne zahteve in postopke preverjanja, kar pa je težko doseči ob dejstvu, da imajo države različne tehnične zmogljivosti, obstoječe nacionalne sheme e-identitet in zakonodajne posebnosti. Nekatere države razvijajo EUDIW na podlagi referenčne implementacije, druge pa vzpostavljajo povsem nove platforme, kar lahko privede do razlik med implementacijami.

Poleg tehničnih razlik obstajajo tudi neenotnosti v razumevanju pravnih zahtev uredbe eIDAS 2.0 in izvedbenih aktov. Pomanjkanje končnih standardov in zamude pri potrjevanju tehničnih specifikacij še dodatno otežujejo prizadevanja držav, da bi začele pravočasno testiranje in integracijo svojih rešitev. Pomanjkanja končnih standardov in specifikacij zlasti na področjih formatov potrdil, primernosti WSCD, protokolov za predstavitev podatkov in zagotavljanja zasebnosti pri uporabi denarnice bistveno otežuje nadaljnji razvoj denarnic. Poleg tega obstajajo tudi izzivi glede vključevanja ponudnikov storitev in ponudnikov potrdil o atributih, ki morajo biti tehnično in pravno usposobljeni za sodelovanje v ekosistemu.

Interoperabilnost pa ni odvisna le od enotnih specifikacij, temveč tudi od ustrezne razpoložljivosti testnih okolij, učinkovitega certificiranja in natančno opredeljenih vlog v ekosistemu. Pomembno je tudi zagotoviti, da bodo implementacije denarnic, ki temeljijo na referenčni implementaciji ali neodvisnih rešitvah, sposobne varnega preverjanja in predstavitve podatkov ne glede na državo ali tip ponudnika storitve. Evropska komisija poskuša s pomočjo pilotnih projektov, ko so POTENTIAL [5], v katerem sodeluje tudi Slovenija ter EU Digital Wallet Consortium (EWC) [6], Nobid [7] in DC4EU [8], zagotoviti podporo državam pri razvoju interoperabilnih rešitev. Kljub temu pa se izkušnje iz teh projektov še niso povsem prenesle v enotno prakso.

Na dolgi rok bo uspeh EUDIW močno odvisen od tega, ali bo mogoče zagotoviti neprekinjeno sodelovanje med regulatorji, razvijalci, ponudniki storitev in uporabniki. Interoperabilnost ne pomeni le tehnične usklajenosti, temveč zahteva tudi vzpostavitev trajnih mehanizmov sodelovanja, izmenjave znanja in skupnega nadzora nad kakovostjo storitev. Zato je bistveno, da se interoperabilnost razume kot dinamičen proces, ki ga bo treba redno nadgrajevati in prilagajati novim razmeram, tehnologijam in uporabniškim pričakovanjem.

3 Elektronska potrdila o atributih in vloga njihovih ponudnikov

Elektronska potrdila o atributih so temeljni gradniki ekosistema EUDIW, saj omogočajo prenos zaupanja iz preverjenih virov na način, ki je strojno berljiv, interoperabilen in varen. Atributi predstavljajo značilnosti posameznika ali organizacije, kot npr. da ima vozniško dovoljenje, končano izobrazbo, stalno bivališče, pooblastilo za zastopanje podjetja, poklicne kvalifikacije, članstvo v zbornici, davčno številk, ipd. Ti atributi so shranjeni v obliki strukturiranih podatkovnih zapisov, ki jih izdajajo zaupanja vredne ustanove. Ključni element sistema je, da uporabnik lahko te podatke hrani v svoji denarnici in jih po potrebi predstavi različnim ponudnikom storitev, ne da bi ob tem razkril prekomerne količine osebnih podatkov oz. uporabi t.i. selektivno razkritje atributov [9].

Osrednjo vlogo pri izdaji osnovne digitalne identitete ima ponudnik PID. Ponudnik PID zagotovi temeljno identiteto uporabnika, ki je podlaga za izdajanje drugih potrdil. Vzporedno z njim delujejo izdajatelji drugih potrdil o atributih kot so univerze, zdravstvene ustanove, poklicne zbornice, podjetja in druge institucije, ki so pooblaščene za potrjevanje specifičnih dejstev o posamezniku. Ta dva tipa izdajateljev se razlikujeta predvsem po obsegu odgovornosti in naravi podatkov, ki jih potrjujeta, vendar morata oba zagotavljati visoko stopnjo zaupanja, doslednost pri izdaji in skladnost s predpisi uredbe eIDAS 2.0 ter splošno uredbo o varstvu podatkov GDPR.

Pomembno je, da se vsa potrdila izdajo v skladu s standardi, opredeljenimi v izvedbenih aktih in ARF, ter da se omogoči njihova poznejša ponovna izdaja brez ogrožanja zasebnosti. To pomeni, da mora biti proces obnove

potrdil možen brez potrebe po celoviti ponovni registraciji uporabnika, kar zahteva zanesljivo vezavo potrdil na identiteto, ne da bi se pri tem razkrile nepotrebne informacije. Zanesljivost in razširjenost potrdil o atributih sta neposredno odvisni od tehnične in organizacijske usklajenosti ponudnikov. Zato se znotraj ARF natančno opredeljujejo formati potrdil, življenjski cikel izdajanja in preklica, možnosti ponovne izdaje ter postopki preverjanja veljavnosti. Poleg tega morajo biti izdajatelji vključeni v mehanizme, ki omogočajo interoperabilnost na ravni EU, kar vključuje registracijo, certificiranje in obveščanje o izdanih oziroma preklicanih potrdilih. Elektronska potrdila o atributih tako ne predstavljajo zgolj digitalne različice papirnih dokumentov, temveč omogočajo bolj finozrnat, varnejši in nadzorovan način izmenjave informacij, ki je nujen za vzpostavitev zaupanja vrednega digitalnega okolja v Evropi.

Zaradi potrebe po visoki ravni zaščite podatkov je nujno, da tako ponudnik PID kot ponudniki potrdil o atributih delujejo v skladu s predpisi o varstvu osebnih podatkov (GDPR) in uporabljajo tehnike, kot je ZKP.

4 ZKP kot temelj za varovanje zasebnosti v EUDIW

ARF namenja posebno pozornost zaščiti zasebnosti uporabnikov. Eden ključnih mehanizmov za doseganje tega cilja je uporaba kriptografskih metod ZKP. ZKP predstavlja sredstvo, s katerim lahko uporabnik EUDIW dokaže določeno lastnost (npr. da je polnoleten ali da ima določeno licenco) brez razkritja same vrednosti atributa ali dodatnih podatkov.

Pomembna prednost ZKP je tudi preprečevanje povezljivosti med različnimi transakcijami uporabnika. Če bi uporabnik za vsako storitev vedno znova razkrival iste identifikacijske podatke, bi to omogočilo sledenje in profiliranje. Z ZKP pristopi pa lahko vsakokrat ustvari edinstven dokaz, ki ne razkriva identitete in ne omogoča povezovanja različnih transakcij. Tako se bistveno izboljša varstvo zasebnosti in zmanjša tveganje zlorabe osebnih podatkov.

Tehnologija ZKP pa ne pomeni le prednosti za uporabnika. Tudi ponudniki storitev imajo od nje koristi, saj jim omogoča zanesljivo preverjanje pogojev brez potrebe po shranjevanju ali obdelavi osebnih podatkov. To zmanjšuje obremenitve glede skladnosti z GDPR, saj ponudniki ne prevzemajo odgovornosti za varovanje podatkov, ki jih sploh ne prejmejo. Poleg tega ZKP odpira vrata za uporabo EUDIW v sektorjih, kjer je varovanje zasebnosti še posebej pomembno – na primer v zdravstvu, financah ali pri glasovanju na daljavo.

Izvedba ZKP v evropskem kontekstu pa prinaša tudi svojevrstne izzive. Tehnologija mora biti standardizirana, preverljiva in izvedljiva v vseh državah članicah. Zahteva tudi močno podporo na ravni infrastrukture, vključno z varnimi kriptografskimi napravami in digitalnimi podpisi, ki zagotavljajo integriteto dokazov. V okviru delovnih skupin Evropske komisije se pripravljajo visoko nivojske zahteve, ki pa še niso potrjene. Končne verzije zahtev bodo objavljene v eni naslednjih verzij ARF. V nadaljevanju pa je podanih nekaj kandidatov zahtev iz delovnega dokumenta Evropske komisije [10] za vključitev v ARF:

- *Temeljne funkcije ZKP*: Shema ZKP mora omogočati dokazovanje prisotnosti atributov v PID ali potrdilih, njihove veljavnosti, nepreklicanosti in vezave na WSCD, pri čemer so vsi atributi skriti. Poleg tega naj podpira možnost prikritja izdajatelja potrdila.
- *Dokaz posedovanja*: Shema mora omogočati dokazilo, da uporabnik razpolaga z določenim tipom potrdila.
- *Zasebna vezava*: ZKP naj omogoča zasebno povezovanje PID in potrdila, tako da dokaže prisotnost istega atributa v obeh.
- *Psevdonimi*: Shema naj omogoča ustvarjanje preverljivih psevdonimov, izpeljanih iz tajnega atributa in identifikatorja ponudnika storitve, brez razkritja samega atributa.

- *Uporabnost v različnih kontekstih*: ZKP mora biti uporabna v bližnjih (npr. osebna prisotnost) in oddaljenih (npr. spletna storitev) scenarijih, brez bistvenega vpliva na uporabniško izkušnjo.
- *Združljivost z obstoječimi formati*: Naj omogoča dokazovanje nad PIDs in potrdili v formatih ISO/IEC 18013-5 in SD-JWT VC.
- *Brez sledenja*: ZKP ne sme uvajati nobenih komunikacij ali metapodatkov, ki bi omogočali sledenje uporabniku.
- *Standardizirani algoritmi*: Vsi uporabljeni algoritmi morajo biti standardizirani pri telesih, ki jih priznava Evropska komisija.
- *Skladnost z visoko ravnjo zanesljivostjo*: Uporaba ZKP ne sme ovirati zagotavljanja visoke ravni zanesljivosti avtentikacije uporabnika.

4.1. Implementacija in uporaba ZKP v EUDIW

Obstajata dva prevladujoča pristopa k ZKP, ki se obravnavata v okviru procesa priprave zahtev ARF: to sta BBS+ in zk-SNARK. Prvi omogoča ustvarjanje več-sporočilnega podpisa, pri katerem je mogoče razkriti le podmnožico podatkov iz posameznega sporočila, hkrati pa ohraniti dokaz o pristnosti celote. Entiteta, ki pozna podpis in izvirni niz podpisanih sporočil, lahko ustvari dokaz s katerim razkrije samo podmnožico teh sporočil. Tak dokaz je mogoče preveriti zgolj z uporabo javnega ključa podpisnika in razkritih sporočil iz te podmnožice. BBS+ velja za bolj primernega za okolja z omejenimi viri, kot so mobilne naprave, saj ne zahteva visoke računske moči, vendar pa zahteve spremembe v procesu podpisovanja potrdil o atributih. Po drugi strani pa zk-SNARK temelji na dokazovanju veljavnosti trditev z uporabo aritmetičnih vezij in ponuja visoko stopnjo prilagodljivosti. Omogoča dokazovanje kompleksnih pogojev (npr. dohodek nad določenim pragom), pri čemer so vhodni podatki popolnoma skriti.

Tabela 1 prikazuje razlike med tehnologijama ZKP, ki sta kandidata za uporabo v kontekstu EUDIW.

Tabela 1: Primerjava BBS+ in zk-SNARK pristopov.

4.1.1.	**Lastnost**	**4.1.2.**	**BBS+**	**4.1.3.**	**Zk-SNARK**
4.1.4.	**Selektivno razkritje**	4.1.5.	DA	4.1.6.	DA
4.1.7.	**Podpora za kompleksne trditve**	4.1.8.	NE	4.1.9.	DA
4.1.10.	**Računska zahtevnost**	4.1.11.	NIZKA	4.1.12.	VISOKA
4.1.13.	**Zahteva začetni setup**	4.1.14.	NE	4.1.15.	OBIČAJNO DA

Vir: G - Zero Knowledge Proof (delovni dokument) [6]

Tabela kaže, da je izbira tehnologije močno odvisna od konkretnega primera uporabe, tehničnih zmožnosti uporabniške naprave in zahtev glede zasebnosti. Možnost uporabe in izbira konkretnih tehnologij ZKP je še vedno predmet razprave v skupinah EU, ki se ukvarjajo s pripravo ARF in tehničnih specifikacij, zato končni mehanizmi uporabe ZKP še niso potrjeni.

Za ilustracijo je v nadaljevanju naštetih nekaj primerov uporabe ZKP v EUDIW:

- Uporabnik želi dokazati, da je starejši od 18 let, za dostop do spletne trgovine ali dogodka. ZKP omogoča, da se iz potrdila izlušči zgolj ta podatek, ne da bi se razkril točen datum rojstva, ime ali kakršna koli druga osebna informacija.
- Namesto CAPTCHA lahko spletne strani uporabijo ZKP, s katerim uporabnik dokaže, da je človeško bitje s potrjeno identiteto. Rešitev povečuje dostopnost, zlasti za invalide, in zmanjšuje zlorabe.
- Uporabnik, ki večkrat obišče isto spletno stran, se lahko identificira z istim psevdonimom, brez razkrivanja svoje identitete. ZKP omogoča takšno vezavo prek derivacij atributov, ki ostajajo skriti.
- ZKP podpira t. i. "privacy-preserving revocation", kjer se lahko preveri, ali je potrdilo še veljavno, ne da bi razkrili identitete uporabnika ali podatka o njegovi dejavnosti.

5 Zaključek

Uvedba EUDIW je eden najambicioznejših projektov digitalne preobrazbe Evropske unije. Predstavlja tehnološki, pravni in organizacijski izziv, ki zahteva usklajeno sodelovanje vseh držav članic, razvijalcev, zakonodajalcev, ponudnikov storitev in končnih uporabnikov.

Ključ do uspešne uvedbe leži v vzpostavitvi trdnega zaupanja med vsemi deležniki. To zaupanje lahko temelji le na transparentnih pravilih, interoperabilnih standardih, varnosti ter učinkovitem varovanju zasebnosti. Tehnologija ZKP se v tem kontekstu kaže kot nepogrešljiv gradnik, saj ščiti uporabnika pred neželenim razkritjem podatkov in hkrati poenostavlja protokole za preverjanje identitete. Vendar pa so končne zahteve glede uporabe ZKP še predmet razprav in usklajevanja med državami članicami in komisijo. Pričakovati je, da se bo razprava na to temo nadaljevala v drugi polovici leta 2025.

Literatura

[1] Uredba (EU) št. 910/2014 o elektronski identifikaciji in storitvah zaupanja za elektronske transakcije na notranjem trgu, https://eur-lex.europa.eu/eli/reg/2014/910/oj/eng, pridobljeno 1. 8. 2025

[2] Uredba (EU) 2024/1183, https://eur-lex.europa.eu/eli/reg/2024/1183/oj/eng, pridobljeno 1. 8. 2025

[3] Uredba (EU) št. 2016/67, https://eur-lex.europa.eu/legal-content/SL/TXT/?uri=CELEX:32016R0067, pridobljeno 1. 8. 2025

[4] EUDI Wallet Architecture and Reference Framework 2.4.0, https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework, pridobljeno 1. 8. 2025

[5] European Digital Identity Wallet, https://www.digital-identity-wallet.eu, pridobljeno 1. 8. 2025

[6] EUDI Wallet Consortium, https://eudiwalletconsortium.org, pridobljeno 1. 8. 2025

[7] NoBID Consortium, https://www.nobidconsortium.com, pridobljeno 1. 8. 2025

[8] DC4EU, https://www.dc4eu.eu, pridobljeno 1. 8. 2025

[9] ETSI TR 119 476 V1.2.1, https://www.etsi.org/deliver/etsi_tr/119400_119499/119476/01.02.01_60/tr_119476v010201p.pdf, pridobljeno 1. 8. 2025

[10] EU Digital Identity Wallet , https://eu-digital-identity-wallet.github.io, pridobljeno 1. 8. 2025

Zmogljiva integracija brez REST strežnika ali mikrostoritev na primeru Python-a in .NET

Matjaž Prtenjak

Endava d.o.o, Ljubljana, Slovenija
matjaz@matjazev.net

V današnjem času umetne inteligence in podatkovno vodenih odločitev ni več vprašanje ali, temveč, kako hitro lahko razvijalci vklopimo napredne funkcionalnosti v naše aplikacije. Python je nesporni kralj podatkovne znanosti in umetne inteligence – vendar mnogi razvijalci še vedno ustvarjamo poslovne aplikacije v C# in .Net. Prispevek demonstrira neposredno izvajanje Python kode znotraj .Net aplikacij, povezovanje z orodji, kot so: NumPy, Pandas, Matplotlib, TensorFlow, PyTorch, OpenCV, itd., uporabo virtualnih okolij, tesno integracijo z .Net tipi (tudi naprednimi, kot je Span). Prispevek je smiselno nadaljevanje mojih prispevkov iz OTS 2019 (Vue), 2022 (Blazor) in 2023 (Blazor Hybrid), vendar pa je vseeno zaključen prispevek, neodvisen od prejšnjih prispevkov in predznanja. Nadaljevanje pa je v smislu, da .Net okolje zopet nadgrajujemo in povezujemo z zunanjim svetom.

Ključne besede:
.Net,
Python,
integracije,
REST,
Blazor.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.15

1 Uvod

Python in .NET sta orodji/okolji, ki blestita na različnih področjih. .NET je robusten, strukturiran in primeren za večja podjetja, medtem ko Python s svojo preprostostjo in obsežno zbirko knjižnic izstopa predvsem na področju analize podatkov, umetne inteligence in avtomatizacije.

V povezavi ponujata najboljše iz obeh svetov: zmogljivost in širino uporabe.

Ta članek opisuje, kako lahko razvijalci uporabijo Python-ove zmogljivosti neposredno znotraj svojih .NET aplikacij.

Programsko kodo v Python-u bomo klicali direktno iz C# programske kode, kot da bi bila del C# kode, kar pomeni, da ne bomo uporabljali kakšnih mikrostoritev, povezav preko REST, sporočilnih vrst ali česa podobnega.

Celotne programske kode ne bo v članku, temveč jo lahko snamete s spleta, z GitHub-a, na naslovu: https://github.com/MPrtenjak/OTS2025-NET-And-Python

1.1. Zakaj povezati .Net in Python?

Glavni razlog je razpoložljivost brezplačnih in zmogljivih Python knjižnic. Tipičen primer je obdelava PDF dokumentov, kjer Python nudi knjižnice, kot je PyPDF2, brezplačno, medtem ko so .NET alternative pogosto plačljive.

V povezavi s Python svetom lahko:

- hitro prototipiramo in testiramo zamisli,
- uporabljamo napredne algoritme strojnega učenja,
- avtomatiziramo rutinske procese,
- razširimo funkcionalnosti brez dodatnih stroškov.

1.2. Namen prikazane aplikacije

Ker je bistvo prispevka v prikazu integracije .Net in Python okolja, ne bomo razvili »uporabne« aplikacije, temveč nekaj primerov integracije med okoljema. V vseh primerih bo realno delo opravljala Python koda .Net del rešitve pa bo služil samo za pripravo podatkov in prejem rezultatov.

Razvili bomo pet različnih primerov integracije, od najlažjega do (malce) bolj kompleksnega:

1. `HelloWorld`: klasičen programski primer 'Pozdravljen svet! Pozdravlja te <ime>.', kjer bo spremenljivko '`ime`' podal .Net del, sporočilo generiral Python in rezultat spet prejel .Net.
2. `PdfExtractor`: primer, kjer bomo Python kodi podali PDF dokument, Python bo v dokumentu prebral vso besedilo in nam ga vrnil kot niz znakov.
3. `ShowGraph`: primer, kjer bomo s pomočjo Python knjižnic `seaborn` in `matplotlib` ustvarili graf ter ga kot niz `byte`-ov vrnili .Net kodi, ki ga bo prikazala.
4. `OpenAI`: primer, kjer se bo Python preko knjižnice `openai` povezal z `OpenAI` ter generiral kratko otroško zgodbico o zmajčku.
5. `HTMLExtractor`: v tem primeru pa bomo z uporabo Python-ovih knjižnic `selenium`, `bs4` in programom `webdriver` pridobili žive podatke o cenah delnic na Ljubljanski borzi.

.Net del programa bo razvit v treh projektih:

1. Glavni del programa bo .Net knjižnica, ki bo v resnici skrbela za povezavo in predajo podatkov med .Net in Python-om.
2. Projekt s programom, ki teče v ukazni vrstici ter kliče prej omenjeno knjižnico.
3. REST API projekt s petimi API funkcijami, ki posledično kličejo prej omenjenih pet funkcij v Python-u.

2 Povezave med Python in .Net

Danes različne tipe programske opreme ali pa programsko opremo, ki je razvita v različnih ogrodjih, z različnimi programskimi jeziki, največkrat kličemo z medsebojno uporabo REST API funkcij ali pa slednje povežemo z uporabo sporočilnih vrst.

Vendar pa, kot je omenjeno že na začetku, danes ne bomo govorili o tovrstni povezavi, temveč bomo .Net in Python povezali neposredno v eno celoto.

Za povezavo med .Net in Python lahko uporabimo različne knjižnice, jaz pa bom danes omenil tri ter prikazal kratko tabelo z razlikami med njimi.

Tabela 1: Pregled razlik med tremi knjižnicami, uporabnimi za povezavo python in .Net.

Značilnost	IronPython	Python.Net	CSnakes
Podprta različica Pythona	Primarno Python 2.7; omejena podpora za Python 3	Podpira Python 2.x in 3.x	Podpira Python od 3.9 dalje
Integracija z .Net	Python koda teče na .Net-ovem Common Language Runtime (CLR)	Omogoča Python kodi dostop do .Net knjižnic; uporablja standardni CPython interpreter	Python v .Net teče z uporabo direktnih klicev Python-ovega C-API
Zmogljivost	Relativno najslabša hitrost delovanja med temi tremi	Na splošno boljša zmogljivost; neposreden dostop do .Net knjižnic	Visoka zmogljivost; neposredna integracija na C-nivoju
Podpora za Python knjižnice tretjih oseb	Omejena podpora, zlasti za module s C-razširitvami	Dobra podpora; omogoča uporabo večine Python knjižnic, tudi s C-razširitvami	Popolna podpora Python knjižnicam, vključno s C-razširitvami in virtualnimi okolji
Nitnost in GIL	Nima globalnega interpreter locka (GIL); boljša podpora večnitnosti	Podvržen GIL; omejena resnična večnitnost	Podpira CPython 3.13 način "free-threading"
Razvojno okolje	Integracija z Visual Studiom; podpora za .Net orodja	Zahteva standardna orodja za Python razvoj	Podpora za Visual Studio in druge IDE-je; omogoča "hot reload" Python kode
Primernost za uporabo	Idealno za aplikacije, ki močno temeljijo na .Net in vključujejo nekaj Python skriptiranja	Primerno za Python aplikacije, ki potrebujejo dostop do .Net knjižnic	Primerno za .Net aplikacije, ki zahtevajo visoko zmogljivo integracijo s Python knjižnicami

S pregledom na tabelo lahko sami ocenite primernost posamezne knjižnice za vaše potrebe, jaz pa se bom tokrat osredotočil na knjižnico CSnakes.

3 Postopek

V tem prispevku ne bom opisoval knjižnice kot takšne, saj je podroben opis seveda dostopen na spletu, bom pa opisal konkreten način uporabe ter postopke, ki sem jih uporabljal.

Kot vsako drugo knjižnico, lahko tudi CSnakes uporabite na različne načine in tudi sama omogoča več načinov inicializacije in uporabe. A ne glede na način uporabe, morate nekako skozi par korakov:

1. Priprava Python okolja:
   a. lahko vnaprej instalirate sami ali
   b. prepustite CSnakes knjižnici, da ga avtomatično naloži s spleta in instalira.
2. Priprava potrebnih knjižnic:
   a. lahko vnaprej instalirate sami ali
   b. prepustite CSnakes knjižnici, da jih poišče na spletu in instalira.
3. Uporaba kode v našem .Net programu.

3.2. Priprava Python okolja

V kolikor Python okolja ne pripravite vnaprej, lahko knjižnici zaukažete, naj ga sama poišče in instalira. Pri tem boste naleteli na sledeče težave:

1. Python se fiksno instalira v podmapo uporabnikove glavne mape, torej nekam na: `\users\<ime>\AppData\Roaming\SCnakes…`
   a. Če nič drugega, to pomeni, da bi imel vsak uporabnik vašega programa svojo kopijo Python-a.
2. Uporabnik mora imet dovolj pravic za instalacijo in zagon programov is podmape `\users`.
3. Program mora imeti dostop do spleta.

Seveda pa knjižnica »ni neumna« in pred instalacijo preveri, ali se Python na iskani lokaciji že nahaja in v tem primeru ga seveda ne instalira ponovno.

> 😎 ***Moje priporočilo:*** *Vaš instalacijski program naj vnaprej instalira ustrezno različico Python-a, najbolje kar ob vaš program in CSnakes naj potem uporablja to različico.*

3.3. Priprava Python knjižnic

Podobno kot pri instalaciji samega Python okolja, lahko tudi instalacijo Python knjižnic prepustite CSnakes knjižnici ali pa jih vi sami instalirate že vnaprej.

V primeru avtomatične instalacije Python knjižnic s strani Csnakes-a slednji uporabi program `pip` v Python podmapi in s pomočjo pip-a instalira vse zahtevane knjižnice. Seznam knjižnic mu podate vi preprosto tako, da jih naštejete v dokumentu z imenom `requirements.txt`.

> 😎 ***Moje priporočilo:*** *Vaš instalacijski program naj vnaprej pripravi virtualno Python okolje, kamor vi že vnaprej instalirate vse potrebne Python knjižnice.*

Kaj je Pythonovo virtualno okolje?

Pythonovo virtualno okolje je kot peskovnik za vaš projekt — ločeno, čisto okolje, kjer lahko nameščate knjižnice, ne da bi vplivali na preostali sistem. Ko ustvarite okolje z ukazom:

```
python -m venv moje_okolje
```

pravzaprav rečete Python-u: *Ustvari mi izolirano okolje (z imenom 'moje_okolje', kjer lahko delam, ne da bi karkoli pokvaril drugje).*

To okolje ima svojo kopijo interpreterja in svojo mapo `site-packages`, kjer se nahajajo vse knjižnice — ločeno od sistemske namestitve Python-a.

Zakaj je to uporabno?

1. **Izogibanje konfliktom**: različni projekti pogosto potrebujejo različne verzije knjižnic. Virtualna okolja vam omogočajo, da jih ločite — en projekt lahko brez težav uporablja Django 3.2, drugi pa Django 4.0.
2. **Čistejši razvoj**: ne obremenjujete sistemskega Python-a s knjižnicami, ki jih potrebujete le za vaš projekt. Upravljanje je enostavnejše, odstranjevanje pa brez posledic.
3. **Ponovljivost**: z ukazom `` `pip freeze > requirements.txt` `` lahko zamrznete trenutno stanje okolja, nekdo drug pa ga lahko natančno poustvari z `` `pip install -r requirements.txt` ``.
4. **Varno eksperimentiranje**: želite preizkusiti novo knjižnico ali različico? To lahko storite v virtualnem okolju, brez tveganja za druge projekte.

😎 ***Eureka!*** *Vse te točke skupaj pa so natanko tisto, kar potrebujete vi v vašem programu. Vaš program namreč ne sme vplivati na druge Python in sistemske nastavitve na uporabnikovem računalniku!*

Priprava navideznega Python okolja

Priprava Python virtualnega okolja je skrajno preprosta in tukaj prilagam skripto, ki utvari virtualno okolje z imenom `moj-peskovnik`, ter vanj instalira knjižnice `PyPDF2`, `seaborn` in `matplotlib`:

```
python -m venv moj-peskovnik
call moj-peskovnik\Scripts\activate

call pip install PyPDF2
call pip install seaborn
call pip install matplotlib
```

4 Konkreten primer

V tem poglavju bom prikazal celoten postopek razvoja .NET programa, ki bo uporabil Python za pridobitev besedila iz poljubnih PDF datotek.

4.1. Priprava Python okolja

Že v fazi razvoja celotne aplikacije si lahko – kot .Net razvijalci – pripravimo ločeno Python okolje, kjer bomo zadeve testirali. In natanko to lahko kasneje uporabimo tudi pri instalaciji naše aplikacije na uporabnikov računalnik.

Zatorej je najbolje izbrati kar »embeded« Python, torej Python, ki ga bolj ali manj instaliramo preprosto tako, da razširimo ZIP datoteko v neko mapo.

Prilagam Windows CMD datoteko, ki:

1. prenese Python s spleta,
2. ga razširi v podmapo,
3. ga skonfigurira,
4. instalira virtualna okolja,
5. ustvari delovno virtualno okolje ter ga aktivira,
6. in na koncu še instalira PyPDF2 knjižnico, ki jo potrebujemo za branje PDF datotek:

```
@echo off
setlocal

set PYTHON_FOLDER=Python312
set PYTHON_ZIP=python-embed.zip
set          PYTHON_URL=https://github.com/astral-sh/python-build-standalone/releases/
download/20250626/cpython-3.12.11+20250626-x86_64-pc-windows-msvc-pgo-full.tar.zst
set VENV_NAME=virtual-env

echo Prenos Python paketa...
curl -L -o %PYTHON_ZIP% %PYTHON_URL%

echo Razsirjanje v mapo %PYTHON_FOLDER%...
tar -xf %PYTHON_ZIP%
ren python %PYTHON_FOLDER%
del %PYTHON_ZIP%

echo Instalacija virtualnega okolja...
%PYTHON_FOLDER%\install\python.exe -m pip install virtualenv

echo Ustvarjanje virtualnega okolja...
%PYTHON_FOLDER%\install\python.exe -m virtualenv  %VENV_NAME%

echo Aktivacija virtualnega okolja...
call %VENV_NAME%\Scripts\activate.bat

echo Namestitev PyPDF2...
pip install PyPDF2

echo Koncano! Python je v mapi '%PYTHON_FOLDER%', virtualno okolje pa v '%VENV_NAME%'.

Endlocal
```

😎 ***Dobro je vedeti!*** *Github.com/astral-sh je spletišče, kjer lahko dobite različne 'embeded' Python verzije! Kot rečeno 'embeded' pač pomeni da ni instalacije, temveč samo razpakirate ZIP datoteko in začnete uporabljati Python!*

4.2. Python koda za branje PDF datoteke

Python torej imamo instaliran in lahko napišemo preprosto skripto, ki bo prebrala vse podane PDF datoteke in iz njih izluščila besedilo:

```
import PyPDF2
import sys

def extract_pdf_texts(file_paths: list[str]) -> list[str]:
    contents = []
    for path in file_paths:
        try:
            with open(path, 'rb') as f:
                reader = PyPDF2.PdfReader(f)
                text = ""
                for page in reader.pages:
                    text += page.extract_text() or ""
                contents.append(text)
        except Exception as e:
            contents.append(f"[ERROR reading {path}: {e}]")
    return contents

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python extract.py <pdf1> <pdf2> ...")
    else:
        pdf_paths = sys.argv[1:]
        results = extract_pdf_texts(pdf_paths)
        for i, content in enumerate(results, 1):
            # Prikažemo samo prvih 500 znakov
            print(f"\n--- PDF {i} ---\n{content[:500]}...")
```

Za potrebe .Net aplikacije seveda ne potrebujemo dela kode, ki se izvaja, če skripto poženemo v ukazni vrstici. Za potrebe .NET integracije potrebujemo samo in izključno funkcijo `extract_pdf_texts`, ki jo bomo v C# kodi poklicali.

> 😎 ***Opomba:*** *Če pogledate podpis funkcije extract_pdf_texts, boste videli tudi podatkovne tipe za vhodne in izhodne podatke. Funkcija torej sprejme seznam stringov (list[str]) (imena PDF datotek) in tudi vrne seznam stringov (vsebina posameznih datotek).*
>
> *S stališča Python kode to ni potrebno, Python je zelo svobodomiseln, kar se tiče podatkovnih tipov. Je pa to zelo priporočljivo za lažje povezovanje te kode v .Net okolje.*
>
> *Na ta način bo lahko razvojno okolje .Net takoj ugotovilo pričakovan podatkovni tip in ponudilo IReadOnlyList<string> kot vhod in izhod.*
>
> *To je nekaj podobnega kot JavaScript in TypeScript. Tudi JavaScript ne potrebuje oznak podatkovnih tipov, pa smo sedaj začeli uporabljati TypeScript, ki pa jih ima in je nam lažje … računalniku je tako ali tako vseeno* 😳

Če torej izvedemo prvo skripto, ki nam vzpostavi Python okolje, lahko takoj preizkusimo to Python skripto s klicem: `python.exe extract.py test1.pdf test2pdf test3pdf`.

4.3. Priprava .NET okolja

Kot omenjeno že na začetku, bom v članku prikazal minimalno delujočo kodo, na GitHub-u (https://github.com/MPrtenjak/OTS2025-NET-And-Python) pa lahko najdete popolno kodo z več primeri ter tako .NET REST API aplikacijo kot tudi preprosto aplikacijo v ukazni vrstici.

Tukaj bomo zatorej razvili samo aplikacijo v ukazni vrstici - z minimalno kode!

V ukazni vrstici lahko pripravimo prazen projekt z imenom `ExtractPdf2Txt`:

```
dotnet new console -n ExtractPdf2Txt -f net9.0
cd ExtractPdf2Txt
```

Dodati je potrebno še CSnakes knjižnico:

```
dotnet add package csnakes.runtime
```

In že lahko pričnemo z delom.

4.4. Python del našega projekta

Python kodo smo že zapisali v datoteki `extract.py` in zato to datoteko tudi skopiramo v naš projekt `ExtractPdf2Txt`.

> 😎 ***Opomba:*** *Python kode seveda ni potrebno imeti v glavni mapi vašega .Net projekta. Lahko se nahaja kjerkoli, je pa smiselno, da je nekako skupaj s projektom.*
>
> *Osebno imam navado vso Python kodo dajati v podmapo Python znotraj mojega .Net projekta.*

Ker naša Python funkcija za delovanje potrebuje zunanjo knjižnico `PyPDF2`, moramo v projekt dodati tudi datoteko `requirements.txt`, kjer naštejemo vse potrebne zunanje Python knjižnice.

> 😎 ***Opomba:*** *Datoteka requirements.txt je zelo pomembna, saj z njo knjižnici CSnakes naročite, katere zunanje knjižnice naj naloži, da bo vaš program deloval.*
>
> *V kolikor vi vnaprej pripravite virtualno okolje, to sicer ni pomembno, škodi pa tudi ne* 😊*.*
>
> *Seveda se lahko datoteka imenuje tudi drugače, vendar pa morate potem to novo ime sporočiti knjižnici CSnakes, v kolikor se držite standarda, pa vam ni potrebno storiti nič več.*

.Net okolje se mora zavedati Python kode, da jo lahko pregleda

Velika pridobitev CSnakes knjižnice je v dejstvu, da se zelo dobro integrira v .Net razvojno okolje, kar pomeni, da pozna podatkovne tipe, pozna vse funkcije v vaših Python datotekah in podobno.

Zato morate prevajalniku namigniti, naj tovrstne datoteke pregleda. To storite tako, da Python datoteke in datoteko `requirements.txt` označite kot dodatne datoteke, ki jih bo pregledal C# prevajalnik. V vašo `.csproj` datoteko morate zatorej dodati sledeče:

```
<ItemGroup>
  <AdditionalFiles Include="requirements.txt">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </AdditionalFiles>
  <AdditionalFiles Include="extract.py">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </AdditionalFiles>
</ItemGroup>
```

4.5. Glavni program

In sedaj smo prišli do dela, kjer moramo zares napisati naš C# program, ki bo sprejel seznam PDF datotek in izpisal besedilno vsebino podanih datotek.

Program mora torej narediti sledeče:

1. Preveriti vhodne parametre in za vsakega izmed njih preveriti, ali ima končnico PDF in ali obstaja datoteka s podanim imenom.
2. V kolikor datotek ni, to javiti uporabniku in končati program.
3. Inicializirati knjižnico CSnakes.
4. Pridobiti kazalec na Python datoteko `extract.py`.
5. V prej omenjeni datoteki izvesti funkcijo `extract_pdf_texts`, ji podati seznam datotek in sprejeti vrnjeno besedilo.
6. Izpisati prejeto besedilo.

Program

Prilagam program, ki natanko izvede omenjenih 6 korakov.

> 😎 ***Opomba:*** *To je seveda minimalen, a povsem delujoč program. Želja je, da vam prikažem pomembne dele programa, zato je preverjanje parametrov, delitev programske kode na funkcije in podobno pač opuščeno!*

```
using CSnakes.Runtime;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System.Reflection;

internal class Program
{
  private static void Main(string[] args)
  {
    var pdfFiles = args
        .Where(arg => File.Exists(arg) &&
                      Path.GetExtension(arg).Equals(".pdf",
StringComparison.OrdinalIgnoreCase));

    if (!pdfFiles.Any())
    {
      Console.WriteLine("No PDFs to convert.");
      return;
    }

    var exeFolder = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)!;
    var baseFolder = Path.GetFullPath(Path.Combine(exeFolder, @"..\..\..\"));

    var host = Host.CreateDefaultBuilder(args)
      .ConfigureServices(s => s
        .WithPython()
        .FromFolder(Path.Combine(baseFolder, @"..\Python312\install"), "3.12")
        .WithVirtualEnvironment(Path.Combine(baseFolder, @"..\virtual-env"))
        .WithPipInstaller())
      .Build();

    var env = host.Services.GetRequiredService<IPythonEnvironment>();
    var module = env.Extract();
    var texts = module.ExtractPdfTexts(pdfFiles.ToArray());

    foreach (var text in texts)
      Console.WriteLine($"------------\n{text}\n------------\n");
  }
}
```

Pomembnejši deli programa

Začetek programa je preprost in gre samo za preverjanje vhodnih podatkov.

Nato določimo mapo, kjer se nahaja naš izvršljiv program (`exeFolder`) in na podlagi tega določimo, kje se nahaja naš projekt (`baseFolder`).

> 😎 ***Opomba:*** *Kot že omenjeno, je to minimalni program katerega namen je prikazati delovanje knjižnice in zato predvidevam (če sledite navodilom v tem prispevku), da program preizkušate v razvojnem okolju in da slednje program prevede v podmape bin/debug/net9.0.*
>
> *S tem namenom se moram za iskanje našega projekta (baseFolder) vrniti tri nivoje višje.*

Inicializacija knjižnice CSnakes

Knjižnico inicializiramo tako kot je v .Net dokaj standardno, namreč da najprej definiramo gostitelja (`host`) in ga nato zgradimo (`Build`).

Knjižnici CSnakes povemo:

1. da bomo uporabili Python (`WithPython`),
2. da se Python nahaja v podmapi `Python312` in da ima verzijo `3.12` (`FromFolder`),
3. da želimo uporabiti virtualno okolje z imenom `virtual-env` (`WithVirtualEnvironment`),
4. ter da želimo naj knjižnica uporabi `pip`, v kolikor je potrebno v virtualno okolje še instalirati kakšno knjižnico (`WithPipInstaller`).

Izvajanje Python kode

Z izgradnjo našega gostitelja lahko v kodi kadarkoli dosežemo objekt, ki je namenjen interakciji s Python kodo (`IPythonEnvironment`).

S tem objektom pridobimo možnost dostopa do vseh Python datotek, v našem primeru pač (`env.Extract`)

> 😎 ***Opomba:*** *Python datoteka, s katero delamo, se imenuje extract.py, zato se tudi naš modul znotraj .Net imenuje Extract. V kolikor bi v programu imeli še druge Python datoteke, recimo statistika.py ali numericne_metode.py, bi do slednjih v .Net dostopali kot env.Statistika() oz. env.NumericneMetode().*

Ko imamo dostop do modula, pa lahko v njem izvedemo katero koli funkcijo, ki se v modulu nahaja. V našem modulu se nahaja samo funkcija `extract_pdf_texts`, zato je tudi edina metoda, ki nam je v .Net dosegljiva pač `ExtractPdfTexts`.

> 😎 ***Opomba:*** *Kot ste lahko že opazili, knjižnica CSnakes inteligentno prevaja Python standarde »besede z malimi začetnicami in ločene s podčrtaji« v .net standarde »besede z velikimi začetnicami«.*

In to je vse. Klic Python funkcije nam vrne besedila vseh podanih PDF datotek in vse, kar je še potrebno storiti, je to izpisati uporabniku.

S tem smo preprosto in učinkovito povezali naše .Net okolje, naš C# program s Python okoljem.

5 Demonstracijska aplikacija dosegljiva na GitHub-u

V tem prispevku sem prikazal minimalni delujoč primer, na GitHub-u pa si lahko naložite demonstracijsko aplikacijo, ki prikaže razvoj sledečih modulov:

1. `HelloWorld`: klasičen programski primer 'Pozdravljen svet! Pozdravlja te <ime>.', kjer bo spremenljivko '`ime`' podal .Net del, sporočilo generiral Python in rezultat spet prejel .Net.
2. `PdfExtractor`: primer, kjer bomo Python kodi podali PDF dokument, Python bo v dokumentu prebral vso besedilo in nam ga vrnil kot niz znakov.
3. `ShowGraph`: primer, kjer bomo s pomočjo Python knjižnic `seaborn` in `matplotlib` ustvarili graf, ter ga kot niz `byte`-ov vrnili .Net kodi, ki ga bo prikazala.
4. `OpenAI`: primer, kjer se bo Python preko knjižnice `openai` povezal z `OpenAI` ter generiral kratko otroško zgodbico o zmajčku.
5. `HTMLExtractor`: v tem primeru pa bomo z uporabo Python-ovih knjižnic `selenium`, `bs4` in programom `webdriver`, pridobili žive podatke o cenah delnic na Ljubljanski borzi.

Literatura

[1] CSnakes, https://tonybaloney.github.io/CSnakes/, pridobljeno 1. 8. 2025

[2] Install .NET on Windows - .NET | Microsoft Learn, https://learn.microsoft.com/en-us/dotnet/core/install/windows, pridobljeno 1. 8. 2025

[3] Welcome to Python.org, https://www.python.org/, pridobljeno 1. 8. 2025

[4] Programska koda celotne aplikacije, https://github.com/MPrtenjak/OTS2025-NET-And-Python, pridobljeno 1. 8. 2025

Uporaba javanske knjižnice za velike jezikovne modele

Andrej Krajnc, Vojko Ambrožič, Bojan Štok

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

Veliki jezikovni modeli omogočajo, da običajnim aplikacijam dodamo funkcionalnosti umetne inteligence. Pametni asistenti z uporabo orodij omogočajo, da veliki jezikovni model preko povratnega klica pokliče poslovno metodo v aplikaciji. V javanskem okolju je najbolj priljubljena knjižnica Langchain4j. S pomočjo Langchain4j lahko uporabimo različne implementacije velikih jezikovnih modelov (OpenAI, Ollama, JLama ...), ne da bi nam bilo treba spreminjati programsko kodo. V prispevku bomo podrobneje predstavili knjižnico Langchain4j in Quarkusovo razširitev Quarkus Langchain4j, ki uporabo knjižnice Langchain4j še bolj poenostavi, saj omogoča deklarativno uporabo knjižnice večinoma le z uporabo anotacij. Tako lahko hitro razvijamo nove funkcionalnosti. Podobni pristop uporablja Spring AI v aplikacijah SpringBoot. Pametnega asistenta COBISS, ki smo ga predstavili lani, smo nadgradili z uporabo knjižnice Langchain4j. Tako je precej ponavljajoče se kode odpadlo. Na kratko bomo predstavili tudi ogrodje Quarkus, ki omogoča zelo hiter zagon aplikacij in je konkurenčno ogrodjema Spring in Jakarta EE.

Ključne besede:
COBISS,
COBISS Lib,
Langchain4j,
Quarkus,
veliki jezikovni modeli.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.16

1 Uvod

V zadnjem času se v aplikacije vedno bolj vgrajujejo funkcionalnosti, ki s pomočjo uporabe velikih jezikovnih modelov običajnim aplikacijam dodajo zmogljivosti umetne inteligence. Učenje velikih jezikovnih modelov poteka na osnovi dostopnih podatkov na spletu, zato so dobri v generiranju odgovorov na splošna vprašanja, vendar pa modeli pogosto nimajo domensko specifičnih znanj in zato včasih niso najbolj priročni v primerih, kjer so domensko specifična znanja zelo pomembna. Rešitev za takšne situacije je lahko implementacija lastnega pametnega asistenta.

Na konferenci OTS 2024 smo predstavili izgradnjo pametnega asistenta COBISS, ki smo ga implementirali v okviru aplikacije COBISS Lib za knjižničarje [1]. COBISS Lib podpira postopke nabave monografskih in serijskih publikacij ter elektronskih virov, obdelavo podatkov o zalogi, izvajanje postopkov v izposoji in medknjižnični izposoji ipd.

Ker sta aplikacija in dokumentacija obsežni, smo se odločili razviti asistenta, ki bi knjižničarjem pomagal pri vsakodnevnem delu. Uporabili smo pristop vključevanja agentov in zunanjih virov znanja RAG (ang. Retrieval-Augmented Generation), za jezikovni model pa smo uporabili Chat GPT-4o, čeprav bi lahko uporabili tudi katerikoli drugi model.

Razvili smo programe v Javi 21, ki so markdown-datoteke priročnikov razbile po odstavkih. Za vsak odstavek smo od ChatGPT zahtevali, da pripravi povzetek. Nato smo za vsak tak povzetek pripravili vektor (ang. embeddings) ter ga shranili v vektorsko bazo, ki smo jo kreirali z uporabo javanske knjižnice JVector. Ko uporabnik postavi vprašanje, od ChatGPT zahtevamo, da generira vektor in zazna jezik uporabnika. V vektorski bazi poiščemo pet (nastavitev) najbližjih vektorjev/besedil (ang. cosine similarity) glede na iskalni vektor. Nato od ChatGPT zahtevamo, da za uporabnikovo zahtevo na osnovi najdenih besedil generira odgovor v zaznanem jeziku. Razvili smo javanski odjemalec do ChatGPT z uporabo odprtokodne knjižnice OpenAI-Java.

Celotno pot smo želeli prehoditi sami in pri tem dobiti lastne izkušnje. V prispevku za konferenco OTS 2024 smo najavili, da bomo v prihodnje uporabili najnovejše pristope, s pomočjo katerih lahko veliko hitreje in enostavneje vgradimo funkcionalnosti umetne inteligence. Odločili smo se za uporabo ogrodja Langchain4j, ki je eno od najpopularnejših tovrstnih ogrodij za programski jezik Java.

2 Ogrodje Langchain4j

Langchain4j [2] je odprtokodna javanska knjižnica, ki omogoča enostavno integracijo velikih jezikovnih modelov LLM (ang. Large Language Model) v aplikacije, napisane v Javi. Ustvarjena je bila leta 2023, ko se je pojavila potreba po javanski različici priljubljene knjižnice LangChain, ki je bila napisana v Pythonu. Knjižnica je prilagojena javanskemu ekosistemu in uporablja poznane koncepte, kot so razredi, vmesniki in anotacije. Zasnovana je za razvijalce, ki želijo vgraditi umetno inteligenco v svoje javanske aplikacije. Zadnja verzija Langchain4j 1.2.0 je bila izdana julija 2025.

Knjižnica Langchain4j omogoča, da lahko uporabimo različne implementacije velikih jezikovnih modelov (Ollama, OpenAI, Azure OpenAI, Google Vertex AI Gemini, Jlama, MistralAI …) in različne implementacije vgrajenih shramb (ang. embeddings stores), npr. Cassandra, PGVector (PostgreSQL), Elasticsearch, Redis, Infinispan, MongoDB Atlas, Neo4j itd. Na relativno enostaven način lahko v aplikaciji testiramo različne modele ali shrambe, saj nam ni treba spreminjati aplikacije, ampak spremenimo le konfiguracijo.

Knjižnica Langchain4j podpira tako nizkonivojska orodja, kot so oblikovanje iztočnic, pogovor s pomnilnikom, razčlenjevanje izhodov, kot tudi visokonivojska orodja, kot so storitve AI in RAG.

Langchain4j sam poskrbi za komunikacijo z jezikovnimi modeli in upravljanje konteksta pogovora. Prav tako omogoča vodenje pogovornega spomina (ang. conversation memory), s čimer si lahko aplikacije zapomnijo, kaj je bilo povedano v prejšnjih sporočilih, kar je ključno za naravno interakcijo.

Langchain4j se dobro integrira v sodobna javanska ogrodja, kot so Quarkus, Spring Boot, Helidon in Micronaut. Pomemben mejnik se je zgodil, ko je Red Hat vključil Langchain4j kot uradno razširitev v Quarkus, kar je prineslo še večjo priljubljenost Langchain4j v javanskem ekosistemu.

2.1. Uporaba Langchain4j v javanskih strežniških aplikacijah

Knjižnico Langchain4j lahko uporabljamo v javanskih aplikacijah na različne načine.

V klasičnih javanskih aplikacijah, kjer se uporablja standardna verzija Jave (ang. Java Standard Edition), lahko Langchain4j uporabljamo na preprost način kot navadno knjižnico razredov. Takšen pristop je primeren za enostavne primere, skripte, CLI-orodja, testiranje ipd.

Primer uporabe Langchain4j kot knjižnice razredov:

```
ChatModel chatModel = OpenAiChatModel.builder().apiKey(OPENAI_API_KEY).build();
String answer = chatModel.chat("Kaj je Langchain4j?");
```

V javanskih strežniških aplikacijah (ang. Enterprise Java) lahko uporabljamo Langchain4j na bolj napreden način, saj lahko uporabimo različne stvari, kot so anotacije, vrivanje odvisnosti (ang. dependency injection) in druge tehnologije, ki so značilne za javanske strežniške aplikacije. Takšen pristop je primeren za izgradnjo vmesnikov REST, mikrostoritev in raznih spletnih aplikacij.

Langchain4j lahko uporabljamo v ogrodjih, kot so Spring, Quarkus, Micronaut, Helidon ipd. Kar nekaj ogrodij je integriralo Langchain4j na način, ki še olajša uporabo umetne inteligence v javanskih strežniških aplikacijah. Tako obstajajo integracije, kot so Langchain4j Spring, Quarkus Langchain4j, Langchain4j Microprofile (SmallRye LLM) itd. Te integracije omogočajo, da lahko umetno inteligenco v aplikacijah uporabimo z zgolj nekaj vrsticami kode.

Primer kode v Quarkus Langchain4j:

```
@ApplicationScoped
public class LlmService {
   @Inject
   OpenAiChatModel chatModel;

   public String answer(String question) {
      return chatModel.chat(question);
   }
}
```

2.2. Quarkus Langchain4j

Čeprav večina naših aplikacij temelji na uporabi Java EE oz. Jakarta EE, smo se pri implementaciji pametnega asistenta COBISS odločili za uporabo Quarkus Langchain4j [3].

Quarkus je sodobno javansko ogrodje, zasnovano za hitro in učinkovito izvajanje v oblačnih okoljih, npr. v Kubernetes, OpenShift, Docker Swarm in podobno [4]. Prva verzija 1.0 je bila narejena znotraj podjetja Red Hat leta 2019, zadnja LTS verzija 3.20.2 je iz julija 2025. Glavni cilj ogrodja Quarkus je optimizirati zagon in delovanje aplikacij, zato ga odlikujeta izjemno hiter zagon in majhna poraba pomnilnika. Ogrodje ima zelo dobro podporo za navidezni stroj **GraalVM**, s pomočjo katerega lahko ustvarjamo izvorne slike (ang. native image). Izvorne slike so majhne binarne datoteke, ki se zaženejo zelo hitro in so med izvajanjem neodvisne od platforme Java.

Quarkus je celovito strežniško ogrodje (ang. full stack server framework), ki ne vključuje uporabniškega vmesnika (HTML/JS/CSS), ampak ponuja vse ključne tehnologije za razvoj celotne strežniške aplikacije, tj. od podatkovnega sloja do programskih vmesnikov, varnosti in povezav z zunanjimi sistemi. Quarkus temelji na veliko strežniških standardih, pri razvoju katerih je že leta sodelovalo podjetje Red Hat. Tako ogrodje podpira vrivanje odvisnosti preko standarda CDI (ang. Contexts & Dependency Injection), izdelavo vmesnikov REST preko standarda JAX-RS, dostop do podatkovnih baz preko standarda JPA (Java Persistence API), omogoča uporabo transakcijskega programskega vmesnika JTA (Java Transaction API), itd. Quarkus je združljiv z javanskimi standardi, a hkrati prinaša inovacije, ki jih v drugih ogrodjih pogosto ni.

Quarkus Langchain4j je razširitev, ki prinaša zmogljivosti umetne inteligence v okolje Quarkus na način, skladen z njegovimi načeli: hiter zagon, majhna poraba pomnilnika in oblačni razvoj. Razširitev omogoča, da razvijalci uporabljajo jezikovne modele LLM (ang. Large Language Model) znotraj aplikacij Quarkus z minimalno konfiguracijo, izkoriščajo pa lahko vse funkcionalnosti Quarkusa za razvoj in integracijo z drugimi storitvami.

Ker je Langchain4j uradni del Quarkusa, je z njo mogoče brez težav povezati jezikovne modele LLM z obstoječo infrastrukturo (npr. podatkovne baze, sporočilni sistemi, varnostni mehanizmi). Deluje brez težav z izvornimi slikami (GraalVM), kar pomeni, da lahko zgradimo mikrostoritve, temelječe na umetni inteligenci, ki so lahke, hitro odzivne in pripravljene za izvajanje v oblačnih okoljih, kot Kubernetes, OpenShift, Docker Swarm itd. Poleg tega omogoča enostavno vključevanje agentov in zunanjih virov znanja (RAG) v tipičen Quarkusov razvojni cikel.

Primer implementacije enostavnega klepeta:

```
@ApplicationScoped
@RegisterAiService()
public interface DemoChat {

  String chat(@MemoryId int id, @UserMessage String message);

  @SystemMessage("you are java programmer")
  String chatJava(@MemoryId int id, @UserMessage String message);
}
```

Z anotacijo @SystemMessage lahko damo dodatna navodila sistemu, z anotacijo @MemoryId določamo ID klepeta, z anotacijo @UserMessage posredujemo sporočilo uporabnika.

Če uporabimo multimodalen model jezik-slika (ang. multimodal language-image), lahko razpoznavamo vsebino slike ali slike generiramo. Eden izmed odprtokodnih modelov je LLaVa [5].

Primer kode za razpoznavanje slike:

```
@ApplicationScoped
@RegisterAiService()
public interface DemoImageService {

  @UserMessage(value = "What do you see")
  String describeImage(@ImageUrl String imageUrl);

  @UserMessage(value = "Extract only text from image")
  String extractText(@ImageUrl String imageUrl);
}
```

Nekateri modeli omogočajo tudi generiranje slike. Primer kode:

```
@ApplicationScoped
@RegisterAiService()
```

```
public interface DemoImageService {
  Image generateImage(String message);
}
```

3 Implementacija pametnega asistenta COBISS z uporabo Langchain4j

Prvo verzijo pametnega asistenta COBISS smo razvili z uporabo ogrodja **Jakarta EE** na aplikacijskem strežniku **WildFly**. Asistenta smo nadgradili tako, da smo namesto ogrodja Jakarta EE uporabili Quarkus Langchain4j. Komponente za dostop do velikih jezikovnih modelov (OpenAI-Java) smo zamenjali s knjižnico **Langchain4j**. Prav tako smo vektorsko bazo **jVector** nadomestili z bazo **PGVector** (na osnovi PostgreSQL).

Aplikacijo smo testirali z uporabo različnih modelov **OpenAI** in z uporabo odprtokodnih modelov, kot so »llama3.3«, »deepseek«, ki so bili nameščeni na našem lokalnem strežniku in dostopni preko vmesnika **Ollama**.

Največja sprememba je bila na področju priprave in obdelave dokumentov, ki se nato shranijo v vektorsko bazo in uporabljajo pri iskanju konteksta v okviru strategije **RAG** (ang. Retrieval-Augmented Generation**)**.

3.1. Inicializacija modelov in podatkovnih shramb

Običajno se ob dvigu aplikacije ali prvi uporabi pametnega asistenta na osnovi konfiguracije inicializirajo različni modeli in podatkovne shrambe.

V primeru pametnega asistenta COBISS uporabljamo model za pogovore (ang. chat model) ter model za vdelave (ang. embedding model). Primer kode za inicalizacijo modela za pogovore:

```
ChatModel chatModel = OpenAiChatModel.builder().
  .apiKey(OPENAI_API_KEY)
  .modelName(“gpt-4o”)
  .temperature(0)
  .maxTokens(3000)
  .build();


 ChatModel chatModel = OllamaLanguageModel.builder().
  .baseUrl(OLLAMA_URL)
  .modelName("llama3.3")
  .temperature(0)
  .maxTokens(3000)
  .build();
```

Primer kode za inicalizacijo modela za vdelave:

```
EmbeddingModel embeddingModel =  OpenAiEmbeddingModel.builder()
  .apiKey(OPENAI_API_KEY)
  .modelName("text-embedding-3-small")
  .build();


EmbeddingModel embeddingModel =  OllamaEmbeddingModel.builder()
  .baseUrl(OLLAMA_URL)
  .modelName("llama3.3")
  .build();
```

Primer kode za inicializacijo podatkovne shrambe vdelav:

```
EmbeddingStore<TextSegment> embeddingStore = PgVectorEmbeddingStore.builder()
  .host(PGVECTOR_HOST)
  .port(PGVECTOR_PORT)
  .createTable(recreateTable)
  .dropTableFirst(recreateTable)
  .dimension(MODEL_DIMENSION)
  .table(TABLE_NAME)
  .user(USERNAME)
  .password(PASWORD)
  .database(DATABASE)
  .build();
```

3.2. Priprava poslovnih podatkov

Pri uporabi strategije RAG je ključnega pomena ustrezna priprava poslovnih podatkov za hrambo v vektorski bazi. Knjižnica Langchain4j daje podporo za branje dokumentov iz različnih virov, kot so lokalni diski ali spletni naslovi. Podprti formati vključujejo HTML, PDF, TXT in druge. V našem primeru je bila vsebina priročnikov v **HTML**-obliki.

Prvi korak: Razčlenjevanje dokumentov

Dokumente je treba najprej razčleniti, torej pretvoriti njihovo vsebino v besedilno obliko, ki jo lahko obdelujemo. Langchain4j omogoča uporabo različnih razčlenjevalnikov (ang. parsers) za ta namen.

V prvem koraku poiščemo relevantne HTML-dokumente:

```
try (Stream<Path> stream = Files.walk(Paths.get(documentsPath))) {
  stream.filter(Files::isRegularFile).filter(path -> {
    String fileName = path.getFileName().toString().toLowerCase();
    return fileName.endsWith(".html");
  }).forEach(path -> parseDocument(path);
} catch (IOException e) {
  e.printStackTrace();
}
```

Nato preberemo vsak najdeni dokument in mu po potrebi dodamo več metapodatkov:

```
DocumentParser parser = new ApacheTikaDocumentParser();
Document document = FileSystemDocumentLoader.loadDocument(path, parser);
document.metadata().put(Document.URL, documentUrl);
document.metadata().put("document_type", »LOAN« );
```

Drugi korak: Preoblikovanje vsebine (opcijsko)

Po razčlenjevanju lahko dokumente dodatno preoblikujemo: odstranimo nepotrebne dele, izboljšamo formatiranje ali dodamo metapodatke. Namen tega koraka je izboljšati kakovost informacij, ki bodo kasneje predstavljene jezikovnemu modelu.

```
DocumentTransformer transformer = new DocumentTransformer(
  public Document transform(Document document) {
    String text = document.text();
    text = format(text);
    document = new Document(text.trim(), document.metadata());
```

```
    return document;
  }
);
Document transformedDocument = transformer.transform(document);
```

Tretji korak: Razdelitev vsebine na segmente

Eden ključnih korakov je razdelitev dokumenta na manjše, bolj obvladljive enote (ang. chunks). Veliki jezikovni modeli imajo omejeno velikost vhodnega okna (ang. context window), zato mora biti vsak segment dovolj majhen, da ga model lahko obdela v enem pozivu.

Langchain4j ponuja razrede, ki omogočajo razdelitev vsebine glede na število besed, stavkov ali znakov, in po potrebi omogočajo prekrivanje med segmenti za boljši kontekst.

```
DocumentSplitter splitter = DocumentSplitters.recursive(5000, 100);
List<TextSegment> segments = splitter.split(document);
for (TextSegment segment : segments) {
  saveEmbedding(segment, documentType);
}
```

Četrti korak: Izračun vektorjev in shramba v vektorsko bazo

Za vse kreirane segmente izračunamo numerično reprezentacijo vsebine ter izračunane vektorje shranimo v vektorsko bazo:

```
Embedding embedding = embeddingModel.embed(segment).content();
embeddingStore.add(embedding, segment);
```

Shranjena vrstica vsebuje tekst, metapodatke in izračun večdimenzionalnega vektorja segmenta.

3.3. Prikaz poslovnih podatkov

RAG (ang. Retrieval-Augmented Generation) je metoda, pri kateri veliki jezikovni model (LLM) pri generiranju odgovorov uporablja zunanje podatkovne vire. Namesto da bi se zanašal le na lastno znanje, sistem najprej poišče relevantne informacije v vnaprej pripravljeni vektorski bazi, nato pa te vsebine vključi v poziv (ang. prompt) v modelu. Tako nastane odgovor, ki je bolj dejstveno utemeljen, aktualen in prilagojen konkretni podatkovni zbirki.

RAG združuje prednosti iskanja po vsebini in naravnega jezika za natančnejše, zanesljivejše odgovore.

Iztočnica uporabnika

Uporabnik zastavi vprašanje ali poda zahtevo v obliki besedilnega vnosa (t.i. prompt).

Iskanje po vektorski bazi

Sistem pretvori vprašanje v vektorsko predstavitev ter poišče najbolj relevantne dokumente ali odlomke iz vektorske baze podatkov (npr. PGVector), ki so semantično podobni vprašanju.

Primer kode za iskanje:

```
Embedding embeddedQuestion = embeddingModel.embed(question).content();
EmbeddingStore<TextSegment> embStore = getEmbeddingStore();
EmbeddingSearchRequest searchRequest = EmbeddingSearchRequest.builder()
  .queryEmbedding(embeddedQuestion)
  .maxResults(5)
  .minScore(0.8)
```

```
  .build();
EmbeddingSearchResult<TextSegment> searchResult = embStore.search(searchRequest);
```

Priprava odgovora

Najdeni odlomki se dodajo kot kontekst jezikovnemu modelu, ki na tej osnovi generira relevanten odgovor, podprt z dejstvi.

```
ChatMemory chatMemory = MessageWindowChatMemory.withMaxMessages(20);
SystemMessage systemMsg = new SystemMessage("""
Use these guidelance to help the user:
    - You are a librarian
    - When listing, use list notation
    - Answer the user prompt with help of supplied text
    """);
chatMemory.add(systemMsg);
searchResult.matches().forEach(match -> {
     chatMemory.add(UserMessage.from(match.embedded().text()));
});
UserMessage userQuestion = UserMessage.from(question);
chatMemory.add(userQuestion);
AiMessage aiAnswer = chatModel.chat(chatMemory.messages()).aiMessage();
chatMemory.add(aiAnswer);
```

Če primerjamo odgovore, ki nam jih dajeta OpenAI in Ollama, lahko rečemo, da OpenAI daje boljše odgovore in to v krajšem času, vendar pa je uporaba orodja Ollama na lastnem strežniku lahko cenejša rešitev, ki zagotavlja tudi zasebnost, saj vsi podatki ostajajo v lokalnem okolju.

Priprava odgovora v obliki toka

Včasih ne želimo čakati na dokončno oblikovan odgovor, temveč želimo odgovor dobivati sproti, kot se pripravlja. Tako prihaja odgovor v koščkih v obliki toka, kot smo ga vajeni iz orodij, kot je na primer ChatGPT. Na ta način lahko zagotovimo hitrejši odziv na uporabniškem vmesniku, kar predstavlja boljšo uporabniško izkušnjo.

Če želimo dobiti odgovore na takšen način, moramo uporabljati drug model za pogovore, in sicer model v obliki toka (ang. streaming model).

Primer kode za inicializacijo modela za pogovore v obliki toka:

```
StreamingChatModel streamingChatModel= OpenAiStreamingChatModel.builder().
  .apiKey(OPENAI_API_KEY)
  .modelName(“gpt-4o”)
  .temperature(0)
  .maxTokens(3000)
  .build();


 StreamingChatModel streamingChatModel = OllamaStreamingChatModel.builder().
  .baseUrl(OLLAMA_URL)
  .modelName("llama3.3")
  .temperature(0)
  .maxTokens(3000)
  .build();
```

V primeru uporabe tokov je tudi koda za pripravo odgovora drugačna:

```
@POST
@Path("/streamHelp")
@Consumes({"application/json"})
public Response streamHelp(AiRequest data) {
  StreamingOutput stream = output -> {
    PrintWriter writer = new PrintWriter(new OutputStreamWriter(output, StandardCharsets.UTF_8));
    aiService.getStreamHelp(data, writer);
  };
  return Response.ok(stream).build();
}

CountDownLatch latch = new CountDownLatch(1);
streamingChatModel.chat(chatMemory.messages(), new StreamingChatResponseHandler() {
  @Override
  public void onPartialResponse(String partialResponse) {
    writer.write(partialResponse);
    writer.flush();
  }
  @Override
  public void onCompleteResponse(ChatResponse completeResponse) {
    writer.println("\n$DOCUMENTS\n");
    for (AiDocument document : documents) {
      writer.write("Document: " + document.getDocumentUrl() + "\n");
    }
    writer.flush();
    latch.countDown();
  }
  @Override
  public void onError(Throwable error) {
    writer.write("Error: " + error.getMessage());
    writer.flush();
    latch.countDown();
  }
});
latch.await();
```

4 Orodja Langchain4j

V Langchain4j je vgrajen koncept orodij (ang. tools) in klicanja funkcij (ang. function calling).

Orodja LangChain4j [6] so zunanje funkcije ali storitve, ki jih jezikovni model lahko po potrebi uporabi kot pomoč. To modelu omogoča, da ne odgovarja na vprašanja zgolj na klasičen način, temveč odgovore obogati, tako da pokliče zunanje funkcije, kot so na primer matematični izračuni, iskanje podatkov v bazi, pridobivanje podatkov s spleta ali uporaba programskih vmesnikov (API).

Novo orodje (ang. tool) lahko ustvarimo na različne načine. Najbolj enostaven način je, da implementiramo običajno javansko metodo in jo označimo z anotacijo @Tool. Ko model sklepa, da uporabnik želi nekaj, kar orodje zna narediti, ga samodejno pokliče.

Primer kode za definiranje novega orodja za seštevanje:

```
public class CobissTool {
  @Tool("Avtor knjige z naslovom")
  int getBookAuthor(String title) {
    List<Book> books = cobissSearch.searchTitle(title);
    if (books.size() > 0) return books.get(0).getAuthor();
    return null;
  }
}
```

Novo orodje registriramo na enostaven način:

```
@RegisterAiService(tools=CobissTool.class)
```

Ko uporabnik pošlje sporočilo, jezikovni model oceni, ali je katero od orodij primerno za pomoč pri generiranju odgovora in v primeru, da uporabnik želi pridobiti avtorja neke knjige, uporabi orodje za pridobivanje avtorja in na koncu odgovori.

```
String answer = assistant.chat("Kdo je avtor knjige z naslovom Na klancu?");
```

Primer odgovora na koncu je: "Avtor knjige z naslovom Na klancu je Ivan Cankar.".

Orodij Langchain4j zaenkrat še nismo uporabili v pametnem asistentu COBISS, jih pa nameravamo v prihodnje.

5 Zaključek

V prispevku smo prikazali implementacijo pametnega asistenta COBISS z uporabo javanske knjižnice Langchain4j. Langchain4j je v tem trenutku zagotovo eno najbolj vročih orodij za izgradnjo javanskih aplikacij, ki uporabljajo zmožnosti jezikovnih modelov.

Glede na prvotni pristop, kjer smo vse preizkusili ročno, nam je Langchain4j zelo olajšal izgradnjo pametnega asistenta COBISS, saj ponuja številne uporabne funkcionalnosti. Orodje podpira najrazličnejše implementacije jezikovnih modelov, preklopi med njimi pa so enostavni, saj običajno za uporabo drugega modela ni treba spreminjati programske kode, temveč le konfiguracijo.

Zaenkrat smo za implementacijo pametnega asistenta COBISS uporabili osnovne funkcionalnosti Langchain4j, v prihodnje pa nameravamo uporabiti tudi orodja Langchain4j na način, da bodo jezikovni modeli klicali naše funkcije.

Preizkusiti nameravamo tudi uporabo vnaprej pripravljenih okolij Docker za lokalno poganjanje jezikovnih modelov (Docker Model Runner) [7]. Ker se modeli tako poganjajo lokalno, ni potrebe po klicanju zunanjih programskih vmesnikov, s čimer zmanjšamo stroške in ohranjamo zasebnost pri uporabi jezikovnih modelov.

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] LangChain4j, https://github.com/langchain4j/langchain4j/, obiskano 29. 7. 2025

[3] Quarkus LangChain4j, https://docs.quarkiverse.io/quarkus-langchain4j/dev/index.html/, obiskano 29. 7. 2025

[4] Quarkus, https://quarkus.io/, obiskano 29. 7. 2025

[5] LLaVa, https://llava-vl.github.io/, obiskano 29. 7. 2025

[6] LangChain4j Tools (Function Calling), https://docs.langchain4j.dev/tutorials/tools, obiskano 29. 7. 2025

[7] Docker Model Runner, https://docs.docker.com/ai/model-runner/, obiskano 29. 7. 2025

Protokol MCP - Kako povezati obstoječe aplikacije in agente

Jani Šumak

Inova IT d.o.o., Maribor, Slovenija
jani.sumak@inova.si

Novembra 2024 je družba *Anthropic*, najbolje poznana po razvoju skupine velikih jezikovnih modelov (angl. Large Language Models, v nadaljevanju LLM) po imenu Claude, objavila odprti protokol za modelni kontekst (angl. Model Context Protocol, v nadaljevanju protokol MCP). K sodelovanju so kmalu pristopili tudi drugi razvijalci in ponudniki rešitev, ki se poslužujejo velikih jezikovnih modelov, med drugim Google in OpenAI. Temeljni namen protokola je standardizacija interakcije med LLM aplikacijami (angl. LLM application) ter zunanjimi podatkovnimi viri in orodji. Protokol se osredotoča na vprašanje, kako velikim jezikovnim modelom in LLM aplikacijam podati kakovosten kontekst s pomočjo katerega lahko generirajo boljše rezultate ali izvajanje aplikacije (agenti). V uvodu bomo povzeli delovanje velikih jezikovnih modelov in izpostavili nekatere pomanjkljivosti. Na to bomo pregledali rešitve, ki naslavljajo enake ali podobne težave kot protokol MCP. Po uvodnem delu se bomo posvetili protokolu MCP. Prispevek bomo zaključili s prikazom konkretni implementaciji protokola MCP in primerov uporabe.

Ključne besede:
veliki jezikovni modeli,
pozivanje,
LLM Aplikacije,
protokol mcp,
agenti.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.17

1 Uvod

Z javno dostopnostjo prve iteracije aplikacije ChatGPT[22] in popularizacijo velikih jezikovnih modelov – tukaj gre zasluga tudi odprtokodnim modelom, kot so Llama[1] - so se pričela prizadevanja omejevanja stohastičnosti[2] aplikacij, ki uporabljajo velike jezikovne modele. Onstran pogovornih aplikacij (angl. chat-based applications) je (bila) integracija velikih jezikovnih modelov z obstoječimi ali novimi aplikacijami zahteven postopek.

Druga omejitev je meja znanja (angl. knowledge cutoff). Po končanem razvoju je model omejen na podatke, ki so bili na voljo tekom učenja in prilagajanja modela. V kolikor želimo, da ima model "dostop" do novejših podatkov ali podatkov, ki jih ni bilo v učnem naboru, na primer posodobljeno dokumentacijo ogrodja, ki ga uporabljamo, je potrebno omogočite vnos teh podatkov v poziv ali dovoliti, da model, natančneje aplikacija, ki uporablja model, te podatke pridobi in jih vključi v poziv.

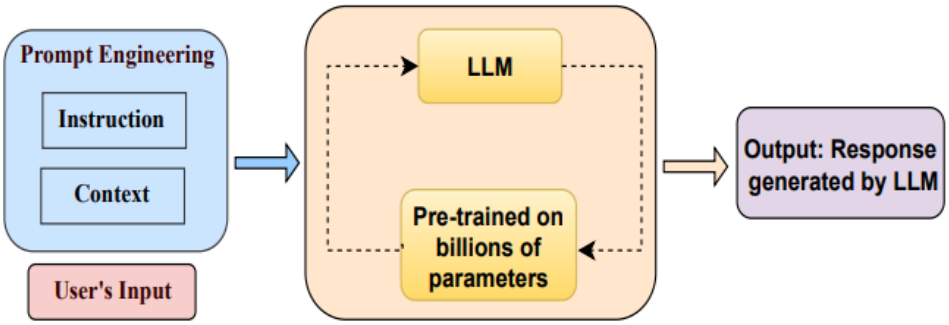
Rešitve, ki naslavljajo te omejitve, lahko razdelimo na tri pristope: nastavljanje dekodiranja[23], prilagajanje modelov in inženiring pozivov (ang. prompt engineering). Temeljna razlika med temi pristopi je, da se prva dva osredotočita na delovanje modela, medtem ko se slednji osredotoča vnos, natančneje na omejevanje procesa iskanja v vektorskem prostoru. Kot bomo pokazali, protokol MCP sodi ali dopolnjuje slednje rešitve.

2 Razvoj inženiranja pozivov

2.1. Pozivanje brez in z več primeri

Inženiring pozivov je veščina oblikovanja pozivov, ki poveča zmogljivosti velikih jezikovnih in vizualno-jezikovnih modelov. Poziv je sestavljen iz navodil, konteksta in uporabnikovega vnosa. [3] Pristop je precej razširjen in se ga poslužujejo tudi večji ponudniki velikih jezikovnih modelov, na primer v obliki sistemskih pozivov.[4]

Povezave pozivov in delovanja modela so opazili raziskovalci prvih modelov GPT. Namesto specializiranih modelov za posamezna področja, so raziskovalci ugotovili, da lahko uporabijo namensko pripravljene pozive s katerimi so lahko model uspešno uporabili na nalogah za katere ni imel posebej pripravljenih podatkov. [5] Tej tehniki, pozivanje brez primerov (angl. zero-shot prompting) je sledil poizkus, da v pozivu navedemo več primerov, t.i. pozivanje z več primeri (angl. few-shot prompting, ki še dodatno optimizira rezultat.[6]



Slika 1: Inženiranje pozivov. [7]

[22] ChatGPT je bil javno dostopen 30. novembra 2022. Prva iteracije je uporabljala model GPT 3.5.

[23] Gre za spreminjanje nastavitev modela, ki vplivajo na ustvarjeni rezultat. Uporabljajo se parametri, kot sta temperatura in top-p.

Oba pristopa ste v osnovi enaka. Pozivanje brez primerov pred uporabnikov poziv doda dodatna navodila in po potrebi kontekst. Pozivanje z več primeri pred uporabnikov poziv doda enega ali več primerov rešitve naloge. Čeprav oba pristopa dosežeta boljše rezultate z istimi modeli, je potrebno veliko truda za pripravo kakovostnih pozivov.

2.2. Verižno sklepanje

Kljub uspešnosti zgoraj opisanih pristopov veliki jezikovni modeli niso uspešno reševali naloge, kjer je bilo potrebno simulirati razmišljanje, na primer artimetika. Jason Wei je s soavtorji predstavil nov pristop, t.i. verižno sklepanje (angl. chain of thought). Verižno sklepanje model vodi v dodajanje t.i. vmesnih miselnih korakov. Namesto, da bi model generiral rezultat, model generira vmesne korake, ki vodijo do natančnejšega rezultata. [8]

Podobno kot pri pozivanju z več primeri je priprava kakovostnih pozivov za verižno sklepanje dolgotrajen in zahteven postopek. Kmalu so raziskovalci odkrili in preizkusili tehnike, ki do neke mere avtomatizirajo postopek verižnega sklepanja. Med bolj znanimi je tehnika, kjer se pozivu doda preprost stavek, da naj postopa po korakih, npr. "Let's think step by step."[9]

2.3. Uporaba orodji

Shunyu Yao je s sovatorji združil ugotovitve verižnega sklepanja in raziskovanja zmožnosti velikih jezikovnih modelov, da načrtujejo, in predlagali uporabo orodji. Verižno sklepanje in sorodne tehnike ne odpravijo omejitve na učne podatke modela. Kolikor se modelu doda možnost vključevanja zunanjih virov, na primer dostopa do programskega vmesnika Wikipedija, se bistveno izboljšajo generiranje rezultatov. [10]

Zmožnost modelov, da kličejo funkcije ali uporabljajo zunanja orodja, se sicer lahko doseže na način kot je opisan v zgoraj navedenem članku, a postopek težko skaliramo. Modele se lahko nauči uporabe orodja nadzorovanim učenjem, kar poenostavi integracijo novih orodji. [11]

Na temelju tovrstnih ugotovitev je družba OpenAI je modelom GTP 3.5 in GTP 4 dodala zmožnost klicanju funkcij in uporabe orodji. [12] Omenjena modela sta lahko uporabljala v naprej pripravljene funkcije ali funkcije, ki jih je pripravil uporabnik, tako da se je klicu dodal seznam orodji. Model pri generiranju rezultata odloči ali uporabo eno ali več funkcij ali ne. V času nastajanja tega prispevka je večina funkcij že dodanih v grafični vmesnik orodja ChatGPT.

2.4. Dodatek: agenti

Preden se posvetimo protokolu MCP, ki neposredno odgovarja na zastavljeno težavo, je potrebno omeniti koncept agentov. Agenti sicer niso neposredno povezani s protokolom MCP ali razvojem pozivnega inženiringa, se pa pogostop pojavljajo v povezavi z uporabo orodji in tehnikami priprave pozivov.

Agenti so aplikacije, kjer generiran rezultat velikih jezikovnih modelov nadzoruje potek izvedbe. Kadar govorimo o agentih ne govorimo o samostojnih aplikacijah, ki sami nadzorujejo svojo izvedbo, ampak o zmožnosti aplikacije, da v potek izvedbe vključi generiran rezultat modela. Agenti imajo lahko večje ali manjše stopnje integracije in uporabe rezultatov modela, zato govorimo o spektru agentskih aplikacij. Kar te aplikacije povezuje je omenjen vpliv generiranega rezultat na izvedbo. [13]

Agenti so seveda možni brez klicanja orodji in ne potrebujejo naprednih tehnik pozivov. Toda ker agenti delujejo z zunanjimi viri, na primer posodobljajo dokument na našem računalniku, si je v praksi težko zamisliti agente, ki ne lahko uporabljajo orodja in se ne poslužujejo tehniki pozivanja kot smo jih omenili zgoraj.

3 Protokol MCP

3.1. O protokolu

Če se vrnemo k uporabi funkcij in orodji, smo nakazali, da je težava število integracij, ki jih je potrebno narediti, da lahko integrirano programski vmesnik z modeli ali ponudniki modelov.

Anthropic je dne 25. Novembra 2024 objavil prvo revizijo protokola MCP. [14] Protokol je naslovil težavo številnih integracij, ki bi jih morali izdelati, da bi asistenti – njihov izraz za aplikacije, ki uporabljajo velike jezikovne modele – lahko uporabljali zunanje vire ali obstoječe aplikacij.

Nedolgo za tem sta protokol podprla tudi OpenAI in Google. Prvi implicitno z vključitvijo v njihovo ogrodje za razvoj agentov [15], slednji z izrecno omembo na konferenci Google I/O. S podporo treh največjih ponudnikov velikih jezikovnih modelov je protkol postal *de facto* standard.

Protokol nastaja javno, na omrežju Github. [16] Vsakdo lahko prispeva ali začne razpravo o protokolu na omrežju Github. V času nastajanja tega članka je zadnja različica protokola 3. različica z dne 18. junija 2025. Vsaka nova različica v uvodnem delu vključuje seznam sprememb v primerjavi s prejšnjo različico.

3.2. Osnove protokola

Protokol MCP je razdeljen na naslednja poglavja: uvod, osnovni protokol, odjemalec, strežnik in shema. Protokol standardizira integracijo ne glede na končno uporabo aplikacije, bodisi da gre za integrirano okolje za razvijalce, pogovori vmesnik ali po meri narejeno rešitev z uporabo LLM-ov. Protokol določa tako arhitekturni vidik, kot tudi format komunikacije.

Kot bomo videli v nadaljevanju protokol uporabi protokol JSON-RPC 2.0[17] za pošiljanje sporočil med strežnikom in odjemalcem. Kot analogijo protokol izrecno omenja protokol za jezikovne strežnike.

MCP standardizira:

- Deljenje konteksta z velikimi jezikovnimi modeli,
- Deljenje informacij o orodjih in zmogljivostih stražnikov in zmogljivostih odjemalcev,
- Izgradnjo izmenljivih in združljivih integracij terdelovnih potekov.

Protokol uporablja *JSON-RPC 2.0* sporočila za vzpostavitev komunikacije med:

- **Gostitelji**: LLM aplikacije, ki vzpostavijo povezave,
- **Odjemalci**: sestavni deli gostiteljske aplikacije, ki skrbijo za povezavo s strežnikom,
- **Strežniki**: proces, ki zagotavljajo kontekst in zmogljivosti.

MCP predpostavlja stanovitno povezavo med odjemalcem in strežnikom, ki komunicirata s pomočjo protokola JSON-RPC.

Strežniki odjemalcem nudijo:

- Vire: kontekst in podatke,
- Pozive: predloge sporočil in delovnih potekov,
- Orodja: funkcije, ki jih lahko LLM aplikacija pokliče.

Dodatno lahko **odjemalec** strežnikom nudi:

- Vzorčenje: strežniki lahko zaženejo delovanje agenta ali rekurzivno interakcijo z velikim jezikovnim modelom,
- Korene: odjemalec lahko strežniku omeji poizvedbe o enoličnih identifikatorjih ali dotočenem sistemu,
- Pridobivanje: poizvedbe s strani strežnika za pridobivanje dodatnih informacij o uporabniku.

Varnost in zaupanje sta sestavna dela protokola. Protokol predpisuje in zahteva, da se vsa dejanja izvajajo na podlagi izrecne privolitve. Postopek privolitve mora biti jasen, zato morajo biti tudi vmesniki razumljivi. Podatki, ki jih gostitelj ali strežnik hranita, ne smejo biti hranjeni brez uporabnikovega privoljenja. Podatki morajo biti ustrezno zaščiteni.

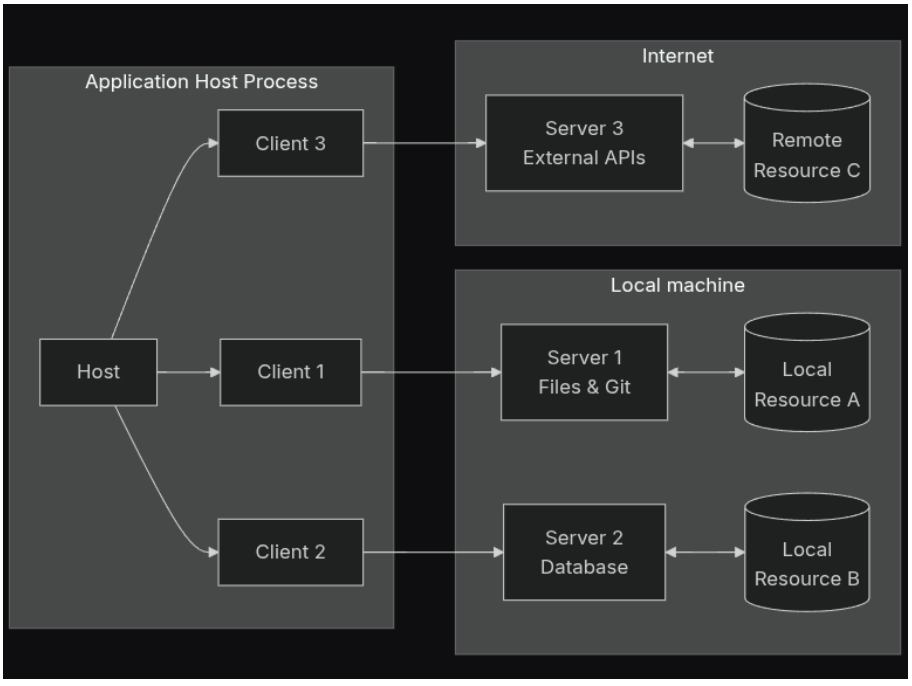
Nadalje protokol zahteva, da so orodja, ki jih nudi strežnik odjemalcu, varna in se smejo uporabiti samo s privoljenjem uporabnika. Njihova uporaba mora biti jasna in razumljiva uporabniku. Tudi za vzorčenje velja, da lahko strežniki uporabijo vzorčenje samo, če so meje vzorčenja jasne in je uporabnik predhodno privolil.

Protokol ne more vsiliti ali zagotoviti varnostih omejitev in zahtev, zato razvijalcem zadaja, da morajo graditi robustne postoka avtorizacije in podajanja soglasja, zagotavljati jasno dokumentacijo o varnosti sistema, implementirani ustrezen nadzor dostopanja in zaščite podatkov, slediti varnostnim napotkom in dobrim praksa ter vključiti varnost v vsako funkcionalnost njihovega sistema.

Polega navedenega protokol določa tudi načine nastavljanja, sledenja napredovanja, prekinitve izvajanja, sporočanja napak in beleženja.

3.3. Arhitektura

Kot je že bilo omenjeno so osnovne komponente protokola gostitelj, odjemalec in strežnik. Vsak gostitelj ima lahko več odjemalcev, tako da sta odjemalca in aplikacija ločena, kar omogoča, da se isti odjemalec doda več aplikacijam in ena aplikacija lahko vključuje različne odjemalce.



Slika 2: Inženiranje pozivov.[16]

Gostitelj:

- upravlja več odjemalcev,
- nadzoruje povezave, dovoljenja in življenjske cikle odjemalcev,
- uveljavlja varnost,
- razrešuje zahteve za avtorizacijo,
- usklajuje integracijo z velikim jezikovnimi modeli in vzorčenjem,
- upravlja agregacijo konteksta med odjemalci.

Odjemalec:

- za vsako sejo vzpostavi stanovitno povezavo s strežnikom,
- izvaja pogajanje o protokolu in izmenjavo zmogljivosti,
- dvosmerno usmerja protokolna sporočila,
- uprava z naročanjem in obvestili,
- zagotavlja varnost med strežniki.

Strežniki:

- Izpostavijo vire, orodja in pozive z uporabo osnovnih gradnikov protokola,
- Delujejo neodvisno od odjemalca,
- Zahteve vzorce,
- Spoštuje varnostne omejitve,
- Lahko deluje lokalno ali kot storitev na daljavo,
- Naj bo enostaven in se omeji na jasno določene zmogljivosti, kompleksne naloge naj prepusti gostitelju,
- Naj deluje izolirano,
- Ne smejo dostopati do celotnega pogovora ali imeti vpogleda v druge strežnike - za komunikacijo med strežniki skrbi gostitelj,
- Strežniki in odjemalci lahko postopoma dodaja funkcionalnosti in se razvijajo neodvisno drug od drugega.

Pogajanje o zmogljivostih

Kot je razvidno protokol določa majhen nabor zmogljivosti odjemalca in strežnika. Da bi odjemalec in strežnik vedela kater funkcionalnosti imata na voljo, morat sporočiti zmogljivosti in njihove omejitve.

Ob inicializaciji strežnik odjemalcu sporoči nabor naročnin, orodji in predlog za pozive. Odjemalec sporoči strežniku kater zmogljivosti podpira, npr. vzorčenje, obvestila. Oba morata spoštovati zmogljivosti in omejitve tekom seje.

3.4. Transportni protokoli, življenjski cikel, avtorizacija, varnost in pripomočki

MCP določa dva standardna transporta:

- **stdio**: strežnik se izvaja kot lokalni proces in komunicira prek stdin in stdout.
- **pretočni HTTP** (angl. streamable HTTP): strežnik uporablja protkol HTTP, strežnik palahko odgovori z enim samim JSON objektom ali s tokom SSE.

Življenjski cikel sledi točno določenim korakom:

- inicializacija seje: prvotna zahteva, prvoten odziv in potrditev,
- faza uporabe,
- postopek izklopa,
- prekinitev povezave.

V postopku inicializacije odjemalec strežniku pošlje različico protokola, podatke o sebi in svojih zmogljivostih. Strežnik mora potrditi različico protokola in poslati lastne podatke in seznam zmogljivosti.

V fazi uporabe odjemalec in strežnik spoštujeta različico protokola, ki sta jo dogovorila, medtem ko se nabor zmogljivosti strežnika lahko spreminja, tako da strežnik pošlje odjemalcu sporočilo, da naj odjemalec posodobi seznam zmogljivosti.

V postopku izklopa protokol predvideva, da se uporabijo metode transportnega mehanizma.

Protokol MCP postavlja varnost v ospredje z implementacijo OAuth 2.1.[18] Strežniki delujejo morajo preverjati žetone. Žetoni se pridobijo za vsak strežnik posebej in se ne smejo deliti.

Nadalje protokol nalaga razvijalcem, da:

- validirajo vhodne podatke in žetone,
- nadzorujejo dovoljenja in dostope ter preprečujejo pisanje zaupnih podatkov v sporočila ali dnevnike,
- seje naj bodo varne in edinstvene.

3.5. Koreni, vzorčenje in pridobivanje

Odjemalca komunicirajo s strežnikom in strežnik postavljajo meje ali kontekst. Odjemalci lahko dajo strežniku na voljo dostop do lokalnega datotečnega sistem, a ga lahko sočasno omejijo na enega ali več korenov.

Odjemalec v fazi inicializacije navede zmogljivosti, ki jih podpirajo. Podroben seznam pa pošljejo na strežnikovo poizvedbo z ustrezno metodo.

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "roots/list"
}
```

Slika 3: Zahteva za seznam korenskih dostopov.

Vzorčenje je zmožnost odjemalca, da strežniku omogoči dostop do modela. Strežnik model uporabi, da oblikuje sporočila, generira slike ipd. Na ta način lahko tudi strežnik deluje kot agent in za izvajanje ukaza uporabi model, ki ga uporablja aplikacija. Odjemalec lahko strežniku izpostavi dodane informacije o modelih, ki so na voljo, kot na primer hitrost, ali celo namigne, kako naj modele izbira.

Kadar strežnik za izvedbo ukaza potrebuje dodatne informacije s strani uporabnika lahko pridobi dodatne informacije od odjemalca, v kolikor jih odjemalec podpira. Postopek poteka podobno kot izvajanje ukazov, pri

čemer je strežnik tisti, ki sledi navodilo za izvajanje ukaza ali poizvedbe in odjemalec validira in odobri ali zavrne poizvedbo.

3.6. Pozivi, viri, orodja in pripomočki

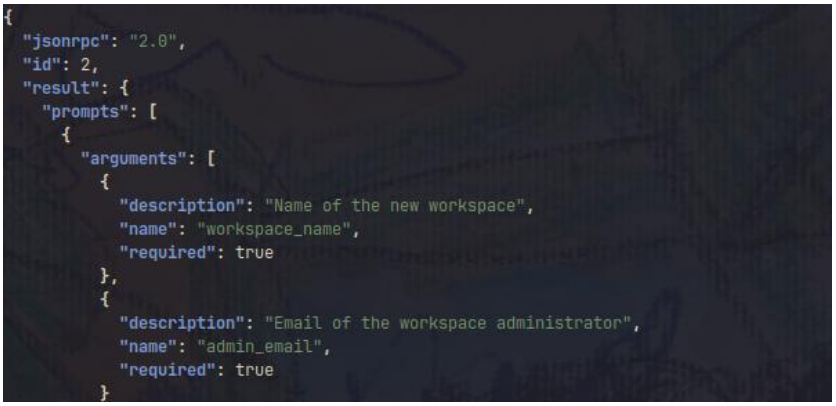
Trije primitivni tipi MCP protokola so pozivi, viri in orodja. Pozivi so predloge, ki jih uporablja in nadzira uporabnik, na primer s poševnico v pogovorni aplikaciji. Viri so datoteke in drugi besedilni viri, ki jih pridobi in uporablja aplikacija. Orodja so funkcije, ki jih lahko uporabi LLM s klici na MCP strežnik.

Strežnik mora v fazi inicializacije sporočiti kater primitivne tipe podpira.

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "capabilities": {
      "prompts": {},
      "resources": {},
      "tools": {}
    },
    "protocolVersion": "2025-06-18",
    "serverInfo": {
      "name": "magnum-mcp-server",
      "version": "1.0.0"
    }
  }
}
```


Slika 4: Inicializacija strežnika.

Strežnik ne rabi navesti vseh primitivov, ki jih uporablja, mora pa jih navesti. Seznam se pridobi s pošiljanjem ukaza s pravilno navedeno metodo, npr. Prompts/list.



```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "prompts": [
      {
        "arguments": [
          {
            "description": "Name of the new workspace",
            "name": "workspace_name",
            "required": true
          },
          {
            "description": "Email of the workspace administrator",
            "name": "admin_email",
            "required": true
          }
```


Slika 5: Seznam pozivov.

Vsak primitivni tip ima drugačno podatkovno strukturo. Pozivi, kot je razvidno, imajo ime, naziv (opcijsko), opis (opcijsko) in seznam argumentov. Od odjemalca pa pričakujejo vlogo (uporabnik ali asistent) in vsebino. Vsebina je nadaljnje razdelana glede na tip, torej besedilo, zunanji vir, slike ipd.

Podobnemu vzorcu sledijo viri in orodja. Strežnik navede seznam virov ali orodji v predpisani podatkovni strukturi, odjemalec pa pošlje zahtevo v predpisani obliki. Viri so morda posebnost, ker ne pričakujejo vsebine, ampak pravilno oblikovan enolični identifikator vira. Pri oblikovanju enoličnih identifikatorjev protokol MCP ne omejuje nabora, navaja pa splošne sheme kot so https://, file:// in git://.

Poleg primitivnih tipov strežniki lahko omogočajo tudi zmožnosti dopolnjevanja za izboljšano interaktivnost, beleženj in paginacijo.

3.7. Shema

Protokol vključuje tudi shemi, kjer so definirani ukazi, podatkovni modeli in tipi. Za shemi uporablja programski jezik Typescript[19] ali JSON shemo[20], ki se ga s pomočjo orodji enostavno pretvori v druge programske jezike, kot na primer vmesnike (angl. interface) v jeziku golang.

Shema

4 Praktični primeri

Na omrežju Github je v okviru organizacije Model Context Protocol objavljen seznam vzorčnih in vidnejših strežnikov.[21] Prav tako je na zgoraj navedeni spletni strani prosto dostopni tečaj, kjer je prikazan postopek implementacije strežnika.

Kljub zapletenosti protokola, predvsem na mestih, ki omogoča agentske lastnosti na strežniku, je implementacija strežnika dokaj enostavna. Na omrežju Github je na voljo več paketov za razvoj programske opreme za različne programske jezike. [22]

4.1. Strežnik za dostop do datotečnega sistema

Če pogledamo enega izmed uradnih vzročnih primerov, strežnik za dostop do datotečnega sistema. Strežnik je vzorčna implementacija, zato koda ne sledi nujno vsem dobrim praksam.

Strežnik se nahaja v eni datoteki. Ko zažene strežnik navede, da podpira orodja.

```
// Server setup
const server = new Server(
  {
    name: "secure-filesystem-server",
    version: "0.2.0",
  },
  {
    capabilities: {
      tools: {},
    },
  },
);
```

Slika 6: Strežnik MCP za datotečni sistem – inicializacija strežnika.

Seznam orodji našteje v namenskem upravljalniku.

```
// Tool handlers
server.setRequestHandler(ListToolsRequestSchema, async () => {
  return {
    tools: [
      {
        name: "read_file",
        description: "Read the complete contents of a file as text. DEPRECATED: Use read_text_file instead.",
        inputSchema: zodToJsonSchema(ReadTextFileArgsSchema) as ToolInput,
      },
      {
```

Slika 7: Strežnik MCP za datotečni sistem – seznam orodji.

Odjemalec kliče orodja v skladu s protokolom, strežnik pa jih pokliče tako, da iz ukaza pridobi orodje in ga požene z namensko metodo.

```
server.setRequestHandler(CallToolRequestSchema, async (request) => {
  try {
    const { name, arguments: args } = request.params;

    switch (name) {
      case "read_file":
      case "read_text_file": {
        const parsed = ReadTextFileArgsSchema.safeParse(args);
        if (!parsed.success) {
          throw new Error(`Invalid arguments for read_text_file: ${parsed.error}`)
        }
        const validPath = await validatePath(parsed.data.path);
```

Slika 8: Strežnik MCP za datotečni sistem – uporaba orodji.

Strežnik lahko zažene lokalno z ukazom "*npx -y @modelcontextprotocol/server-filesystem ~*". Strežnik uporablja protokol transportni protokol stdio in povežete s svojo aplikacijo LLM.

4.2. Strežnik za dostop do internega sistema za beleženje projektov

Na družbi Inova IT imamo interno orodje za beleženje dela na projektih. Aplikacija dnevno poziva zaposlene, da sporočijo na katerem projektu so delali. Ker je delo na projektih povezano z rednimi in izrednimi odsotnostmi od dela, aplikacija beleži in omogoča upravljanje dopustov.

Za potrebo tega prispevka in interno demonstracijo smo implementirali vzorčni strežnik za dostop do zalednega sistema aplikacije. Strežnik ima na voljo vzorčne pozive, orodja in celo izvede prijavo. Pri slednjem sicer ne sledi vse varnostmi zahtevam protokola, saj gre za vzorčno implementacijo.

Strežnik je bil zgrajen pomočjo orodja Claude code. [23]. Agent je dobil dostop do specifikacije Open API 3.0, ki jo s pomočjo anotacij samodejno generira zaledni sistem. S pomočjo specifikacije programskega vmesnika je agent pripravil ostnutek strežnika s seznamom pozivov in orodji. Tako kot pri zgornjem primeru so orodja del kode in za njih skrbi namenski upravljalnik.

```
func (s *MCPServer) handleRequest(ctx context.Context, req *MCPRequest) *MC
  response := &MCPResponse{
    Jsonrpc: "2.0",
    ID:      req.ID,
  }

  switch req.Method {
  case "initialize":
    response.Result = s.handleInitialize(req.Params)
  case "notifications/initialized":
    return nil
  case "tools/list":
    result, err := s.handleToolsList(req.Params)
    if err != nil {
      response.Error = &MCPError{
        Code:    ErrorInternal,
        Message: err.Error(),
      }
    } else {
      response.Result = result
    }
  case "tools/call":
    result, err := s.handleToolCall(ctx, req.Params)
    if err != nil {
      // Map error messages to appropriate error codes
      code := ErrorToolExecutionFailed
      if strings.Contains(err.Error(), "unknown tool") {
        code = ErrorToolNotFound
      } else if strings.Contains(err.Error(), "authentication required") {
        code = ErrorAuthenticationRequired
      } else if strings.Contains(err.Error(), "permission denied") {
```

Slika 9: Upravljalnik za orodja.

Strežnik smo uporabili v zgoraj omenjenim Claude code. Čeprav Claude code ni namenjen splošni uporabi, ampak je prilagojena za delo s kodo, je bil rezultat dober, saj je bilo mogoče podati zahtevo za izpis delovnih okolji in ustvariti novo okolje.

Primeru uporabe, ki so bili preizkušeni, so sicer preproste. Drži tudi, da je trenutno grafični vmesnik internega orodja boljši za izvedbo teh ukazov, a morebitne bodoče integracije s koledarjem, kjer bi zahtevo za dopust primerjali s stanjem v uporabnikovem koledarju nakazujejo na zanimive primere uporabe.

Strežnik je napisan v programskem jeziku Golang z minimalno uporabo knjižnic. Koda je dostopna na Githubu v organizaciji *inovait*.[24]

5 Zaključek

Avtor ocenjuje, da je protokol MCP velik korak v razvoju aplikacij, ki uporabljajo velike jezikovne modele. Protokol MCP sicer ne rešuje temeljnih omejitev velikih jezikovnih modelov, omogoča pa lažjo izgradnjo aplikacij, ki jih uporabljajo. Uporaba, kot je bilo nakazano, ni omejena na uporabnike, ampak je lahko del večjega procesa, ki ga deloma ali celoti izvajajo agenti.

Kljub številnim primerom implementacije in podporo programskim jezikom, je implantacija strežnika zahtevno delo. Tukaj avtor opozarja predvsem na varnostne zahteve. Aplikacije, ki del izvajanja prepustijo velikim jezikovnim modelom ali agentom, ki jih uporabljajo, izpostavljajo velike površine za napade. Agenti lahko pobrišejo datoteke ali baze, delijo zaupne ali osebne podatke ali prek predlog pozivi okužijo aplikacije.

Literatura

[1] huggingface.co/docs/transformers/en/model_doc/llama, Modeli Llama, obiskano 29. 7. 2025

[2] BENDER Emily M., GEBRU Timnit, McMILLAN-MAJOR Angelina, SHMITCHELL Shmargaret "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?", Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, Association for Computer Machinery – ACM, New York, str. 616.

[3] SAHOO Pranab, AYUSH Kumar Singh, SRIPARNA Saha, VINIJA Jain, SAMAT Sohel Mondal, AMAN Chadha "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications." ArXiv abs/2402.07927 (2024), str. 1.

[4] NEUMANN Anna, KIRSTEN Elisabeth, ZAFAR Muhammad Bilal, SINGH Jatinder "Position is Power: System Prompts as a Mechanism of Bias in Large Language Models (LLMs)", Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency, Association for Computing Machinery, New York, str. 574.

[5] RADFORD Alec, WU Jeffrey, CHILD Rewon, LUAN David, AMODEJ Dario, SUTSKEVER Ilya "Language models are unsupervised multitask learners". OpenAI blog, 1(8):9, 2019, str. 2.

[6] BROWN Tom et al "Language models are few-shot learners", arXiv preprint arXiv:2005.14165, str. 6.

[7] SAHOO Pranab, str. 1.

[8] WEI Jaso et al. "Chain-of-thought prompting elicits reasoning in large language models", Advances in Neural Information Processing Systems, številka 35, letnik 24824–24837, 2022, str. 3.

[9] SAHOO Pranab, str. 2.

[10] 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, Ruanda, str. 3-4.

[11] YUIJA Qin et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs, arXiv preprint arXiv:2307.16789, str. 3.

[12] https://platform.openai.com/docs/guides/tools, Navodila za uporabo orodji, obiskano dne 29. 7. 2025

[13] https://huggingface.co/docs/smolagents/en/conceptual_guides/intro_agents, Spletni tečaj za delo na agentih, obiskano dne 29. 7. 2025

[14] https://www.anthropic.com/news/model-context-protocol, Protokol MCP, obiskano dne 29. 7. 2025

[15] https://github.com/modelcontextprotocol/modelcontextprotocol, Repozitorij protokola MCP, obiskano dne 29. 7. 2025

[16] https://openai.github.io/openai-agents-python/mcp, obiskano dne 29. 7. 2025

[17] https://www.jsonrpc.org/, obiskano dne 29. 7. 2025

[18] https://modelcontextprotocol.io/specification/2025-06-18/architecture, obiskano dne 29. 7. 2025

[19] https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/ OAuth 2.1, obiskano dne 29. 7. 2025

[20] https://github.com/modelcontextprotocol/modelcontextprotocol/tree/main/schema/2025-06-18/schema.ts, , obiskano dne 29. 7. 2025

[21] https://github.com/modelcontextprotocol/modelcontextprotocol/tree/main/schema/2025-06-18/schema.json, , obiskano dne 29. 7. 2025

[22] https://github.com/modelcontextprotocol/servers, Seznam MCP strežnikov, obiskano dne 29. 7. 2025

[23] https://github.com/modelcontextprotocol?q=sdk&type=all&language=&sort=, Seznam paketov za razvoj programske opreme, obiskano dne 29. 7. 2025

[24] https://www.anthropic.com/claude-code, Claude code, obiskano dne 29. 7. 2025

[25] https://github.com/inovait/magnum-mcp, Repozitorij MCP strežnika za interno orodje na Inova IT, obiskano dne 7. 8. 2025.

Inovativni procesi zagotavljanja kakovosti razvoja programskih rešitev

Luka Prah, Mihael Kegl

Databox, Ptuj, Slovenija
luka.prah@databox.com, miha@databox.com

Članek obravnava sodobne pristope k zagotavljanju kakovosti (QA) v razvoju programske opreme, s poudarkom na proaktivni vlogi QA in njegovi vključitvi v zgodnje faze projektnega cikla. Predstavljen je koncept "shift-left", ki QA postavlja kot ključnega sogovornika pri analizi zahtev, oblikovanju funkcionalnosti ter uporabniške izkušnje. Prispevek temelji na praktičnih izkušnjah podjetja Databox, kjer QA proces vključuje jasno strukturirane faze: analiza zahtev, načrtovanje testiranja, priprava testnih primerov, raziskovalno in funkcionalno testiranje, avtomatizacija, regresija, poročanje ter poizidne aktivnosti. Posebna pozornost je namenjena tudi uporabi testnih orodij (Qase, Playwright, Asana) in pomenu komunikacije ter sodelovanja z razvojnimi in produktnimi ekipami. V članku so izpostavljeni tudi sodobni izzivi QA, kot so skalabilnost testiranja, upravljanje z regresijami in tehničnim dolgom, ter prilagajanje QA agilnemu razvoju. V zaključku članek ponuja priporočila za uspešno uvajanje inovativnih QA praks, kot so zgodnje vključevanje, CI/CD integracija, odgovornost za kakovost in retrospektiva kot orodje za nenehne izboljšave. Članek dokazuje, da QA ni zgolj faza testiranja, temveč strateški dejavnik, ki bistveno vpliva na kakovost, stabilnost in uporabniško izkušnjo končnega produkta.

Ključne besede:
zagotavljanje kakovosti,
shift-left pristop,
testni načrt,
QA strategija,
avtomatizacija.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.18

1 Uvod

Namen članka

V podjetju Databox smo se z namenom izboljšanja stabilnosti in dviga kakovosti produkta lotili sistemske prenove procesa zagotavljanja kakovosti (QA). Članek opisuje ključne faze QA procesa, ki ga izvajamo pri razvoju novih funkcionalnosti in večjih projektov, z vključitvijo QA ekipe že v zgodnji fazi razvoja.

Pomen kakovosti v razvoju programske opreme

Kakovost v razvoju programske opreme pomeni več kot le odpravo napak. Gre za celosten pristop, ki zagotavlja, da končni izdelek izpolnjuje pričakovanja uporabnikov, je stabilen, varen, zmogljiv in dolgoročno vzdrževan. V sodobnem svetu, kjer uporabniki pričakujejo brezhibno delovanje in kjer konkurenca na trgu ne dopušča veliko napak, kakovost postaja, oziroma bi morala postati ključno merilo uspeha.

Razlogi, zakaj je kakovost ključnega pomena:

1. Zadovoljstvo uporabnikov

Uporabniki pričakujejo intuitivne, hitre in zanesljive aplikacije. Slaba uporabniška izkušnja pomeni izgubo zaupanja in posledično strank. Dobra kakovost pomeni večje zadovoljstvo in dolgoročno zvestobo.

2. Zmanjšanje stroškov

Napake, odkrite v poznejših fazah razvoja ali v produkciji, so bistveno dražje za odpravo kot tiste, ki jih zaznamo zgodaj. S tem, ko QA sodeluje že v fazi načrtovanja, se zmanjšajo stroški popravkov in ponovnega razvoja.

3. Hitrejši čas do trga

Zanesljiv QA proces omogoča hitrejše izdaje brez potrebe po dolgotrajnih popravkih po produkciji. Z avtomatizacijo testiranja in dobro pripravljeno testno dokumentacijo lahko ekipa razvija in izdaja hitreje.

4. Zmanjševanje poslovnega tveganja

Slaba kakovost lahko vodi v izgubo podatkov, varnostne ranljivosti, pravne težave ali negativno medijsko pokritost. QA deluje kot zaščitni mehanizem pred takšnimi tveganji.

5. Gradnja dobrega ugleda

Podjetje, ki sistematično izdaja kakovostno programsko opremo, si gradi ugled zaupanja vrednega ponudnika. To neposredno vpliva na konkurenčnost in rast podjetja.

6. Omogočanje skalabilnosti

Dobro strukturiran QA proces omogoča, da podjetje lažje uvaja nove funkcionalnosti, podpira večje število uporabnikov in integrira različne komponente sistema brez ogrožanja stabilnosti.
Vse zgoraj našteto kaže, da kakovost ni strošek, temveč naložba – dolgoročna in strateška. Vlaganje v kakovost pomeni vlaganje v stabilnost, rast in uspeh podjetja. Zato mora biti QA del DNK vsake razvojne ekipe.

2 Kakovost kot strateška komponenta v razvojnem procesu

2.1. Prehod iz reaktivnega v proaktivni QA pristop

Tradicionalno je bila kakovost programske opreme obravnavana kot zadnji korak v razvojnem ciklu – nekaj, kar se preverja tik pred izdajo produkta. V takšnem **reaktivnem pristopu** QA deluje kot "lovilec napak", kar pomeni, da poskuša odkriti napake, ki so se že zgodile med razvojem. Ta pristop pogosto vodi do visokih stroškov popravil, zamud pri izdaji ter nezadovoljstva uporabnikov.
V sodobnem razvoju programske opreme pa kakovost ne more biti zgolj posledica testiranja, temveč mora biti **vgrajena v proces razvoja od samega začetka**. To pomeni prehod na **proaktivni QA pristop**, kjer je QA vključen že v zgodnjih fazah projektnega cikla – pri oblikovanju zahtev, načrtovanju funkcionalnosti in oblikovanju uporabniške izkušnje.
Ključne značilnosti proaktivnega QA pristopa:

- **"Shift-left" filozofija**: QA se pomakne levo po časovnici razvoja – v fazo načrtovanja in analize zahtev. To omogoča zgodnjo identifikacijo potencialnih napak in nejasnosti.

- **Sodelovanje z drugimi vlogami**: QA aktivno sodeluje z produktnimi vodji (PM), oblikovalci (UX/UI) in razvijalci že pri oblikovanju funkcionalnosti. S tem postane pomemben sogovornik pri odločitvah.

- **Preventivno razmišljanje**: Namesto da bi se QA osredotočal le na to, kaj ne deluje, razmišlja o tem, kaj bi *lahko* šlo narobe – in kako to preprečiti.

- **Sistematično načrtovanje testiranja**: Testni načrt nastaja že sočasno z oblikovanjem zahtev. Vključuje oceno tveganj, predlog izboljšav ter določitev strategije testiranja (ročno, avtomatizirano, raziskovalno …).

- **Poudarek na uporabniški izkušnji (UX)**: QA ne testira le tehnične pravilnosti, temveč tudi smiselnost uporabniških tokov, odzivnost, dostopnost in vizualne detajle.

Koristi prehoda na proaktivni QA pristop:

Prehod na proaktivni QA pristop prinaša številne prednosti, med katerimi je ena ključnih bistveno manjše število napak v poznejših fazah razvoja, ko so popravki običajno najdražji in najbolj zamudni. Zaradi zgodnjega vključevanja QA v razvojne aktivnosti se pospeši celoten proces, kar omogoča hitrejši razvoj in krajši čas do izdaje ("time to market"). Poleg tega se izboljša sodelovanje med ekipami, saj QA postane aktivni sogovornik produktnim vodjem, razvijalcem in oblikovalcem. Rezultat takšnega pristopa je višja kakovost končnega izdelka ter manj stresa in urgentnih situacij ob zaključevanju projektov.
Proaktivni QA pristop spreminja vlogo testiranja iz zadnje ovire v strateški gradnik uspešnega razvoja. QA postane ključni partner v procesu odločanja in zagovornik celostne kakovosti – od ideje do produkcije.

Koncept "shift-left" in vključevanje QA že v zgodnjih fazah

Koncept **"shift-left"** označuje premik aktivnosti zagotavljanja kakovosti (QA) **v zgodnejše faze razvojnega procesa**. Gre za zavestno spremembo miselnosti in organizacije dela, kjer QA ni več le faza pred izdajo produkta, temveč integralni del načrtovanja, analize in dizajna.

Kaj pomeni "shift-left"?

V klasičnem »waterfall« pristopu QA sledi po fazah razvoja in implementacije. V sodobnih, agilnih okoljih pa se QA premika levo – proti začetku časovnice projekta:

Slika1: Tradicionalni pristop testiranja.

Shift-left pristop: Start QA → Načrt → Razvoj → QA/Testiranje → Izdaja. To pomeni, da je QA vključen že v zgodnjih fazah projekta, kjer sodeluje pri analizi poslovnih zahtev, definiranju kriterijev sprejema (AC), načrtovanju funkcionalnosti in tehnične arhitekture, oblikovanju uporabniške izkušnje (UX/UI) ter oceni tveganj in izvedljivosti.

Slika2: Shift-left pristop testiranja.

Zgodnje vključevanje QA je izjemno pomembno, saj omogoča, da se napake odkrijejo že med razmislekom o funkcionalnosti, ko je njihovo odpravljanje še najcenejše in najmanj moteče. Poleg tega se s tem pristopom izognemo napačnim predpostavkam, ki bi se sicer lahko prenesle v razvoj in povzročile večje težave kasneje. QA s svojim sodelovanjem v zgodnjih fazah pripomore tudi k boljši in natančnejši dokumentaciji, saj zastavlja ključna vprašanja že pri pripravi specifikacij (PRD) ali oblikovalskih zasnov (Figma). Pomembno je tudi, da se testni načrt začne razvijati vzporedno z načrtovanjem funkcionalnosti, kar bistveno pohitri in olajša kasnejšo izvedbo testiranja.

Tabela1: Konkretni primeri vključevanja QA v zgodnje faze.

Faza	Vloga QA
Kick-off projektov	Prisotnost na sestanku, razumevanje vizije, izpostavitev tveganj
Analiza zahtev	Identifikacija testabilnih zahtev, vprašanja o nejasnostih
PRD + Figma	Predlogi izboljšav za UX/UI, priprava testnega skeleta
Ocena virov	Sodelovanje pri planiranju QA virov in časovnic

Pristop "shift-left" prinaša številne prednosti, med katerimi izstopa zmanjšanje stroškov zaradi zgodnjega odkrivanja napak, kar pomeni manj popravkov v kasnejših fazah razvoja. Ob tem se poveča tudi učinkovitost QA ekipe, saj se testiranje izvaja bolj ciljno in pravočasno. Takšen pristop spodbuja boljše sodelovanje z ostalimi

oddelki, kot so produktni vodje (PM), razvojne in oblikovalske ekipe, kar vodi k bolj usklajenemu razvoju. Rezultat je večja kakovost končnega izdelka, obenem pa takšen sistem omogoča lažje in bolj obvladljivo skaliranje QA procesov tudi pri večjih in kompleksnejših projektih.
Koncept "shift-left" omogoča, da QA postane **partner pri oblikovanju izdelka**, ne zgolj nadzornik njegovega delovanja. Vključevanje QA v zgodnjih fazah vodi do premišljenega, zanesljivega razvoja in bistveno boljše uporabniške izkušnje – kar je v končni fazi tudi glavni cilj vsake programske rešitve.

3 Proces zagotavljanja kakovosti pri razvoju večjih funkcionalnosti

3.1. Koraki QA cikla na primeru Databox

Pri razvoju večjih funkcionalnosti je ključno, da je QA cikel jasno definiran, strukturiran in hkrati dovolj prilagodljiv, da omogoča učinkovito sodelovanje vseh vpletenih ekip. V podjetju Databox uporabljamo uveljavljen pristop, ki temelji na preizkušenih korakih zagotavljanja kakovosti skozi celoten življenjski cikel funkcionalnosti – od začetne ideje do izdaje in tudi po njej.
QA cikel se začne z **analizo zahtev**, kjer QA inženir pregleda tehnične in poslovne specifikacije (PRD), Figma dizajne ter aktivno sodeluje na kick-off sestankih. Namen te faze je razumeti obseg funkcionalnosti, izpostaviti morebitne nejasnosti in že zgodaj definirati testabilne kriterije.
Sledi **načrtovanje testiranja**, kjer QA pripravi začetni Test Plan, opredeli strategijo testiranja (ročni testi, avtomatizacija, regresija, raziskovalno), identificira potrebne tipe testov in oceni potrebne vire (čas, okolja, orodja, ljudi). Na tej točki se začne tudi razmislek o tveganjih in prioritetah.
V naslednji fazi QA pristopi k **pripravi testnih primerov**, ki temeljijo na potrjenih kriterijih sprejema. Najprej se zgradi skelet primerov, nato pa se ti dopolnjujejo skozi testiranje, zlasti pri bolj odprtih ali kompleksnih funkcionalnostih.
Vzpostavitev **testnega okolja in testware** je pomembna podlaga za učinkovito testiranje. QA v sodelovanju z DevOps zagotovi, da testno okolje odraža produkcijsko, da so podatki ustrezno pripravljeni, ter da so vzpostavljena vsa potrebna orodja (npr. Qase, Playwright, orodja za beleženje napak).
Testiranje se začne z **raziskovalnim pristopom**, kjer QA prosto raziskuje funkcionalnost in išče očitne vizualne ali uporabniške pomanjkljivosti. To testiranje pogosto razkrije področja, ki jih specifikacije ne pokrivajo, a so za končno izkušnjo pomembna.
Ko so osnovne zadeve preverjene, QA nadaljuje s **funkcionalnim testiranjem**, kjer sistematično preverja vse predvidene tokove uporabe, robne primere in napake. Vse zaznane napake se zabeležijo v orodju (Asana), hkrati pa QA sodeluje pri prioritizaciji njihove odprave.
Po odpravi ključnih napak in ponovnem testiranju le teh, sledi **regresijsko testiranje**, kjer QA preveri, ali nove spremembe niso vplivale na obstoječe delovanje sistema. Ta faza je pogosto podprta z avtomatizacijo – v našem primeru s pomočjo Playwright skript.
Zaključek testiranja vključuje pripravo testnega poročila (kratkega ali razširjenega), preverjanje odprtosti vseh QA nalog in komunikacijo z ostalimi deležniki, da je funkcionalnost pripravljena za produkcijo.
Po izdaji sledijo še aktivnosti, ki vključujejo obvezni dimni test (smoke test) v produkcijskem okolju, spremljanje delovanja (monitoring), sodelovanje v retrospektivi ter posodabljanje testnih primerov glede na novo pridobljene izkušnje. Vse zaznane napake se prenesejo v "Product bugs" projekt, ker se prioritizirajo in planirajo za aktivnosti po izdaji, QA pa zagotovi, da se znanje iz projekta prenese naprej. Le to zagotavljamo tudi z redni QA Guild sestanki, ker se strukturirano in planirano deli znanje z ekipo, ter tudi širše drugim ekipam in obratno. Takšen strukturiran, a prilagodljiv QA proces omogoča zanesljivo uvajanje večjih funkcionalnosti brez ogrožanja kakovosti produkta ali uporabniške izkušnje.

4 Praktični primer: Proces testiranja nove funkcionalnosti

Pri razvoju kompleksnih funkcionalnosti, kot je na primer nov modul za konfiguracijo vizualizacij v analitični platformi, je uspešnost testiranja neposredno povezana s kakovostjo sodelovanja med različnimi ekipami ter z uporabo ustreznih orodij in dokumentacije. V nadaljevanju predstavljamo praktičen potek QA procesa v takšnem projektu.

Ključ do uspeha je zgodnja **vključitev produktnega vodje (PM)**, **oblikovalcev (UX/UI)** in **razvijalcev**, skupaj z QA inženirji. Proces se začne z **izhodičnim sestankom**, kjer PM predstavi specifikacije (PRD), oblikovalci priložijo Figma prototipe, QA pa že na tem mestu zastavi pomembna vprašanja glede robnih primerov, nejasnosti v tokovih ter predlaga izboljšave.

Sledi priprava testnega plana in skelet testnih primerov. Pri tem QA aktivno uporablja orodja, kot so:

- **Qase** [2] za strukturirano dokumentacijo testnih primerov, testnih planov in izvedbo testnih ciklov;
  Slack za tekočo komunikacijo z razvojem, PM in ostalimi deležniki (vključno z objavo napredka in blockerjev v #QA kanalih);
  Asana [3] za organizacijo nalog, označevanje QA pripomb ter beleženje napak in predlogov za izboljšave znotraj projekta.

QA vodi testiranje po korakih: začne z raziskovalnim testiranjem, nato izvede funkcionalne preglede, nato pa pripravi regresijski testni cikel. Med testiranjem aktivno dopolnjuje testne primere, poroča o napakah ter sproti komunicira prioritete z razvojno in produktno ekipo.

Celoten proces se dokumentira v Qase in Asana, končni izdelek pa je testno poročilo, ki vključuje stanje testnih primerov, statistiko uspešnosti, odprte težave ter končno priporočilo glede pripravljenosti za izdajo. Komunikacija v vsaki fazi je ključna – QA redno obvešča vse vpletene o stanju testiranja, tveganjih in morebitnih blokadah.

Ta pristop jasno pokaže, da QA ni le zaključna kontrola, temveč most med načrtovanjem, izvedbo in izdajo – z osredotočenostjo na kakovost, komunikacijo in uporabniško vrednost.

5 Avtomatizacija testiranja

Avtomatizacija testiranja je ključni element sodobnega QA pristopa, še posebej pri agilnem razvoju, kjer so hitre in pogoste izdaje stalnica. Čeprav ročno testiranje ostaja nepogrešljivo pri raziskovalnih in kompleksnih primerih, avtomatizacija bistveno prispeva k hitrosti, ponovljivosti in zanesljivosti testiranja.

Odločitev za avtomatizacijo temelji na več dejavnikih. Prvi pomemben kriterij je pogostost izvajanja testov – scenariji, ki se večkrat ponavljajo, kot so npr. dimni testi ali regresijski prehodi, so idealni kandidati za avtomatizacijo. Prav tako je pomembna stabilnost funkcionalnosti – avtomatiziramo le tiste dele sistema, ki so dovolj dodelani in imajo jasno definirane uporabniške tokove. Poleg tega avtomatizacija omogoča dolgoročen prihranek časa in stroškov, saj nadomešča ročno izvajanje ponavljajočih se testov. Zmanjšuje tudi možnost napak zaradi človeškega faktorja, saj zagotavlja doslednost pri izvajanju.

V podjetju Databox za avtomatizirano testiranje uporabniškega vmesnika, API in vizualnih testov uporabljamo orodje **Playwright** [1], ki omogoča zanesljivo izvajanje end-to-end testov v več brskalnikih. Playwright je prilagojen za moderne spletne aplikacije in omogoča testiranje v okoljih Chromium, Firefox in WebKit. Poleg preverjanja interakcij in vizualnih komponent omogoča tudi napredno validacijo prek nadzora omrežnih zahtevkov in zalednih podatkov.

Izdelava avtomatiziranih testnih skript pa ni zgolj tehnična naloga, temveč zahteva strateški pristop. Ključnega pomena je dobra organizacija kode, na primer z uporabo vzorca Page Object Model, ki ločuje selektorje od logike testa. Prav tako je pomembna ponovna uporaba funkcij in pomočnikov (helperjev), pregledno poimenovanje

testov in pisanje razlagalnih komentarjev za lažje vzdrževanje. Testne skripte je treba redno posodabljati glede na spremembe v uporabniškem vmesniku ali poslovni logiki, da ohranimo njihovo učinkovitost in zanesljivost [4].
Za maksimalen učinek je avtomatizacija vključena tudi v **CI/CD procese**. Integracija z orodji, kot so GitHub Actions, GitLab CI ali Jenkins, omogoča avtomatsko izvajanje testov ob vsaki spremembi kode ali odprtem pull requestu. Na ta način se v zgodnji fazi odkrijejo regresije, preprečijo izdaje z napakami, razvijalci in QA pa prejmejo hitro povratno informacijo. Avtomatizirani testi tako postanejo sestavni del vsakodnevnega delovnega toka in ne zgolj orodje, ki se uporablja občasno.
Avtomatizacija testiranja dopolnjuje ročne preglede in omogoča višjo stopnjo zaupanja v stabilnost programske opreme. Z učinkovito uporabo orodij, kot je Playwright, ter vključevanjem testiranja v CI/CD procese, lahko QA ekipe bistveno povečajo hitrost in zanesljivost razvoja ter omogočijo kakovostne izdaje brez nepotrebnega tveganja.

6 QA kultura in odgovornost

V sodobnih razvojnih okoljih QA ni več zgolj vloga, temveč sestavni del kulture podjetja. Eden najpomembnejših konceptov, ki se v zadnjih letih vse bolj uveljavlja, je **QA kot lastnik kakovosti**. To pomeni, da QA ni tam le zato, da poišče napake, temveč ima popolno odgovornost za kakovost funkcionalnosti – od začetne specifikacije do končne izdaje. QA aktivno skrbi za preverjanje vseh vidikov funkcionalnosti, od tehnične pravilnosti do uporabniške izkušnje, in zagotavlja, da izdelek izpolnjuje visoke standarde kakovosti.
Za uspešno izvajanje takšne vloge je ključna **samoiniciativnost**, saj QA pogosto prevzema naloge brez izrecnih navodil, prepoznava morebitna tveganja še pred razvojem in predlaga izboljšave še preden se pojavijo težave. Enako pomembna sta **transparentnost in sodelovanje z ekipo** – QA mora komunicirati jasno, pogosto in konstruktivno z razvijalci, produktnimi vodji in oblikovalci. Le z odprto komunikacijo in skupnim razumevanjem ciljev je mogoče zagotoviti usklajeno in učinkovito testiranje.
Posebno pomembno vlogo pri vzdrževanju kakovosti in učenju iz preteklih izkušenj ima **retrospektiva**. To ni zgolj formalnost ob zaključku projekta, temveč dragoceno ogrodje za izboljšave. QA na retrospektivo prinese vpogled v to, kaj je šlo dobro, kje so bile težave, kako učinkovita je bila komunikacija in kateri procesi bi lahko bili boljši. Takšna povratna informacija ne koristi le QA ekipi, temveč vsem v projektu, saj omogoča stalno izboljševanje razvojnega procesa.
Kultura kakovosti se torej ne vzpostavi s pravilniki in orodji, temveč z aktivnim vključevanjem QA v vse faze razvoja, z jasno dodeljeno odgovornostjo in s skupno zavezanostjo ekipe k izboljševanju. QA v tem kontekstu ni le »lovilec napak«, ampak partner v razvoju, usmerjen v ustvarjanje najboljših možnih rešitev.

7 Izzivi in priložnosti sodobnega QA

Z razvojem tehnologij, UI (Umetne inteligence), hitrejšimi cikli izdaj in vedno večjimi zahtevami uporabnikov se QA sooča z novimi izzivi – hkrati pa se odpirajo številne priložnosti za nadgradnjo vloge in pristopov. V nadaljevanju so predstavljeni trije ključni vidiki, s katerimi se sodobne QA ekipe pogosto srečujejo.
Eden glavnih izzivov je **skalabilnost testiranja**. Ko produkt raste in postaja kompleksnejši, se povečuje tudi število funkcionalnosti, scenarijev, kombinacij in platform, ki jih je treba preizkusiti. Ročno testiranje vseh možnosti postane neobvladljivo, zato je ključnega pomena uvedba avtomatizacije, pametno upravljanje testnih primerov, prioritizacija testov in uvajanje orodij, ki omogočajo učinkovito razširjanje QA aktivnosti brez linearnega povečevanja števila testerjev.
Poleg tega QA ekipe pogosto prevzemajo breme **upravljanja z regresijo in tehničnim dolgom**. Z vsakim novim dodatkom ali spremembo v kodi se poveča tveganje, da bo nekaj obstoječega prenehalo delovati. Če regresijsko testiranje ni dobro načrtovano ali avtomatizirano, se napake lahko prikradejo v produkcijo. Hkrati je naloga QA, da opozarja na tehnični dolg – zastarele teste, preobremenjeno infrastrukturo za testiranje ali ponavljajoče se težave,

ki jih razvoj še ni naslovil. Proaktivno zaznavanje in beleženje teh področij prispeva k dolgoročni stabilnosti produkta.
Sodobna QA vloga se mora tudi prilagoditi zahtevam **agilnega razvoja**, kjer se funkcionalnosti razvijajo iterativno in v zelo kratkih ciklih. To pomeni, da mora biti QA sposoben hitro razumeti nove zahteve, pripraviti ustrezno testno strategijo in izvajati teste znotraj istega sprinta kot razvoj. V agilnem okolju QA ni več zadnja postaja, temveč aktiven partner v razvojnem procesu – redno sodeluje pri izpiljevanju nabora zahtev, planiranju, pregledu sprintov in dnevnih usklajevanjih.
Kljub številnim izzivom pa prav tu ležijo tudi največje priložnosti QA. S pravo kombinacijo tehničnega znanja, razumevanja produkta, komunikacijskih veščin in strateškega razmišljanja lahko QA postane osrednji člen v razvoju kakovostnih digitalnih rešitev. Z vlaganjem v avtomatizacijo, izboljševanjem procesov in krepitvijo sodelovanja z ostalimi ekipami QA ne samo ohranja svojo vrednost, ampak jo sčasoma še povečuje.

8 Zaključek

Zagotavljanje kakovosti v razvoju programske opreme ni več izolirana aktivnost, temveč postaja osrednji element celotnega razvojnega procesa. V članku smo predstavili sodoben, proaktiven in strateški pristop k QA, ki temelji na zgodnjem vključevanju QA inženirjev v faze načrtovanja, sodelovanju z različnimi oddelki ter uvajanju avtomatizacije in nenehnega izboljševanja. Ključni elementi uspešnega QA procesa vključujejo: jasno analizo zahtev, dobro strukturirane testne načrte, prilagodljivo uporabo testnih orodij, aktivno komunikacijo, učinkovito regresijsko testiranje, testno dokumentacijo, ter poizidne aktivnosti in retrospektivo.
Proaktivni pristop, uveljavljen v podjetju Databox, jasno pokaže, da QA ni le funkcija iskanja napak, ampak partner pri razvoju, odgovoren za stabilnost, uporabniško izkušnjo in dolgoročno kvaliteto produkta.
Na podlagi predstavljenih praks lahko podamo nekaj ključnih **priporočil za uvajanje inovativnih QA procesov**:

- Vključite QA že v fazi načrtovanja funkcionalnosti, kjer lahko prispeva k definiciji zahtev, sprejemnih kriterijev in oceni tveganj.
- Vzpostavite kulturo odgovornosti, kjer QA ni zgolj kontrola, temveč nosilec kakovosti, ki ima moč sooblikovati rešitve.
- Uporabljajte sodobna testna orodja (kot so Playwright, Qase, Testmo, Testrail ipd.), ki omogočajo avtomatizacijo in strukturirano dokumentacijo.
- Poskrbite za CI/CD integracijo testiranja, ki omogoča hitro in zanesljivo validacijo ob vsaki spremembi kode.
- Krepite sodelovanje med QA, razvojem, produktnimi vodji in oblikovalci – kakovost je skupna odgovornost, ne ločena funkcija.
- Ne pozabite na retrospektivo – sistematična analiza opravljenega testiranja je ključno orodje za nenehno izboljševanje procesov.

Sprejemanje inovativnih QA praks pomeni dolgoročno vlaganje v stabilnost, hitrost in kakovost razvoja. Podjetja, ki uspešno vključijo kakovost kot strateško komponento, si zagotovijo konkurenčno prednost, boljšo uporabniško izkušnjo in trajnostno rast.

Literatura

[1] https://playwright.dev/, Microsoft, obiskano julij 2025

[2] https://qase.io/, Qase, obiskano julij 2025

[3] https://asana.com/, Asana, obiskano julij 2025

[4] OTS 2023 Sodobne informacijske tehnologije in storitve: Zbornik 26. konference, Testno ogrodje za razvijalce - ali kako doseči, da razvijalci celovito testirajo, Mitja Krajnc, Tadej Ciglič, Boris Ovčjak, Databox

Zagotavljanje kakovosti pri integriranju z zunanjimi viri podatkov

Tadej Volgemut

Databox, Ptuj, Slovenija
tadej.volgemut@databox.com

Zagotavljanje kakovosti pri povezovanju z zunanjimi viri podatkov predstavlja enega najpomembnejših izzivov v podatkovno usmerjenih sistemih, kot je Databox. Raznolikost programskih vmesnikov (ang. application programming interface, API), pomanjkljiva dokumentacija in nenapovedane spremembe zahtevajo natančno načrtovan in prilagojen pristop zagotavljanja kakovosti (QA), ki vključuje tehnične, procesne in vsebinske plasti preverjanja. V prispevku predstavimo, kako pristopamo k zagotavljanju kakovosti integracij: od začetne analize in testiranja povezav, do validacije podatkovne točnosti in konsistentnosti prikaza metrik. Opišemo uvedbo notranjih ogrodij, kot je Integration Testing Framework (ITF), ki omogoča razvijalcem samostojno testiranje že med implementacijo, in uporabo avtomatizacije ter umetne inteligence (AI) za generiranje testnih primerov in zaznavanje anomalij. Poudarek je tudi na vplivu kakovosti na uporabniško izkušnjo in na vlogi QA pri preprečevanju težav v produkciji. Prispevek temelji na praktičnih izkušnjah, ki so rezultat sodelovanja med razvojem in QA ekipo, ter ponuja pogled v smeri večje standardizacije in napredne uporabe AI v prihodnosti.

Ključne besede:
zagotavljanje kakovosti,
integracije,
API,
avtomatizirano testiranje,
umetna inteligenca.

Univerzitetna založba Univerze v Mariboru

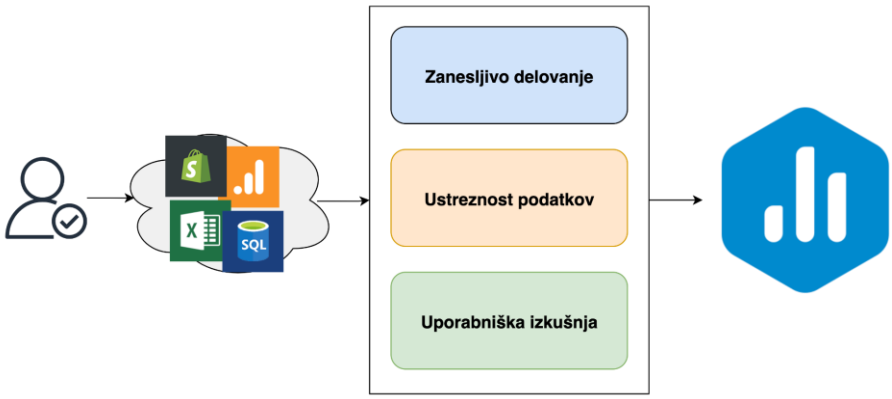
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.19

1 Uvod

Integracije z zunanjimi viri podatkov predstavljajo hrbtenico podjetja Databox. Uporabnikom ponujamo povezovanje z različnimi viri in s tem dostop do podatkov več kot 100 ponudnikov (npr. Google Analytics 4, Shopify, Excel, baze podatkov) kar skupaj tvori podatkovni ekosistem, na katerem temelji več naših produktov. Glede na raznolikost integracij ter kompleksnost podatkovnih tokov in struktur je zagotavljanje kakovosti ključnega pomena – tako na ravni povezovanja s ponudnikom kot tudi pri nadaljnji obdelavi in prikazu podatkov. V prispevku se osredotočamo na tri ključna področja zagotavljanja kakovosti:

- Zanesljivost delovanja – od konsistentne tehnične implementacije in varne vzpostavitve povezave z zunanjimi ponudniki do stabilnega in zanesljivega prenosa podatkov.
- Točnost in ustreznost podatkov – podatki morajo biti pravilno interpretirani, tehnično in semantično ujemajoči ter umeščeni v ustrezen kontekst uporabe.
- Uporabniška izkušnja – preglednost, intuitivnost in enostavna uporaba so pomembni vidiki kakovosti, ki dopolnjujejo tehnično plat rešitve.

Ker kakovost obravnavamo celostno, jo v prispevku predstavljamo skozi procesne pristope, tehnične rešitve in – posebej – testiranje. Poudarek namenjamo vlogi ekipe za zagotavljanje kakovosti (QA) od začetne analize, skozi implementacijo, do izdaje in vzdrževanja integracije. Izzive in pristope prikazujemo na konkretnih primerih iz prakse.
Raznolikost integracij ter pogosto pomanjkljiva dokumentacija ali nepričakovane spremembe na strani ponudnikov [1] dodatno krepijo pomen sistematičnega pristopa k zagotavljanju kakovosti.
Slika 1 ponazarja, kako trije ključni vidiki kakovosti (zanesljivost, točnost, uporabniška izkušnja) neposredno vplivajo na uspešnost in sprejetost integracij v Databox okolju. Da jih preverjen uporabnik lahko uporablja, in da ga obdržimo, mora njihovo delovanje biti zanesljivo, podatki ustrezni, adopcija pa enostavna in intuitivna.



Slika 1: Vloga kakovosti integracije v Databox.

2 Zunanji viri podatkov kot temelj ekosistema

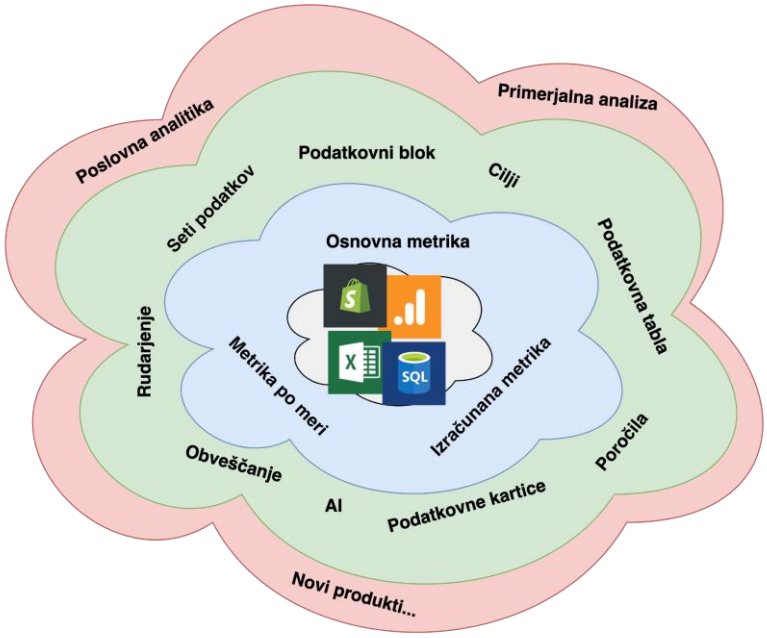
2.1. Sistemske razlike med integracijami

Integracije, ki jih ponujamo uporabnikom, so raznolike iz tehničnega in vsebinskega vidika. Razvoj in QA morata imeti v mislih predvsem te lastnosti:

- Preverjanje identitete (avtentikacija): osnovna (npr. *Basic*), ki zahteva uporabniško ime in geslo, ali napredna (npr. *OAuth 2.0*), ki vključuje pridobivanje in uporabo žetona (ang. *token*), pogosta je tudi avtentikacija z API ključem (*API key*), ki zahteva le vnaprej pripravljen ključ.

- Dodeljevanje pravic (avtorizacija): definira kaj lahko uporabnik počne. Za nas so pomembe predvsem bralne (ang. *read-only*) pravice (npr. *read:analytics*).
- Strukturo in obseg vhodnih in izhodnih podatkov: ali vstopamo prek parametrov spletnega naslova (ang. *URL parameters*), ali s podatki v telesu (ang. *body*) sporočila. Izstopni format je lahko različnih tipov (*JSON*, *XML*) in razdeljen na več strani (*paginacija*). Preferiramo paginiran *JSON*.
- Omejitve poizvedb (rate limiting): optimalno bi bilo, da nismo omejeni. Ponavadi pa je to določeno s številom zahtev, ki jih lahko naredimo v danem času.
- Svojevrstno tolmačenje podatkovne semantike: kar je pogosto še bolj kritično. Tako lahko dve integraciji, ki na prvi pogled podajata enako metriko (npr. "št. uporabnikov"), v praksi merita popolnoma različne stvari.

Skupni imenovalec vseh ponudnikov so njihovi podatki, ki jih ovrednotimo, oplemenitimo in ponudimo uporabniku. Slika 2 prikazuje konceptualno pot podatka: od njegove pridobitve iz zunanjega vira, prek pretvorbe v metriko, do prikaza v posameznem produktu znotraj Databox okolja.



Slika 2: Podatek gre v metriko, ki je osnova za funkcionalnost na nekem produktu.

2.2. Skupni QA pristopi

Določeni pristopi, ki jih uporabljamo pri strukturiranju kode, obravnavi napak in validaciji podatkov, so podrobneje opisani tudi v tehničnem blogu [1]. Ključni poudarki so naslednji:

- Koda mora biti ustrezno strukturirana in implementirana po smiselnih vzorcih, le tako je lahko dodajanje novih in vzdrževanje obstoječih integracij hitro in učinkovito.
- Delegiranje napak mora biti sistemsko - uporabnik se ne sme znajti v situaciji, ko ne ve kaj je šlo narobe (npr. napaka na avtentikaciji, nezadostne pravice za ogled določenega podatka, ali presežena meja poizvedb v nekem času).
- Preverjanje pravilnosti podatkov pomeni, če npr. metrika Total users prikazuje podatke za vse uporabnike. Točnost na drugi strani pove če se vrednosti, ki jih pokažemo uporabniku ujemajo s tistimi na izvorni platformi (npr. Google Analytics dashboard).

- Notranja orodja in ogrodja (ang. *frameworks*) pomagajo pri hitrosti in zanesljivosti prejšnjih treh točk. V naslednjih poglavjih bomo podrobneje predstavili ogrodje, ki omogoča posnetek metrike in primerjavo z živimi podatki.

Naše izkušnje kažejo, da največ težav ne povzroča tehnična izvedba sama po sebi, temveč funkcionalnosti, ki pogosto niso dokumentirane ali so implementirane v nasprotju z ponudnikovo dokumentacijo. Velik izziv predstavljajo tudi večje spremembe, ki jih ponudniki včasih objavijo zelo pozno, ali jih sploh ne. Zato QA pristop pri integracijah ne more biti generičen – vsak primer zahteva prilagojeno testiranje, pogosto tudi ročno analizo in kontekstualno razlago.
Kljub tehnični raznolikosti integracij uvajamo enoten nabor preverjanj, ki tvorijo osnovo našega pristopa k zagotavljanju kakovosti. Ti koraki vključujejo:

- Preverjanje povezave – osnovna validacija avtentikacije in avtorizacije ter uspešne vzpostavitve povezave z virom podatkov.
- Validacija podatkovnega toka – preverjanje, ali integracija pravilno pridobi ključne metrike v ustrezni strukturi in obsegu.
- Testiranje zgodovinskih podatkov – preverjanje, ali API vrne celovit odziv za daljše časovne obsege ter ustrezno podpira paginacijo.
- Semantična skladnost metrik – usklajenost med pridobljenimi podatki in pričakovano vsebino, kot jo prikazuje izvorna platforma.
- Regresijsko preverjanje – primerjava trenutnih rezultatov z vnaprej zajetimi posnetki s čimer odkrivamo morebitne napake pri izboljšavah.

Ti koraki se izvajajo s pomočjo notranjih orodij in testnih ogrodij, kot je Integration Testing Framework (ITF), in predstavljajo sistematiziran pristop, ki ga uspešno uporabljamo tudi pri kompleksnih integracijah – na primer pri integraciji z Google Analytics 4, kjer se pokaže večina omenjenih izzivov.

2.3. Poseben primer: GA4 testni pristop

Kot konkreten primer predstavljamo povezovanje z zunanjim virom Google Analytics 4 (GA4), kjer se združujejo številni izzivi, značilni za kompleksne API-je: avtentikacija prek protokola OAuth 2.0, poizvedovanje prek strukturiranih zahtevkov ter obravnava kvot in omejitev. V nadaljevanju opisujemo, kako v praksi izvajamo preverjanje kakovosti pri tej integraciji.

Avtentikacija in avtorizacija
GA4 uporablja protokol OAuth 2.0, pri čemer mora uporabnik najprej izvesti prijavo v svoj Google račun. Sistem nato prejme:

- žeton za dostop (ang. *access token*), ki omogoča začasni dostop do API-ja,
- žeton za osvežitev (ang. *refresh token*), ki omogoča avtomatsko pridobivanje novega dostopnega žetona, brez ponovne prijave uporabnika.

Dovoljenja se določijo ob avtentikaciji z zahtevanim bralnim (ang. *readonly*) dostopom. Ta obseg dovoljuje zgolj branje analitičnih podatkov in predstavlja osnovni primer avtorizacije.

Struktura vhodnih in izhodnih podatkov

Pri integraciji z Google Analytics 4 uporabljamo Google Data API, kjer je osnovna metoda za pridobivanje metrik *runReport*. Ta metoda omogoča prilagojeno poizvedbo metrik in dimenzij glede na časovni razpon, filtre in parametre sortiranja. Uporablja se za večino standardnih poročil in je zato osrednji del QA režima znotraj naše integracije. Podatke pošiljamo v strukturirani obliki (*JSON*), ki vključuje:

- identifikator vira podatkov (npr. `property: "properties/123456"`) - nastavimo v parametru URL,
- seznam dimenzij in metrik (npr. `"dimensions": [{ "name": "date" }], "metrics": [{ "name": "activeUsers" }]`),
- filtriranje in razvrščanje.

Primer 1: Vhodni podatki za poizvedbo GA4 RunReport

```
{
  "dimensions": [{ "name": "date" }],
  "metrics": [{ "name": "activeUsers" }],
  "dateRanges": [{ "startDate": "2024-01-01", "endDate": "2024-01-31" }]
}
```

Primer 2: Izhodni podatki po uspešni poizvedbi

```
{
  "dimensionHeaders": [{ "name": "date" }],
  "metricHeaders": [{ "name": "activeUsers" }],
  "rows": [
    {
      "dimensionValues": [{ "value": "2024-01-01" }],
      "metricValues": [{ "value": "1243" }]
    }
  ]
}
```

Pri preverjanju kakovosti je ključna skladnost med tem, kar pričakujemo (npr. po vizualizaciji na strani Google Analytics), in tem, kar vrne API. Preverja se:

- celovitost podatkov (ali so zajeti vsi dnevi),
- ujemanje metrik (ali vrednosti ustrezajo vmesniku),
- doslednost strukture odziva tudi pri praznih ali delno napolnjenih rezultatih.

Omejitve poizvedb

GA4 API omejuje število zahtevkov na uporabnika, projekt in kombinacijo dimenzij/metrik:

- 10 zahtevkov na sekundo na uporabnika,
- 50.000 vrstic podatkov na dan,
- Quota Error (HTTP 429) je vrnjen, ko presežemo omejitve.

Zato je nujno, da testno okolje zajema primere:

- kako aplikacija ravna ob napaki *HTTP 429 Too Many Requests*,
- ali so implementirani algoritmi za ponovni poizkus z zakasnitvijo (ang. *backoff*),
- in ali lahko ob preobremenjenosti sistem degradira svojo funkcionalnost (npr. prednaloženi podatki - ang. *cache*).

Pri testiranju take integracije vključujemo naslednje vidike:

- preverjanje veljavnosti žetonov ter ukrepanje v primeru njihove neveljavnosti (npr. zaradi poteka veljavnosti, preklica ali napake),
- preverjanje pravilnosti vhodnih podatkov (zahtevki z napačno strukturo, manjkajoče metrike),
- preverjanje odziva API-ja ob napačni avtorizaciji (npr. odstranjen obseg),
- simulacijo povečanega prometa, ki povzroči dosego rate limita.

3 Procesi zagotavljanja kakovosti integracij z zunanjimi viri

3.1. Priprava in analiza API-ja

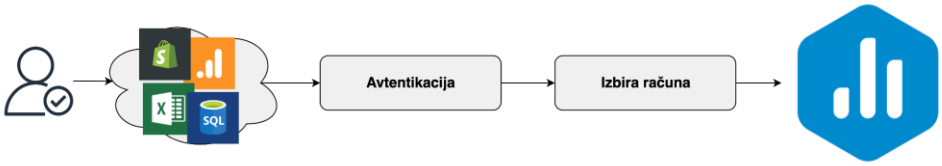
Pred pričetkom implementacije integracije se API željenega ponudnika skrbno preveri ali ustreza kriterijem vključitve. Pregleda se dostopna dokumentacija in vzpostavi nekaj inicialnih poizvedb, ponavadi kar prek orodja za delo z API-ji, Postman [2]. Z analizo rezultatov integraciji določimo lastnosti, ki smo jih opisali na začetku prejšnjega poglavja.

3.2. Usmeritve QA v začetni fazi

Za QA je pomembno, da se vključimo že v tej fazi, saj tako dobimo prvi vtis o kvaliteti in delovanju API-ja, količini in naravi metrik, ki jih bomo preverjali, predvsem pa imamo možnost opozoriti na potencialne napake, ki bi se lahko zaznale šele med funkcionalnim testom. Vsebinska in tehnična pričakovanja uskladimo neposredno z razvojnim oddelkom, po potrebi pa tudi z vodjo projekta. Vse to je ključno pri podajanju ocene testiranja in pripravi testnega plana.

3.3. Skriptiranje in avtomatizacija povezave

Kasneje si QA pripravi podrobnejši seznam metrik, ki jih bo preverjal ter naredi podporne skripte in avtomatske teste. S skriptami si pomagamo pri analizi podatkovnih struktur, ki jih API vrača. Na primer, že zgodaj lahko preverimo, če poizvedba vrne vse podatke v enem odgovoru, ali večih (paginacija), če so vključeni vsi podatki izbranega časovnega okvirja in ali je struktura podatkov pravilna. Z avtomatskimi testi pa želimo avtomatizirati ponavljajoče delo in pokriti najbolj kritična področja, na primer povezovanje z računom integracije (najpogosteje pri integracijah z *API key* avtentikacijo), s čem potrdimo da povezovanje integracije s ponudnikom deluje, kar je pogoj, da lahko začnemo pridobivati podatke. Slika 3 prikazuje osnovni potek povezovanja integracije z zunanjim virom, kjer QA s pomočjo avtomatskih testov preveri, ali je povezava uspešna in ali sistem lahko prične z zajemom podatkov.



Slika 3: Uporabnik poveže svojo integracijo z Databox.

3.4. Preverjanje točnosti in ujemanja metrik

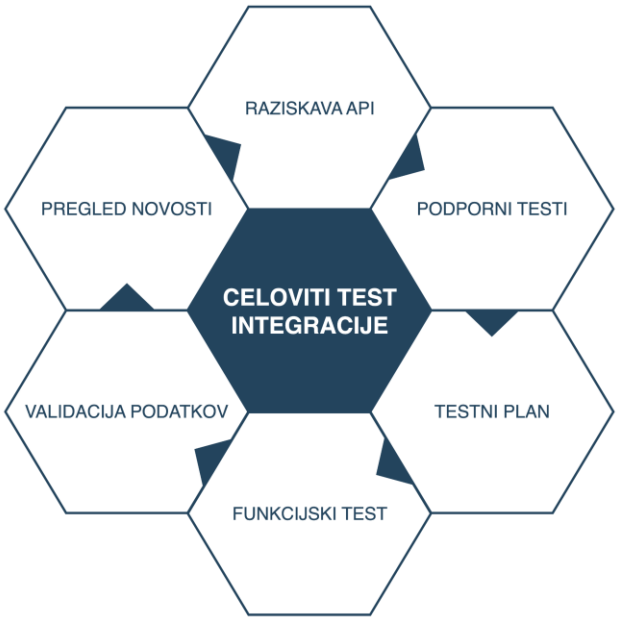
Velik delež testov zaradi raznolikosti še vedno ostaja ročen, in to je preverjanje točnosti podatkov. Zraven ustreznosti (format, časovna cona, uporaba enot, opis metrike), kar lažje avtomatiziramo, je pomembno, da se

pridobljeni podatki ujemajo z izvornimi. Pri veliki količini metrik je ta korak obsežen, in zahteva vzporedno primerjavo podatkov na naših vizualizacijah s prikazom na ponudnikovem uporabniškem vmesniku.
Pri podatkih izstopata 2 izziva:

- Vsi podatki niso na voljo na obeh straneh: zgodi se, da API vrača več, kot prikazuje ponudnik na svojem uporabniškem vmesniku. Točnosti takšnih podatkov ne moremo preveriti zato se velikokrat zanašamo na uporabniški odziv.
- Podatki niso enotni že pri ponudniku: tukaj lahko gre za (a) različne interpretacije metrike v API-ju in uporabniškem vmesniku ponudnika ali (b) napako. V tem primeru je ključno poznavanje dokumentacije in komunikacija z ponudnikom, kar z našo pomočjo opravi vodja projekta.

3.5. Komunikacija in prioritizacija

Tukaj je ključna komunikacija tako z vodjem projekta kot z razvojnim oddelkom. Uporabniški vmesniki ponudnikov so velikokrat kompleksni in iskanje metrik je le redko intuitivno. Ker je razvojni oddelek skozi začetno raziskavo že deloma spoznal vmesnik, nam s svojim znanjem lahko prihrani ogromno časa. Ker smo ponavadi omejeni z viri za testiranje, se je večkrat potrebno uskladiti glede pomembnosti metrik, in skladno s tem sestaviti testni plan. Če zunanji vir prek API-ja ponuja večje število metrik (npr. 125), ekipa za zagotavljanje kakovosti v sodelovanju z razvojem določi ključne metrike, ki jih preverimo podrobneje. Preostale vključimo v širši nabor osnovnih preverjanj (dimni test), pri katerih ne izvajamo ročne validacije vrednosti, temveč zgolj preverimo njihovo prisotnost in tehnično skladnost. Integracije z večjim povpraševanjem imajo ponavadi tesnejše roke dostave, kjer prav tako moramo biti pragmatični.
Celoviti test integracije kot ga prikazuje slika 4, se teoretično ne zaključi in je živ, dokler jo ponujamo uporabnikom. Zelo pomemben del je spremljanje novosti, kar pomeni preverjanje objav ponudnikov za večjimi spremembami, ki bi lahko vplivale na našo integracijo, ali nenazadnje o novih metrikah, ki bi jih lahko ponudili. Za vsako integracijo imamo vzpostavljen kanal, kamor prihajajo takšne novosti (ponavadi je to RSS naročnina, ali pa vsaj prijava na elektronske novice). QA te kanale redno spremlja in potencialne spremembe skomunicira z vodjem integracijskega oddelka.



Slika 4: Celoviti test, ki se začne z raziskavo, in živi dokler je integracija omogočena v Databox.

4 Tehnični pristopi in avtomatizacija

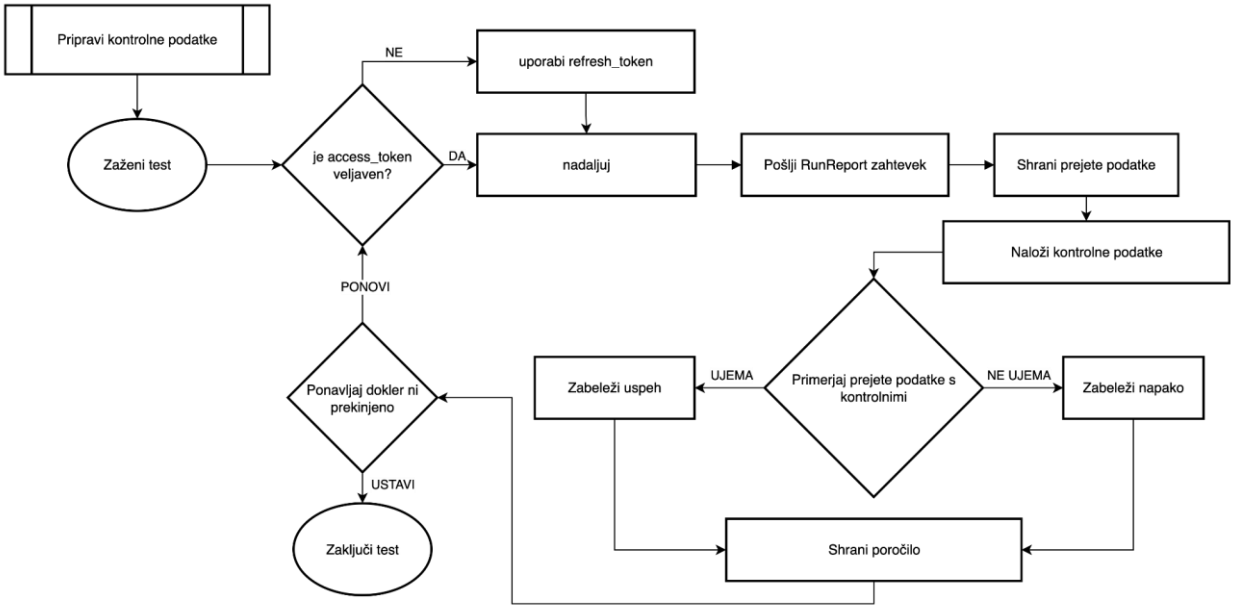
Za učinkovito zagotavljanje kakovosti je dobro, da je QA dovolj tehnično podkovan, saj lahko le tako celostno pripomore k razvoju. Razvili smo interno ogrodje za preverjanje podatkov, ki je namenjeno razvijalcem integracij, ob enem širimo pokritost testnih primerov z avtomatskimi testi, in sledimo AI trendom.

4.1. Interno ogrodje za preverjanje podatkov ITF

QA je zasnoval ogrodje za preverjanje točnosti in pravilnosti podatkov (Integration Testing Framework, ITF). Gre za dodano funkcionalnost, ki je pogojno vključena v poslovno logiko pridobivanja in priprave podatkov. Deluje tako, da se najprej posname poizvedba na API, nato pa se sestavi test enot (*unit test*), ki žive podatke primerja s posnetkom. Test se požene po vsaki večji spremembi, praviloma na strani razvijalca, in je vključen tudi v proces neprekinjene integracije in dostave (*CI/CD*). Tu se testi sprožijo samodejno in omogočajo hitro zaznavo nepričakovanih sprememb v metrikah. Ogrodje zagotavlja, da poseg v poslovno logiko ni vplival na točnost in pravilnost podatkov. Njegov cilj je razvijalcem omogočiti preverjanje ustreznosti kode že med implementacijo, zmanjšati odvisnost od ekipe za zagotavljanje kakovosti ter povečati zaupanje v stabilnost obsežnejših sprememb. Ob enem pa je bil namen tudi zmanjšati obseg končnega dela za QA, ter pohitriti izdajo. Uporaba se je izkazala za učinkovito, predvsem pri ponudnikih z velikim številom metrik, kot je Google Analytics. Prototip in znanje je QA predal integracijskemu oddelku, ki je ogrodje uspešno vkomponiral v svoj razvojni cikel.

4.2. Avtomatski testi

Prve testne primere smo avtomatizirali z orodjem Katalon Studio [3], ki je omogočalo relativno dobro izvedbo. Kljub temu smo se odločili za prehod na ogrodje Playwright [4], saj bolje ustreza našim potrebam po *zmogljivosti*, *hitrosti izvajanja*, *prijaznosti razvijalcem* in podpori več programskih jezikov. Playwright omogoča učinkovite teste tipa konec-konec (ang. *end to end, E2E*) v modernih brskalnikih, ima veliko razvojno skupnost in je zaenkrat brezplačen. Implikacijo v še eno ogrodje, namenjeno tudi razvijalcem, predstavlja naš prispevek iz leta 2023 [5].
Vedno več virov usmerjamo v avtomatizacijo testnih primerov v *Playwright*, saj lahko le tako ohranjamo ali celo zvišujemo trend dostave v hitrosti in kvaliteti. Slika 5 prikazuje avtomatiziran tok testiranja integracije GA4 z uporabo orodja Playwright. Test vključuje preverjanje avtorizacije, zajem podatkov in validacijo rezultatov v primerjavi s pričakovanimi vrednostmi. Kontrolni podatki se pripravijo ročno. Vstop v test je lahko enkratni ročni zagon, ali pa akcija na *CI/CD*. Najprej se preveri veljavnost žetona, nato se pridobi podatke in se jih preveri.



Slika 5: diagram poteka avtomatiziranega testa za GA4, ki pridobi in preveri podatke.

Zraven Playwright oddelki ali posamezni inženirji uporabljajo še druge tehnologije. Iz vidika proaktivnosti, kreativnosti in samostojnosti je to zelo dobrodošlo, se pa QA vseeno zavzema za standardizacijo postopkov in orodij, vzpodbuja k deljenju znanja in nestandardnih primerov uporabe. Tak primer je uporaba ogrodja za obremenitveno testiranje *K6* [6], ki počasi postaja formalizirana, saj bo koristna predvsem za potrebe zagotavljanja konsistence pri obravnavi velike količine podatkov.

4.3. Umetna inteligenca

Umetna inteligenca (ang. *Artificial Intelligence, AI*) je zagotovo nepogrešljiva in z velikim potencialom, tudi na področju QA. Vzpodbujamo celosten AI pristop, kar pomeni da ga želimo vključevati na vseh nivojih dela. Od priprave testnih planov, generiranja testnih primerov, do preoblikovanja in izboljšav testnih ogrodij in izdelave virtualnih pomočnikov (ang. *bot*). Predstavili bomo dva.

Izdelava testnih primerov in avtomatskih testov s pomočjo AI

Vsebine pomoči na Databox so obsežne, predvsem za integracije, ki jih ponujamo uporabnikom. To predstavlja ogromno količino znanja, ki ga lahko posredujemo AI in relativno hitro dosežemo visoko stopnjo pokritosti. Z ustreznim zahtevkom (ang. *prompt*) ki zajema referenco do naših vsebin, smo npr. v kratkem času dobili testne scenarije, ki smo jih lako dalje pretvorili v uvozno datoteko, katero smo naložili v sistem za upravljanje s testi (ang. *Test management tool*). Iz predlaganih testnih korakov nam je AI pomagal tudi pri generiranju avtomatskih testov na različnih stopnjah kvalitete integracije. Pri tem je seveda ključno zavedanje, da se tudi AI lahko zmoti, zato je bila procesna umestitev preverjanja ustreznosti generiranih rezultatov nujna. S tem ja QA postal veliko hitrejši in agilnejši.

Nastavitev in treniranje virtualnega QA pomočnika za testiranje integracij

Dodaten vir znanja, ki ga lahko predamo AI je javna dokumentacija ponudnikov, koda naših implementacij, in nenazadnje naše dosedanje izkušnje. S tem izhodiščem smo ustvarili virtualnega pomočnika (ang. *AI bot*), ki ga sproti učimo z vodenjem in dodajanjem ključnih informacij. Ta pomočnik je nepogrešljiv sogovornik pri zagotavljanju kvalitete novih integracij, ali testiranju sprememb na starejših. Pomagal je npr. identificirati neočitne pasti, opozoriti na prihajajoče večje spremembe API-ja, in preverjati ustreznost in točnost podatkov. Prednost

pristopa s pomočnikom je ta, da je na voljo ostalim članom, se sproti uči, in lahko dobi res širok spekter znanja. Med tem, ko smo spet pridobili na hitrosti in učinkovitosti, seveda ne smemo pozabiti na preverjanje generiranih rezultatov.

5 Vpliv kakovosti na uporabniško izkušnjo

Ko uporabnik pride k nam, ga želimo obdržati, zato je ključna dovolj dobra uporabniška izkušnja (ang. *UX*). Ogromni sistemi, predvsem v domeni poslovne inteligence (*ang. Business Intelligence - BI*), so kompleksni že sami po sebi. Slaba uporabniška izkušnja v obdobju, ko si posameznik ali podjetje ne more privoščiti dolgotrajnega uvajanja in čakati na odpravljene napake, negativno vpliva na uporabniški cikel in rast podjetja. Pri konkurenci opazimo veliko slabih praks. Tudi dobre so, katere želimo posnemati, ali biti še boljši.

Primeri slabih izkušenj pri nas niso pogosti, pa vseeno niso zanemarljivi. Na primer ponavljajoče večje odstopanje pri podatkih uporabnika, kritične napake v času predstavitve partnerjem, prekinjene povezave, izdaje z napakami, … zagotovo ne vplivajo dobro na zadovoljstvo naših uporabnikov.

Slabe izkušnje poskušamo omiliti z dobrimi praksami in aktivnim odnosom do uporabnika. Zato komuniciramo z uporabnikom, ki ima težave, poskušamo razumeti njegovo frustracijo, mu vsebinsko pomagati, in kar je najpomembneje, ga poskušamo obdržati, ko zaznamo da želi oditi. Na primer, pri integraciji z Facebook (Meta) API-jem smo uporabniku v manj kot dveh urah posredovali razlago napake, popravek pa je bil vključen v naslednjo izdajo.

QA ima tudi tukaj pomembno vlogo. Sodeluje npr. pri iskanju ponovitve redke napake pri veliki stranki, z hitrim odzivom doprinese k hitri in kvalitetni izdaji popravkov, celostno prispeva k razvoju procesov in ogrodij, ki dolgoročno izboljšujejo uporabniško izkušnjo.

6 Zaključek

V prispevku smo predstavili tehnične in procesne pristope zagotavljanja kakovosti pri povezovanju z zunanjimi viri podatkov v sistemu Databox. Poudarili smo pomen razumevanja raznolikosti API-jev, potrebe po semantični skladnosti podatkov in vlogo QA pri celotnem življenjskem ciklu integracije – od načrtovanja do vzdrževanja.

Naši rezultati kažejo, da sodelovanje med razvojem in QA bistveno izboljša učinkovitost, zanesljivost in hitrost integracij. Posebej se je izkazalo, da lahko z notranjimi ogrodji (kot je ITF) ter avtomatskimi testi razvijalci že v zgodnji fazi validirajo pravilnost svoje implementacije, kar razbremeni QA in skrajša čas do izdaje.

V prihodnosti želimo nadgraditi vpeljane prakse z večjo stopnjo standardizacije in poglobljeno uporabo umetne inteligence – tako pri generiranju testnih primerov kot pri odkrivanju sprememb. Želimo tudi nadaljevati z odpiranjem QA orodij širšemu razvojnemu ekosistemu in z boljšim povezovanjem uporabniške povratne informacije v proces kakovosti. Verjamemo, da lahko predstavljene izkušnje in pristopi služijo kot vzorčni model tudi za druge sisteme, ki se soočajo z izzivi integracije podatkov iz zunanjih virov.

Literatura

[1] Databox, Our methods and challenges of API integration, https://databox.com/our-methods-and-challenges-of-api-integration, obiskano julij 2025.

[2] https://www.postman.com/, Postman, obiskano julij 2025

[3] https://katalon.com/, Katalon, obiskano julij 2025

[4] https://playwright.dev/, Microsoft, obiskano julij 2025

[5] OTS 2023 Sodobne informacijske tehnologije in storitve: Zbornik 26. konference, Testno ogrodje za razvijalce - ali kako doseči, da razvijalci celovito testirajo, Mitja Krajnc, Tadej Ciglič, Boris Ovčjak, Databox

[6] https://k6.io/, Grafana Labs, obiskano julij 2025

Avtomatizirani testi uporabniških vmesnikov ogrodja .NET MAUI

Alen Granda

Alcad d.o.o., Slovenska Bistrica, Slovenija
alen.granda@alcad.si

V razvoju mobilnih aplikacij postaja zagotavljanje stabilnosti in pravilnega delovanja uporabniškega vmesnika vse pomembnejše, še posebej pri večplatformskih rešitvah, kot to ponuja ogrodje .NET MAUI. Ročno preverjanje delovanja aplikacij je zamudno, ponovljivost testov je omejena. Zato predstavlja avtomatizacija testiranja uporabniškega vmesnika ključen korak k večji zanesljivosti in učinkovitemu razvojnemu procesu. Prispevek obravnava celovit postopek vzpostavitve avtomatskih testov uporabniških vmesnikov za aplikacijo, razvito znotraj ogrodja .NET MAUI. Prikazan je razvoj testov z uporabo knjižnice Appium, priprava platformno specifične kode za operacijski sistem Android ter vključitev celotnega postopka v cevovod neprekinjene integracije in postavitve na platformi GitLab. Poseben poudarek je na izvajanju testov znotraj vsebnika, ki teče v operacijskem sistemu Linux. Podan je tudi opis vseh korakov, vključno z zagonom virtualne naprave, vzpostavitvijo strežnika Appium in izvedbe testov. Prispevek se zaključi s pregledom ključnih prednosti in slabosti pristopa ter ponudi smernice za nadaljnjo optimizacijo tovrstnih sistemov testiranja.

Ključne besede:
.NET MAUI,
CI/CD,
zagotavljanje kakovosti,
UI testi,
Appium.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.20

1 Uvod

Zaradi naraščajočih zahtev po večplatformskih aplikacijah, ki morajo delovati zanesljivo tako na mobilnih napravah kot na namiznih računalnikih, se razvojni procesi vse bolj osredotočajo na avtomatizacijo in integracijo kakovostnih mehanizmov testiranja. Ogrodje .NET MAUI [2] razvijalcem omogoča razvoj enotne programske kode za različne platforme, kot so Android, iOS, Windows in macOS. Vendar ta večplatformskost predstavlja izziv tudi pri testiranju uporabniških vmesnikov UI (ang. User interface), ki so pogosto podvrženi napakam, povezanih z vizualnimi elementi, postavitvami in interakcijami [3].

Ročno testiranje aplikacij hitro postane neučinkovito in nezanesljivo, še posebej pri hitrem razvojnem ciklu in pogostih spremembah uporabniškega vmesnika [4]. Pogoste spremembe, ki vplivajo na logiko ali izgled vmesnika, lahko hitro povzročijo napake, ki jih je brez avtomatiziranih testov težko pravočasno zaznati. Zato je avtomatizacija testiranja ključna za zagotavljanje kakovosti in stabilnosti končnih rešitev. V prispevku predstavimo, kako smo v projekt ogrodja .NET MAUI vpeljali avtomatizirane teste uporabniškega vmesnika z uporabo knjižnice Appium [6] ter jih vključili v proces neprekinjene integracije CI (ang. Continuous integration) s pomočjo platforme GitLab [7]. Testi so zasnovani tako, da omogočajo ponovljivo preverjanje funkcionalnosti aplikacije v različnih razvojnih fazah, kar dolgoročno izboljšuje zanesljivost produkta [5].

Poseben poudarek namenjamo zasnovi in vzpostavitvi cevovoda CI, ki izvaja teste znotraj vsebnika (ang. Container). Takšna zasnova omogoča ponovljivost, razširljivost in enostavnejše upravljanje testnega okolja ter zmanjšuje odvisnost od ročnega nastavljanja okolij na različnih napravah ali agentih. V našem primeru smo se za namen izvedbe testov osredotočili izključno na platformo Android, saj je to trenutno edina podprta mobilna platforma, ki jo podjetje uporablja v produkciji. Testiranje drugih platform, kot sta iOS ali Windows, v tej fazi razvoja ni prednostno, kar je vplivalo tudi na tehnično poenostavitev celotnega procesa.

Prispevek je strukturiran tako, da bralca najprej seznani z uporabljenimi tehnologijami, kot so .NET MAUI, Appium, Docker in GitLab. V nadaljevanju postopoma predstavimo pripravo osnovnega uporabniškega testa v okolju .NET MAUI, vključno z uporabo platformno specifične kode za Android. Sledi podroben opis, kako testno okolje prenesemo v vsebnik, ki omogoča izolirano izvajanje testov na operacijskem sistemu Linux. Sledi integracija avtomatiziranega zagona testov v cevovod CI, kjer so opisani vsi potrebni koraki za zagon virtualne naprave Android, strežnika Appium, namestitve aplikacije in izvajanja testov. Prispevek se zaključi z analizo prednosti in slabosti predstavljenega pristopa ter s kratkim povzetkom ključnih ugotovitev.

2 Uporabljene tehnologije

Za vzpostavitev avtomatiziranega testiranja uporabniškega vmesnika v okolju .NET MAUI smo uporabili tri ključne tehnologije: ogrodje .NET MAUI [2] za razvoj aplikacije, knjižnico Appium [6] za avtomatizacijo testov uporabniškega vmesnika ter platformo GitLab [7] za upravljanje z različicami kode in izvajanje procesov neprekinjene integracije in postavitve CI/CD. Vsaka izmed teh tehnologij ima pomembno vlogo pri zagotavljanju stabilnega in ponovljivega testnega okolja.

.NET MAUI

Ogrodje .NET MAUI [2] je ogrodje podjetja Microsoft za razvoj večplatformskih aplikacij z enotno programsko kodo, katerih zgrajena koda teče na operacijskih sistemih Android, iOS, Windows in macOS. Predstavlja naslednika ogrodja Xamarin.Forms in temelji na tehnologiji .NET 6+.

Njegova glavna prednost je možnost deljenja večino izvorne kode med vsemi podprtimi platformami, kar zmanjšuje kompleksnost razvoja in omogoča hitrejšo dostavo funkcionalnosti. Poleg tega se promovira uporaba arhitekture MVVM [13] (ang. Model-View-ViewModel), ki olajša ločevanje predstavitvene plasti od poslovne logike ter posledično izboljša berljivost, vzdržljivost in testabilnost programske kode. Arhitektura MVVM omogoča strukturirano gradnjo aplikacije, kar je ključnega pomena za dolgoročno vzdrževanje in širitev projekta.

V okviru prispevka smo pripravili projekt za testiranje programske kode znotraj rešitve, ki služi kot predloga za vse projekte v podjetju. Rešitev vključuje osnovno strukturo aplikacije z definiranimi sloji uporabniškega vmesnika in poslovne logike, kar omogoča lažjo ponovljivost in standardizacijo razvoja. Nadgradili smo jo z avtomatiziranimi testi uporabniškega vmesnika in jih integrirali v proces neprekinjene integracije in postavitve CI/CD, kar omogoča avtomatsko potrjevanje funkcionalnosti ob vsaki spremembi izvorne kode. V našem primeru smo se osredotočili izključno na platformo Android, saj je fokus podjetja osredotočen le na izbran operacijski sistem.

Appium

Appium [6] je odprtokodno orodje za avtomatizacijo testiranja uporabniških vmesnikov mobilnih aplikacij. Temelji na standardu WebDriver [8] in omogoča izvajanje testov brez potrebe po spremembah izvorne programske kode aplikacije.

Appium podpira več programskih jezikov (npr. C#, Java, Python, JavaScript) in omogoča testiranje aplikacij na platformah Android in iOS, kar ga naredi izjemno prilagodljivega za uporabo v večplatformskih okoljih. V ozadju Appium deluje kot strežnik, ki posluša zahteve testnega odjemalca z uporabo protokola WebDriver in jih usmerja na ustrezen gonilnik – npr. gonilnik UiAutomator2 za Android ali gonilnik XCUITest za operacijski sistem iOS [8]. Gonilnik nato komunicira z mobilno napravo ali virtualno napravo in izvaja dejanske ukaze, kot so:

- pritisk na gumb,
- vnos besedila v polja,
- preverjanje vidnosti elementov,
- pomikanje po zaslonu,
- čakanje na prisotnost določenih elementov ipd.

Ta arhitektura omogoča, da Appium deluje kot posrednik med testno logiko, ki jo pišemo v domenskem programskem jeziku, kot je C#, in napravo ali virtualno napravo, na katerem teče aplikacija. Appium torej simulira resničnega uporabnika – klikne gumbe, vnese besedilo, preverja postavitve in odzive elementov uporabniškega vmesnika. To omogoča preverjanje funkcionalnosti na način, ki je najbližji dejanskemu uporabniškemu scenariju, kar je ključnega pomena za zagotavljanje kakovostne uporabniške izkušnje.

V našem primeru Appium deluje v povezavi z virtualno napravo Android, kjer izvaja avtomatizirane teste UI za aplikacijo, razvito znotraj ogrodja .NET MAUI. V kombinaciji s funkcionalnostjo platforme GitLab smo Appium integrirali v avtomatiziran testni cevovod, kjer se testi UI sprožijo samodejno po vsaki združitvi vej programske kode ali pred izdajo nove različice aplikacije.

GitLab

GitLab [7] je celovita platforma, ki združuje nadzor nad izvorno programsko kodo, sledenje napakam in močan sistem za izvajanje procesov CI/CD. Ključna prednost platforme je integrirani cevovod CI/CD, ki omogoča samodejno izvajanje testov, gradnjo in nameščanje aplikacij ob vsaki spremembi kode.

V našem primeru smo GitLab uporabili za avtomatizacijo postopka testiranja, kjer se ob vsaki združitvi glavnih vej programske kode sproži cevovod, ki v vsebniku zažene testni strežnik Appium in izvede vnaprej definirane teste UI. S tem zagotovimo, da se morebitne napake odkrijejo zgodaj in da testno okolje ostaja ponovljivo in neodvisno od posameznih razvijalcev. Z uporabo funkcionalnosti GitLab CI/CD lahko tako dosežemo višjo stopnjo avtomatizacije, hitrejšo povratno informacijo po spremembah ter boljšo sledljivost napak znotraj razvojnega procesa.

3 Zasnova zagona testov znotraj vsebnika

Za namen avtomatizacije testiranja uporabniških vmesnikov, zgrajenih v ogrodju .NET MAUI, smo pripravili lastno sliko Docker (ang. Docker image), zasnovano kot nadgradnja slike, opisane v [9]. Slika temelji na osnovni sliki podjetja Microsoft, ki vsebuje vse potrebne gradnike za gradnjo, testiranje in objavo projektov .NET. Osnovna slika vključuje orodja, kot so komplet za razvoj programske opreme .NET (ang. .NET SDK), komplet za razvoj programske opreme operacijskega sistema Android (ang. Android SDK) ter gradbene komponente za aplikacije Android, kar omogoča celovit razvoj znotraj vsebniškega okolja.

V našem primeru smo sliko nadgradili z dodatnimi komponentami, ki so ključne za izvajanje testov UI:

- Tehnologija NodeJS [10] – nameščena je bila za delovanje strežnika Appium. Naložena je bila trenutno zadnja stabilna verzija 23.
- Knjižnica Appium – nameščena je bila z uporabo upravitelja paketov npm, saj Appium deluje kot aplikacija NodeJS. Dodatno smo namestili še gonilnik appium-uiautomator2-driver [12], ki je potreben za testiranje naprav Android.
- Virtualna naprava operacijskega sistema Android AVD (ang. Android virtual device), ki omogoča izvajanje aplikacij v virtualnem okolju. Uporabili smo vmesnik za programiranje aplikacij operacijskega sistema Android verzije 34, naprava je bila Pixel 5 [11].

Za izvajanje testov smo pripravili vsebnik na podlagi opisane slike in zasnovali postopek, ki se izvede znotraj tega okolja. Potrebno je bilo izvesti več korakov, ki so opisani v nadaljevanju.

3.1. Priprava testov v ogrodju .NET MAUI

Prvi korak v procesu avtomatiziranega testiranja je priprava same testne kode, ki se izvaja znotraj vsebnika. Za ta namen smo uporabili ogrodje za pisanje testov v jeziku C# v kombinaciji s knjižnico Appium. Naša osnovna testna logika je zapisana v razredu UITest1, ki vsebuje metodo AppLaunches. Ta metoda poskrbi, da se ob zagonu aplikacije ustvari zaslonski posnetek zaslona naprave (Slika 1). Test uporabi metodo GetScreenshot() za zajem trenutnega stanja zaslona naprave in ga shrani kot datoteko s formatom PNG, kar omogoča vizualno potrditev, da je aplikacija uspešno zagnana.

Za uspešno izvajanje testov na virtualni napravi operacijskega sistema Android smo pripravili platformno-specifično konfiguracijo, ki inicializira okolje Appium in vzpostavi povezavo z virtualno napravo. To konfiguracijo smo zapisali v razred AppiumSetup, ki vsebuje metodi, ki se izvedeta

```
12  public class UITest1 : BaseTest
13  {
14      [Test]
        1 reference
15      public void AppLaunches()
16      {
17          App.GetScreenshot().SaveAsFile($"{nameof(AppLaunches)}.png");
18      }
19  }
```

Slika 1: Demonstracijska testna metoda ob zagonu aplikacije .NET MAUI.

enkrat pred začetkom vseh testov in enkrat po koncu (Slika 2).

Ta nastavitev omogoča Appiumu, da:

- vzpostavi povezavo z lokalnim strežnikom Appium,
- uporablja gonilnik UIAutomator2, ki je priporočeni gonilnik za operacijski sistem Android,
- prepozna aplikacijo po paketu (AppPackage) in začetni aktivnosti (AppActivity),
- samodejno zažene virtualno napravo z vnaprej definiranim profilom AVD,
- obdrži stanje aplikacije s parametrom NoReset, da se izogne odstranitvi knjižnic za hitro objavo paketa.

Kot lahko opazimo, smo v nastavitvah povečali časovno omejitev zagona aplikacije iz privzetih 60 sekund na kar 6 minut. Do te spremembe je prišlo, ker je postopek inicializacije okolja Appium, zagona virtualne naprave in vzpostavitve povezave z aplikacijo izredno dolgotrajen – še posebej, kadar se izvaja na računalnikih z omejenimi strojno-programskimi zmogljivostmi. V praksi se je izkazalo, da lahko celoten proces od zagona vsebnika do zaključka testov traja tudi 30 minut. To predstavlja eno ključnih slabosti poganjanja testov UI znotraj vsebnikov, saj je postopek še vedno neoptimiziran, strojno zahteven in časovno potraten. Zaradi teh razlogov tovrstno testiranje trenutno ni primerno za hitre in pogoste iteracije, kot jih zahtevajo moderne prakse CI/CD.

```
9   [SetUpFixture]
    1 reference
10  public class AppiumSetup
11  {
        3 references
12      private static AppiumDriver? driver;
13
        1 reference
14      public static AppiumDriver App => driver ?? throw new NullReferenceException("AppiumDriver is null");
15
16      [OneTimeSetUp]
        0 references
17      public void RunBeforeAnyTests()
18      {
19          var androidOptions = new AppiumOptions
20          {
21              AutomationName = "UIAutomator2",
22              PlatformName = "Android",
23          };
24
25          androidOptions.AddAdditionalAppiumOption(MobileCapabilityType.NoReset, "true");
26          androidOptions.AddAdditionalAppiumOption(AndroidMobileCapabilityType.AppPackage, "com.impol.mauitemplate");
27          androidOptions.AddAdditionalAppiumOption(AndroidMobileCapabilityType.AppActivity, $"com.impol.mauitemplate.MainActivity");
28          androidOptions.AddAdditionalAppiumOption("uiautomator2ServerLaunchTimeout", 360000);
29          androidOptions.AddAdditionalAppiumOption("uiautomator2ServerInstallTimeout", 360000);
30          androidOptions.AddAdditionalAppiumOption("avd", "Pixel_5_API_34");
31
32          driver = new AndroidDriver(new Uri("http://127.0.0.1:4723/"), androidOptions, TimeSpan.FromMinutes(10));
33      }
34
35      [OneTimeTearDown]
        0 references
36      public void RunAfterAnyTests()
37      {
38          driver?.Quit();
39      }
40  }
```

Slika 2: Domensko specifična konfiguracija sistema Appium.

Poleg zgornjih nastavitev smo v glavni aktivnosti aplikacije Android MainActivity.cs dodali naslednji atribut:

[Register("com.app.mauitemplate.MainActivity")]

Ta atribut zagotovi, da je razred MainActivity pravilno registriran znotraj zagonske datoteke aplikacije in da ga Appium lahko najde ter zažene. Brez tega atributa Appium izpiše napako, da zagonska aktivnost ne obstaja ali je ne more zagnati, kar bi preprečilo uspešno inicializacijo testnega okolja.

3.2. Zagon strežnika Appium

Programska koda je pripravljena, sedaj sledi opis postopka zagona testov znotraj vsebnika. Najprej smo zagnali strežnik Appium z uporabo ukaza:

appium --log-level error

Appium je poslušal na privzetih vratih protokola TCP 4723 in čakal na povezavo z odjemalcem (t.j. testnim projektom .NET MAUI). Le-ta mu ob zagonu testov pošilja ukaze za izvajanje testov nad virtualno napravo.

3.3. Zagon virtualne naprave Android

Da bi bilo okolje primerno za avtomatsko izvajanje v cevovodu CI/CD, kjer ni grafičnega uporabniškega vmesnika, smo virtualno napravo zagnali v t. i. načinu brez glave (ang. Headless). Za ta namen smo uporabili naslednji ukaz:

emulator -avd "Pixel_5_API_34" -gpu off -noaudio -no-boot-anim -netdelay none -no-window -no-snapshot -memory 4096 -partition-size 4096 -no-accel

Pojasnilo parametrov:

- -avd "Pixel_5_API_34" – ime profila virtualne naprave AVD (Pixel 5 z API 34), ki je bil prej konfiguriran in registriran.
- -gpu off – izklop grafičnega pospeševanja, saj ni potrebe po tej funkcionalnosti v načinu brez glave.
- -noaudio – onemogoči zvok v virtualni napravi.
- -no-boot-anim – preskoči animacijo ob zagonu naprave, kar pohitri zagon.
- -netdelay none – izklopi simulacijo mrežne zakasnitve.
- -no-window – zažene virtualno napravo brez okna, v ozadju.
- -no-snapshot – onemogoči uporabo posnetkov virtualne naprave za večjo stabilnost.
- -memory 4096 – dodeli 4 GB pomnilnika za boljšo zmogljivost.
- -partition-size 4096 – dodeli 4 GB prostora za virtualni disk, da se preprečijo napake ob namestitvi aplikacij.
- -no-accel – onemogoči strojno pospeševanje, saj v nekaterih vsebnikih to ni na voljo.

Izbrana konfiguracija omogoča zanesljiv in hitrejši zagon virtualne naprave znotraj vsebnika, kar je ključno za ponovljivo testno okolje.

3.4. Namestitev in zagon aplikacije na virtualni napravi

Preden smo lahko zagnali teste UI z ukazom dotnet test, smo morali najprej zagnati aplikacijo na virtualni napravi. Če tega koraka ne izvedemo, se pri zagonu testov pojavi napaka:

Activity name '.com.app.mauitemplate.MainActivity' used to start the app doesn't exist or cannot be launched! Make sure it exists and is a launchable activity.

Do te napake pride, ker virtualna naprava ne vsebuje nobene aplikacije, ki bi jo Appium lahko zagnal ali nadziral. Zato smo aplikacijo ročno namestili in zagnali s pomočjo ukaza:

```
dotnet build -t:Run -f net8.0-android /p:AndroidDevice=Pixel_5_API_34
App.MauiTemplate/App.MauiTemplate.csproj
```

S tem smo zagotovili, da je aplikacija dejansko nameščena in zagnana na virtualni napravi, kar je pogoj za uspešen zagon testov UI.

3.5. Zagon testov

Po zagonu virtualne naprave, strežnika Appium in uspešnem zagonu aplikacije na napravi, smo lahko iz konzole sprožili zagon testov z ukazom:

```
dotnet test
```

Testi so se z uporabo knjižnice Appium povezali z virtualno napravo in izvedli vse definirane korake. Rezultati testov so se prikazali v konzoli.

4 Avtomatizacija testov uporabniških vmesnikov

Postopek zagona testov uporabniškega vmesnika, kot je bil opisan v prejšnjem poglavju, smo integrirali v cevovod CI platforme GitLab z namenom doseči popolno avtomatizacijo preverjanja delovanja aplikacije. V okviru obstoječega cevovoda smo dodali namenski postopek (ang. job), ki skrbi za pripravo okolja in izvedbo testov znotraj vsebnika. Cevovod sledi naslednjim korakom (Slika 3):

- Zažene se virtualna naprava Android v načinu brez glave.
- Počaka se na popoln zagon virtualne naprave z večkratnim preverjanjem lastnosti sys.boot_completed.
- Ko je virtualna naprava pripravljena, se v ozadju zažene strežnik Appium.
- Aplikacija .NET MAUI se zgradi in zažene na virtualno napravo.
- V zadnjem koraku se izvedejo avtomatizirani testi UI z uporabo ukaza dotnet test.

Testi so zasnovani tako, da se po potrebi poženejo do trikrat zapored. Razlog za to odločitev je v tem, da se v praksi pogosto zgodi, da prva iteracija testov odpove zaradi prehitrega zagona aplikacije ali strežnika Appium, preden se virtualna naprava v celoti vzpostavi in stabilizira. V takem primeru ni smiselno takoj označiti testov kot neuspešne, saj je lahko razlog zgolj v časovni nedoslednosti priprave okolja. Z dodatnimi poskusi zmanjšamo število lažno negativnih rezultatov in s tem povečamo zanesljivost testnega procesa.
Pomembno je poudariti, da gre za precej kompleksen in časovno zahteven proces. Četudi se vsi koraki izvedejo uspešno, lahko en sam zagon testov traja tudi od 20 do 30 minut. Glavni razlogi za dolgotrajnost so:

- počasna vzpostavitev virtualne naprave v virtualiziranem okolju brez strojne pospešitve,
- zagon strežnika Appium in vzpostavitev povezave z napravo,
- grajenje aplikacije in njen prenos na virtualno napravo,
- morebitni vmesni izpadi ali napake, ki sprožijo dodatne poskuse izvajanja testov.

Takšna arhitektura za avtomatizacijo testov UI sicer zagotavlja zanesljivo preverjanje aplikacije v kontroliranem okolju, vendar ni optimalna za hitro iterativno razvijanje. Zaradi njene počasnosti in porabe virov predstavlja ta pristop predvsem osnovo za občasno preverjanje stabilnosti uporabniškega vmesnika, ne pa za vsakodnevno uporabo znotraj intenzivnega cikla cevovoda CI/CD.

5 Zaključek

V prispevku smo predstavili celoten postopek priprave avtomatiziranega testiranja uporabniškega vmesnika za mobilno aplikacijo, razvito na podlagi ogrodja .NET MAUI. Najprej smo pripravili osnovne teste UI s pomočjo knjižnice Appium, kjer smo ob zagonu aplikacije posneli zaslonski posnetek naprave. V nadaljevanju smo opisali, kako je bilo potrebno za delovanje testov implementirati tudi platformno specifično kodo za operacijski sistem Android, vključno z nastavitvijo gonilnika ter ustrezno označiti glavno aktivnost v aplikaciji. Poseben poudarek smo namenili selitvi testnega okolja v vsebnik, ki teče na operacijskem sistemu Linux. Vse skupaj smo vključili v cevovod platforme GitLab, kjer se virtualna naprava, strežnik Appium in sama aplikacija samodejno vzpostavijo ter se poskrbi za izvedbo testov z možnostjo ponovnega zagona v primeru neuspeha. S tem pristopom smo dosegli popolno avtomatizacijo testiranja v nadzorovanem okolju.

Med ključne prednosti pripravljene rešitve štejemo predvsem njeno avtomatizacijo, saj se celoten postopek odvije brez ročnega posredovanja. Zagon testov v vsebniku omogoča neodvisnost od operacijskega sistema, kar pomeni, da lahko teste izvajamo kjerkoli, kjer imamo na voljo podporo k tehnologiji Docker. Poleg tega takšna rešitev nudi ponovljivost rezultatov, saj je vsaka testna seja zagnana v svežem okolju, kar zmanjšuje vpliv zunanjih dejavnikov. Postopek predstavlja tudi dobro osnovo za nadaljnje razširitve, kot so testiranje različnih scenarijev, platform ali vključitev poročanja o pokritosti.

Po drugi strani ima takšen pristop tudi svoje slabosti. Izvajanje testov na virtualni napravi Android v glavi cevovoda CI je zelo časovno zahtevno – posamezna seja lahko traja tudi 30 minut, kar je za hitro iteracijo v razvojnem ciklu pogosto predolgo. Dodatno kompleksnost prinaša konfiguracija okolja. Pravilna nastavitev virtualne naprave, strežnika Appium in aplikacije zahteva precej tehničnega znanja in natančnosti. Delovanje virtualne naprave brez strojnega pospeševanja je počasno, kar še dodatno zmanjšuje učinkovitost in odzivnost testov.

Kljub navedenim izzivom predstavlja pripravljena rešitev učinkovit temelj za avtomatizirano testiranje aplikacij .NET MAUI. Omogoča zgodnje zaznavanje napak v uporabniškem vmesniku in pripomore k višji kakovosti končnega izdelka. V prihodnosti bi bilo smiselno nadalje optimizirati hitrost zagona, razdeliti teste glede na prioriteto ter poiskati možnosti za vzporedno izvajanje testov na različnih napravah.

```
.ui_test:
  script:

  - echo "Waiting for emulator to boot..."
  - |
    for i in {1..30}; do
      if adb shell getprop sys.boot_completed | grep -m 1 "1"; then
        echo "Emulator booted."
        break
      else
        echo "Waiting for emulator... ($i)"
        sleep 10
      fi
    done

  - echo "Waiting extra time for emulator to stabilize..."
  - sleep 10

  - echo "Starting Appium..."
  - nohup appium > appium.log 2>&1 &

  - sleep 10

  - echo "Building and running .NET MAUI app..."
  - dotnet build -t:Run -f net8.0-android /p:AndroidDevice=Pixel_5_API_34 MauiTemplate/MauiTemplate.csproj

  - echo "Running tests with up to 3 retries if needed..."
  - |
    attempt=1
    max_attempts=3      Alen Granda, yesterday • SAP-19590 -Try with demo pipeline
    while [ $attempt -le $max_attempts ]; do
      echo "Test attempt #$attempt"
      dotnet test --logger "trx;LogFileName=test_results_$attempt.trx"
      exit_code=$?

      if [ $exit_code -eq 0 ]; then
        echo "Tests passed on attempt #$attempt"
        break
      else
        echo "Tests failed on attempt #$attempt"
      fi

      if [ $attempt -eq $max_attempts ]; then
        echo "All $max_attempts attempts failed. Exiting with failure."
        exit 1
      fi

      attempt=$((attempt + 1))
      echo "Retrying tests in 15 seconds..."
      sleep 15
    done
```

Slika 3: Postopek cevovoda v platformi GitLab, namenjen izvedbi testov uporabniškega vmesnika.

Literatura

[1] https://medium.com/innovies-club/running-android-emulator-in-a-docker-container-19ecb68e1909, Running Android Emulator in a Docker Container, Salem Amr, obiskano 3. 7. 2025.

[2] https://dotnet.microsoft.com/en-us/apps/maui, .NET Multi-platform App UI, obiskano 3. 7. 2025.

[3] https://www.lambdatest.com/blog/visual-testing-challenges, The Challenges of Visual Testing [With Solutions], Gazala Saniya, obiskano 3. 7. 2025.

[4] https://muuktest.com/blog/manual-software-testing, Manual Software Testing: A Practical Guide, The MuukTest Team, obiskano 3. 7. 2025.

[5] https://www.testim.io/blog/test-automation-benefits/, Benefits of Test Automation: A Complete Guide, obiskano 3.7. 2025.

[6] https://appium.io/docs/en/latest/, Appium, obiskano 3. 7. 2025.

[7] https://about.gitlab.com/, About GitLab, obiskano 3. 7. 2025.

[8] https://appium.io/docs/en/2.1/intro/drivers/, Intro to Appium drivers, obiskano 3. 7. 2025.

[9] GRANDA Alen "Neprekinjena integracija in postavitev MAUI mobilnih aplikacij", OTS 2023 Sodobne informacijske tehnologije in storitve: Zbornik 26. konference, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 115-126.

[10] https://nodejs.org/en, Node.js – Run JavaScript everywhere, obiskano 4. 7. 2025.

[11] https://developers.google.com/android/images, Factory images for Nexus and Pixel devices, obiskano 4. 7. 2025.

[12] https://github.com/appium/appium-uiautomator2-driver, Appium UiAutomator2 Driver, obiskano 4. 7. 2025.

[13] https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm, Model-View-ViewModel (MVVM), obiskano 7. 7. 2025.

Dostopnost mobilnih in spletnih aplikacij

Mateja Hazl, Gašper Funda, Filip Božić, Mitja Kočevar

Inova IT d.o.o., Maribor, Slovenija
mateja.hazl@inova.si, gasper.funda@inova.si, filip.bozic@inova.si,
mitja.kocevar@inova.si

Dostopnost digitalnih rešitev je ključnega pomena za zagotavljanje enakih možnosti uporabe vsem uporabnikom, ne glede na njihove fizične, senzorne ali kognitivne omejitve. Prispevek obravnava smernice WCAG 2.2 ter Evropski akt o dostopnosti (EAA), ki postavljata temeljne zahteve za oblikovanje vključujočih digitalnih storitev. Podrobno so predstavljeni praktični pristopi k implementaciji dostopnosti v spletnih, iOS in Android aplikacijah, s poudarkom na uporabi semantično pravilnih struktur, prilagodljivosti vsebine, zagotavljanju ustreznega barvnega kontrasta in implementaciji tehnologij, kot so bralniki zaslona, glasovno upravljanje, zunanje vhodne naprave ter prilagoditve za osebe z gibalnimi, vidnimi ali kognitivnimi oviranostmi. Prispevek izpostavlja tudi najpogostejše napake, s katerimi se razvijalci srečujejo pri implementaciji dostopnosti, ter ponuja priporočila in najboljše prakse za njihovo odpravo. Predstavljena so ključna orodja za ročno in avtomatsko testiranje, ki omogočajo učinkovito preverjanje skladnosti z zahtevami dostopnosti že v zgodnjih fazah razvoja. Cilj prispevka je ponuditi celovit, praktično naravnan pregled strategij za razvoj zakonodajno skladnih in uporabniško prijaznih digitalnih rešitev, ki izboljšujejo uporabniško izkušnjo za vse in prispevajo k trajnostni digitalni vključenosti.

Ključne besede:
dostopnost,
WCAG 2.2,
EAA,
semantična struktura,
HTML,
SwiftUI,
Jetpack Compose.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.21

1 Uvod

V sodobni digitalni družbi mobilne aplikacije predstavljajo ključno orodje za vključevanje uporabnikov v digitalno okolje. Dostopnost digitalnih storitev je tako postala pomemben element razvoja programske opreme, saj omogoča enakovredno uporabo tudi uporabnikom z različnimi oblikami oviranosti.

Po podatkih Svetovne zdravstvene organizacije ima približno 1,3 milijarde ljudi (16 % svetovne populacije) neko obliko invalidnosti [20]. Ta odstotek pa narašča zaradi staranja prebivalstva in večje razširjenosti kroničnih bolezni. Zato je vključujoče oblikovanje aplikacij ključno za zagotavljanje enakega dostopa in polne udeležbe vseh uporabnikov v digitalnem okolju. Da bi razvijalci in oblikovalci lahko učinkovito naslovili te izzive, so bile oblikovane mednarodno priznane smernice, ki opredeljujejo ključne zahteve za dostopnost digitalnih vsebin.

Smernice za dostopnost spletnih vsebin (angl. Web Content Accessibility Guidelines, WCAG) so priporočila za izboljšanje dostopnosti spletnih vsebin, predvsem za osebe z invalidnostmi, kot so težave z vidom, sluhom ali kognitivne motnje [1]. Trenutna različica WCAG, verzija 2.2, obsega 13 smernic, razvrščenih v štiri temeljna načela dostopnosti. Ta načela določajo merila, ki jih mora spletna vsebina izpolnjevati in jih bomo v nadaljevanju podrobneje opisali.

Zaznavnost (angl. Perceivable): informacije in komponente uporabniškega vmesnika morajo biti predstavljene na način, ki omogoča njihovo zaznavo (niso nevidne za vse čute).

- Vsebina kot so slike, video, audio in podobno morajo imeti alternativna besedila.
- Multimedijske vsebine vključujejo podnapise ali napise, po potrebi oboje.
- Vsebina spletne strani mora biti predstavljiva na različne načine (npr. z uporabo bralnika zaslona)
- Vsebina mora biti vizualno in zvočno dostopna.

Upravljivost (angl. Operable): komponente uporabniškega vmesnika in navigacija morajo biti upravljive, kar pomeni, da vmesnik ne sme zahtevati interakcij, ki jih uporabnik ne more izvesti.

- Vsa funkcionalnost mora biti dostopna preko tipkovnice.
- Uporabnik ima dovolj časa za branje in interpretacijo vsebine.
- Vsebina spletne strani ne sme povzročati epileptičnih napadov ali drugih telesnih reakcij.
- Uporabniku je olajšana navigacija po spletni strani in iskanje vsebine.
- Uporabnik ima možnost uporabe različnih vhodnih naprav (npr. naprave za glasovno upravljanje).

Razumljivost (angl. Understandable): uporabniki morajo razumeti informacije in delovanje uporabniškega vmesnika.

- Besedilo spletne strani je berljivo in razumljivo.
- Vsebina se pojavlja in deluje na predvidljiv način.
- Uporabniki imajo možnost preprečevanja napak in popravljanja, kadar do njih pride.

Zanesljivost (angl. Robust): vsebina in delovanje spletne strani mora dovolj zanesljivo, da jo lahko uporablja širok spekter uporabniških agentov (npr. tehnologije za podporo).

- Spletna stran je združljiva s trenutnimi in prihodnjimi uporabniškimi orodji.

WCAG 2.2 opredeljuje tudi tri nivoje skladnosti z zgoraj naštetimi smernicami [1, 2]:

- **Nivo A** predstavlja osnovno raven dostopnosti. Spletna stran, ki ne dosega tega standarda, verjetno vsebuje ovire, ki onemogočajo dostopnost nekaterim skupinam ljudi in jih je treba odpraviti.
- **Nivo AA** dodatno prispeva k dostopnosti spletnih vsebin. Na tej ravni mora spletna stran izpolnjevati vse kriterije uspešnosti nivojev A in AA.
- **Nivo AAA** predstavlja najvišjo možno raven skladnosti in pomeni najvišji standard spletne dostopnosti. Na tej ravni morajo biti izpolnjeni vsi kriteriji nivojev A, AA in AAA. Ta raven ni vedno izvedljiva ali primerna za vse organizacije.

Evropski akt o dostopnosti (EAA) je uredba Evropske unije, katere cilj je med drugim zagotoviti, da imajo osebe z invalidnostjo dostop do digitalnih izkušenj brez ovir. Akt je bil sprejet leta 2019 in usklajuje zahteve glede dostopnosti med državami članicami EU. Da dosežemo skladnost z zahtevami EAA, moramo doseči skladnost s smernicami nivoja AA WCAG [2].

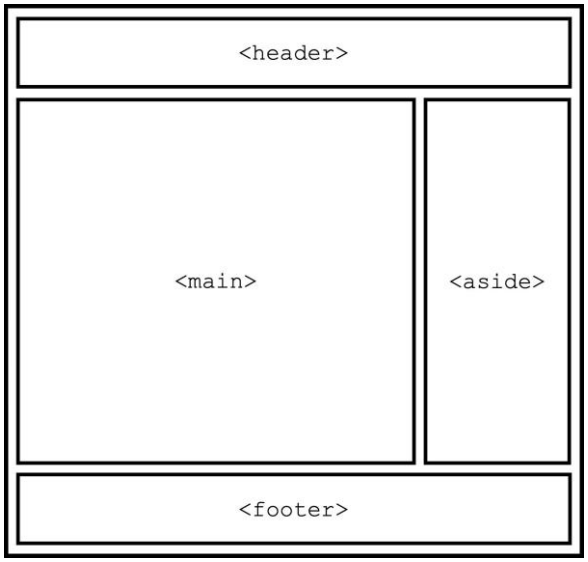
2 Spletne aplikacije

2.1. Struktura HTML dokumenta in semantične značke

Eden temeljnih vidikov razvoja dostopnih spletnih aplikacij je uporaba ustrezne strukture HTML dokumenta in pravilna raba semantičnih HTML značk. Semantično pravilno strukturirane strani omogočajo uporabnikom bralnikov zaslona neposreden dostop do glavne vsebine ter hitrejše premikanje med posameznimi razdelki, kar bistveno poveča učinkovitost interakcije z vsebino [3].
Semantično pravilna struktura spletne vsebuje naslednje ključne regije [3]:

- ***<header>*** označuje zgornji del spletne strani. Ta običajno vključuje logotip, iskalno funkcijo in glavno navigacijo.
- ***<footer>*** predstavlja spodnji del spletne strani, ki običajno vsebuje informacije, kot so kontaktni podatki, izjava o zasebnosti, splošni pogoji uporabe in podobno.
- ***<nav>*** označuje navigacijski del spletne strani, ki je namenjen povezavam za premikanje po vsebini znotraj spletnega mesta. Gre lahko za glavno navigacijo (označeno z atributom *aria-label="glavna navigacija"*), ali pa za stranske in lokalne navigacije. Trenutno izbrano povezavo vedno označimo z *aria-current="page"*.
- ***<main>*** označuje osrednje vsebinsko področje spletne strani. V skladu s smernicami dobrih praks naj bo ta element uporabljen le enkrat na posamezni spletni strani.
- ***<aside>*** označuje vsebinske sklope, ki pojasnjujejo ali komentirajo osrednjo vsebino strani.

Regije, ki se lahko na spletni strani pojavijo večkrat, je potrebno unikatno identificirati z atributom *aria-labelledby* (če obstaja ustrezna vidna oznaka) ali *aria-label* (za nevidne oznake) [3]. Shemo semantično pravilne strukture spletne strani lahko vidimo na sliki 1.



Slika 1: Semantično pravilna struktura spletne strani.

Poleg regij spletnih strani organiziramo vsebino na spletni strani z naslovi, ki jih tehnologije za dostopnost uporabljajo za navigacijo znotraj strani. V skladu s smernicami dobrega razvoja morajo biti naslovi (*<h1>*–*<h6>*) strukturirani hierarhično. Ker naslovi odražajo logično in semantično strukturo dokumenta, je priporočljivo, da se ravni ne preskakuje. Tako naj naslovu *<h1>* ne sledi neposredno naslov *<h3>*, saj lahko to zmede uporabnike in negativno vpliva na dostopnost [3]. Primer nepravilne strukture naslovov vidimo na sliki 2.

```
<h1>Heading level 1</h1>
<h3>Heading level 3</h3>
<h4>Heading level 4</h4>
```

Slika 2: Primer semantično nepravilne strukture naslovov.

Obstaja več semantičnih značk, ki omogočajo označevanje posameznih enot vsebine spletne strani. Semantično pravilno označena vsebina spletne strani lahko vsebuje naslednje značke, ki pomembno izboljšajo dostopnost [3]:

- ***<article>*** in ***<section>*** sta znački, ki predstavljata samostojne vsebinske enote, neodvisne od preostalega konteksta (npr. objava na forumu, komentar, izdelek v spletni trgovini).
- ***<ul>, <ol>, <li>, <dl>, <dt>, <dd>*** so značke, ki se uporabljajo za označevanje različnih vrst seznamov.
- ***<blockquote>, <q>, <cite>*** so značke za označevanje delov besedila, ki so pripisani drugemu avtorju.
- ***<summary>*** in ***<details>*** omogočata uporabniku, da prikaže ali skrije dodatne podrobnosti. Z uporabo teh elementov je zagotovljena podpora za interakcijo s tipkovnico.

Pri vključevanju vizualnih vsebin v spletno stran je ključnega pomena uporaba ustreznih semantičnih elementov, ki označujejo prisotnost vizualne vsebine in njen pripadajoči napis (angl. caption). V ta namen uporabimo element *<figure>*, v katerega vstavimo element *<figcaption>*. Poleg tega mora značka *<img>* vključevati alternativno besedilo, ki je kratek opis vsebine slike. Določimo ga z atributom *alt*, kar omogoča razumevanje vsebine slike tudi uporabnikom, ki ne morejo videti slike. Primer uporabe pravilnih značk in atributov za vključevanje slik lahko vidimo na sliki 3. Dekorativnim slikam je priporočljivo nastaviti prazen atribut alt (*alt=""*), ki omogoča bralnikom zaslona, da takšne slike prezrejo. Poznamo tudi funkcijske slike, ki jih pogosto srečamo v gumbih, povezavah in drugih interaktivnih elementih. Besedilna alternativa za te slike mora opisovati akcijo, ki jo slika sproži, in ne samo opis njene vizualne podobe (npr. *"Iskanje"* namesto *"Povečevalno steklo"*) [4].

```
<figure role="group" aria-labelledby="figure-caption">
    <figcaption id="figure-caption">An elephant at sunset.</figcaption>
    <img src="…" alt="A silhouette of an elephant walking through a field at sunset." />
</figure>
```

Slika 3: Uporaba značk <*figure*> in <*figcaption*> ter atributa alt za dostopno označevanje slik.

Za implementacijo dostopnih tabel celice glave povežemo z njihovimi vrednostmi preko ustreznih semantičnih oznak <*th*> in <*td*>. Kadar tabela vsebuje tako glavo stolpcev kot tudi glavo vrstic, je v elementih <*th*> nujna uporaba atributa scope. Celice glave, ki označujejo stolpce, morajo vključevati atribut *scope="col"*, medtem ko morajo celice glave, ki označujejo vrstice, vsebovati atribut *scope="row"*. S tem se zagotovi pravilna semantična povezava med glavo in ustrezno podatkovno celico. Priporoča se, da se v tabelo vključi tudi element <*caption*>, ki nudi kratek opis vsebine tabele [5].

2.2. Video in avdio

Pomemben vidik dostopnosti avdio in video vsebin na spletnih straneh so predvajalniki, ki jih uporabljamo za prikaz in predvajanje. HTML elementa <*video*> in <*audio*> ponujata vgrajene predvajalnike video in avdio vsebin, ki imajo omejen nabor funkcionalnosti za podporo dostopnosti. Če želimo kontrole predvajalnika (npr. za predvajanje, pavzo, glasnost) implementirati sami, morajo te izpolnjevati nekaj zahtev. Da je predvajalnik dostopen, mora biti vsaka kontrola predvajalnika dostopna za uporabo s pomočjo tipkovnice, element, ki je trenutno aktiven, pa mora biti jasno označen. To dosežemo tako, da za implementacijo kontrol uporabimo HTML elemente kot je <*button*>. Vsaka kontrola mora imeti tudi jasno označen namen in funkcionalnost, kar dosežemo z uporabo atributov, kot je *aria-label*. Med videom, kontrolami predvajalnika in morebitnimi napisi mora biti zagotovljen dovoljšen kontrast, ki zagotavlja boljšo berljivost [6].

Da je video ali avdio vsebina dostopna vsem uporabnikom, mora nuditi podporo za napise in podnapise. Podnapise ali napise podpremo tako, da v HTML element <*video*> ali <*audio*> dodamo element <*track*>, ki vsebuje povezavo do datoteke s podnapisi v formatu WebVTT. Uporabo značke track lahko na primeru kode vidimo na sliki 4. Poleg tega mora uporabniku biti dostopen opis video vsebine (angl. video description, described video). Ta vsebuje opis vizualnih informacij, ki so ključne za razumevanje vsebine videa (npr. besedilo, prikazano v videu). HTML ne nudi privzetega načina za vključitev opisa vizualnih informacij v video, vendar obstajajo različni pristopi, s katerimi je to mogoče doseči [6]. Transkript vsebine (angl. transcript) predstavlja besedilno različico govornih in negovornih informacij v videu ali avdiu. Ta se običajno vključi v spletno stran kot povezava postavljena nad ali pod predvajalnikom [7].

```
<video controls src="/assets/friday.mp4">
  <track
    default
    kind="captions"
    srclang="en"
    src="/assets/friday.vtt" />
</video>
```

Slika 4: Uporaba značke <*track*> za vključitev podnapisov ali napisov.

2.3. Obrazci

Za boljšo dostopnost obrazcev (angl. forms) je pomembna dosledna uporaba semantično pravilnih HTML elementov, ki uporabnikom omogočajo razumevanje in uspešno izpolnjevanje obrazcev.

Vsak element na obrazcu mora biti jasno označen z uporabo elementa *<label>*, kar lahko vidimo na sliki 5. Ta naj bo povezana s pripadajočim kontrolnikom preko atributa *for*, katerega vrednost se ujema z vrednostjo atributa *id*, ki je nastavljen na kontrolniku. V vsakem obrazcu moramo zagotoviti navodila, ki uporabnikom pomagajo razumeti, kako izpolniti obrazec. Navodila so lahko del elementa *<label>*, ali pa jih, v primeru da je namen kontrolnika vizualno dovolj jasen, podamo s pomočjo nevidnega atributa *aria-label* ali *aria-describedby*.

```
<label for="firstname">First name:</label>
<input type="text" name="firstname" id="firstname"><br>
```

Slika 5: Označevanje kontrolnikov z elementom *<label>*.

Sorodne kontrolnike združujemo z uporabo elementa *<fieldset>*, ki logično poveže skupino elementov, na primer niz povezanih možnosti ali polj. Za pojasnilo namena skupine se uporablja element *<legend>*, ki s kratkim in jasnim opisom predstavi vsebino skupine [8]. Primer označevanja sorodnih kontrolnikov lahko vidimo na sliki 6.

```
<fieldset>
    <legend>Interests</legend>
    <div>
        <input type="checkbox" name="art" id="checkbox_1">
        <label for="checkbox_1">Art</label>
    </div>
    <div>
        <input type="checkbox" name="music" id="checkbox_2">
        <label for="checkbox_2">Music</label>
    </div>
    <div>
        <input type="checkbox" name="sport" id="checkbox_3">
        <label for="checkbox_3">Sport</label>
    </div>
</fieldset>
```

Slika 6: Označevanje skupine sorodnih kontrolnikov.

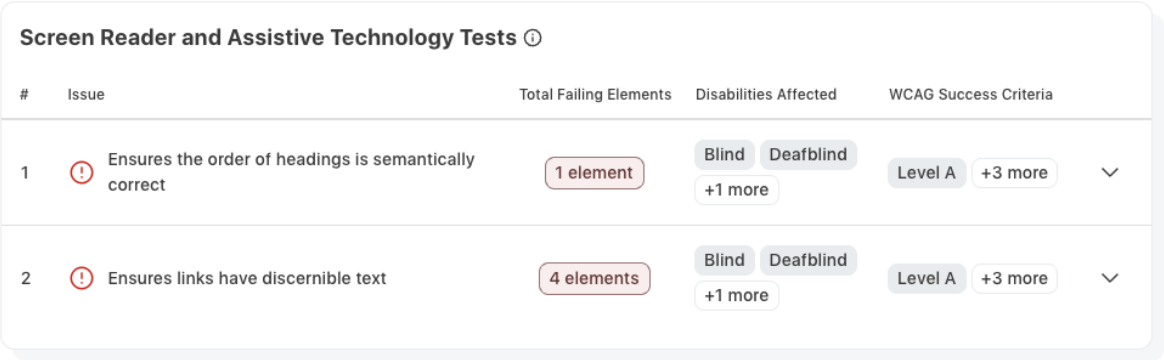
Za validacijo obrazcev je najbolje uporabiti čim več mehanizmov, ki so že podprti v HTML5. Tako na primer za obvezna polja uporabimo atribut *required*, ki avtomatsko nastavi tudi *aria-required="true"* za bralnike zaslonov. HTML5 podpira še validacijo za datume in ure, elektronsko pošto, številke, obsege in druge. Sporočilo o vrsti napake mora biti jasno, nedvoumno in vizualno dostopno [8].

Pri spletnih straneh z daljšimi obrazci se priporoča razdelitev na več logično povezanih korakov. Prvi korak mora vsebovati informacijo o tem, koliko korakov še sledi. Uporabnika obveščamo o trenutnem napredku preko značke *<title>*, preko glavnega naslova strani, ali pa z uporabo elementa *<progress>*. Če je možno, ponudimo uporabniku povezavo za vrnitev na prejšnje korake. Ob tem vključimo besedilo, ki je lahko neviden, in označuje končane korake ter trenutni korak [8].

2.4. Orodja

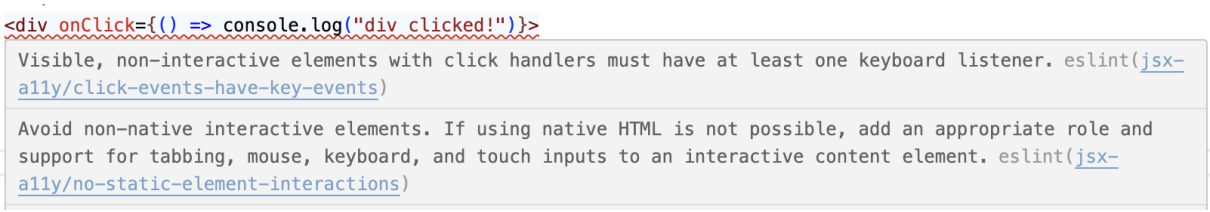
Za izboljšanje dostopnosti spletnih strani obstaja več pristopov, orodij in vtičnikov, ki razvijalcem pomagajo zagotoviti, da so spletne vsebine dostopne vsem uporabnikom. Poznamo več tipov orodij za izboljšanje dostopnosti, nekaj pa jih bomo opisali v nadaljevanju.

Preverjevalniki spletne dostopnosti (angl. web accessibility evaluation tools) analizirajo HTML spletne strani in opozorijo na odstopanja od najboljših praks ter smernic WCAG [9]. Primer rezultata uporabe preverjevalnika spletne dostopnosti lahko vidimo na sliki 7.



Slika 7: Rezultat preverjevalnika dostopnosti spletne strani.

Nekatera odstopanja od smernic za spletno dostopnost je mogoče zaznati s statično analizo programske kode. Za preverjanje programske kode v projektih, ki uporabljajo ogrodje React, lahko uporabimo knjižnico *ESLint* in vtičnik *eslint-plugin-jsx-a11y*, ki opozarja na napake glede dostopnosti v programski kodi. Vtičnik zazna napake kot so uporaba dogodkov za miško ali tipkovnico na neinteraktivnih elementih, izpuščanje alt atributa na slikah, in podobno [10]. Primer napake lahko vidimo na sliki 8.



Slika 8: Primer uporabe napačne prakse, ki jo označi vtičnik *eslint-plugin-jsx-a11y*.

Za testiranje dostopnosti spletne strani je nujno tudi dobro poznavanje funkcij bralnikov zaslona in redno preverjanje ali ti pravilno navigirajo po naši spletni strani. Prav tako je dobro preverjati ali struktura naše spletne strani omogoča hitro in pravilno navigacijo s pomočjo tipkovnice.

2.5. Pogoste napake

V tem poglavju bomo predstavili nekatere najpogostejše napake, ki smo jih zaznali med postopkom razvoja in prilagajanja spletne strani z namenom izboljšanja njene dostopnosti.

Struktura spletne strani in semantične značke

Prva pomanjkljivost, ki smo jo zaznali v obstoječem projektu, je bila nepravilna struktura HTML dokumenta. Na spletni strani je manjkala oznaka <*main*>, prav tako pa naš meni, ki vsebuje povezave na glavne strani, ni vseboval oznake <*nav*>.

Poleg manjkajočih elementov smo naleteli tudi na nepravilno strukturo glavnih naslovov. V skladu z zahtevami uporabniškega vmesnika so naslovu <*h1*> na večini straneh sledili naslovi <*h3*>, kar ne ustreza pravilni hierarhiji naslovov. To napako smo odpravili z uvedbo možnosti za urejevalce vsebine, da vstavijo naslov <*h2*>, ki mu lahko dodelijo slog naslova <*h3*>. To smo storili tako, da smo znački <*h2*> dodali ime razreda "*h3*", ki je v CSS oblikovan enako kot značka <*h3*>. Tako smo spletno stran vizualno uskladili z zahtevami uporabniškega vmesnika. Podobno smo spreminjanje slogov omogočili tudi za ostale nivoje naslovov.
Nepravilna uporaba oziroma neuporaba semantičnih značk je predstavljala eno izmed ključnih pomanjkljivosti obstoječe implementacije. V procesu izboljšav smo morali na več mestih dodati elemente <*ul*> in <*li*>, na primer v seznamu povezav znotraj elementa <*nav*> ter v vseh vrtiljakih (angl. carousel).
Čeprav smo uporabljali značko <*footer*>, pri delu strani, ki je vključeval kontaktne podatke, nismo uporabili ustrezne značke <*address*>. Pri slikah nismo uporabljali značk <*figure*> in <*figcaption*>. Te smo naknadno vključili ne le ob slikah, temveč tudi pri prikazu interaktivnega globusa, kjer smo z opisnim elementom <*figcaption*> omogočili slepim in slabovidnim uporabnikom boljše razumevanje vsebine. Element <*caption*> smo dodali tudi tabelam, s čimer smo urednikom omogočili dodajanje opisov vsebine tabel.
Pri citiranih besedilih smo uporabili značko <*blockquote*>, prav tako pa smo uporabili značko <*cite*> za označevanje avtorjev citatov.

Alternativna besedila

Alternativna besedila slik so imela dve glavni pomanjkljivosti. Prva od teh je bila, da urejevalcem vsebine nismo omogočili vnosa alternativnih besedil povsod, kjer so se prikazovale slike. Poleg tega pa obstoječa alternativna besedila niso sledila smernicam dobrih praks, kot jih priporoča Harvard [11]. V odgovor na te težave smo omogočili urejevalcem vsebine vnos alternativnih besedil za vse slike na spletni strani in dodatno zagotovili, da vsi vnesena alternativna besedila sledijo najboljšim praksam.

***aria-label* za povezave in gumbe**

Naša spletna stran je vsebovala številne povezave, ki niso imele besedilnega opisa (npr. element v vrtiljaku ali ikona kot povezava) ali pa so bile vključene v besedilo, brez ustreznega opisa povezave. Podobne težave smo zaznali tudi pri gumbih, ki so včasih vsebovali le ikono ali pa zelo kratko besedilo, ki ni dovolj jasno opisoval akcije, ki jo gumb sproži. Te pomanjkljivosti smo odpravili tako, da smo omogočili urejevalcem vsebine vnos atributa aria-label za vse povezave in gumbe na spletni strani. S tem smo omogočili, da lahko bralniki zaslona preberejo natančnejši opis cilja povezave (namesto na primer zgolj *"preberi več"*) ali pa podrobneje opišejo dejanje, ki ga gumb izvaja (npr. *"Odpri nastavitve dostopnosti"*).

Interaktivni elementi

Pri interaktivnih elementih smo zaznali več napak, ki so negativno vplivale na dostopnost spletne strani. Prva napaka je bila uporaba značke <*div*> z atributom *onClick*, ki ni omogočala interakcije s tipkovnico. To smo odpravili tako, da smo, kjer je bilo to mogoče, zamenjali značko <*div*> z značko <*button*>, saj ta že privzeto podpira vse potrebne interakcije s tipkovnico.
Druga napaka je bila gnezdenje interaktivnih elementov, v našem primeru elementa <*input*> znotraj elementa <*details*>. Ta praksa lahko povzroči težave pri branju elementov z bralniki zaslona in oteži interakcijo z njimi. Zato smo premaknili element <*input*> tako, da ni več vključen znotraj drugih interaktivnih elementov.
Poleg tega smo zagotovili, da vsi elementi, ki jih prikazujemo znotraj pomikajočih se starševskih elementov (npr. vrtiljakov), podpirajo fokusiranje z uporabo tipke Tab (če tega ne podpirajo že privzeto, kot npr. <*a*> ali <*button*>). To omogoči uporabnikom, ki uporabljajo spletno stran s pomočjo tipkovnice, da izberejo in vidijo elemente zunaj vidnega območja. To dosežemo s pomočjo atributa *tabIndex*.
Za interaktivne elemente, ki se premikajo izven zaslona (npr. povezave v celozaslonskih mobilnih menijih), smo poskrbeli, da so neaktivni ko niso vidni. Tako jih bralniki zaslona ne berejo in jih uporabniki ne morejo izbrati z uporabo tipke Tab. To smo dosegli z uporabo atributa *inert*.

Napisi, podnapisi in transkript v video vsebinah
Na video elementih smo zagotovili, da lahko urejevalci vsebine naložijo datoteke v formatu VTT, ki vsebujejo napise in/ali podnapise za video. Poleg tega smo urejevalcem vsebine omogočili dodajanje povezave do transkripta videa v opisu slike. Z uporabo atributa aria-label smo dodali podporo za podroben opis namena videa in njegove vsebine.

Kontrasti
Paleta barv, uporabljena na naši spletni strani, v nekaterih primerih ni izpolnjevala zahtev WCAG AA standarda glede kontrasta. Ker sprememba obstoječe palete barv ni bila mogoča, smo v nastavitve dostopnosti vključili možnost, da uporabnik vklopi visoko kontrastni način. Ta možnost spremeni vse elemente črne barve v rumene, vse preostale barve pa se spremenijo v črne, kar zagotavlja zadosten kontrast med elementi na strani.

Ostale izboljšave
Poleg že omenjenih izboljšav smo uvedli še nekaj dodatnih nastavitev, ki jih uporabniki lahko vklopijo na spletni strani. Na primer, uporabniki lahko s pomočjo enega klika tipke Tab hitro preskočijo na glavno vsebino ali na gumb za nastavitve dostopnosti. Prav tako imajo uporabniki možnost izklopa vseh animacij na spletni strani, kar je še posebej pomembno za uporabnike z vestibularnimi motnjami gibanja. Ta nastavitev se samodejno vključi tudi za uporabnike, ki imajo vklopljeno nastavitev zmanjšane dinamičnosti animacij (angl. prefers reduced motion).

3 iOS

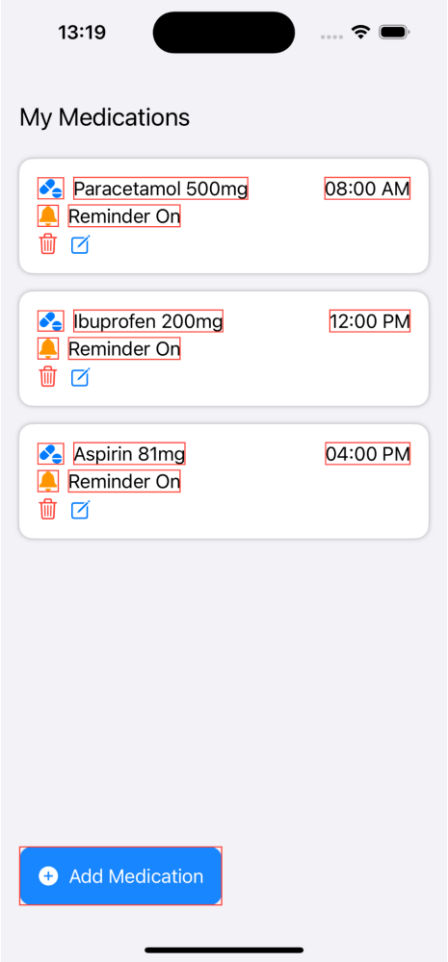
Apple daje velik poudarek dostopnosti v okviru svojih platform, pri čemer iOS vključuje obsežen nabor funkcij in orodij, ki omogočajo enostavno implementacijo dostopnosti v mobilnih aplikacijah, s ciljem zagotoviti, da so aplikacije dostopne vsem uporabnikom, ne glede na njihove fizične ali kognitivne zmožnosti. Upoštevanje smernic za dostopnost v iOS ne pomeni zgolj izpolnjevanja formalnih zahtev, temveč izboljšuje uporabniško izkušnjo za vse in zmanjšuje tveganje za izključevanje pomembnega dela uporabnikov.

3.1. Semantični uporabniški elementi

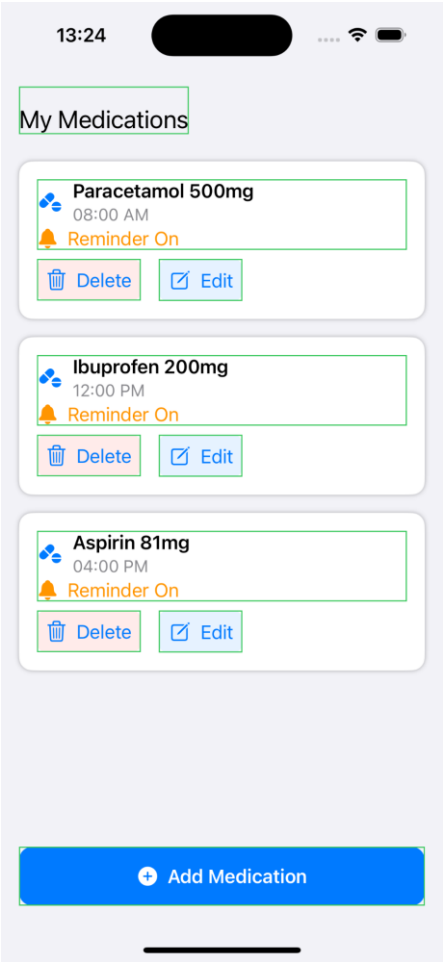
Privzeti elementi v SwiftUI in UIKit (npr. gumbi, labele) že vključujejo osnovno podporo za dostopnost [12]. Apple omogoča dodatno prilagoditev z uporabo *Accessibility API*, kjer so ključne lastnosti:

- *accessibilityLabel(_:)*: opis vsebine elementa,
- *accessibilityValue(_:)*: opis trenutne vrednosti elementa,
- *accessibilityHint(_:)*: sporočilo uporabniku o posledici dejanja,
- *accessibilityAction(( ) -> Void)*: omogoča dodajanje lastnih dostopnostnih dejanj, omogočajo podpornim tehnologijam, kot je VoiceOver, interakcijo z elementom,
- *accessibilityHidden(Bool)*: izključitev elementov iz dostopnostnega drevesa,
- *accessibilityAdd/RemoveTraits(AccessibilityTraits)*: dodajanje ali odstranjevanje lastnosti (npr. označevanje elementa kot gumb).

Združevanje povezanih elementov (npr. stikala in oznake) v logične enote omogoča, da jih VoiceOver predstavi kot eno komponento. To izboljša razumevanje vsebine in poenostavi navigacijo.



Slika 9: Primer neustrezne implementacije.



Slika 10: Primer ustrezne implementacije.

Na levi sliki je prikazan primer slabo implementirane dostopnosti, kjer elementi niso ustrezno združeni v logične skupine. Takšna struktura otežuje delovanje zaslonskih bralnikov, saj ti ne morejo smiselno povezati sorodnih informacij, kar uporabnikom z okvarami vida otežuje razumevanje vsebine. Poleg tega so interaktivni elementi, kot so gumbi, premajhni in brez ustreznih opisov, kar negativno vpliva na njihovo dostopnost prek asistivnih tehnologij. Neustrezen barvni kontrast dodatno zmanjšuje berljivost, kar vodi v slabo uporabniško izkušnjo.

Desna slika prikazuje primer dobre prakse, kjer so sorodni elementi ustrezno združeni in imajo jasno določen pomen. Vsak element vsebuje opisne oznake, ki jih zaslonski bralniki zlahka interpretirajo, kar izboljša razumljivost vsebine za uporabnika. Interaktivni elementi so ustrezno dimenzionirani ter dopolnjeni z opisnimi oznakami in namigi, kar zagotavlja enostavnejšo, varnejšo in dostopnejšo uporabo.

3.2. Načela za izboljšanje dostopnosti

Semantični elementi predstavljajo temelj dostopnosti, poleg njihove pravilne uporabe je potrebno upoštevati tudi dodatne smernice, ki zagotavljajo skladnost z najboljšimi praksami in povečujejo uporabniško izkušnjo za osebe z različnimi oblikami oviranosti [13]:

- **Podpora dinamični velikosti pisave:** Vmesnik mora omogočati prilagoditev velikosti besedila, vključno z do 200-odstotnim povečanjem, brez prekrivanja, prirezovanja ali izgube vsebine.

- **Dostopnost interaktivnih elementov:** Vsi interaktivni elementi, kot so gumbi, stikala in vnosna polja, morajo biti dosegljivi z VoiceOver-om in drugimi asistivnimi tehnologijami.

- **Podpora za komponente po meri:** Enaka pravila veljajo tudi za komponente, razvite po meri, ki morajo biti ustrezno integrirane v semantično drevo.

- **Logičen vrstni red fokusiranja:** Vrstni red prehajanja med elementi mora slediti logični hierarhiji uporabniškega vmesnika. Če privzeta struktura ni ustrezna, jo je mogoče prilagoditi z uporabo metode *.accessibilitySortPriority(_:)* Fokus ne sme preskakovati pomembnih informacij ali se ujeti v nepomembnih elementov.

- **Oznake za pomenske slike:** Vsaka slika, ki nosi pomembno informacijo, mora imeti ustrezen opis dostopnosti, medtem ko morajo biti dekorativne slike izključene iz dostopnostnega drevesa.

- **Izključitev nepomembnih elementov:** Dekorativni ali nedejavni elementi morajo biti izključeni iz dostopnosti z uporabo lastnosti *.accessibilityHidden(true).*

3.3. Orodja

VoiceOver (bralnik zaslona)

VoiceOver [19] je vgrajeni bralnik zaslona, ki omogoča slepim in slabovidnim uporabnikom interakcijo z aplikacijo prek gest in zvočnih opisov. Testiranje z VoiceOver vključuje:

- preverjanje navigacije po elementih (gesta swipe levo/desno),

- preverjanje logičnega vrstnega reda,

- oceno pravilnosti oznak (label, value, traits),

- preverjanje dostopnosti vseh dejanj.

VoiceOver se omogoči v **Settings > Accessibility > VoiceOver** ali preko **Accessibility Shortcut** (npr. trojni klik na stranski gumb) [14, 17].

Switch Control (nadzor s preklopi)

Switch Control omogoča uporabnikom interakcijo z aplikacijami prek fizičnih ali programskih stikal (npr. namenski gumbi, tipkovnice ali igralni krmilniki), vendar uporaba dodatne strojne opreme ni nujna, saj omogoča Switch Control uporabo obstoječih elementov naprave, kot so gumbi na telefonu, dotiki zaslona ali celo dotiki na zadnjo stran naprave.

AssistiveTouch

AssistiveTouch [12] uporabnikom omogoča nadomestitev kompleksnih gest, kot so stiskanje, vlečenje ali uporaba več prstov, z enostavnimi, prilagodljivimi meniji. Ta možnost omogoča izvajanje širokega nabora opravil, kot so prilagajanje glasnosti, zaklepanje zaslona, uporaba več prstnih gest, ponovni zagon naprave ali nadomestitev fizičnih gumbov s preprostim dotikom. Funkcionalnost je ključna za osebe z gibalnimi omejitvami, saj omogoča intuitivno in učinkovito interakcijo brez potrebe po zahtevnih fizičnih dejanjih.

Voice control (nadzor z glasom)

Voice Control [18] omogoča upravljanje aplikacij izključno z glasovnimi ukazi, brez potrebe po fizičnem dotiku zaslona. Ko je funkcionalnost omogočena, uporabniki izgovarjajo ukaze, ki posnemajo običajna dejanja, izvedena z dotikom, kar omogoča izvajanje celotne navigacije in interakcije z uporabniškim vmesnikom. Dodatne možnosti vključujejo ukaza *show numbers*, ki na zaslonu prikaže številčne oznake za vse interaktivne elemente, ter *show names*, ki namesto številk prikaže njihove opisne oznake.

Filtri za barve in zmanjšanje gibanja

Filtri za barve in možnost zmanjšanja gibanja predstavljajo pomembni funkcionalnosti za uporabnike z barvno slepoto ter tiste, ki imajo težave z zaznavanjem premikajočih se animacij. Filtri omogočajo prilagoditev barvne sheme zaslona z namenom izboljšanja kontrasta in prepoznavnosti vsebine, možnost zmanjšanja gibanja zmanjšuje ali popolnoma odstrani kompleksne animacije in prehode, ki lahko povzročijo nelagodje ali težave pri orientaciji.

Dostopnost z zunanjo tipkovnico

Dostopnost z zunanjo tipkovnico omogoča uporabnikom interakcijo z mobilno napravo brez uporabe zaslona na dotik, kar je posebej pomembno za osebe z omejeno mobilnostjo. iOS podpira uporabo fizičnih tipkovnic, kot je Magic Keyboard, ali drugih združljivih tipkovnic, povezanih prek Bluetooth ali USB. Funkcionalnost polnega dostopa s tipkovnico (angl. *Full Keyboard Access*) omogoča upravljanje naprave z bližnjicami na tipkovnici, pri čemer sistem označi element na zaslonu, ki je trenutno v fokusu.

3.4. Testiranje

Accessibility Inspector

Xcode vključuje orodje Accessibility Inspector, namenjeno vizualnemu preverjanju dostopnostnih lastnosti znotraj iOS aplikacij [16]. To orodje omogoča razvijalcem analizo in validacijo različnih atributov uporabniškega vmesnika brez potrebe po ročnem preizkušanju z VoiceOver-jem. Ključne funkcionalnosti Accessibility Inspectorja so:

- Pregled lastnosti posameznih elementov (npr. *label*, *trait*, *value*).
- Preverjanje, ali elementi prejemajo fokus v pričakovanem zaporedju in da je navigacija logična.
- Preverjanje skladnosti barvnih kontrastov in odzivnost interaktivnih komponent na uporabniške interakcije.

Accessibility Inspector je posebej uporaben za hitro odkrivanje napak, kot so manjkajoče oznake ali nepravilno določeni atributi, kar omogoča izboljšanje dostopnosti brez dolgotrajnega ročnega testiranja.

Testiranje barvnega kontrasta

V okviru iOS razvoja Apple omogoča uporabo vgrajenih orodij in simulacij znotraj iOS Simulatorja, kjer je mogoče preizkusiti različne scenarije, kot so simulacija barvne slepote ali visoko kontrastni načini. S tem lahko razvijalci preverijo:

- ali imajo vsi pomembni vizualni elementi zadostno kontrastno razmerje (najmanj 4.5:1 za običajno besedilo in 3:1 za večje pisave, skladno z WCAG),
- ali ključne informacije niso predstavljene zgolj z uporabo barve (na primer: napake označene samo z rdečo barvo).

Testiranje prilagoditve velikosti besedila

Prilagoditev velikosti besedila je ena ključnih funkcionalnosti za zagotavljanje dostopnosti, saj omogoča uporabnikom s slabšim vidom ali posebnimi potrebami lažje branje vsebine. Uporabniki lahko v nastavitvah sistema povečajo ali zmanjšajo velikost pisave, pri čemer se pričakuje, da aplikacija ustrezno reagira na te spremembe brez izgube funkcionalnosti ali berljivosti. Pri preverjanju je treba upoštevati naslednje vidike:

- vso besedilo mora ostati v celoti vidno in se ne sme prekrivati drugih elementov,
- uporabniški vmesnik mora omogočati dinamično prilagajanje večjim pisavam, ne da bi se pri tem porušila struktura ali uporabnost zaslona.

Avtomatska testiranja

Poleg ročnega preverjanja dostopnosti so na voljo tudi avtomatizirane rešitve, ki omogočajo hitrejše in bolj sistematično odkrivanje napak. Ena izmed takšnih platform je BrowserStack, ki omogoča testiranje mobilnih aplikacij na različnih napravah in konfiguracijah, brez potrebe po fizičnem dostopu do teh naprav.

3.5. Najpogostejše napake pri implementaciji dostopnosti

Pri implementaciji dostopnosti v iOS aplikacijah se pogosto pojavljajo napake, ki lahko bistveno vplivajo na uporabniško izkušnjo oseb z različnimi oblikami oviranosti. Najpogostejši primeri vključujejo:

- **Manjkajoče oznake** (*accessibilityLabel*) – če gumbi, ikone ali druge interaktivne komponente nimajo ustrezno določenih oznak, jih VoiceOver ne more pravilno predstaviti uporabniku, kar otežuje razumevanje njihove funkcionalnosti.
- **Dostopnost nepomembnih elementov** – dekorativne slike in ozadja ne bi smela biti del dostopnostnega drevesa. V takšnih primerih je potrebno nastaviti lastnost *isAccessibilityElement* = *false*, s čimer se izognemo nepotrebnim informacijam za uporabnika.
- **Nelogičen vrstni red fokusiranja** – če VoiceOver prehaja elemente v nelogičnem zaporedju, lahko to povzroči zmedo in frustracijo uporabnika. Vrstni red mora slediti naravnemu toku vsebine na zaslonu.
- **Uporaba samo barve za prenos pomena** – informacija ne sme biti posredovana zgolj z barvo (npr. »rdeče označena polja so obvezna«), temveč mora biti podprta z dodatnim besedilnim ali semantičnim indikatorjem.
- **Neodzivni ali nedosegljivi elementi** – napačna uporaba lastnosti *isUserInteractionEnabled* ali neustrezna implementacija gest lahko prepreči dostopnost določenih funkcionalnosti.

4 Android

V letu 2025 platformo Android uporablja 3,3 milijarde uporabnikov [28], zato lahko predpostavljamo, da je od tega kar 495 milijonov uporabnikov platforme Android z določeno obliko invalidnosti. Zaradi tako velikega deleža je ključnega pomena, da razvijalci zagotavljajo dostopne aplikacije, saj s tem ne le izboljšajo uporabniško izkušnjo, temveč preprečijo izgubo pomembnega dela uporabnikov in ohranijo konkurenčnost na trgu.

Platforma Android vključuje številne smernice in orodja, ki razvijalcem pomagajo zagotoviti dostopnost aplikacij. Med temi je še posebej pomemben Jetpack Compose – sodoben deklarativen okvir za gradnjo uporabniških vmesnikov – ki omogoča bolj nadzorovano in konsistentno upravljanje z dostopnostjo. Prispevek podrobno obravnava priporočila, načela in prakse za razvoj dostopnih Android aplikacij, zlasti s poudarkom na komponentah Material Design, prilagojenem oblikovanju, semantični opremljenosti komponent ter testiranju in orodjih.

4.1. Android Accessibility Guidelines

Google v okviru Android Accessibility Guidelines konkretizira omenjena WCAG načela [22] z naborom priporočil, ki pokrivajo zasnovo, razvoj in testiranje mobilnih aplikacij [25]. Smiselno je ta priporočila integrirati že v zgodnjih fazah razvoja, saj tako preprečimo poznejše spremembe, ki so stroškovno bolj zahtevne. Razumevanje dostopnosti lastnega izdelka prispeva k večji uporabnosti tudi za splošno populacijo, saj prilagoditve pomagajo tako osebam z nizkim vidom, gluhoto, motnjami pozornosti, gibalnimi omejitvami kot tudi tistim, ki se znajdejo v situacijskih oviranostih, kot je zlomljena roka.

4.2. Material Design in Jetpack Compose

Material Design [26] kot oblikovalski sistem daje velik poudarek dostopnosti. Njegova filozofija temelji na predpostavki, da je dostopnost privzeta vrednota, ne dodatek. Uporaba ogrodja Jetpack Compose skupaj s komponentami Material 3 olajša integracijo dostopnostnih funkcionalnosti, saj komponente že privzeto vključujejo ustrezno semantiko, omogočajo odzivnost na sistemske nastavitve (npr. povečava pisave, temni način), zagotavljajo ustrezne kontraste, velikosti interaktivnih površin in napredne možnosti prilagajanja. S tem razvijalcem omogočajo, da na enostaven in konsistenten način zagotovijo skladnost z dostopnostnimi smernicami, zmanjšajo ročno delo ter pospešijo razvoj vključujočih uporabniških vmesnikov.

4.3. Načela za izboljšanje dostopnosti aplikacij

Semantika

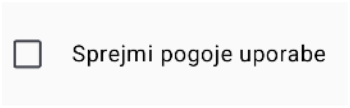
Semantika (angl. Semantics) predstavlja dodatne informacije o pomenu in vlogi UI komponent. Poleg osnovnih lastnosti lahko z uporabo semantičnih lastnosti omogočimo boljše razumevanje elementov za storitve dostopnosti, avtomatsko izpolnjevanje obrazcev in testne okvire.

V Jetpack Compose lahko semantiko nastavimo prek *Modifier.semantics* { }. Vsak element ima lahko določene semantične lastnosti, kot so:

- *role*: vrsta komponente (npr. gumb, stikalo, potrditveno polje),
- *stateDescription*: opis trenutnega stanja (npr. "Omogočeno"),
- *contentDescription*: opis pomena (npr. "Posnemi fotografijo"),
- *liveRegion*: način obveščanja uporabnika ob spremembah; *LiveRegionMode.Polite* (počaka na branje) ali *LiveRegionMode.Assertive* (prekine trenutno branje),
- *heading()*: za naslove in podnaslove,
- *paneTitle*: za ločevanje med pogovornimi okni in zasloni,
- *error*: za dodatna sporočila napak,
- *progressBarRangeInfo*: informacije o napredku,
- *collectionInfo* in *collectionItemInfo*: podatki o seznamih in elementih v njih,
- *customActions*: za bolj kompleksne geste (npr. poteg za odstranitev).

Ustrezno nastavljena semantika omogoča bralnikom zaslona, da elemente pravilno predstavijo, uporabniki pa se lažje orientirajo in upravljajo aplikacijo. Če komponenta vključuje več podkomponent (npr. Gumb z ikono in besedilom), se njihova semantika običajno združi z uporabo *mergeDescendants* = *true*. Če uporabljamo lastne nizkonivojske komponente, moramo semantiko eksplicitno definirati.

Spodnji primer prikazuje dobro prakso združevanja elementov, kjer se *Checkbox* in pripadajoče besedilo združita v enoten dostopnostni element, kar poenostavi navigacijo za uporabnike bralnikov zaslona. Dodajanje semantičnih lastnosti, kot so *role*, *contentDescription*, *stateDescription* in *onClick*, zagotavlja jasno komunikacijo namena in stanja elementa ter omogoča interakcijo brez potrebe po natančnem ciljanju posamezne komponente.

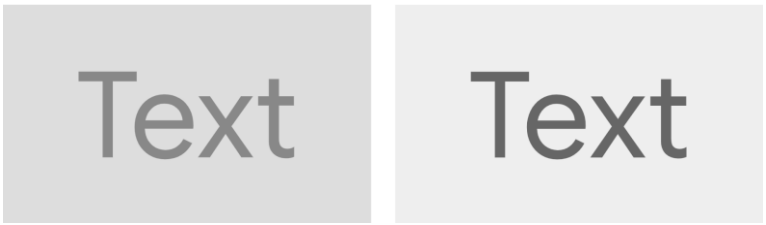



```
Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 24.dp)
        .focusable()
        .semantics(mergeDescendants = true) {
            role = Role.Checkbox
            contentDescription = "Sprejmi pogoje uporabe"
            stateDescription = if (acceptedTerms) "Sprejeto" else "Ni sprejeto"
            liveRegion = LiveRegionMode.Polite
            onClick {
                acceptedTerms = !acceptedTerms
                true
            }
        }
) {
    Checkbox(
        checked = acceptedTerms,
        onCheckedChange = { acceptedTerms = it },
        modifier = Modifier
    )
    Spacer(modifier = Modifier.width(8.dp))
    Text(text = "Sprejmi pogoje uporabe")
}
```

Slika 11: Primer uporabe semantičnih lastnosti za lastno komponento.

Povečanje vidnosti besedila

Za vsako besedilo v aplikaciji naj bo kontrast med besedilom in ozadjem dovolj visok [23]. Če je besedilo manjše od 18pt (14pt krepko), je priporočeno razmerje kontrasta vsaj 4.5:1, za večja besedila je priporočeno razmerje vsaj 3:1.



Slika 12: Nizek barvni kontrast (levo) in primeren kontrast (desno). [23]

Zagotavljanje ustrezne velikosti interaktivnih elementov

Interaktivni elementi morajo biti dovolj veliki, da jih lahko uporabniki zlahka vidijo in aktivirajo. Priporočena je velikost ciljne površine vsaj 48x48 dp ali več. Večje površine pripomorejo k boljši dostopnosti predvsem za uporabnike z gibalnimi ovirami ali tresočimi rokami. Minimalno površino je mogoče doseči z dodanimi odmiki (*padding*)

```
IconButton(
    onClick = onCloseClicked,
    modifier = Modifier
        .align(Alignment.TopEnd)
        .padding(24.dp)
) {
    Icon(
        modifier = Modifier.size(24.dp),
        imageVector = Icons.Default.Close,
        contentDescription = "Zapri"
    )
}
```

Slika 13: Primer komponente s priporočeno velikostjo ciljne površine.

Opisovanje elementov uporabniškega vmesnika

Vsak element naj ima opis (*contentDescription*), ki jasno razloži njegov namen [21]. Ta opis omogoča bralnikom zaslona, da uporabniku posredujejo ustrezne informacije. Elementom tipa *Text* ni potrebno dodatno nastavljati opisa, saj Androidove storitve dostopnosti že samodejno preberejo vsebino besedila kot opis.

Pri dodajanju opisov za uporabniške elemente je treba upoštevati naslednje dobre prakse:

- Vrsta UI elementa ne sodi v opis. Bralniki zaslona samodejno napovejo tako vrsto kot opis elementa. Na primer, če izbira gumba sproži oddajo obrazca, naj bo opis gumba "Oddaj", ne pa "Gumb za oddajo".

- Vsak opis mora biti unikaten. Tako lahko uporabniki bralnikov zaslona prepoznajo, da se fokus ponavlja na elementu, ki je bil že izbran. To je še posebej pomembno pri seznamih (npr. *LazyList*), kjer mora vsak element imeti svoj edinstven opis, npr. ime kraja v seznamu lokacij.

- Dekorativni grafični elementi ne potrebujejo semantičnih lastnosti, zato jih je priporočljivo odstraniti z uporabo *Modifier.clearSemantics()* ali *Modifier.semantics(mergeDescendants = false)* { }. S tem se takšni elementi izključijo iz semantičnega drevesa in ne motijo uporabniške izkušnje za uporabnike dostopnostnih orodij.

```
Button(
    onClick = { onLoginClicked() },
    enabled = acceptedTerms,
    modifier = Modifier
        .fillMaxWidth()
        .focusable()
) {
    Icon(
        imageVector = Icons.AutoMirrored.Filled.ArrowForward,
        contentDescription = "Prijavi se"
    )
}
```

Slika 14: Primer komponente s pravilnim opisom vsebine.

4.4. Orodja

TalkBack

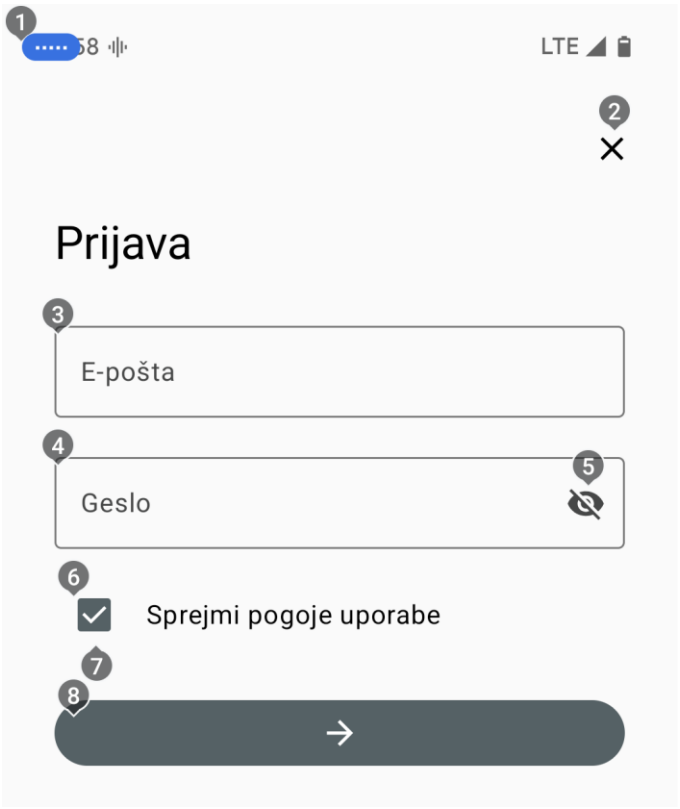
TalkBack [28] je vgrajeni bralnik zaslona v Androidu, ki uporabnikom z okvarami vida omogoča, da upravljajo napravo brez pogleda na zaslon. Ko je TalkBack vklopljen, uporabniki uporabljajo poteze, kot so drsanje prsta po zaslonu za brskanje po elementih ter dvojni dotik za aktivacijo izbranega elementa. Vsaka komponenta, ki je interaktivna ali vizualno pomembna, mora imeti ustrezne semantične oznake, saj bo TalkBack uporabniku prebral kombinacijo teh podatkov: vrsto elementa, njegovo oznako in stanje. Na primer, gumb z opisom "Oddaj" bo TalkBack prebral kot "Gumb, Oddaj. Dvojni dotik za aktivacijo". Za stikala in preklopne gumbe TalkBack prebere tudi stanje, kot npr. "Vklopljeno stikalo. Dvojni dotik za spremembo".

TalkBack ima tudi funkcionalnosti za premikanje med naslovi, obrazci, povezavami in napakami, če so ti ustrezno označeni z *heading()*, *error*, *liveRegion* ipd. Priporoča se, da razvijalci redno testirajo aplikacijo z vklopljenim TalkBack, saj gre za najbolj neposreden način preverjanja dostopnosti za slepe in slabovidne uporabnike. To omogoča hitro odkrivanje težav, ki jih avtomatizirana orodja pogosto ne zaznajo.

Voice Access

Voice Access [31] omogoča upravljanje Android naprav z glasovnimi ukazi. Na voljo je od Android 5.0 dalje in je ključnega pomena za uporabnike z gibalnimi ovirami. Trenutno je na voljo le v angleščini, francoščini, italijanščini, nemščini in španščini.

Ko je Voice Access vklopljen, uporabnik uporablja vnaprej definirane govorne ukaze za upravljanje elementov na zaslonu. Android sistem na zaslonu prikaže številčne oznake za vsak interaktivni element, kar omogoča uporabniku, da s preprostim ukazom (npr. "tap 7"), izvede želeno dejanje. Poleg tega so na voljo ukazi, kot so "scroll down", "go back", "type name", kar pokriva širok nabor interakcij z aplikacijo.



Slika 15: Primer številčnih oznak za Voice Access.

Za dobro delovanje Voice Access mora aplikacija uporabljati deskriptivne opisne vrednosti (*contentDescription*), saj sistem temelji na teh informacijah za generiranje akcijskih ukazov. Na primer, gumb z oznako "Pošlji sporočilo" bo omogočil uporabniku, da ukaz izvede z besedno zvezo "tap Pošlji sporočilo". Izogibati se je potrebno podvajanju opisov ali uporabi generičnih opisov, saj lahko to vodi do nejasnosti pri prepoznavanju elementov. Prav tako je priporočljivo, da vsak dinamično ustvarjen element (npr. v seznamih) dobi svoj edinstven in logičen opis.

Voice Access omogoča tudi navigacijo po obrazcih, izbiro možnosti, uporabo tipkovnice na zaslonu in celo izvajanje kompleksnih zaporedij ukazov. Zaradi teh zmožnosti je nujno, da je aplikacijski vmesnik semantično bogat in dosledno strukturiran. Redno testiranje s Voice Access funkcijo pomaga odkriti pomanjkljivosti, ki jih druge metode testiranja morda ne zaznajo.

Prilagajanje velikosti prikaza in pisave

Velikost prikaza in velikost pisave sta pomembni nastavitvi dostopnosti, ki ju mnogi uporabniki prilagajajo glede na svoje potrebe – bodisi zaradi slabšega vida, bralnih težav ali uporabe naprave z večje razdalje. Android omogoča spreminjanje teh vrednosti v nastavitvah sistema, aplikacije pa se morajo biti sposobne pravilno in smiselno odzvati na te spremembe [27].

Pri testiranju aplikacije je ključno preveriti, kako se uporabniški vmesnik prilagaja ob spremembi teh nastavitev. Povečana pisava lahko povzroči, da se besedilo prelomi, izreže ali prekriva druge elemente, medtem ko povečana velikost prikaza vpliva na celotno postavitev komponent in razmerja med njimi. Kritični deli, kot so gumbi, meniji, obrazci in kartice, morajo ostati berljivi, klikabilni in jasno strukturirani.

Zunanje naprave za vnos

Android sistem podpira navigacijo prek zunanjih naprav za vnos, kot so USB ali Bluetooth tipkovnice ali stikala (npr. Switch Access). Razvijalci morajo zagotoviti, da so vsi pomembni elementi vmesnika dostopni preko tipkovnice, in da je zaporedje fokusiranja logično in predvidljivo. Vsak interaktiven element naj bo fokusabilen, kar lahko v Jetpack Compose dosežemo z uporabo *Modifier.focusable()* in *Modifier.focusRequester()*. Poleg tega je pomembno, da aplikacija ustrezno vizualno označi trenutno fokusiran element, bodisi z okvirjem, spremembo barve ali osenčenjem, da je jasno razvidno, kje se nahaja fokus.

V aplikacijah z zahtevnejšo navigacijo (npr. obrazci, dolgi seznami, dialogi) je priporočljivo testiranje s fizično tipkovnico ali Android emulatorjem z omogočenimi stikali. Preveriti je treba, ali je vsak element dosegljiv, ali se fokus obnaša predvidljivo in ali se ob pritisku na Enter ali preslednico izvede ustrezno dejanje. Dodatno naj bodo vsa pojavna okna (npr. modalna okna, dialogi), pravilno fokusirana ob prikazu ter omogočajo vračanje fokusa na prejšnji element po zaprtju.

4.5. Testiranje

Ročno testiranje

Ročno testiranje predstavlja ključen korak pri razvoju dostopnih aplikacij in mora biti vključeno kot redna praksa v vsak razvojno-testni cikel. Čeprav so avtomatizirana orodja učinkovita pri zaznavanju osnovnih napak, ne morejo nadomestiti dejanske uporabniške izkušnje oseb z različnimi oblikami oviranosti. Zato ima ročno preverjanje delovanja aplikacij z omogočenimi dostopnostnimi storitvami na fizičnih napravah posebno vrednost. Za učinkovito izvedbo ročnega testiranja se običajno preverja več ključnih vidikov [24]:

- **Omogočanje dostopnostnih storitev** (npr. TalkBack, Voice Access, zunanja naprava) in uporaba aplikacije izključno z njimi za oceno dejanske uporabniške izkušnje.
- **Zaporedje fokusiranja med elementi**, ki mora biti logično in predvidljivo.
- **Pravilna oznaka interaktivnih elementov**, bodisi prek *contentDescription* ali drugih semantičnih lastnosti.
- **Vidnost vizualnega fokusa**, ki mora jasno označevati trenutno izbran element tako pri uporabi dotika kot tipkovnice.
- **Odzivnost komponent**, vključno z zagotavljanjem minimalnih dimenzij ciljnih površin.
- **Prilagodljivost na sistemske nastavitve**, kot so povečana velikost pisave, sprememba velikosti prikaza, kontrastni način in barvne prilagoditve.

Priporoča se, da se ročno testiranje izvaja ob vsaki večji spremembi funkcionalnosti, in ne le na koncu razvojnega cikla. Zgodnje odkrivanje težav zmanjša stroške popravkov, hkrati pa omogoča razvoj bolj vključujoče uporabniške izkušnje. Na dolgi rok pa je za resnično preverjanje uporabnosti priporočljivo vključiti tudi testiranje z dejanskimi uporabniki z različnimi oblikami oviranosti. Njihove izkušnje pogosto razkrijejo pomankljivosti, ki jih razvijalci sami težko opazijo.

Compose UI Check

Compose UI Check [30] v Android Studiu omogoča samodejno preverjanje dostopnosti med predogledom komponent. Opozarja na težave z raztezanjem besedila, kontrasti in velikostmi elementov ter pomaga pri zagotavljanju skladnosti skozi različne velikosti zaslonov. Ta funkcionalnost je vgrajena neposredno v Compose

Preview in omogoča, da razvijalci že v fazi razvoja zaznajo težave, ki bi lahko negativno vplivale na uporabnike s posebnimi potrebami. Preverjanje vključuje:

- zaznavanje besedila z nizkim kontrastom v primerjavi z ozadjem,
- odkrivanje komponent, katerih velikost ne dosega priporočenih minimalnih dimenzij (npr. 48x48 dp),
- preverjanje dolžine besedil, ki se lahko raztegnejo izven okvirov ali povzročijo težave pri prilagajanju na večje zaslone.

Opozorila so prikazana neposredno v zavihku Problems znotraj Android Studia, kar omogoča hitro navigacijo do težavnih komponent in njihovo odpravo.

Compose UI Check je še posebej koristen v kombinaciji z uporabo različnih konfiguracij predogledov (*@Preview*) z različnimi nastavitvami velikosti pisave, načini kontrasta in jezikov, saj omogoča preverjanje odzivnosti komponent na prilagoditve uporabniškega sistema.

Accessibility Scanner

Aplikacija Accessibility Scanner omogoča pregled zaslona in poda konkretne predloge za izboljšave. Preverja opise, kontraste, interaktivnost in uporabo pripomočkov. Uporablja okvir Accessibility Test Framework, ki je del Android Accessibility Suite. Uporabnik zažene aplikacijo in izbere cilj aplikacije, ki jo želi analizirati. Scanner nato posname trenutni zaslon in na njem izriše vizualne oznake težav z dostopnostjo, kot so:

- manjkajoči *contentDescription* atributi za slike ali gumbe,
- nizko razmerje kontrasta med besedilom in ozadjem,
- premajhne ciljne površine interaktivnih elementov,
- prekrivajoči se elementi,
- nezdružljivost s pripomočki za dostopnost.

Za vsako zaznano težavo poda priporočilo za izboljšavo ter omogoči neposreden skok v razvojnem orodju (če se uporablja v povezavi z Android Studiem). Prav tako omogoča izvoz rezultatov v obliki poročila, ki je koristno za spremljanje napredka pri odpravljanju napak.

Accessibility Scanner je še posebej uporaben v zgodnjih fazah razvoja in pri regresijskem testiranju, saj zagotavlja hitro vizualno povratno informacijo o skladnosti aplikacije s smernicami dostopnosti. Redna uporaba tega orodja pomaga razvijalcem ohranjati visoko raven dostopnosti skozi celoten življenjski cikel aplikacije.

BrowserStack

BrowserStack [29] je spletna platforma, ki omogoča oddaljeno testiranje aplikacij na različnih napravah in konfiguracijah. To je uporabno pri zagotavljanju dostopnosti na različnih verzijah Androida in tipih naprav, brez fizičnega dostopa do vseh modelov.

Platforma omogoča dostop do široke zbirke fizičnih naprav z različnimi velikostmi zaslonov, ločljivostmi in sistemskimi različicami, kar je ključno za preverjanje, ali aplikacija ohranja svojo dostopnostno skladnost v raznolikem okolju. Uporabniki lahko preko brskalnika ali API-ja naložijo APK in preizkusijo aplikacijo z omogočenimi funkcijami dostopnosti, kot so TalkBack, Zoom, veliki tekst ali visoki kontrast. Omogoča tudi

snemanje sej, kar je uporabno za dokumentacijo testiranja in ponovljivost testnih scenarijev. V povezavi z avtomatiziranimi testi (npr. z uporabo Espresso ali Appium) je mogoče integrirati testiranje dostopnosti v CI/CD procese, kar pomaga pri zgodnjem zaznavanju regresij na področju uporabnosti in dostopnosti.

5 Zaključek

Dostopnost digitalnih rešitev je ključni element sodobnega razvoja programske opreme in prispeva k zagotavljanju enakih možnosti za vse uporabnike, ne glede na njihove fizične, senzorne ali kognitivne sposobnosti. Skladnost s smernicami WCAG ter upoštevanje platformno specifičnih priporočil za splet, Android in iOS omogoča oblikovanje rešitev, ki so robustne, intuitivne in vključujoče. Uporaba semantično pravilnih struktur, podpora dinamičnim prilagoditvam, zagotavljanje ustreznega kontrasta ter implementacija tehnologij za asistenco, kot so bralniki zaslona in glasovno upravljanje, so temeljni koraki k univerzalni dostopnosti.

Ključnega pomena je, da dostopnost ni obravnavana kot dodatek v zaključnih fazah razvoja, temveč kot integralni del celotnega življenjskega cikla programske rešitve. Vključevanje ročnega testiranja, uporaba avtomatiziranih orodij in sodelovanje z dejanskimi uporabniki z oviranostmi prispevajo k odkrivanju in odpravljanju pomanjkljivosti, ki jih drugače ni mogoče zaznati. Takšen pristop ne zagotavlja le skladnosti z zakonodajo in standardi, temveč prinaša tudi konkurenčne prednosti, saj povečuje kakovost uporabniške izkušnje za vse.

Z vidika prihodnosti bo dostopnost še naprej predstavljala strateško področje razvoja, kjer bodo v ospredju prilagodljivost, standardizacija in uporaba inteligentnih orodij za validacijo. Le s sistematičnim in vključujočim pristopom lahko zagotovimo digitalno okolje, ki je pravično, trajnostno in pripravljeno na potrebe različnih skupin uporabnikov.

Literatura

[1] https://www.w3.org/WAI/, "Web Accessibility Initiative", obiskano: 4. 7. 2025.

[2] https://www.wcag.com/compliance/european-accessibility-act/, "The European Accessibility Act: Technical Aspects of Compliance", obiskano 4. 7. 2025.

[3] https://www.w3.org/WAI/tutorials/page-structure/, "Page Structure Tutorial", obiskano 4. 7. 2025.

[4] https://www.w3.org/WAI/tutorials/images/, "Images Tutorial", obiskano 4. 7. 2025.

[5] https://www.w3.org/WAI/tutorials/tables/, "Tables Tutorial", obiskano 4. 7. 2025.

[6] https://www.w3.org/WAI/media, "Media", obiskano 4. 7. 2025.

[7] https://www.w3.org/TR/media-accessibility-reqs/, "Media Accessibility User Requirements", obiskano 4. 7. 2025.

[8] https://www.w3.org/WAI/tutorials/forms/, "Forms Tutorial", obiskano 4. 7. 2025.

[9] https://www.w3.org/WAI/test-evaluate/tools/list/, "Web Accessibility Evaluation Tools List", obiskano 4. 7. 2025.

[10] https://www.npmjs.com/package/eslint-plugin-jsx-a11y, "eslint-plugin-jsx-a11y - npm", obiskano 4. 7. 2025.

[11] https://accessibility.huit.harvard.edu/describe-content-images, "Write helpful Alt Text to describe images", obiskano 4. 7. 2025.

[12] https://developer.apple.com/accessibility/ios/, "Accessibility Programming Guide for iOS", obiskano: 19. 6. 2025.

[13] https://developer.apple.com/design/human-interface-guidelines/accessibility/overview/, "Human Interface Guidelines: Accessibility", obiskano: 19. 6. 2025.

[14] https://nshipster.com/uiaccessibility/, "Accessibility", obiskano: 19. 6. 2025.

[15] https://www.raywenderlich.com/7476-improving-ios-accessibility-with-voiceover, "Improving iOS Accessibility with VoiceOver", obiskano: 19. 6. 2025.

[16] https://designcode.io/ios-design-handbook-design-for-accessibility, "Design for Accessibility – iOS Design Handbook", obiskano: 19. 6. 2025.

[17] https://support.apple.com/en-us/111794, "Make Your Apps More Accessible with VoiceOver", obiskano: 19. 6. 2025.

[18] http://support.apple.com/en-us/111778, "Make Your iOS App Accessible", obiskano: 19. 6. 2025.

[19] https://support.apple.com/sl-si/guide/iphone/ipha4375873f/ios, "Uporaba funkcije VoiceOver na iPhonu"obiskano: 19. 6. 2025.

[20] https://www.who.int/health-topics/disability#tab=tab_1, "WHO Disability", obiskano 2.7.2025.

[21] https://developer.android.com/guide/topics/ui/accessibility/principles, "Principles for improving app accessibility", obiskano 2.7.2025.

[22] https://www.w3.org/WAI/WCAG22/Understanding/intro#understanding-the-four-principles-of-accessibility, "The four principles of accessibility", obiskano 2.7.2025.

[23] https://developer.android.com/guide/topics/ui/accessibility/apps, "Make apps more accessible", obiskano 2.7.2025.

[24] https://developer.android.com/develop/ui/compose/accessibility/testing, "Accessibility Testing", obiskano 2.7.2025.

[25] https://developer.android.com/guide/topics/ui/accessibility, "Building accessible apps", obiskano 2.7.2025.

[26] https://m3.material.io/foundations/overview/principles, "Material Design", obiskano 2.7.2025.

[27] https://support.google.com/accessibility/android/answer/6006564, "Android Accessibility Overview", obiskano 4.7.2025.

[28] https://www.demandsage.com/android-statistics/, "Android Statistics", obiskano 2.7.2025.

[29] https://www.browserstack.com/accessibility-testing, "BrowserStack", obiskano 2.7.2025.

[30] https://developer.android.com/develop/ui/compose/tooling/debug, "Compose Debug", obiskano 2.7.2025.

[31] https://support.google.com/accessibility/android/answer/6151848?hl=en, "Voice Access", obiskano 2.7.2025.

Vzpostavitev skupnosti pospeševalcev uporabe umetne inteligence v EU

Milan Zorman, Bojan Žlahtič, Tanja Botić, Aleš Holobar

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
milan.zorman@um.si, bojan.zlahtic@um.si, tanja.botic@um.si, ales.holobar@um.si

Področje umetne inteligence zaradi svoje interdisciplinarne narave zahteva strokovno znanje s področij matematike, statistike, računalništva in seveda domenskega znanja na področju uporabe. Umetna inteligenca se hitro razvija, nenehno se pojavljajo novi algoritmi in tehnike, ki zahtevajo stalno učenje. Izzivi poleg tehničnih vključujejo etične vidike, ublažitev pristranskosti in preglednost v postopkih odločanja. Široka uporaba umetne inteligence v različnih panogah zahteva različne nabore spretnosti in prilagodljivost. Umetna inteligenca tako ostaja zahtevno in dinamično področje. Pospeševalci uporabe umetne inteligence imajo ključno vlogo pri učinkoviti uporabi širokega nabora znanj . Ponujajo smernice za krmarjenje po zapletenem okolju tehnologij umetne inteligence in zagotavljajo usklajenost s predpisi EU, etičnimi standardi in družbenimi vrednotami. Pomagajo opredeliti priložnosti za uvedbo umetne inteligence, jih uskladiti s posebnimi potrebami in izzivi v EU ter spodbujati inovacije in gospodarsko rast. Preko platforme EU.FFICIENT (https://eufficient.eu/) lahko obstoječi in novi pospeševalci dostopajo do najnovejših praktičnih primerov in učnega gradiva, ki jim pomaga pridobiti nove spretnosti in znanja. Platforma EU.EFFICIENT združuje štiri tematske skupnosti, in sicer, Napredno proizvodnjo, Biomedicinsko tehniko, Mobilnost ter Umetno inteligenco in digitalne tehnologije. Z zagotavljanjem centralizirane platforme za soustvarjanje pospešujemo in izboljšujemo sodelovanje med akademskim svetom in industrijo ter tako ustvarjamo pozitiven vpliv na družbo.

Ključne besede:
umetna inteligenca,
CEF,
skupnost pospeševalcev,
uporaba UI v Evropi,
EU.FFICIENT.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.22

1 Uvod

Področje umetne inteligence zaradi svoje interdisciplinarne narave zahteva strokovno znanje s področij matematike, statistike, računalništva in seveda domenskega znanja na področju uporabe. Umetna inteligenca se hitro razvija, nenehno se pojavljajo novi algoritmi in tehnike, ki zahtevajo stalno učenje. Izzivi poleg tehničnih vključujejo etične vidike, ublažitev pristranskosti in preglednost v postopkih odločanja. Široka uporaba umetne inteligence v različnih panogah zahteva različne nabore spretnosti in prilagodljivost. Poleg tega je iskanje svetega grala umetne inteligence – splošne inteligence – še vedno izmuzljivo in od raziskovalcev zahteva premikanje meja, premagovanje teoretičnih in praktičnih ovir. Umetna inteligenca tako ostaja zahtevno in dinamično področje.

Pospeševalci imajo ključno vlogo v EU pri učinkoviti uporabi širokega nabora znanj na različnih področjih. Pospeševalci uporabe umetne inteligence ponujajo smernice za krmarjenje po zapletenem okolju tehnologij umetne inteligence in zagotavljajo usklajenost s predpisi EU, etičnimi standardi in družbenimi vrednotami. Pomagajo opredeliti priložnosti za uvedbo umetne inteligence, jih uskladiti s posebnimi potrebami in izzivi v EU ter spodbujati inovacije in gospodarsko rast. Poleg tega vzpodbujajo in omogočajo sodelovanje med različnimi zainteresiranimi stranmi, vključno z vlado, industrijo, akademskimi krogi in civilno družbo ter spodbujajo izmenjavo znanja, sodelovanje in soustvarjanje. Podpirajo pobude za krepitev zmogljivosti (ang. Capacity-Building Initiatives) ter usposabljajo strokovnjake in oblikovalce politik za odgovorno in etično uporabo umetne inteligence.

2 Motivacija

Pospeševalci ne potrebujejo le tehničnih veščin, kot so tehnično znanje ali poznavanje intelektualne lastnine, pač pa tudi mehke veščine. Najučinkovitejši pospeševalci so spretni komunikatorji, ki obvladajo aktivno poslušanje, empatijo in veščine reševanja konfliktov ter so sposobni spodbujati sodelovanje vseh deležnikov. To je še posebej pomembno, kadar soustvarjalci delujejo kot pospeševalci, saj niso več zgolj proizvajalci znanja, temveč igrajo tudi vloge posrednikov znanja, izvajalcev sprememb, strokovnjakov za učenje, znanstvenikov, pospeševalcev in vodij projektov. [1].

Pospeševalci imajo tudi poglobljeno razumevanje akademskega in industrijskega okolja ter izzivov in priložnosti, ki se pojavijo pri združitvi teh dveh svetov. Sposobni so se orientirati v različnih kulturah, jezikih in prioritetah različnih deležnikov ter najti skupne temelje, da bi kar najbolje izkoristili proces soustvarjanja. Prav tako so sposobni vzpostaviti kulturo zaupanja med partnerji, kar močno koristi procesu soustvarjanja. S svojim delom pomagajo krepiti ustvarjalnost, odprtost in medsebojno razumevanje ter so ključni za zagotavljanje učinkovitosti procesa soustvarjanja [2].

3 Metodologija

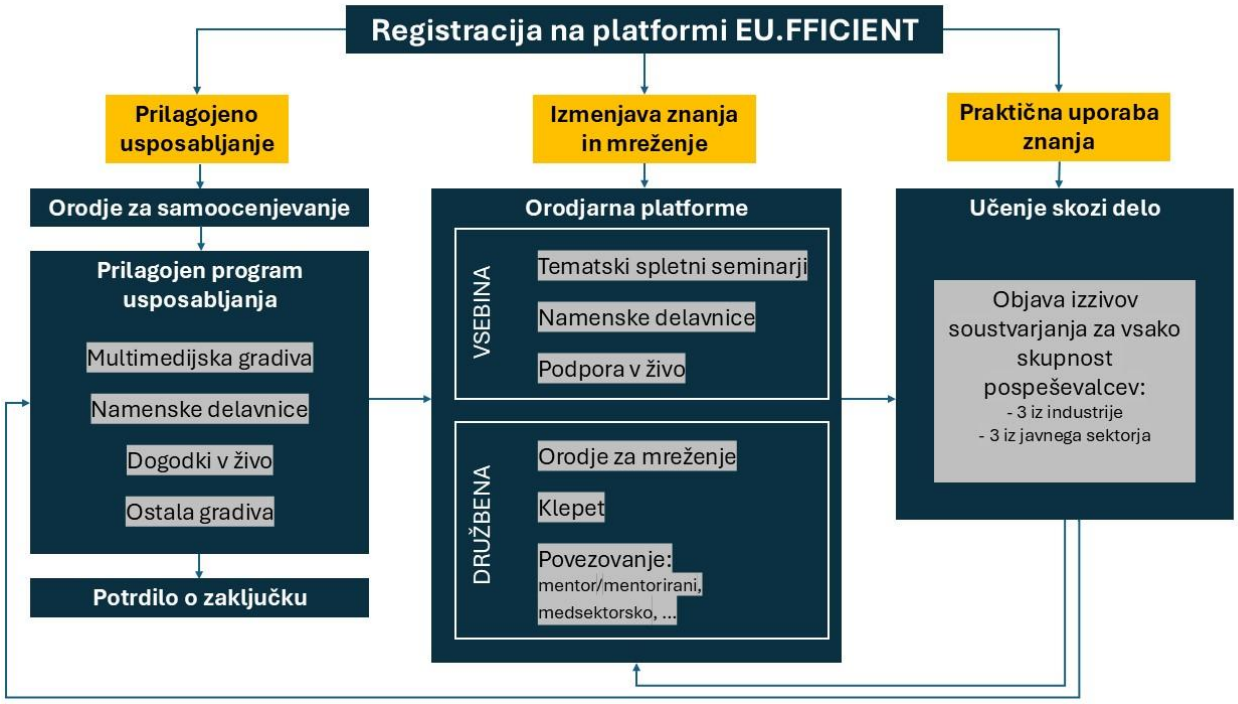
Koncept, ki ga implementira platforma EU.FFICIENT [3], združuje dva gradnika:

(i.) Metodologijo za izgradnjo štirih skupnosti pospeševalcev (Slika 1) in

(ii.) Inovativno spletno platformo, ki olajša izvajanje metodologije projekta (Slika 2).

V projektu EU.FFICIENT (Obzorje Evropa) smo začeli z oblikovanjem štirih skupnosti pospeševalcev (SP, angl. CEF - Communities of Expert Facilitators): Napredna proizvodnja, Biomedicinske tehnologije, Mobilnost in Umetna inteligenca/digitalne tehnologije. Slednji v tem prispevku posvečamo največ pozornosti.

SP umetne inteligence se nekoliko razlikuje od drugih skupnosti, saj umetna inteligenca predstavlja tudi ključno omogočitveno tehnologijo na vseh ostalih treh navedenih področjih. Cilji vseh so odkriti obstoječe vrzeli med začetniki in naprednimi pospeševalci in podpreti procese soustvarjanja; usposobiti druge pospeševalce o posebnih temah, povezanih s soustvarjanjem; izmenjava znanj med pospeševalci (inter- in intradisciplinarno); pomagati uskladiti ponudbo in povpraševanje po inovacijah ter analizirali vpliv novih trendov/tehnologij v procesih soustvarjanja.



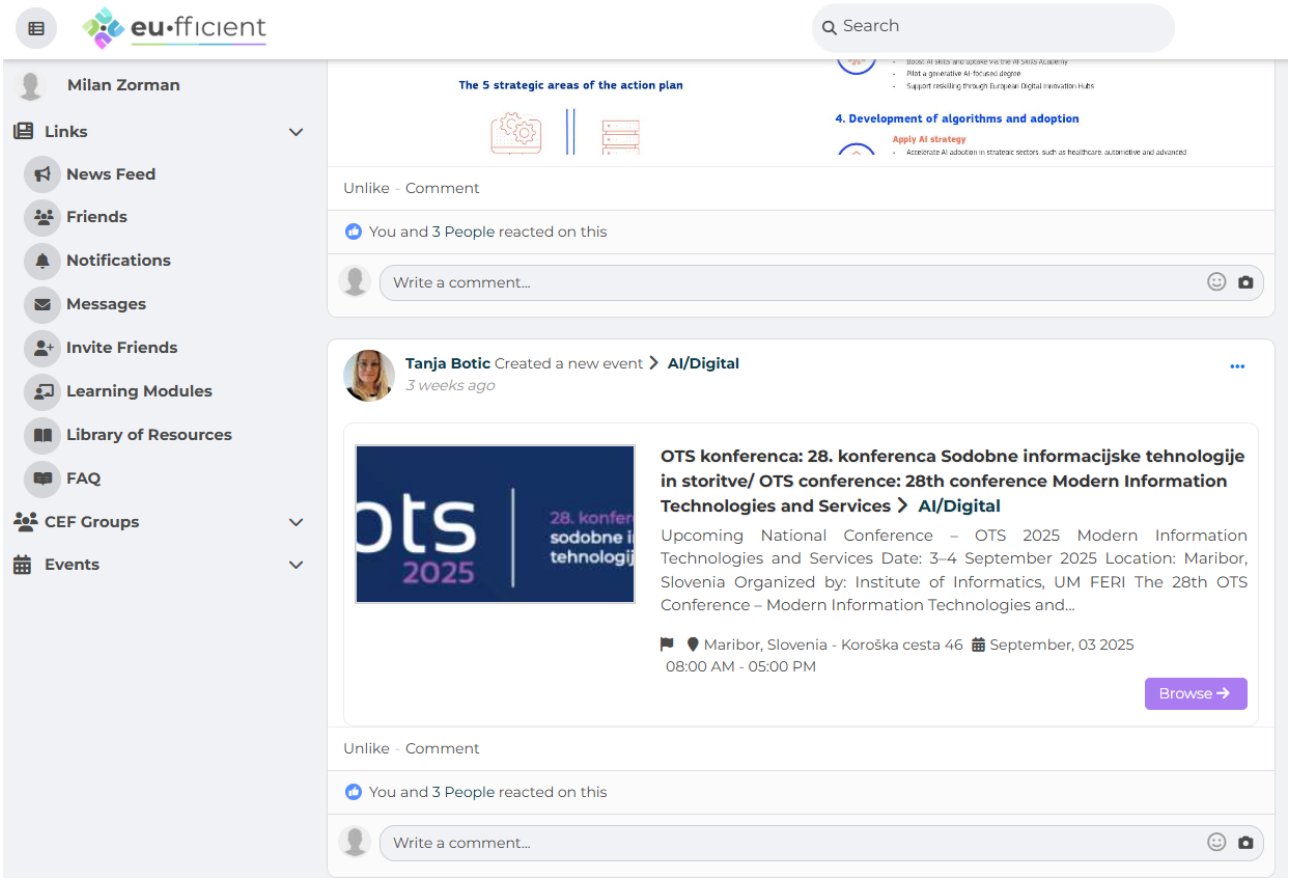
Slika 1: Metodologija projekta EU.FFICIENT.

Pospeševalci umetne inteligence se morajo odzivati na posebnosti napredka umetne inteligence. Imajo poglobljeno tehnično znanje na področju umetne inteligence, strojnega učenja in podatkovne znanosti, kar jim omogoča vodenje razprav o kompleksnih temah, kot so algoritmi, upravljanje podatkov in vrednotenje modelov. Za razliko od pospeševalcev na mnogih drugih področjih morajo pospeševalci umetne inteligence premostiti vrzel med visoko tehnično podkovanimi člani ekipe na eni strani in poslovnimi deležniki na drugi ter zagotoviti usklajenost s širšimi cilji [4, 5].

Druga ključna razlika je njihovo zavedanje etičnih vidikov in izzivov, značilnih za umetno inteligenco. Med nje sodijo pristranskost, pravičnost in preglednost. To znanje pospeševalcem omogoča, da vodijo odgovoren razvoj umetne inteligence. Poleg tega hiter tempo napredka umetne inteligence zahteva nenehno učenje, kar je, v primerjavi z nekaterimi drugimi področji, še posebej izrazita zahteva.

Pospeševalci umetne inteligence pogosto pomembno prispevajo k oblikovanju inovacij in strategije ter izkoriščajo svoje strokovno znanje za doseganje konkurenčne prednosti. Medtem ko pospeševalci na drugih področjih prav tako igrajo strateške vloge, tehnična intenzivnost in transformativni potencial umetne inteligence zahtevata posebno mešanico znanj, zaradi česar so pospeševalci umetne inteligence v edinstvenem položaju za obvladovanje kompleksnosti izvajanih projektov.

Drugi ključni element koncepta EU.FFICIENT je vzpostavljena spletna platforma (Slika 2). Razvita z vitkim pristopom sledi čim večji stroškovni učinkovitosti. Platforma povezuje procese in orodja, ki izhajajo iz metodologije EU.FFICIENT. Trenutno podpira pilotno izvajanje štirih SP, vključno z usposabljanjem pospeševalcev, funkcionalnostmi povezovanja in mreženja, knjižnico virov za soustvarjanje, učnega gradiva ter objavo izzivov od sodelujočih za sodelujoče.



Slika 2: Platforma EU.FFICIENT. [6]

3.1. Pospeševalci in soustvarjalci v kontekstu projekta EU·FFICIENT

Čeprav je umetna inteligenca globalni fenomen, se v EU.FFICIENT osredotočamo predvsem na vključevanje strokovnjakov za umetno inteligenco v EU. Regulativni in politični okviri v EU, kot so predpisi o varstvu podatkov in etične smernice za razvoj in uvedbo umetne inteligence, predstavljajo omejitve, ki niso nujno obravnavane izven EU.

Pospeševalci so posamezniki, ki imajo potrebne veščine in strokovna znanja za vodenje in spodbujanje sodelovanja med sodelujočimi deležniki, tako imenovani soustvarjalci. Pospeševalci običajno delajo za visokošolske zavode in razvojno raziskovalne-raziskovalne organizacije, kjer lahko opravljajo različne funkcije, kot so uradniki za prenos tehnologije, poslovni koordinatorji, poslovni razvijalci, uradniki za evropske/raziskovalno-razvojne in inovacijske projekte ter številne druge. Zaposlujejo jih lahko tudi drugi posredniki v inovacijskem ekosistemu, kot so grozdi (npr. vodje grozdov) in znanstveni in inovacijski parki (npr. vodje inovacij). Nekatera specializirana svetovalna podjetja nudijo tudi storitve pospeševanja za podporo projektom soustvarjanja. Nenazadnje lahko najdemo tudi pospeševalce, ki delajo tako v podjetjih (zlasti v velikih družbah) kot v javnih organizacijah, kot so občine ali javna podjetja, kjer lahko opravljajo različne vloge (npr. vodje oddelka inovacij).

Soustvarjalci v tem kontekstu pomenijo posameznike (iz akademskih krogov, industrije ali javnega sektorja), ki imajo neposredne koristi od procesa soustvarjanja.

Vloga pospeševalcev je vse prej kot lahka. Učinkovit pospeševalec mora imeti impresiven in raznolik nabor znanj in spretnosti. Zahtevano je usposabljanje in redno poznavanje najnovejših trendov in najboljših praks v soustvarjanju. Poleg tega je za pospeševalce vse bolj pomembno, da poznajo ustrezne discipline družboslovja in humanistike (npr. etiko ali vedenjske vidike) ter druge ključne vidike, kot je spodbujanje raziskav, ki upoštevajo enakost spolov in so vključujoče. Spremljanje vsega tega znanja za marsikoga predstavlja precejšen izziv. Čeprav obstaja veliko pobud, ki ponujajo vire za nadgradnjo veščin soustvarjanja pospeševalcev, so ti viri pogosto

razdrobljeni in razpršeni preko stotin platform in spletnih mest. Da bi rešili to težavo, smo vzpostavili namensko platformo in metodologijo, ki pospeševalcem zagotavlja enostaven in prilagojen dostop do najboljših virov za soustvarjanje, pa tudi priložnosti za usposabljanje in mreženje.

3.2. Skupnost Umetne inteligence in digitalnih tehnologij

Skupnost Umetne inteligence in digitalnih tehnologij se nekoliko razlikuje od ostalih treh skupnosti projekta EU.FFICIENT, saj pokriva širok spekter vplivov umetne inteligence in digitalnih tehnologij na večino sektorjev. Nadaljnji razvoj in integracija umetne inteligence in digitalnih tehnologij omogoča industriji avtomatizacijo procesov, optimizacijo proizvodnje in sprejemanje informacijsko podprtih odločitev. S svojim potencialom za povečanje produktivnosti, zmanjšanje stroškov in izboljšanje rezultatov povzročajo te tehnologije pomembne spremembe v številnih panogah, ki presegajo napredno proizvodnjo, mobilnost in zdravje. Na primer, v energetski industriji umetna inteligenca pomaga optimizirati porabo energije, zmanjšati emisije ogljika in povečati učinkovitost energetskih omrežij. Vendar pa umetna inteligenca prinaša tudi nove izzive, povezane z zasebnostjo, kakovostjo podatkov, izvorom podatkov in etiko. Slednje so temelj prizadevanj za zaupanja vredno umetno inteligenco, ki jih vodi Evropska komisija, in ki jih je mogoče obravnavati le z ustrezno integracijo družbenih in humanističnih ved.

4 Izzivi

Hiter razvoj umetne inteligence je preoblikoval panoge, ustvaril nove priložnosti in spodbudil pomembne inovacije. Vendar pa povpraševanje po strokovnem znanju na področju umetne inteligence presega razpoložljivost usposobljenih strokovnjakov, kar vodi do vrste izzivov, s katerimi se morajo spopasti tako organizacije kot posamezniki. Ena glavnih skrbi je pomanjkanje usposobljenih strokovnjakov za umetno inteligenco, kar ima za posledico zelo konkurenčen trg dela, kjer se podjetja trudijo pritegniti in obdržati vrhunske talente. To pomanjkanje ne le ovira razvoj in uvajanje tehnologij umetne inteligence, temveč tudi povečuje neenakosti med organizacijami, ki si lahko privoščijo vlaganje v umetno inteligenco, in tistimi, ki si tega ne morejo privoščiti.
Interdisciplinarna narava umetne inteligence (ta združuje elemente računalništva, matematike, inženirstva in specifičnega znanja na posameznih področjih), strokovnjakom otežuje pridobitev celovitega nabora znanj in spretnosti, potrebnih za odličnost na dotičnem področju. Sledenje hitremu razvoju tehnologij in metodologij umetne inteligence je še ena pomembna ovira, saj sta za ohranjanje konkurenčnosti potrebna nenehno učenje in prilagajanje. Poleg tega etični vidiki, regulatorni mehanizmi EU in potreba po odgovornem razvoju umetne inteligence povečujejo kompleksnost vloge, saj morajo strokovnjaki uravnotežiti inovacije z vplivom na družbo.

V projektu EU.FFICIENT posvečamo posebno pozornost naslednjim omejitvam:

1. *Porazdelitev nabora talentov*: Čeprav strokovno znanje na področju umetne inteligence obstaja po vsem svetu, porazdelitev ni enotna. Nekatere regije ali države imajo lahko bolj uveljavljene ekosisteme umetne inteligence, kar vodi do koncentracije strokovnjakov. To lahko v EU povzroči izzive pri privabljanju in ohranjanju vrhunskih talentov na področju umetne inteligence.

2. *Konkurenčnost trga*: Svetovno povpraševanje po strokovnjakih za umetno inteligenco ustvarja konkurenčen trg, kjer podjetja, raziskovalne ustanove in vlade po vsem svetu iščejo usposobljene strokovnjake. EU se lahko sooči z izzivi pri tekmovanju z drugimi regijami ali sektorji, ki ponujajo višje plače, boljše delovne pogoje ali ugodnejše priseljenske politike za talente na področju umetne inteligence.

3. *Izobraževanje in usposabljanje*: Razvoj strokovnega znanja na področju umetne inteligence zahteva specializirano izobraževanje in usposabljanje na področjih, kot so računalništvo, matematika in

podatkovna znanost. Razpoložljivost in kakovost izobraževalnih programov, povezanih z umetno inteligenco, v EU vplivata na število strokovnjakov, ki se izobražujejo v posameznih državah.

4. *Regulativno in politično okolje*: Regulativni in politični okviri znotraj EU, kot so predpisi o varstvu podatkov (npr. GDPR) in etične smernice za razvoj in uporabo umetne inteligence, lahko oblikujejo privlačnost regije za strokovnjake za umetno inteligenco. Kompleksni ali omejevalni predpisi lahko strokovnjake odvrnejo od odločitve za delo v EU.

5. *Mednarodno sodelovanje*: Čeprav je strokovno znanje na področju umetne inteligence globalno, je sodelovanje med strokovnjaki prek meja bistvenega pomena za napredek na tem področju. EU lahko izkoristi mednarodne pobude za sodelovanje, da bi dostopala do strokovnega znanja in virov zunaj svojih meja, s čimer bi zmanjšala omejitve za domače talente.

Odpravljanje teh omejitev zahteva večplasten pristop, vključno z naložbami v izobraževanje in raziskave, spodbujanjem podpornega ekosistema za inovacije na področju umetne inteligence, izboljšanjem regulativnih okvirov za privabljanje talentov in spodbujanjem mednarodnega sodelovanja za izkoriščanje strokovnega znanja z vsega sveta.

5 Zaključek

Cilj projekta EU.FFICIENT je ustvariti, implementirati in validirati metodologijo za naše skupnosti pospeševalcev, ki vključuje ključne deležnike iz industrije, akademskih krogov, javnega sektorja in družbe, s ciljem izboljšanja strokovnega vodenja procesov soustvarjanja v Evropi.

Pospeševalci kot zaupanja vredni svetovalci in katalizatorji uvajanja umetne inteligence prispevajo k prizadevanjem EU za izkoriščanje potenciala umetne inteligence za reševanje družbenih izzivov, spodbujanje digitalne preobrazbe in povečanje konkurenčnosti na svetovni ravni.

Preko platforme EU.FFICIENT lahko obstoječi in novi pospeševalci dostopajo do najnovejših praktičnih primerov in učnega gradiva, ki jim bo pomagalo pridobiti nove spretnosti in znanja. Pričakujemo koristi od sinergij med člani štirih tematskih skupnosti, kar lahko privede do novih soustvarjenih projektov in partnerstev. Z zagotavljanjem centralizirane platforme za soustvarjanje pospešujemo in izboljšujemo sodelovanje med akademskim svetom in industrijo ter tako ustvarjamo pomemben vpliv na družbo.

Skupnosti pospeševalcev bodo ne glede na osnovno področje (poleg osnovnih štirih bomo identificirali in postavili skupnosti za še dve področji) imeli digitalno podporo in bodo ponujali virtualne spletne module za podporo usposabljanju, mreženju in praktičnemu učenju pospeševalcev, ki se pridružujejo različnim skupnostim za soustvarjanje in vrednotenje znanja.

Zahvala

Delo, predstavljeno v dokumentu, je bilo financirano iz programa Obzorje Evropa, projekt EU.FFICIENT, GA št. 101135297. Izražena stališča in mnenja so mnenja avtorjev in ne odražajo nujno stališč in mnenj EU ali HaDEA (European Health and Digital Executive Agency). Posledično Evropska unija in HaDEA ne odgovarjata za navedbe v prispevku.

Literatura

[1] WIEBECK Victoria, ELIASSON Karin, NESET Tina-Simone »Co-creation research for transformative times: Facilitating foresight capacity in view of global sustainability challenges«. Environmental Science & Policy, Letnik 128, 2022, str. 290-298.

[2] SCHWARZ Roger, The Skilled Facilitator: A Comprehensive Resource for Consultants, Facilitators, Managers, Trainers, and Coaches, Jossey-Bass, 2016.

[3] eufficient.eu, Spletna stran projekta EU.FFICIENT, obiskano 23. 7. 2025

[4] JOHANSEN Bob, The New Leadership Literacies: Thriving in a Future of Extreme Disruption and Distributed Everything, Berrett-Koehler Publishers, 2017.

[5] PETREVSKA NECHKOSKA Renata, MANCHESKI Gjorgji, POELS Geert. Facilitation in Complexity: From Creation to Co-creation, from Dreaming to Co-dreaming, from Evolution to Co-evolution, Springer, 2023.

[6] community.eufficient.eu/home, Platforma EU.FFICIENT, obiskano 23. 7 2025.

Standardi in smernice za digitalne javne storitve

Vesna Vidmar, Vesna Križ Purić, Monika Zupan

Ministrstvo za digitalno preobrazbo Republike Slovenije, Ljubljana, Slovenija
vesna.vidmar@gov.si; vesna.kriz-puric@gov.si; monika.zupan@gov.si

Digitalizacija javnih storitev je del digitalne preobrazbe in predstavlja razvojni cilj Digitalne Slovenije do leta 2030 in osrednji del Strategije digitalnih javnih storitev 2030. Ključni cilj projekta Razvoj novih dinamičnih elektronskih storitev (DES) je poenotenje dostopa do digitalnih javnih storitev z uvedbo enotnih standardov in smernic za spletne storitve in mobilne aplikacije. Javna uprava tako postaja nosilec digitalne preobrazbe slovenske družbe. Na Ministrstvu za digitalno preobrazbo si prizadevamo za partnersko sodelovanje in soustvarjanje uporabniku prijaznih storitev. Vzpostavitev enotnih standardov za digitalne javne storitve omogoča dosledne funkcionalne, vizualne in tehnične zahteve, kar prispeva k bolj dostopnim, varnim, enostavnim in učinkovitim rešitvam. Z enotnimi standardi bomo dosegli višjo kakovost, večjo učinkovitost in prihranke pri razvoju in vzdrževanju digitalnih storitev v javni upravi ter spodbudili širšo uporabo digitalnih kanalov. Pripravljene so že bile smernice za mobilne storitve, ki so že javno dostopne na Portalu nacionalnega interoperabilnostnega okvira (NIO), ki je osrednje nacionalno spletno mesto za objavo digitalnih rešitev in izdelkov javnega sektorja. Za podporo razvijalcem načrtujemo mobilno razvojno platformo (angl. mobile app framework) z vnaprej pripravljenimi gradniki, aplikacijskimi programskimi vmesniki oz. API-ji (ang. Application Programming Interface), odprto in modularno izvorno kodo za posamezne komponente in prototipom aplikacije.

Ključne besede:
digitalne javne storitve,
uporabniška izkušnja,
interoperabilnost in dostopnost,
mobilna razvojna platforma,
soustvarjanje,
trajnostnost.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.23

1 Uvod

Digitalizacija javnih storitev predstavlja enega temeljnih stebrov digitalne preobrazbe Slovenije do leta 2030, kot jo opredeljuje *Strategija digitalnih javnih storitev 2030 (v nadaljevanju: SDJS2030)* [2][3]. Z načrtnim prehodom v digitalno delovanje, javni sektor državljanom in podjetjem omogoča dostop do sodobnih, varnih, vključujočih in uporabniku prilagojenih storitev.
Med štiri ključne cilje SDJS 2030 sodijo:

- razvoj uporabniško usmerjenih in vključujočih digitalnih storitev,
- povečanje zaupanja v digitalno javno upravo,
- krepitev učinkovitosti in odpornosti javnega sektorja ter
- razvoj digitalnih zmogljivosti.

V tem okviru projekt Razvoj novih dinamičnih elektronskih storitev (DES), sofinanciran iz *Načrta za okrevanje in odpornost (NOO)*, predstavlja enega izmed osrednjih strateških instrumentov za uresničevanje teh ciljev. Ključni poudarek projekta je *Poenotenje dostopa do digitalnih javnih storitev*.
Prvi korak k poenotenju digitalnih javnih storitev je vzpostavitev enotnih standardov in smernic za digitalne javne storitve. Ti vključujejo pravila, tehnične okvire in oblikovne smernice, ki omogočajo enotno, varno in učinkovito izvajanje ter razvoj storitev. Poleg izboljšane uporabniške izkušnje pričakujemo tudi pomembne sistemske prihranke in višjo kakovost digitalnih rešitev.
Enotni standardi in smernice za digitalne javne storitve morajo biti zasnovani tako, da podpirajo vse ključne storitve javne uprave, omogočajo njihovo povezovanje ter zagotavljajo enotno in enakovredno uporabniško izkušnjo za vse – tako na nacionalni ravni kot v čezmejnem prostoru.
Na Ministrstvu za digitalno preobrazbo se zavedamo, da so partnersko sodelovanje, soustvarjanje in globoko razumevanje potreb uporabnikov ključni pogoji za razvoj sodobnih, kakovostnih in trajnostnih digitalnih rešitev. Le na ta način je mogoče zagotoviti tudi, da so enotni standardi in smernice dejansko ustrezni za vse digitalne storitve javne uprave.
Del standardov, ki se nanaša na mobilne storitve, je že vzpostavljen in javno dostopen na portalu NIO – *Smernice za razvoj mobilnih aplikacij*[24][1]. Smernice ponujajo celovit nabor informacij za razvoj kakovostnih in varnih mobilnih aplikacij, skladnih s tehnološkimi standardi in pričakovanji uporabnikov. Vključujejo tudi kriterije za presojo smiselnosti uvedbe mobilne aplikacije.
Za podporo tehnični izvedbi standardizacije mobilnih rešitev je v načrtu vzpostavitev mobilne razvojne platforme, ki bo razvijalcem omogočila dostop do ključnih gradnikov[25][4], vključno z vnaprej pripravljenimi knjižnicami, programskimi vmesniki (API-ji), odprto in modularno izvorno kodo ter prototipom aplikacije. Ta infrastruktura bo pomembno prispevala k pospešenemu in usklajenemu razvoju digitalnih storitev v javni upravi.
Z uvedbo enotnih standardov in smernic za digitalne javne storitve (za spletne in mobilne kanale) bomo okrepili kakovost, prepoznavnost in uporabo digitalnih kanalov v javni upravi ter omogočili usklajen, celovit in stroškovno učinkovit pristop k razvoju in vzdrževanju digitalnih rešitev.

[24] Smernice za razvoj mobilnih aplikacij
[25] V skladu s Smernicami MDP za razvoj informacijskih rešitev je gradnik informacijsko podprta funkcionalnost, ki jo različne aplikacije uporabljajo za ponovno uporabo.

2 Namen

Osrednji cilj projekta DES je *vzpostavitev poenotenega dostopa* do digitalnih javnih storitev, z uvedbo enotnih oblikovnih, funkcionalnih in tehničnih standardov – smernic za spletne storitve in mobilne aplikacije.
Poenotenje ne pomeni centralizacije, temveč *uskladitev uporabniške izkušnje* prek različnih portalov in aplikacij – ne glede na institucijo, ki storitev nudi. Cilj je, da uporabniki dostopajo do storitev prek enotno zasnovanih, preglednih in intuitivnih vmesnikov, kar povečuje uporabnost ter zmanjšuje potrebo po učenju in raziskovanju vedno novih sistemov in rešitev.

Ključni cilji vključujejo:

1. **Konsistentna uporabniška izkušnja**
   Z enotnimi standardi se zagotovi, da so vse digitalne storitve (spletne in mobilne) oblikovane po enakih načelih, kar uporabnikom omogoča lažjo uporabo in boljšo orientacijo.
2. **Večja dostopnost in vključevanje**
   Standardi vključujejo smernice za dostopnost. To pomeni, da bodo storitve primerne tudi za ranljive skupine (npr. za starejše, osebe z oviranostmi, ipd.).
3. **Večja varnost in zanesljivost**
   Tehnični standardi pomagajo zagotoviti, da so storitve varne, stabilne in skladne z zakonodajo (npr. GDPR).
4. **Učinkovitost in prihranki**
   Z uporabo skupnih gradnikov, knjižnic in orodij *oblikovalskega sistema*[26] (ang.design system) bomo zagotovili večje prihranke (kadrovski, finančni viri) pri razvoju in vzdrževanju informacijskih rešitev.
5. **Podpora razvoju in inovacijam**
   Razvijalci imajo na voljo skupno razvojno okolje (npr. mobilno razvojno platformo), kar spodbuja hitrejši in kakovostnejši razvoj rešitev.
6. **Prepoznavnost in zaupanje**
   Uporabniki lažje prepoznajo uradne digitalne storitve, kar krepi zaupanje v javno upravo.

Z uvedbo skupnih pravil, smernic in tehničnih okvirov si prizadevamo za *učinkovitejše, zanesljivejše, uporabne in trajnostne* rešitve v digitalnem prostoru javne uprave.

3 Struktura in razvoj standardov

Razvoj enotnih standardov za digitalne javne storitve temelji na partnerskem sodelovanju med MDP in ključnimi deležniki iz različnih institucij javnega sektorja. Proces smo zasnovali kot odprto, inkluzivno in iterativno sodelovanje, ki zagotavlja, da so standardi prilagojeni realnim potrebam uporabnikov ter operativnim izzivom organov javne uprave.
V okviru medresorske delovne skupine, v kateri sodeluje 27 institucij, smo izvedli poglobljeno analizo obstoječih digitalnih storitev v Sloveniji in pregledali relevantne mednarodne dobre prakse. Metodološki pristop je vključeval zbiranje podatkov na podlagi poglobljenih pogovorov in intervjujev s skrbniki storitvenih portalov in aplikacij, fokusnih skupin in delavnic s strokovnjaki in skrbniki storitev, kar je omogočilo identifikacijo ključnih izzivov, potreb in pričakovanj uporabnikov.

Na podlagi pridobljenih vpogledov smo oblikovali konceptualni okvir, ki vključuje temeljna načela, širše smernice in konkretne, merljive standarde. Celoten razvojni proces poteka v duhu agilnosti in stalnega usklajevanja, kar

[26] Oblikovalski sistem je nabor ponovljivih komponent in smernic, ki omogočajo dosledno in učinkovito oblikovanje digitalnih storitev. Vključuje vizualne elemente; barve, tipografijo, ikone, uporabniške vmesniške elemente - (angl. User Interface) komponente; gumbi, obrazci ipd. ter pravila za njihovo uporabo.

omogoča prilagoditve glede na povratne informacije ter zagotavlja, da so končni standardi tehnološko ustrezni in omogočajo integrirano, usklajeno, varno in učinkovito interakcijo državljanov in podjetij z javno upravo.

Standardi, ki so še v fazi razvoja, so zasnovani v skladu s strokovnimi izhodišči projekta ter vpeljujejo *tristopenjsko strukturo:*

- *Načela* – temeljno vodilo ali vrednota (*npr. »dostopnost, preglednost, varnost«),*
- *Smernice* – Konkretizacija načela, ki daje bolj splošno usmeritev
  (npr. »zagotovi se enoten in logično strukturiran dostop do digitalnih storitev z uporabo usklajenih navigacijskih vzorcev, terminologije in iskalnih funkcij.«),
- *Standardi* – konkretne, preverljive zahteve in priporočila (označene z *MORA*, *NE SME*, *PRIPOROČA SE*).

3.1. Temeljna načela enotnih standardov

1. **Preglednost in intuitiven dostop** – enoten katalog storitev, iskalniki, strukturiran dostop:
   Digitalne storitve morajo biti enostavno dostopne in dosledno opisane. Sistematično urejen in voden centralni katalog storitev uporabnikom omogoča lažji dostop ter zagotavlja upravljalcem pregled nad celotnim portfeljem storitev.
2. **Usmerjenost k uporabniku**, **uporabnost** – testiranje uporabnosti, zbiranje povratnih informacij, stalne izboljšave:
   Digitalne storitve so v celoti osredotočene na uporabnika. Njihovo načrtovanje, oblikovanje in delovanje je usmerjeno k doslednemu zagotavljanju odlične uporabniške izkušnje – storitve so soustvarjene z relevantnim deležniki, intuitivne za uporabo, učinkovite pri doseganju ciljev ter zadovoljujejo dejanske potrebe in pričakovanja ljudi.
3. **Dostopnost za vse** – skladnost z WCAG[27], zagotavljanje alternativnih kanalov:
   Digitalne storitve morajo biti zasnovane in razvite tako, da so dostopne in uporabne za vse ljudi, ne glede na njihove zmožnosti, oviranosti ali uporabljeno tehnologijo.
4. **Kakovost in jasnost vsebine** – razumljiv jezik, ažurnost, prepoved nerazumljivih kratic:
   Vsebine o digitalnih storitvah morajo biti kakovostne in ažurne, poimenovanja pa jasna, jedrnata, dosledna in enostavno razumljiva ciljnim uporabnikom, da zagotavljajo zanesljive, relevantne in pravočasne informacije.
5. **Vizualna konsistentnost, enotnost in prepoznavnost** – storitve so vizualno in funkcionalno usklajene, kar se odraža v enotnem izgledu in dosledni interakciji, doseženi z uporabo skupnega oblikovalskega sistema. Takšna konsistentnost krepi prepoznavnost in uporabo digitalnih storitev.
6. **Varnost in zaupanje** – visoki varnostni standardi, preglednost postopkov, varstvo osebnih podatkov.
   Digitalne storitve morajo vlivati zaupanje na način, da zagotavljajo najvišjo raven varnosti in zasebnosti za uporabnike ter odgovorno in transparentno ravnanje s podatki uporabnikov.
7. **Tehnična odličnost in interoperabilnost** – odprti API-ji, upoštevanje tehničnih standardov, omogočanje povezljivosti in izmenjave podatkov:
   Digitalne storitve morajo biti zgrajene na trdnih tehničnih temeljih, slediti sodobnim standardom in omogočati povezljivost ter izmenjavo podatkov.

[27] Spletne strani in vsebine so pripravljene v skladu z mednarodnimi smernicami za dostopnost spletnih vsebin in ustrezajo standardu WCAG (Web Content Accessibility Guidelines).

4 Mobilne aplikacije: smernice in podpora za kakovostni razvoj

Smernice za razvoj mobilnih aplikacij, so namenjene predvsem organom državne uprave. Ponujajo metodološki okvir za presojo, ali je razvoj mobilne aplikacije upravičen ali bi bilo učinkoviteje prilagoditi obstoječo spletno rešitev za mobilne naprave.

Poleg tega smernice vsebujejo pregled prednosti in omejitev posameznih pristopov ter tehnične in organizacijske korake, potrebne za uspešno načrtovanje, razvoj, testiranje in implementacijo mobilnih storitev. Smernice izpostavljajo, da razvoj mobilnih aplikacij ne sme temeljiti zgolj na tehnoloških trendih, temveč na jasno opredeljenih potrebah uporabnikov in organizacije. S spodaj navedenimi kriteriji želimo institucijam olajšati odločitev o smiselnosti uvedbe mobilne aplikacije. Z analizo želimo zagotoviti, da bo aplikacija sledila načelom dobre prakse v javni upravi. To pomeni, da bo uporabnikom omogočala dostop do koristnih informacij in hkrati poenostavljala postopke, ki jih ti opravljajo v okviru javnih storitev.

Ključni kriteriji za presojo ustreznosti uvedbe mobilne aplikacije vključujejo:

- ciljno skupino uporabnikov in njihovo digitalno dostopnost,
- potrebo po obveščanju v realnem času,
- možnost uporabe funkcij mobilnih naprav,
- pogostost uporabe,
- potencial za avtomatizacijo in optimizacijo obstoječih procesov.

Smernice ponujajo strukturiran pristop, ki omogoča učinkovito načrtovanje in izvedbo projektov razvoja mobilnih aplikacij v javnem sektorju.

Življenjski cikel mobilne aplikacije je razdeljen na faze načrtovanja, javnega naročila, razvoja, testiranja, objave, obratovanja in podpore ter upokojitve. Posebna pozornost je namenjena postopku javnega naročanja, kjer je pomembno jasno opredeliti tehnične zahteve in preveriti reference izvajalcev. Testiranje vključuje preverjanje uporabniške izkušnje z različnimi skupinami uporabnikov.

Objava mobilne aplikacije poteka prek uradnih trgovin (Google Play, App Store, AppGallery), pri čemer je treba zagotoviti ustrezno dokumentacijo in promocijo. Podpora uporabnikom je organizirana na treh ravneh, z jasno določenimi kanali za komunikacijo in reševanje težav. Prvi nivo je kontaktni center, ki rešuje pogosta vprašanja s pomočjo ažurne dokumentacije. Če težave ne more rešiti, jo posreduje skrbniku aplikacije, ki oceni, ali gre za napako ali poda odgovor. Napake se nato predajo tretjemu nivoju – razvojni ekipi, ki jih odpravi.

Avtentikacija uporabnikov je ključna za zaščito osebnih podatkov. Dokument opisuje različne metode, vključno z uporabniškim imenom in geslom, večstopenjsko avtentikacijo, biometrijo ter uporabo storitve SI-PASS, ki omogoča varno prijavo in elektronski podpis za domače in tuje uporabnike.

V smernicah so predstavljeni tudi enotni standardi za mobilne aplikacije v državni upravi. Ti vključujejo zahteve glede poimenovanja aplikacij, verzioniranja, lokalizacije, tehničnih specifikacij (npr. podpora za večino naprav), vsebinske in oblikovne politike (npr. začetni zaslon z grbom Republike Slovenije, uporaba tipografije Republika, barvna paleta celostne grafične podobe Republike Slovenije) ter dostopnosti. Mobilne aplikacije morajo biti skladne z Zakonom o dostopnosti spletišč in mobilnih aplikacij (ZDSMA) ter smernicami WCAG. To vključuje podporo za prilagoditev velikosti pisave, opise slik za govorne bralnike in druge funkcionalnosti, ki omogočajo uporabo aplikacij tudi osebam s posebnimi potrebami.

Za zagotavljanje tehnične podpore v procesu standardizacije razvoja mobilnih rešitev načrtujemo vzpostavitev mobilne razvojne platforme. Ta bo razvijalcem omogočala dostop do ključnih gradnikov, orodij, med drugim vnaprej pripravljenih vgrajenih knjižnic, programskih vmesnikov (API-jev), odprte in modularne izvorne kode za posamezne komponente, kar bo razvijalcem olajšalo delo in zagotovilo skladnost z obstoječimi smernicami, kot so celostna grafična podoba Republike Slovenije, enotni standardi spletnih mest državne uprave ter smernice za razvoj mobilnih aplikacij.

Namen platforme je poenotenje in pospešitev razvoja mobilnih rešitev, ki se v praksi pogosto izvajajo nepovezano, z različnimi uporabniškimi vmesniki in funkcionalnostmi.

Cilj je razvoj osnovnih gradnikov, ki vključujejo oblikovni gradnik, gradnik za izmenjavo podatkov, gradnik za avtentikacijo, gradnik za večjezičnost aplikacij ter gradnik za plačilni sistem. Vsak gradnik bo fleksibilen, razširljiv in opremljen z ustrezno dokumentacijo. Oblikovni gradnik bo vključeval knjižnico grafičnih komponent, ki bodo skladne z zakonodajo o dostopnosti in omogočene bodo tudi prilagoditve. Gradnik za izmenjavo podatkov bo omogočal standardizirano komunikacijo z zalednimi sistemi, medtem ko bo gradnik za avtentikacijo podpiral različne metode prijave, vključno z biometrijo, večfaktorsko avtentikacijo in integracijo s storitvijo SI-PASS. Gradnik za večjezičnost bo omogočal enostavno upravljanje prevodov, medtem ko bo gradnik za plačilni sistem omogočal integracijo z UJP[28] plačilnim sistemom in JEP[29] cenikom.
Vse navedeno bo predstavljalo pomembno podporo pri razvoju digitalnih storitev in mobilnih aplikacij v javni upravi.

5 Stanje implementacije in naslednji koraki poenotenja dostopa do digitalnih javnih storitev

V nadaljevanju predstavljamo ključne dosežke projekta do danes in načrtovane aktivnosti za obdobje 2025–2026, ki bodo zagotovile nadaljnji razvoj in uspešno implementacijo enotnih standardov ter smernic.
Ključni dosežki do sedaj:

- oblikovana medresorska delovna skupina s 27 institucijami,
- izvedba obsežne analize obstoječega stanja in tujih dobrih praks (statusna analiza, »benchmarking«, intervjuji, delavnice),
- priprava načrta poenotenega dostopa do leta 2030 in priprava osnutka standardov,
- izvedba več delavnic z agilnimi metodami dela za namen vsebinske uskladitve in zasnove standardov,
- širitev zavedanja o pomenu vključevanja uporabnikov, sodelovanja med deležniki ter soustvarjanja digitalnih storitev.

V letih 2025 in 2026 načrtujemo naslednje aktivnosti:

- pilotna uvedba osnutka enotnih standardov in smernic na portalu eUprava, nadgradnja standardov z ugotovitvami in priprava prototipa idealnih scenarijev digitalnih storitev/portalov;
- razvoj oblikovalskega sistema;
- oblikovanje sistema spremljanja skladnosti;
- usposabljanje javnih uslužbencev in podpora organom javne uprave.

V nadaljnjih fazah razvoja bomo enotne standarde in smernice sistematično nadgrajevali z novimi vsebinskimi sklopi, ki bodo zajemali celovito zasnovo digitalnih storitev, vključno z razvojem *oblikovalskega sistema* ter *standardizacijo uporabniških vmesnikov*.
Poleg tega načrtujemo *integracijo rešitev umetne inteligence* kot orodja za optimizacijo učinkovitosti in personalizacijo digitalnih storitev, s čimer bomo podprli napredne uporabniške scenarije ter izboljšali odzivnost in prilagodljivost sistemov.
Pomemben del nadaljnjega razvoja bo tudi vzpostavitev prepoznavne *blagovne znamke* oziroma *certifikata kakovosti,* ki bo zagotavljal skladnost storitev z enotnimi standardi ter krepil *zaupanje uporabnikov* ter deležnikov javnega sektorja. Za uresničitev poenotenega dostopa do digitalnih javnih storitev bo ključno, da se javne institucije postopoma, a dosledno prilagodijo enotnim standardom in smernicam ter novi krovni blagovni znamki. Ključno podporo bodo pri tem predstavljali mehanizmi za centralno spremljanje skladnosti ter svetovanje in pomoč pri uvajanju rešitev, predvidenih v okviru SDJS 2030.

[28] UJP – Uprava za javna plačila Republike Slovenije
[29] JEP – Jedro elektronskih postopkov; del **informacijske infrastrukture za digitalno poslovanje** z javnim sektorjem

Celostni pristop, ki vključuje enotne standarde in smernice, oblikovalski sistem, vključevanje rešitev umetne inteligence in vzpostavitev sistema spremljanja skladnosti, bo prispeval k *večji interoperabilnosti*, zagotavljanju *visokih standardov varnosti* in *dostopnosti* ter *spodbujal inovacije* v digitalni javni upravi.
Na ta način sledimo strateškim ciljem SDJS 2030, med katerimi izpostavljamo:

- zagotovljeno je učinkovito in varno okolje za opravljanje digitalnih storitev,
- vse digitalne storitve so soustvarjene in usmerjene v uporabnike.

6 Zaključek

Z uvedbo smernic in standardov za dosego poenotenja dostopa do digitalnih javnih storitev Slovenija uresničuje strateško zavezo k uporabniško usmerjeni, dostopni in povezani digitalni javni upravi. Standardi ne pomenijo le tehnične izboljšave, temveč so temelj zaupanja, preglednosti in enotnosti v digitalnem okolju. Z usklajenim pristopom bomo omogočili razvoj storitev, ki jih državljani dejansko potrebujejo in znajo uporabljati, hkrati pa bomo povečali prepoznavnost, učinkovitost ter krepili kulturo soustvarjanja, vključevanja uporabnikov in stalnih izboljšav.
Standardi in smernice so naš kompas v digitalnem svetu. Usmerjajo nas k učinkoviti, dostopni in uporabniku prijazni digitalni prihodnosti.
Naša skupna vizija ostaja jasna: digitalna javna uprava mora biti prepoznavna, enostavna, zanesljiva in prijazna – *enaka za vse uporabnike, ne glede na uporabljeno napravo, dostopnost ali ustanovo, ki storitev ponuja.* To ni le tehnični cilj, temveč zaveza k pravičnemu digitalnemu okolju, ki temelji na zaupanju, odprtosti in vključevanju.

Literatura

[1] Smernice za razvoj mobilnih aplikacij. Pridobljeno 28. 7. 2025 iz: Smernice za razvoj mobilnih aplikacij | Izdelki | Portal NIO

[2] Strategija digitalnih javnih storitev 2030: Strategija_digitalnih_javnih_storitev_2030-1.pdf

[3] Digitalna Slovenija 2030: https://www.gov.si/assets/ministrstva/MKRR/Strategija-razvoja-Slovenije-2030/Strategija_razvoja_Slovenije_2030.pdf

[4] Smernice MDP za razvoj informacijskih rešitev. Pridobljeno 28. 7. 2025 iz: Smernice za razvoj mobilnih aplikacij | Izdelki | Portal NIO

OTS 2025 Sodobne informacijske tehnologije in storitve

Zbornik osemindvajsete 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 osemindvajsete konference OTS 2025 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,
obdelava velepodatkov in podatkovnih tokov,
metode agilnega razvoja,
tehnologije veriženja blokov,
kibernetska varnost.

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

COMTRADE GAMING

dem
Dravske elektrarne Maribor
Skupina hse

EQUALEYES

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.
INFORMATIKA

MIKROpis®

msg PLAUT

niceHash

NOVUMRGI

IRC
RC IRC Celje, d.o.o. / Ltd.

SAVA ZAVAROVALNICA

SOFTWARE SYNERGY
softergee

SRC

tridens
technology

MEDIJSKI POKROVITELJ

Računalniške novice
www.racunalniske-novice.com

ISO 27001, 14001, 9001

INOVA IT

Web & Mobile | IoT | AI | VR | Digital Twins

PRIJAVE IN INFORMACIJE

careers@inova.si

COMTRADE GAMING

About us

We are an independent software provider, delivering open gaming platforms and professional services to both the online and land-based gaming sectors with more than 20 years of experience.

Comtrade Gaming's strengths are the development of technology solutions based on industry standards and legislative requirements.

We are dedicated to fostering a supportive and inclusive workplace culture that values and respects the diverse backgrounds, perspectives, and talents of our employees, empowering them to contribute, collaborate, and grow professionally.

iCore

iGaming Platform

sCore

Gaming Management System

gCore

Game development and distribution platform

Follow us:

comtradegaming.com

Vodilni proizvajalec električne energije iz obnovljivih virov

Z energijo narave ustvarjamo zeleno prihodnost.

Zavezani trajnostnemu razvoju ter zanesljivim in strateškim rešitvam postavljamo temelje energetske prihodnosti – odgovorni do okolja, predani družbi.

EQUALEYES

Vaš tehnološki partner za digitalne rešitve

EQUALEYES je inovativno tehnološko podjetje s 120-člansko ekipo, ki zagotavlja celovite razvojne storitve – od načrtovanja in oblikovanja do razvoja in testiranja programske opreme.

120+ članov ekipe

Naša ekipa združuje širok spekter talentov – od backend in frontend razvijalcev, do oblikovalcev in projektnih vodij.

10+ let izkušenj v industriji

Z več kot desetletjem prisotnosti na trgu smo pridobili bogate izkušnje pri razvoju aplikacij in digitalnih platform za različne industrije, vključno s fintech, e-commerce in ticketingom.

Sodelovanje s startupi in globalnimi podjetji

Naš pristop temelji na prilagodljivosti – bodisi pri hitri gradnji MVP-jev ali skaliranju kompleksnih sistemov za večja podjetja.

Več kot 50 uspešno zaključenih projektov

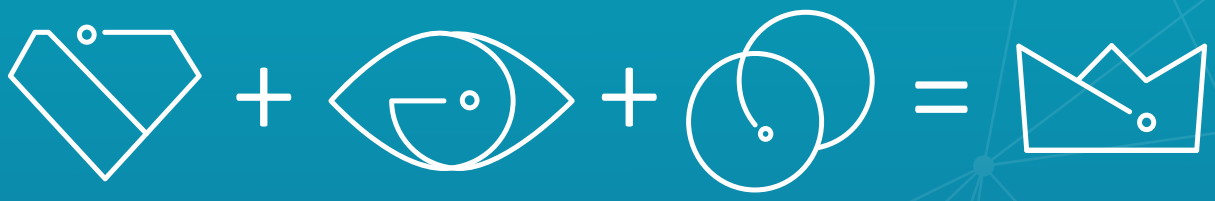
Vsak projekt je priložnost, da uporabimo svoje znanje in ustvarimo rešitve, ki ne samo delujejo, ampak tudi navdušijo uporabnike.

www.equaleyes.com

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.

INFORMATIKA

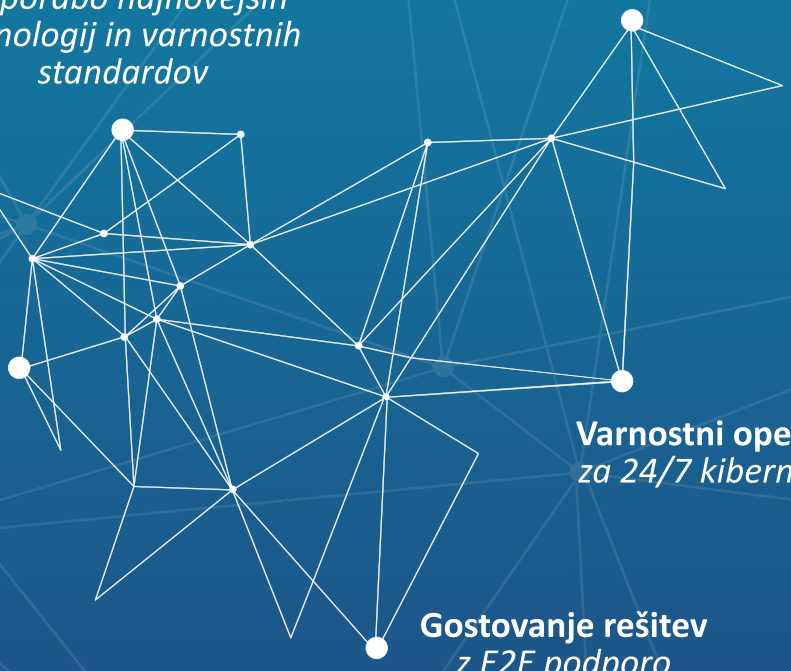
pravi partner za vas



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 kibernetsko 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

Mikropis Holding
Aškerčeva 4a, SI-3310 Žalec, Slovenia, EU
info@mikropis.com

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

ZANESLJIV PARTNER ZAVAROVALNIC

msg Plaut UAP d.o.o. je že več kot 25 let zanesljiv in inovativen partner zavarovalnic. Naše rešitve podpirajo vsako izmed naših strank z omogočanjem po meri prilagojene racionalizacije in avtomatizacije poslovnih procesov, fleksibilnega razvoja izdelkov in optimalne učinkovitosti.

OBSEG STORITEV

- Razvoj celostnih informacijskih sistemov
- Razvoj mobilnih aplikacij
- Rešitve v oblačnih storitvah v mikrostoritveni arhitekturi z uporabo vmesnikov Docker
- Digitalizacija procesov
- Orodje in specializiranost za migracijo podatkov

MSG. UAP – TEHNOLOGIJA, KI DELUJE

- digitalizacija življenjskega cikla
- oblačna rešitev
- odprt API
- SaaS ali na lokaciji
- visoke zmožnosti avtomatizacije
- celovit DevOps
- orodje za migracijo podatkov
- preprosto in stroškovno učinkovito upravljanje

+ 386 2 2356200

andrej.kline@msg-plaut.com

msg-plaut-uap.com

nicehash

THE HASHRATE MARKETPLACE

Vodilni ponudnik platforme za rudarjenje kriptovalut

www.nicehash.com

NOVUMRGI

RGI Group
European leader in the **digital transformation** of the insurance sector

Standard **software** and **cloud solutions** for the **international insurance market**

RC IRC Celje, d.o.o.

Ul. XIV. divizije 14, 3000 Celje

5. DESETLETJE SOUSTVARJAMO INFORMACIJSKO DOBO

- Informacijski sistemi za zdravstvene ustanove
- Informacijski sistemi za celovito podporo poslovnih procesov
- Prenova poslovnih procesov in racionalizacija poslovanja
- Informacijska podpora za poslovno odločanje in upravljanje
- Certifikata kakovosti informacijske tehnologije in informacijske varnosti
- Storitve svetovanja pri uvedbi standardov kakovosti in informacijske varnosti
- Poslovno in informacijsko svetovanje
- Celovito vzdrževanje informacijskih sistemov

Poslovna področja

- Zdravstvo
- Televizija
- Visoko šolstvo
- Telekomunikacije
- Industrija
- Banke

Informacijska cesta, ki povezuje slovenske zgodbe o uspehu.

Inovativnost v službi uspešnosti

RC IRC d.o.o.
Ulica XIV. divizije 14
3000 Celje

T. +386(0)3 427 42 00
F. +386(0)3 427 41 98
E. info@rcc-irc-si
W. www.rcc-irc.si

ISO 9001/TickIT BUREAU VERITAS Certification

ISO 27001 BUREAU VERITAS Certification


ZAV-SAVA.SI
Z roko v roki
nismo nikoli sami.
Zato vam v vsakem trenutku življenja
ponudimo svojo, ko nas potrebujete.
NIKOLI SAMI
SAVA
ZAVAROVALNICA

Z inženirsko natančnostjo

gradimo digitalno prihodnost.

Združujemo strokovno znanje,
inovacije in najnovejše tehnologije.

Spoznajte več o
nas in področjih

www.src.si

SRC

Ljubljana-Naklo-Maribor-Beograd

SRC MB, Gosposvetska cesta 29, 2000 Maribor
+386 1 600 7000, www.src.si

tridens
technology

Tridens je mariborsko IT podjetje, ki je v digitalnem prostoru prisotno že več kot 18 let.

V podjetju se ukvarjamo z razvojem lastnih produktov:

Tridens Monetization • **Tridens EV Charge**

Naša vizija je biti prepoznani kot zaupanja vreden partner za inovativne programske rešitve po celem svetu.

Zaupajo nam:

cisco · TelekomSlovenije · BYD · iskraemeco BY ELSEWEDY ELECTRIC · telemach

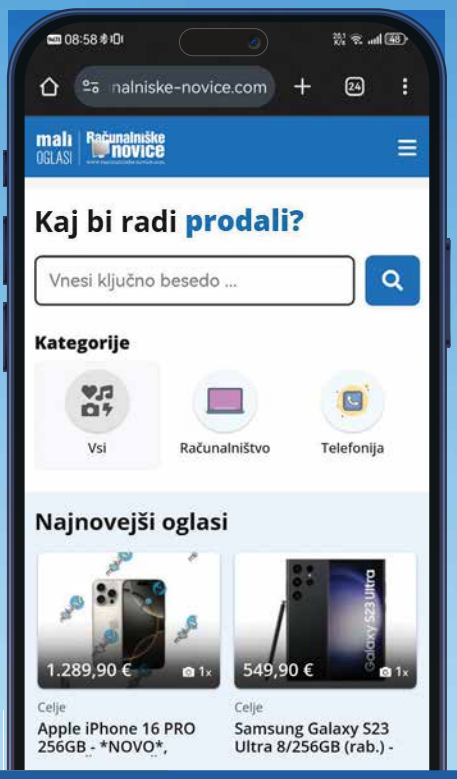
Zaposlujemo!

Poglej prosta delovna mesta na naši spletni strani:
https://tridenstechnology.com/careers/
ali poskeniraj QR kodo.

Računalniške novice

www.racunalniske-novice.com

mali OGLASI



WWW.RACUNALNISKE-NOVICE.COM/MALI-OGLASI

KUPI PRODAJ

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

COMTRADE GAMING

dem Dravske elektrarne Maribor Skupina hse

EQUALEYES

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O. INFORMATIKA

MIKROfis

msg PLAUT

niceHash

NOVUMRGI

RC IRC Celje, d.o.o. / Ltd.

SAVA ZAVAROVALNICA

SOFTWARE SYNERGY softergee

SRC

tridens technology

MEDIJSKI POKROVITELJ

Računalniške novice
www.racunalniske-novice.com

ISBN 978-961-299-036-7

INŠTITUT ZA INFORMATIKO

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

IoT
Varnost
Kriptografija
Web 3
Podatkovne tehnologije
Umetna inteligenca
Poslovna analitika