Univerzitetna založba
Univerze v Mariboru

27. konferenca
sodobne informacijske
tehnologije in storitve

ots 2024

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 2024

Sodobne informacijske tehnologije in storitve

Zbornik 27. konference

Uredniki
Luka Pavlič
Tina Beranič
Marjan Heričko

September 2024

Naslov
Title
OTS 2024 Sodobne informacijske tehnologije in storitve
OTS 2024 Advanced Information Technologies and Services

Podnaslov
Subtitle
Zbornik 27. konference
Conference Proceedings of the 27th 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 2024, Čučko, 2024

Grafične priloge
Graphics material
Vsi viri so lastni, če ni navedeno drugače. Avtorji in Pavlič, Beranič, Heričko (uredniki), 2024

Konferenca
Conference
OTS 2024 Sodobne informacijske tehnologije in storitve

Kraj in datum
Location and date
Maribor, Slovenija, 4. in 5. september 2024

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, Lucija Brezočnik, Saša Brdnik, Luka Četina, Mitja Gradišnik, Tjaša Heričko, Katja Kous, Alen Rajšp, Patrik Rek, Vasilka Saklamaeva (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 2024

Dostopno na
Available at
https://press.um.si/index.php/ump/catalog/book/903

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.946.5:004.7(082)(0.034.2)

SODOBNE informacijske tehnologije in storitve (konferenca) (27 ; 2024 ; Maribor)
OTS 2024 [Elektronski vir] : sodobne informacijske tehnologije in storitve : zbornik 27. konference : [Maribor, 4. in 5. september 2024] / uredniki Luka Pavlič, Tina Beranič, Marjan Heričko. - 1. izd. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2024

Način dostopa (URL): https://press.um.si/index.php/ump/catalog/book/903
ISBN 978-961-286-893-2 (WEB, pdf)
doi: 10.18690/um.feri.4.2024
COBISS.SI-ID 204853763

ISBN 978-961-286-893-2 (pdf)
978-961-286-894-9 (mehka vezava)
978-961-286-895-6 (USB ključ)

DOI https://doi.org/10.18690/um.feri.4.2024

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.). (2024). *OTS 2024 Sodobne informacijske tehnologije in storitve: Zbornik 27. konference.* Univerza v Mariboru, Univerzitetna založba. doi: 10.18690/um.feri.4.2024

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 2024 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. Velik poudarek prispevkov v letošnjem zborniku je na cevovodih neprekinjene dostave informacijskih rešitev, kot tudi spletnih ali mobilnih uporabniških vmesnikih. Še posebej izpostavljene vsebine se letos nanašajo na kibernetsko varnost, sodobne pristope k hranjenju in obdelavi podatkov, ter predpisom in standardizaciji, ki smo ji informatiki izpostaljeni vsakodnevno.

Veseli nas, da konferenca OTS še naprej prispeva k boljši povezanosti IT strokovnjakov, informatikov, arhitektov in razvijalcev naprednih IT rešitev in storitev, kot tudi akademske sfere in gospodarstva.

Ponosni smo na izjemne uspehe sodelujočih podjetij, ki preko znanja, poguma in sposobnosti, da se odlične tehnološko-organizacijske rešitve uspešno vpeljejo v regiji in širše, predstavljajo visoko dodano vrednost za njihove uporabnike.

izr. prof. dr. Luka Pavlič	doc. dr. Tina Beranič	prof. dr. Marjan Heričko
vodja programskega odbora OTS 2024	vodja organizacijskega odbora OTS 2024	predsednik konference OTS 2024

KAZALO

Kako ukrotiti velik jezikovni model nad lokalnim korpusom

Vili Podgorelec, Tadej Lahovnik, Grega Vrbančič

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

V prostrani, neukročeni divjini umetne inteligence se je pojavila in postavila v ospredje nova generativna vrsta: veliki jezikovni modeli. Ti orjaki s svojimi milijardami parametrov tavajo po podatkovni pokrajini, lačni vzorcev in vpogledov v obilje besedil. Toda izkoriščanje njihove moči ni enostaven podvig. Lahko so nepredvidljivi, nagnjeni k halucinacijam in se pogosto težko držijo teme. Predvsem pa jih ni preprosto udomačiti. V članku predstavljamo pristop, s katerim lažje ukrotimo te velikane z uporabo pomenskega povezovanja z lokalnim korpusom besedil. Pristop združuje surovo moč velikih jezikovnih modelov s specifičnostjo in ustreznostjo lokalnih podatkov. Raziskali bomo, kako tak pristop omogoči ne le učinkovito, temveč tudi vsebinsko specifično generiranje odgovorov, pri čemer so zagotovljene natančne in podrobne informacije o vašem specifičnem podjetju ali panogi. V članku poskušamo na preprost način predstaviti zapletenost tovrstnega procesa usposabljanja, vključno s strateško uporabo generiranja z razširjenim iskanjem, ki našemu modelu omogoča učinkovit dostop do lokalnih virov znanja. Če bomo v svojem namenu uspeli, vas prispevek ne bo opremil le s kakšnim novim spoznanjem, ampak boste tudi pripravljeni, da se podate na lastno pustolovščino z generativno umetno inteligenco. Na tej pustolovščini se vam rade volje pridružimo, da skupaj ukrotimo velikega dobrodušnega velikana. Naj se torej krotenje začne!

Ključne besede:
generativna umetna inteligenca
veliki jezikovni modeli
semantično povezovanje
vektorske vložitve
prilagoditev modela

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.1

1 Uvod

Generativna umetna inteligenca (GAI) in veliki jezikovni modeli (LLM), kot je npr. ChatGPT, so v zadnjih letih dosegli izjemen napredek in prepoznavnost. Ti modeli so usposobljeni na ogromnih korpusih besedil, kar jim omogoča izjemno natančno modeliranje in posledično sposobnost generiranja besedil v naravnem jeziku. Zaradi obsežnega učenja nad ogromno količino besedil so ti modeli pridobili široko in poglobljeno razumevanje jezika ter znanje s številnih področij, kar bi lahko primerjali z neko vrsto splošne izobrazbe. Njihova uporabnost se kaže na različnih področjih, predvsem pa so se izkazali pri aplikacijah za odgovarjanje na vprašanja v naravnem jeziku. Sistemi za odgovarjanje na vprašanja, ki temeljijo na LLM, omogočajo uporabnikom, da pridobijo hitre, natančne in kontekstno ustrezne odgovore na kompleksna vprašanja. Takšni sistemi so izjemno koristni na različnih področjih, vključno s tehnično podporo, izobraževanjem, raziskovanjem ali celo v zdravstveni oskrbi. Moč teh sistemov je v njihovi sposobnosti, da razumejo naravni jezik na visoki ravni in se prilagodijo specifičnim potrebam uporabnikov.

Kljub njihovi široki uporabnosti se pogosto pojavi potreba po tem, da se LLM osredotočijo na specifično področje znanja, ki je lahko predstavljeno v lokalnih dokumentih, do katerih model ni imel dostopa med svojim učenjem. Tradicionalni pristop bi zahteval ponovno učenje ali fino uglaševanje modela na novih podatkih, kar pa je proces, ki zahteva izjemne računske vire in je časovno zelo potraten. Ponovno učenje modela na specifičnih podatkih zahteva tudi poglobljeno tehnično znanje in dostop do zmogljive strojne opreme, kar ni vedno izvedljivo.

Da bi se izognili tem izzivom, lahko namesto tega uporabimo pristop, ki modelu omogoča dostop do lokalnih dokumentov brez ponovnega učenja. Tak pristop vključuje uporabo orodij, kot so vektorske baze podatkov in tehnike iskanja podobnosti, ki omogočajo modelu hitro pridobivanje relevantnih informacij iz lokalnih virov. Z vektorskimi bazami podatkov lahko dokumente pretvorimo v vektorske predstavitve, ki omogočajo hitro iskanje in pridobivanje najbolj relevantnih informacij na podlagi vprašanj uporabnikov. Ko LLM pridobi dostop do teh lokalnih virov, lahko informacije iz teh dokumentov spretno vključi v svoje odgovore. S tem pristopom lahko razvijalci informacijskih rešitev izkoristijo moč generativne AI, ne da bi pri tem potrebovali ogromne računske vire za ponovno učenje modelov. To omogoča učinkovitejše, lažje prilagodljive in predvsem bolj ekonomične rešitve za specifične potrebe uporabnikov. Poleg tega tak pristop omogoča hitrejše prilagajanje modela novim podatkom. Ker ni potrebe po dolgotrajnem in zahtevnem procesu ponovnega učenja, se lahko razvijalci hitro odzivajo na spremembe in posodobitve v lokalnih dokumentih. To je še posebej pomembno v dinamičnih okoljih, kjer se informacije pogosto spreminjajo in je hitra prilagodljivost ključna za zagotavljanje relevantnih odgovorov. Uporaba generativne umetne inteligence na ta način prinaša tudi dodatne prednosti, kot so izboljšana natančnost odgovorov in boljša uporabniška izkušnja. Model, ki lahko dostopa do specifičnih informacij v realnem času, je sposoben zagotavljati kontekstno ustreznejše in bolj specifične odgovore, kar povečuje zadovoljstvo uporabnikov in njihovo zaupanje v sistem.

Seveda pa je potrebno tovrsten pristop primerno zasnovati in ga ustrezno realizirati. V trenutni praksi namreč ostaja še veliko izzivov, kako navedene komponente povezati v zanesljiv in dobro delujoč sistem, pri tem pa se izogniti pastem, ki prežijo na razvijalce. V članku predstavljamo rešitev mISLec, ki smo jo zasnovali prav v duhu učinkovite uporabe velikih jezikovnih modelov nad specifičnimi zbirkami lokalnih dokumentov, pri čemer je rešitev karseda neodvisna od uporabljenega pred-naučenega LLM, uporabimo pa lahko tako obstoječe storitve preko klicev API, kot tudi lokalno nameščen veliki jezikovni model. S pravilno uporabo tehnologij in integracijo lokalnih virov informacij lahko ustvarimo zmogljive, prilagodljive in učinkovite sisteme za odgovarjanje na vprašanja v naravnem jeziku, ki ustrezajo specifičnim potrebam uporabnikov brez potrebe po obsežnem ponovnem učenju modelov.

2 Generativna umetna inteligenca in veliki jezikovni modeli

Ljudje uporabljamo jezik kot komunikacijski sistem za medsebojno posredovanje idej, čustev in informacij. Kot dojenčki zaznavamo smiselne pomene besed in z odraščanjem postanemo bolj spretni pri prilagajanju svojega govora. Jezik nam pomaga razumeti sebe in svet okoli nas. Edinstvenost človeka pri učenju jezika je sposobnost posploševanja in posledično učenja iz omejene izpostavljenosti jeziku, še posebej pri otrocih. Kar je praktično neprimerljivo s sodobnimi modeli umetne inteligence. Potrebna so bila desetletja raziskovanja in razvoja sistemov, ki bi lahko ustvarili človeku podobne odzive za naloge obdelave naravnega jezika (angl. Natural Language Processing, NLP), kot so pogovor, samo-dokončanje besedila in prevajanje jezika [1].

Generativna umetna inteligenca (angl. Generative Artificial Intelligence, GAI) in veliki jezikovni modeli (angl. Large Language Models, LLM) predstavljajo enega pomembnejših in odmevnejših napredkov na področju umetne inteligence, saj ponujajo zmogljivosti pri ustvarjalnih oz. generativnih nalogah, kot je na primer ustvarjanje vsebin, ter pri razumevanju naravnega jezika.

V zadnjih letih je generativna UI doživela bliskovit razvoj, ki ga poganjajo napredki v arhitekturah globokega učenja in razpoložljivost velikih podatkovnih zbirk ter računskih virov. Ta razvoj je privedel do pojava več-modalnih modelov, ki lahko hkrati obravnavajo besedilo in slike, kar dodatno širi obseg uporabe generativnih modelov UI. Poleg tega integracija nenadzorovanih in delno nadzorovanih algoritmov inteligentnim sistemom omogoča, da se samodejno odzivajo na ukaze in samostojno ustvarjajo vsebino, pri čemer vsebino črpajo iz predhodno ustvarjenih zbirk podatkov [2].

2.1. Generativna umetna inteligenca

Generativna umetna inteligenca je eno najhitreje razvijajočih se področij v sodobni digitalni pokrajini, ki izkazuje izjemno zmogljivost pri ustvarjanju visokokakovostne, kontekstualno ustrezne vsebine, ki je skorajda ne ločimo od tiste, ki jo ustvari človek. V obdobju, ko aplikacije, kot je ChatGPT, postavljajo rekorde za najhitreje rastočo uporabniško bazo z izkazovanjem znanja neodvisnega od domen, se GAI uveljavlja kot pomemben generator vsebin v digitalnem prostoru. Napredki na področju strojnega učenja (angl. Machine Learning, ML) in globokega učenja (angl. Deep Learning, DL) so omogočili razširitev tradicionalnih nalog umetne inteligence kot so regresija, klasifikacija in priporočila, na ustvarjanje edinstvene, realistične in kreativne vsebine [3].

GAI omogoča inovacije na različnih področjih, vključno s poslovnimi modeli in storitvami. Na primer, sistemi za podporo strankam lahko zdaj uporabljajo GAI za predlaganje ustreznih odzivov na pogovore, kar povečuje učinkovitost in kakovost storitev. Prav tako omogoča avtomatizacijo in optimizacijo procesov v podjetjih, kar lahko vodi do boljšega odločanja in izboljšanja operativnih učinkovitosti.

Najprepoznavnejše rešitve v okviru GAI vključujejo generativne modele, kot so GAN (angl. Generative Adversarial Networks), ki se uporabljajo za ustvarjanje realističnih slik (npr. rešitvi DALL-E [4] in Midjourney [5]) in videoposnetkov, ter transformerje, ki so temelj modelov, kot je ChatGPT [4], in se uporabljajo za ustvarjanje besedil. GAN modeli so pokazali izjemne rezultate na področju umetnosti, zabave in oblikovanja, kjer lahko ustvarjajo izjemno realistične in kreativne vizualne vsebine. Transformerji pa so revolucionirali področje obdelave naravnega jezika, kjer se uporabljajo za različne naloge, kot so prevajanje, povzemanje besedil in ustvarjanje pogovorov [2].

Raziskave in razvoj na področju GAI so osredotočeni na izboljšanje hitrosti, zmogljivosti in učinkovitosti teh modelov. Kljub temu pa ostajajo številna vprašanja o temeljnih načelih, aplikacijah in družbeno-ekonomskem vplivu GAI še vedno neraziskana. To predstavlja izziv, saj jasna podoba in razumevanje generativne umetne inteligence še nista dokončno oblikovana. GAI v splošnem prinaša številne priložnosti za inovacije in izboljšave v različnih industrijah, vendar je hkrati pomembno, da se zavedamo tudi izzivov, ki jih prinaša, ter si prizadevamo za odgovorno in etično uporabo te tehnologije.

2.2. Veliki jezikovni modeli

Veliki jezikovni modeli (angl. Large Language Models, LLM) so skupina pristopov temelječih na metodah in tehnikah globokega učenja, usposobljeni za razumevanje, generiranje in obdelavo besedil v naravnem jeziku. Med najbolj znanimi in uporabljenimi LLM je serija generativnih prednaučenih transformerjev (angl. Generative Pre-trained Transformer, GPT), ki vključuje različice, kot je na primer GPT-4, razvite s strani OpenAI. LLM so učeni na ogromnih količinah besedilnih podatkov, kar jim omogoča prepoznavanje kompleksnih vzorcev v jeziku ter generiranje koherentnih in kontekstualno ustreznih besedil. Osnovni princip delovanja velikih jezikovnih modelov temelji na uporabi transformerjev, napredne arhitekture nevronskih mrež, ki je bila prvič predstavljena v članku »Attention is All You Need« [6]. Tranformerji uporabljajo mehanizem pozornosti (angl. attention mechanism), ki omogoča modelu, da teži k pomembnim delom vhodnega besedila in jih poveže z ustreznimi izhodi. To bistveno izboljša zmogljivost modelov pri obdelavi zaporednih podatkov v primerjavi s predhodnimi pristopi, kot so rekurentne nevronske mreže (angl. Recurrent Neural Network, RNN) [7] in mreže z dolgim kratkoročnim spominom (angl. Long Short-Term Memory, LSTM) [8].

Učenje velikih jezikovnih modelov navadno poteka v dveh korakih: pred-učenje (angl. pretraining) in uglaševanje (angl. fine-tuning). V procesu predučenja je model izpostavljen ogromnim količinam besedil iz različnih virov, kot so knjige, spletni članki in druge oblike pisne komunikacije. V tem koraku se model uči jezikovne strukture, slovnice, konteksta in pomena besed ter fraz. Po pred-učenju je model sposoben generirati besedilo, ki je jezikovno pravilno in kontekstualno ustrezno, vendar morda ni specifično prilagojeno za reševanje določenih nalog. V procesu uglaševanja se model dodatno uči na manjših, bolj specializiranih sklopih podatkov, ki so prilagojeni specifičnim nalogam, kot so prevajanje, povzemanje besedil, odgovarjanje na vprašanja ali analiza sentimenta. To omogoča modelu, da izboljša svojo uspešnost in uporabnost pri reševanju specifičnih problemov.

Osnovni gradniki velikih jezikovnih modelov vključujejo več ključnih komponent. Med najpomembnejše sodita vektorizacija in vložitev (angl. embedding). Vektorizacija je proces pretvorbe besedilnih podatkov v številčne predstavitve, ki jih lahko obdelujejo nevronske mreže. Vložitev pa je tehnika, ki omogoča pretvorbo besed v vektorje z nižjo dimenzionalnostjo, kjer so besede s podobnimi pomeni bližje druga drugi v vektorskem prostoru. Te vektorske predstavitve omogočajo modelu razumevanje semantičnih odnosov med besedami in frazami, kar je ključnega pomena za generiranje kontekstualno ustreznega besedila. Uporaba vektorizacije in vložitve omogoča modelom, da bolje razumejo in obdelujejo kompleksne jezikovne strukture. Vektorizacija prav tako zagotavlja, da so vsi podatki predstavljeni v obliki, ki je primerna za matematično obdelavo, medtem ko vložitev omogoča učinkovitejše in natančnejše razumevanje pomena in konteksta besed. Na primer, model lahko prepozna, da sta besedi "kralj" in "kraljica" semantično povezani ter da imata podobne kontekste uporabe, kar omogoča generiranje bolj naravnih in koherentnih besedil.

Veliki jezikovni modeli imajo široko paleto aplikacij, ki segajo od avtomatskega prevajanja in generiranja besedil do analize sentimenta in pogovornih robotov. Njihova sposobnost razumevanja in generiranja naravnega jezika odpira številne možnosti za izboljšanje komunikacije med ljudmi in stroji. Vendar pa uporaba LLM prinaša tudi izzive, kot so vprašanja o pristranskosti, etičnih vidikih in vplivu na zasebnost. Kljub temu ostajajo veliki jezikovni modeli ena najbolj obetavnih tehnologij v sodobnem razvoju umetne inteligence, ki že zdaj pomembno vpliva na številna področja znanosti in industrije.

2.3. Vektorizacija, žetoni in vložitev

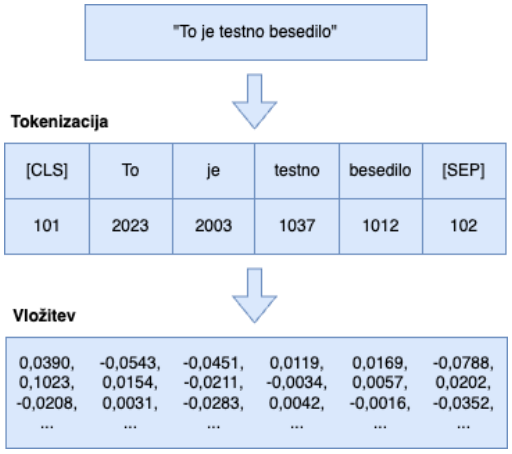
Vektorji igrajo ključno vlogo pri delovanju LLM in GAI. Da bi razumeli njihov pomen, je bistveno razumeti, kaj so vektorji in kako se ustvarjajo ter uporabljajo v LLM. V matematiki in fiziki je vektor objekt, ki ima velikost ter smer in se tipično uporabljajo za predstavitev količin, ki jih ni mogoče v celoti opisati z eno samo številko kot so sila, hitrost ali premik, saj imajo tako velikost kot tudi smer. Na področju LLM se vektorji uporabljajo za predstavitev besedila ali podatkov v numerični obliki, ki jo lahko model razume in obdela. Ta predstavitev je znana kot vložitev (angl. embedding). Vložitve so visokodimenzionalni vektorji, ki zajemajo semantični pomen besed,

stavkov, lahko tudi celotnih dokumentov. Postopek pretvorbe besedila v vložitev omogoča LLM-jem, da izvajajo različne naloge obdelave naravnega jezika, kot so ustvarjanje besedila, analiza občutkov in povzemanje besedil. Ker stroji razumejo samo številke, se podatki, kot so besedilo in slike, pretvorijo v številske vektorje [9].

Izvedba različnih operacij nad takšnimi vektorji, kot je na primer izračun kosinusne podobnosti, lahko uporabimo za namen iskanja oz. ugotavljanja podobnosti med vektorji oz. v primeru procesiranja naravnega besedila ugotavljanja podobnosti med besedami ali besedili. Vektorizacija in vložitev omogočata modelom, da razumejo in obdelujejo kompleksne jezikovne strukture. Prav tako pa zagotavlja, da so vsi podatki predstavljeni v obliki, ki je primerna za matematično obdelavo, medtem ko vložitev omogoča učinkovitejše in natančnejše razumevanje pomena in konteksta besed.

Žetoni (tokens) so osnovne enote podatkov, ki jih obdelujejo LLM-ji. V kontekstu besedila je žeton lahko beseda, del besede (koren besede) ali celo znak, odvisno od pristopa oz. načina tokenizacije. Ko je besedilo posredovano skozi tokenizator, se ustvarjeni žetoni zakodirajo na podlagi specifične sheme kodiranja v vektorje, ki jih LLM lahko razume. Z uporabo tokenizatorja vhodno besedilo razdelimo na besede, dele besed (npr. koren) ali celo posamezne znake, odvisno od uporabljenega tokenizerja. Ko gredo žetoni skozi dekoder, jih je mogoče enostavno znova prevesti v besedilo. Običajno se dolžina konteksta LLM-jev nanaša na enega ključnih dejavnikov razlikovanja med različicami LLM. Tehnično gledano gre za sposobnost LLM, da sprejme določeno število žetonov kot vhod in ustvari drug niz žetonov kot izhod. Tokenizator je prav tako odgovoren za kodiranje poziva LLM (vnosa) v žetone in odgovora LLM (izhoda) nazaj v besedilo [10].

Z vložitvijo dosežejo LLM globoko razumevanje jezika, kar omogoča različne naloge, kot so analiza sentimenta, povzemanje besedila in odgovarjanje na vprašanja z različnimi nivoji razumevanja in zmožnostmi ustvarjanja. So vstopna točka v LLM, vendar se uporabljajo tudi zunaj LLM za pretvorbo besedila v vektorje ob ohranjanju semantičnega konteksta. Na Slika 1 je prikazan postopek tokenizacije in vložitve ene povedi.



Slika 1: Primer tokenizacije in vložitve.

Najpogostejši pristopi k vektorizaciji in vložitvam vključujejo različne tehnike. Ena izmed najpogostejših tehnik je uporaba Word2Vec, ki pretvori besede v vektorje tako, da upošteva njihov kontekst v besedilu. Metoda GloVe (angl. Global Vectors for Word Representation) je še ena priljubljena tehnika, ki temelji na statističnih podatkih o sočasnem pojavljanju besed v besedilu. BERT (angl. Bidirectional Encoder Representations from Transformers) je sodobnejša tehnika, ki uporablja dvostransko kodiranje za razumevanje pomena besed z upoštevanjem konteksta na levi in desni strani besede [10].

Vektorizacija, žetoni in vložitve so tako temeljni gradniki, ki omogočajo delovanje velikih jezikovnih modelov in so ključni za razumevanje ter generiranje naravnega jezika.

2.4. Pomensko povezovanje besedil (RAG)

Retrieval-augmented generation (RAG) predstavlja inovativni pristop k izboljšanju delovanja LLM z namenom, da bi ti lahko odgovarjali na specifična vprašanja v spremenljivem kontekstu. Temelj vseh temeljnih modelov, vključno z LLM-ji, je arhitektura transformatorjev, ki pretvarja ogromne količine surovih podatkov v stisnjene reprezentacije njihove osnovne strukture. Ta osnovna reprezentacija omogoča prilagoditev modela različnim nalogam z dodatnim finim uglaševanjem na označenem, domensko specifičnem znanju.

Pred uvedbo LLM-jev so namreč digitalni pogovorni agenti sledili ročno določenemu toku dialoga, kjer so potrdili namen sogovornika, pridobili zahtevane informacije in dostavili odgovor v obliki univerzalne šablone. Za enostavna vprašanja je ta metoda delovala dobro, vendar je imela svoje omejitve. Pričakovanje in pisanje odgovorov na vsako možno vprašanje, ki bi ga stranka lahko zastavila, je bilo zamudno. Če scenarij za neko vprašanje ni obstajal, pogovorni robot ni imel funkcionalnosti improvizacije [11].

Kljub uporabi LLM pa fino uglaševanje modelov redko zagotavlja popolno pokritost znanja, potrebnega za odgovarjanje na zelo specifična vprašanja v dinamičnem okolju. Leta 2020 je Meta (takrat znana kot Facebook) predstavila ogrodje, imenovano retrieval-augmented generation (RAG) [12], da bi LLM-jem omogočila dostop do informacij, ki presegajo njihove učne podatke. RAG omogoča LLM-jem, da se opirajo na specializirano znanje in tako odgovorijo na vprašanja na bolj natančen način. V RAG sistemu model odgovarja na vprašanja z iskanjem in brskanjem po vsebini v knjigi, namesto da bi se zanašal zgolj na svoje spominske zmožnosti.

RAG vključuje dve fazi: pridobivanje in generiranje vsebine. V fazi pridobivanja algoritmi iščejo in pridobivajo delčke informacij, ki so relevantni za uporabnikov poziv ali vprašanje. V odprtem okolju lahko te informacije prihajajo iz indeksiranih dokumentov na spletu ali iz drugih virov, v zaprtem podjetniškem okolju pa se običajno uporablja ožji nabor internih virov zaradi večje varnosti in zanesljivosti. To zunanjo znanje se nato »priloži« uporabnikovemu pozivu in posreduje jezikovnemu modelu. V fazi generiranja LLM črpa iz obogatenega poziva in svoje notranje reprezentacije učnih podatkov, da ustvari odgovor, ki je prilagojen uporabniku v tistem trenutku. Odgovor se nato posreduje klepetalnemu botu skupaj s povezavami do virov [11]. Poenostavljen koncept RAG je predstavljen na sliki Slika 2.

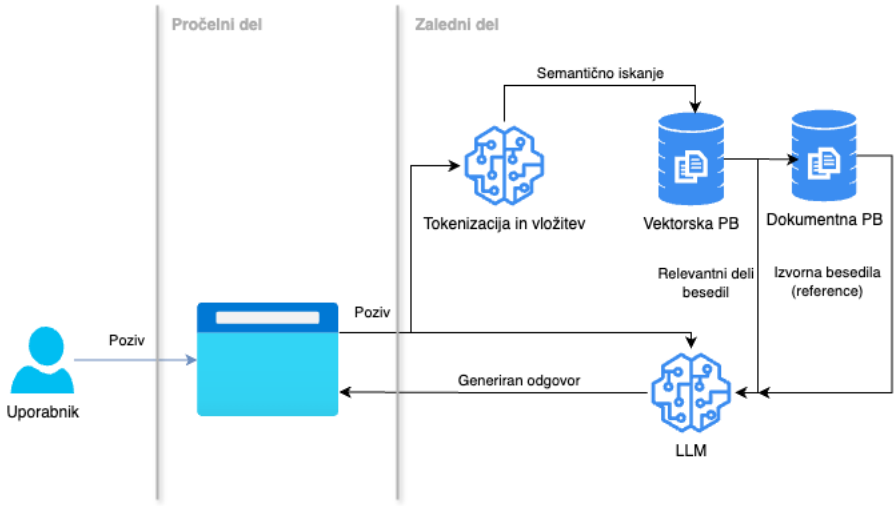
Slika 2: Poenostavljen koncept delovanja RAG.

RAG je trenutno najbolj znano orodje za povezovanje LLM-jev z najnovejšimi, preverljivimi informacijami in zmanjševanje stroškov nenehnega ponovnega učenja in posodabljanja modelov. RAG se zanaša na sposobnost obogatiti uporabniške pozive z relevantnimi informacijami, ki so shranjene v vektorjih, matematičnih predstavitvah podatkov. Vektorske baze podatkov lahko učinkovito indeksirajo, shranjujejo in pridobivajo informacije za stvari, kot so priporočilni sistemi in klepetalni boti. Vendar pa RAG ni popoln in še vedno ostajajo številni izzivi pri pravilnem izvajanju RAG. Eden ključnih je zagotovo na kak način učinkovito in efektivno pridobiti relevantne dele zunanjih podatkov oz. kako izbrati, katere informacije priložiti uporabniškemu pozivu.

3 Zasnova rešitve

Rešitev smo zasnovali z mislimi na čim enostavnejšo integracijo različnih lastnih domenskih tekstovnih korpusov, pri čemer je cilj, da je razvita rešitev neodvisna od uporabljanega velikega jezikovnega modela. Ključ pri vpeljavi lastnih domenskh korpusov je v uporabi ogradja RAG in inteligentni integraciji rezultatov tega z uporabniškim pozivom. Glede na podane želje oziroma zahteve smo zasnovali konceptualno zasnovo naše rešitve, kot je prikazano na sliki Slika 3. Po podanem uporabniškem pozivu se ta preko tokenizacije in vložitve pretvori v vektorje, ki se uporabijo za namen semantičnega iskanja podobnosti z obstoječimi vloženimi vektorji v vektorski podatkovni bazi. Vsak vgrajen vektor poseduje tudi referenco na izvoren dokument hranjen v dokumentni podatkovni bazi. Z rezultati semantičnega iskanja obogaten poziv se nato posreduje uporabljenemu LLM, ki lahko deluje lokalno ali pa v obliki REST API. Odgovor, generiran s strani LLM, posredujemo uporabniku.

Predstavljena konceptualna zasnova rešitve se navezuje na ključno funkcionalnost – pogovornega robota, obogatenega s specifičnim domenskim znanjem v obliki poljubno sestavljenih besedilnih korpusov, pri čemer ni potrebno LLM dodatno uglaševati ali učiti. V ta namen rešitev poleg osnovnje predstavljene funkcionalnosti vključuje tudi funkcionalnosti ustvarjanja različnih korpusov, tokenizacije in vgrajevanja podanih besedil v vektorsko podatkovno bazo, itd.



Slika 3: Konceptualna zasnova rešitve.

4 Rešitev mISLec

V tem poglavju bomo podrobneje predstavili razvito rešitev. Pročelje (angl. *front-end*) rešitve mISLec temelji na ogrodju React, ki omogoča razvoj ponovno uporabljivih komponent. Zaledje (angl. *back-end*) je implementirano v programskem jeziku Python in temelji na ogrodju FastAPI. Podatki in naloženi dokumenti se hranijo v NoSQL podatkovni bazi MongoDB, ki omogoča učinkovito hrambo tako strukturiranih kot tudi nestrukturiranih podatkov.

4.1. Funkcionalnosti in delovanje rešitve

Razvito rešitev sestavlja več komponent, ki omogočajo interakcijo z velikimi jezikovnimi modeli in delo z lokalnimi korpusi. Glavne komponente rešitve so:

- **navigacijski meni**, ki omogoča dostop do različnih delov rešitve,
- **upravljanje z dokumenti**, ki uporabnikom omogoča nalaganje dokumentov, pregled vseh naloženih dokumentov in urejanje vsebine (odstranjevanje odvečnih informacij, kot je npr. vsebina v glavi in nogi dokumenta),
- **vektorizacija dokumentov**, ki velikim jezikovnim modelom omogoča učinkovito obdelavo in razumevanje vsebine dokumentov,
- **upravljanje z lokalnimi korpusi**, ki uporabnikom omogoča ustvarjanje lokalnih korpusov, dodajanje dokumentov v lokalni korpus in vektorizacijo vseh dokumentov v lokalnem korpusu,
- **upravljanje z uporabniki**, ki je na voljo administratorjem in jim omogoča pregled vseh uporabnikov ter spreminjanje njihovih vlog v sklopu rešitve mISLec, in
- **interakcija z velikimi jezikovnimi modeli**, ki uporabnikom omogoča uporabo lokalnega korpusa in komunikacijo z izbranim modelom.

Vektorizacija dokumentov se izvede nad tekstovno vsebino naloženega dokumenta. Ob samem nalaganju dokumenta se njegova vsebina razdeli na manjše enote, tj. posamezne stavke, ki predstavljajo vhod za operacijo vektorizacije. Ta je implementirana s pomočjo knjižnic *transformers* in *torch*. Knjižnica *transformers* posamezne stavke tokenizira, knjižnica *torch* pa ustvarjene žetone (angl. *token*) pretvori v vektorsko obliko, nato pa vektorje še normalizira. Ustvarjeni vektorji se nato shranijo v vektorsko bazo Qdrant, ki omogoča visoko zmogljivo vektorsko iskanje v velikem obsegu. Slika 4 prikazuje implementacijo vektorizacije dokumentov.

```python
def embed_sentences(document: Document) -> Document:
    encoded_input = tokenizer(
        document['raw_sentences_text'], padding=True, truncation=True, return_tensors='pt')

    with torch.no_grad():
        model_output = model(**encoded_input)
        sentence_embeddings = model_output[0][:, 0]
    sentence_embeddings = torch.nn.functional.normalize(
        sentence_embeddings, p=2, dim=1)

    document['embeddings'] = {
        'sentence_embeddings': sentence_embeddings
    }

    operation_info = qdrant_client.upsert(
        collection_name=COLLECTION_NAME,
        wait=True,
        points=models.Batch(
            ids=[str(uuid.uuid4()) for _ in document['raw_sentences_text']],
            payloads=[
                {
                    'text': sentence,
                    'document_id': str(document['_id']),
                    'user_id': str(document['user']),
                }
                for sentence in document['raw_sentences_text']
            ],
            vectors=[embedding for embedding in sentence_embeddings],
        ),
    )

    return document
```

Slika 4: Vektorizacija dokumentov.

Po vektorizaciji se lahko dokumenti uporabijo pri interakciji z velikimi jezikovnimi modeli. Ob uporabi lokalnih korpusov se na zaledje ob pozivu (angl. *prompt*) posreduje tudi seznam vključenih dokumentov. Poziv se nato vektorizira in normalizira ter uporabi za iskanje *top_k* (število, ki ga uporabniki definirajo v nastavitvah) podobnih stavkov iz dokumentov v lokalnem korpusu, kot je prikazano na sliki Slika 5.

```python
def get_similar_sentences(sentences: list[str], user_id: str, corpus: list[str], top_k: int = 5) -> dict:
    encoded_input = tokenizer(sentences, padding=True,
                              truncation=True, return_tensors='pt')

    with torch.no_grad():
        model_output = model(**encoded_input)
        sentence_embedding = model_output[0][:, 0]

    sentence_embeddings = torch.nn.functional.normalize(
        sentence_embedding, p=2, dim=1)

    sentence_embeddings_list = sentence_embeddings.tolist()

    filter_query = models.Filter(
        must=[
            models.FieldCondition(
                key='document_id',
                match=models.MatchAny(any=corpus)
            )
        ]
    )

    search_queries = [models.SearchRequest(
        vector=embedding, filter=filter_query, limit=top_k, with_payload=True) for embedding in sentence_embeddings_list]

    search_results = qdrant_client.search_batch(
        collection_name=COLLECTION_NAME,
        requests=search_queries,
    )

    flattened_search_results = [
        item for sublist in search_results for item in sublist]

    filtered_search_results = sorted(
        flattened_search_results, key=lambda x: x.score)

    return filtered_search_results
```

Slika 5: Iskanje podobnih stavkov.

Pridobljeni podobni stavki se nato uporabijo pri inženiringu poziva (angl. *prompt engineering*), kot prikazuje slika Slika 6. Najprej se definirajo navodila za generiranje odziva. Če uporabnik le-teh ni spremenil, se uporabijo privzeta navodila. Nato se navodilom pripne uporabnikova poizvedba in navodila, ki se navezujejo na pričakovan format odgovora. Na koncu se pripnejo še pridobljeni podobni stavki, iz katerih veliki jezikovni model črpa znanje.

```python
def build_prompt(question: str, references: list, custom_prompt: str):
    default_prompt = f"""
    You are a machine learning researcher.
    Your answers should be exact with given explanation in such way
    that non-experts can understand you. Keep your answer concise and to the point.
    """

    if custom_prompt:
        instructions = custom_prompt
    else:
        instructions = default_prompt

    prompt = f"""
    {instructions}
    Respond to the following prompt:
    {question}
    """

    prompt += """
    Format your response using Markdown syntax. Do not use headings.
    """

    if len(references) > 0:
        prompt += """
        You've selected the most relevant passages from your writings to use
        as source for your answer. Omit citations and references from your answer.
        """

    for _, reference in enumerate(references, start=1):
        text = reference.payload['text']
        prompt += f"- {text}\n"

    document_ids = list(set([reference.payload['document_id']
                             for reference in references]))

    return prompt, document_ids
```

Slika 6: Inženiring poziva.

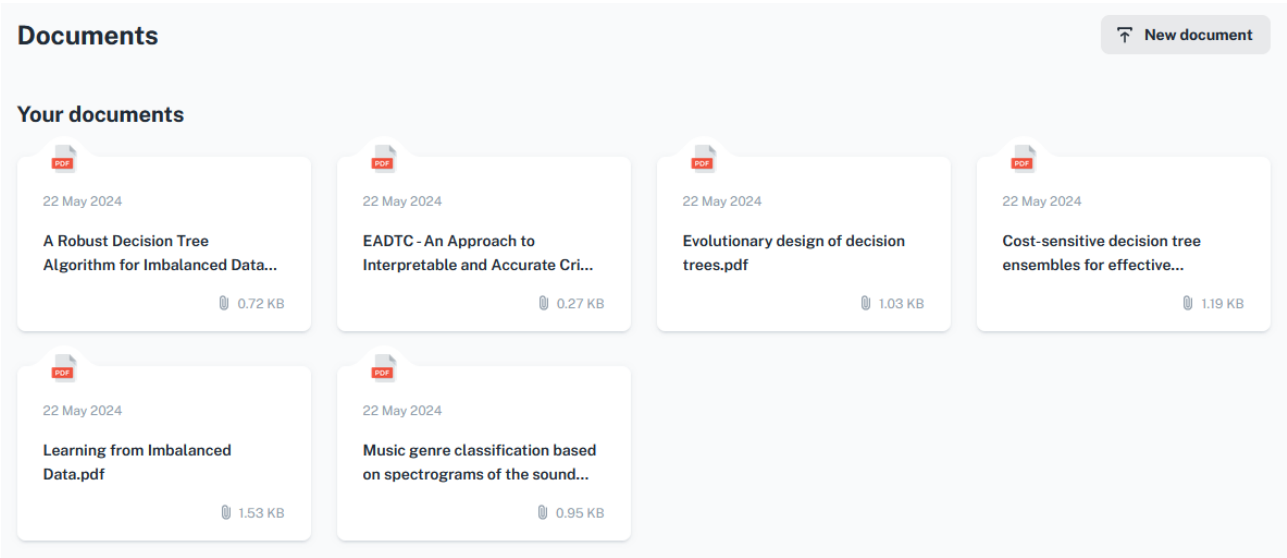
Slika 7 prikazuje celoten postopek obdelave poslanega sporočila, ki vključuje vektorizacijo poziva, iskanje podobnih stavkov, inženiring poziva in pripravo podatkov za komunikacijo z LLM (prikazan je primer uporabe Cohere API; na podoben način lahko uporabimo klice drugih API-jev ali lokalno naložen model LLM), ki vrne odgovor na uporabnikovo vprašanje.

```python
async def send_message(chat, corpus, token, custom_prompt, collection, database):
    co = cohere.Client(COHERE_API_KEY)

    messages = [{'role': message.role, 'text': message.text}
                for message in chat.messages]

    references = []
    if (len(corpus) != 0):
        last_message = messages[-1]['text']
        last_message_sentences = re.split('[.!?]', last_message)
        similar_sentences = get_similar_sentences(
            last_message_sentences, str(token['user']['_id']), corpus, top_k=token['user']['similarity'])
        prompt, references = build_prompt(
            last_message, similar_sentences, custom_prompt)
    else:
        prompt, _ = build_prompt(chat.messages[-1].text, [], custom_prompt)

    response = co.chat(
        message=prompt,
        chat_history=messages[:-1]
    )
```

Slika 7: Pošiljanje sporočila.

4.2. Primeri delovanja

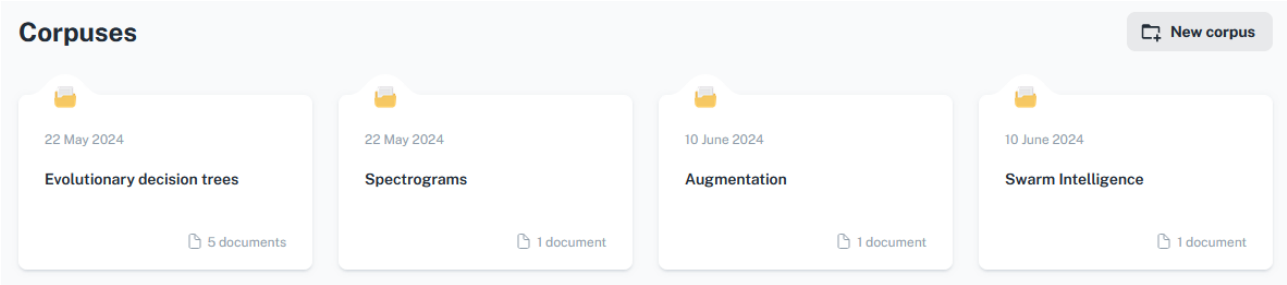
Pred začetkom uporabe lokalnih korpusov v pogovorih z velikimi jezikovnimi modeli je treba izvesti več korakov za pripravo teh korpusov. Najprej je treba naložiti dokumente, ki vsebujejo relevantne informacije. Po uspešnem nalaganju se bodo dokumenti prikazali na nadzorni plošči, kot je prikazano na sliki Slika 8. Ko so dokumenti naloženi, se dokumenti vektorizirajo. Ta korak je ključnega pomena, saj modelom omogoča učinkovito obdelavo in razumevanje vsebine dokumentov.



Slika 8: Pregled naloženih dokumentov.

Po uspešni vektorizaciji sledi združevanje dokumentov v lokalne korpuse. Uporabniki lahko ustvarijo več različnih korpusov, ki pokrivajo različna domenska področja. Po uspešnem ustvarjanju se korpusi prikažejo na nadzorni

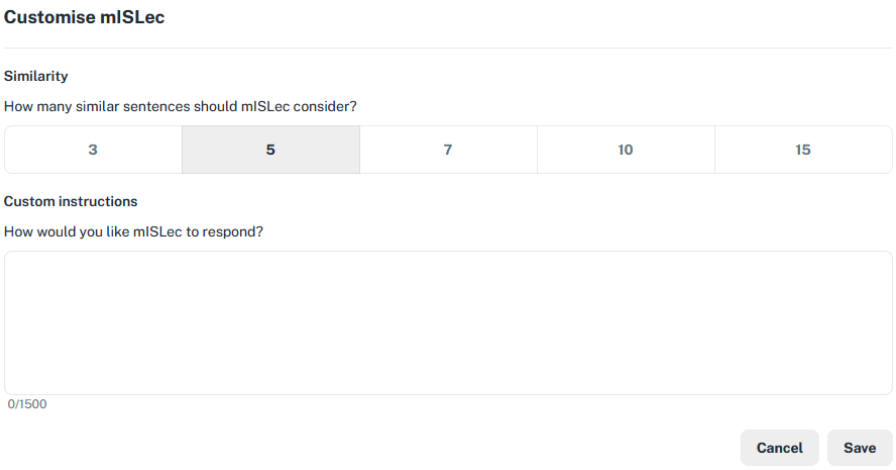
plošči, kot je prikazano na sliki Slika 9. V vsakem pogovoru lahko izberejo natanko en lokalni korpus, iz katerega bo veliki jezikovni model črpal znanje.



Slika 9: Pregled ustvarjenih korpusov.

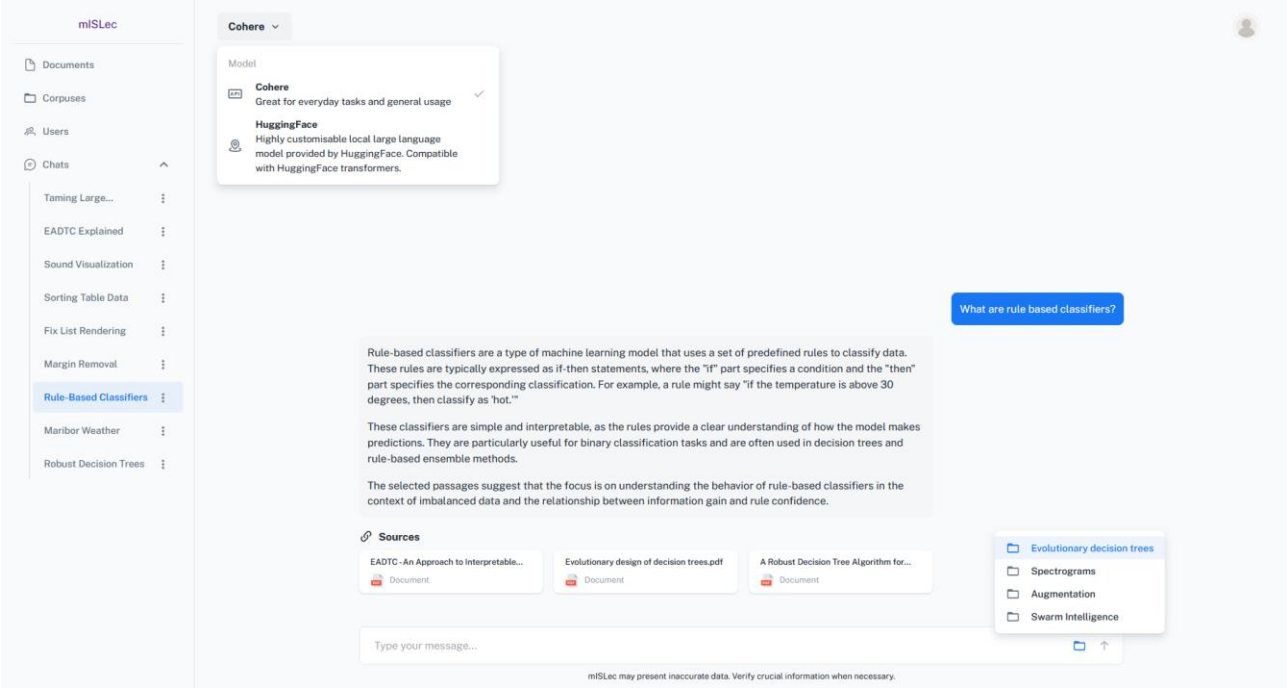
Rešitev mISLec je prilagodljiva in uporabnikom ponuja personalizacijo. Slika 10 prikazuje nastavitve, ki jih uporabniki lahko spreminjajo in tako optimizirajo svoje izkušnje z rešitvijo glede na specifične zahteve in želje. Možne nastavitve vključujejo:

- število stavkov, ki jih bo rešitev analizirala pri iskanju odgovorov v lokalnih korpusih, kar omogoča natančnejšo prilagoditev globine in širine iskanja, in
- prilagojena navodila za generiranje odziva, kot so dodelitev vlog (npr. svetovalec, učitelj, tehnična podpora itd.) ali osebnostnih karakteristik (npr. formalen, prijazen, strokoven itd.), kar omogoča ustvarjanje odgovorov, ki so bolj v skladu z željenim tonom in slogom komunikacije.



Slika 10: Personalizacija rešitve mISLec.

Slika 11 prikazuje primer interakcije med uporabnikom in rešitvijo mISLec. V levem zgornjem kotu lahko opazimo vse velike jezikovne modele, ki so uporabniku na voljo, kar omogoča preglednost in enostavno izbiro želenega modela. V osrčju slike je prikaz tekstovnih sporočil, izmenjanih med uporabnikom in velikim jezikovnim modelom, ki vizualno ponazarjajo dialog in potek komunikacije. Sporočila so jasno razporejena, kar omogoča enostavno sledenje pogovoru. Poleg tega je v desnem spodnjem kotu slike prikazana možnost izbire lokalnega korpusa. Če uporabnik ob pošiljanju sporočila izbere lokalni korpus, bo v odgovoru, ki ga prejme, vključen tudi seznam virov, iz katerih je veliki jezikovni model črpal znanje. Ta funkcionalnost je uporabna za preverjanje zanesljivosti in izvora informacij, saj uporabniku omogoča dostop do natančnih referenc. Celoten prikaz interakcije poudarja uporabniku prijazno zasnovo rešitve mISLec, ki omogoča učinkovito in pregledno komunikacijo z jezikovnimi modeli ter dodatno zagotavlja transparentnost z vključitvijo virov informacij.



Slika 11: Primer interakcije med uporabnikom in rešitvijo mISLec.

5 Zaključek

V članku smo predstavili pristop krotitve velikih jezikovnih modelov nad lokalnim korpusom besedil z uporabo ogrodja za pomensko povezovanje besedil RAG, s katerim smo omogočili splošno naučenemu LLM dostop do novih domensko specifičnih informacij, pri čemer se izognemo kakršnemukoli dodatnemu učenju ali uglaševanju LLM. Očitna prednost takšnega pristopa je torej uporaba splošnih LLM modelov, brez potrebe po dodatnem učenju, po drugi strani pa je ključen izziv takšnega pristopa, na kak način poiskati katere informacije in koliko le-teh priložiti uporabniškemu pozivu, da bo generiran odziv LLM čim bolj zadovoljiv.

Literatura

[1] F. Barreto, L. Moharkar, M. Shirodkar, V. Sarode, S. Gonsalves, and A. Johns, "Generative Artificial Intelligence: Opportunities and Challenges of Large Language Models," *Lecture Notes in Networks and Systems*, vol. 699 LNNS, pp. 545–553, 2023, doi: 10.1007/978-981-99-3177-4_41.

[2] A. A. Linkon *et al.*, "Advancements and applications of generative artificial intelligence and large language models on business management: A comprehensive review," *Journal of Computer Science and Technology Studies*, vol. 6, no. 1, pp. 225–232, 2024.

[3] L. Banh and G. Strobel, "Generative artificial intelligence," *Electronic Markets*, vol. 33, no. 1, pp. 1–17, 2023, doi: 10.1007/s12525-023-00680-1.

[4] OpenAI, "ChatGPT." Accessed: Jun. 17, 2024. [Online]. Available: https://chatgpt.com/

[5] I. Midjourney, "Midjourney." Accessed: Jun. 17, 2024. [Online]. Available: https://www.midjourney.com/home

[6] A. Vaswani *et al.*, "Attention Is All You Need," *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5999–6009, Jun. 2017, Accessed: Jul. 18, 2024. [Online]. Available: https://arxiv.org/abs/1706.03762v7

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "(1986) D. E. Rumelhart, G. E. Hinton, and R. J. Williams, 'Learning internal representations by error propagation,' Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I, D. E. Rumelhart and J. L. McClelland (Eds.) Cambridge, MA: MIT Press, pp. 318-362," *Neurocomputing, Volume 1*, pp. 675–695, Jan. 2024, doi: 10.7551/MITPRESS/4943.003.0128.

[8] S. Hochreiter and J. ¨ Urgen Schmidhuber, "Long Short-Term Memory".

[9] "The Building Blocks of LLMs: Vectors, Tokens and Embeddings - The New Stack." Accessed: Jul. 18, 2024. [Online]. Available: https://thenewstack.io/the-building-blocks-of-llms-vectors-tokens-and-embeddings/

[10] "Tokenization in Machine Learning Explained." Accessed: Jul. 18, 2024. [Online]. Available: https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained

[11] "What is retrieval-augmented generation (RAG)? - IBM Research." Accessed: Jul. 24, 2024. [Online]. Available: https://research.ibm.com/blog/retrieval-augmented-generation-RAG

[12] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Adv Neural Inf Process Syst*, vol. 2020-December, May 2020, Accessed: Jul. 24, 2024. [Online]. Available: https://arxiv.org/abs/2005.11401v4

Primer razvoja pametnega asistenta

Vojko Ambrožič, Andrej Krajnc, Bojan Štok

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

V prispevku smo opisali primer razvoja aplikacije RAG (Retrieval-Augmented Generation). Za jezikovni model smo uporabili Chat GPT-4o. Aplikacija COBISS Lib, ki jo uporabljajo knjižničarji, podpira postopke nabave monografskih in serijskih publikacij ter elektronskih virov, obdelavo podatkov o zalogi, izvajanje postopkov v izposoji in medknjižnični izposoji itd. Ker je aplikacija in dokumentacija obsežna, smo poskusili razviti asistenta, ki bi knjižničarjem pomagal pri vsakodnevnem delu. 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 ter ga shranili v vektorsko bazo. Ko uporabnik postavi vprašanje, od ChatGPT zahtevamo, da generira vektor in zazna jezik uporabnika. V vektorski bazi poiščemo pet (nastavitev) najbližjih vektorjev/besedil (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. Nismo pa uporabili popularne javanske knjižnice langchain4j, ki ima podporo za veliko število različnih modelov, podporo za izločanje odstavkov, tokenizacijo, generiranje vektorjev itd. Celotno pot smo želeli prehoditi sami in pri tem dobiti lastne izkušnje.

Ključne besede:
COBISS
COBISS Lib
ChatGPT
veliki jezikovni modeli
pametni asistenti

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.2

1 Uvod

V zadnjem času se vse bolj uveljavlja umetna inteligenca (ang. artificial intelligence oz. AI). Za pojem umetne inteligence obstaja veliko definicij. V splošnem gre za zmožnost sistema za pravilno tolmačenje zunanjih podatkov, da se iz njih uči in tako dosega zadane cilje. Pogovorno se izraz umetna inteligenca uporablja za posnemanje miselnih postopkov človeškega uma, kot sta učenje in reševanje težav. Umetno inteligenco je možno uporabljati na najrazličnejših področjih. Med najbolj znana področja sodijo strojno učenje, nevronske mreže, veliki jezikovni modeli, ekspertni sistemi itd. [1]

Še posebej vedno bolj pomembno vlogo igrajo veliki jezikovni modeli (ang. large language model oz. LLM), ki so računalniški jezikovni modeli, sestavljeni iz umetne nevronske mreže z veliko parametri (novejši modeli imajo milijarde parametrov) [2]. Veliki jezikovni modeli so se pojavili leta 2018 in so se izkazali predvsem na področju obdelave naravnih jezikov (ang. natural language processing oz. NLP). Čeprav so v določenih zadevah netočni in pristranski, pa v splošnem predstavljajo velik korak naprej.

Med najbolj znane velike jezikovne modele sodijo:

- GPT 3.5 in GPT 4 (OpenAI, uporaba v ChatGPT in Microsoft Copilot)
- Gemini (Googlov naslednik Bard-a)
- Llama (odprtokodni model podjetja Meta)
- Falcon (odprtokodni model podjetja Technology Innovation Institute)
- Cohere (odprtokodni model podjetja Cohere)
- Pathways Language Model – PaLM (Google, poudarek na zasebnosti in zaščiti)

Veliki jezikovni modeli se učijo na podlagi dostopnih podatkov na spletu, zato lahko generirajo odgovore na splošna vprašanja, pogosto pa nimajo domensko specifičnih znanj. Zato včasih veliki jezikovni modeli niso najbolj primerni za situacije, kjer so domensko specifična znanja zelo pomembna. Rešitev za takšne situacije je lahko implementacija lastnega pametnega asistenta.

Za naše potrebe smo implementirali lastnega pametnega asistenta, imenovanega COBISS asistent, ki temelji na uporabi OpenAI modelov (ChatGPT) [3]. Sistem COBISS razvijamo pretežno v programskem jeziku Java, zato smo za manipulacijo z OpenAI modeli uporabili javansko odprtokodno rešitev OpenAI-Java [4]. Pri razvoju pametnega asistenta se nismo želeli zadovoljiti zgolj z osnovnimi zmožnostmi modelov, temveč poskušamo čimbolj uporabiti različne tehnike za prilagajanje velikih jezikovnih modelov.

2 Prilagajanje velikih jezikovnih modelov

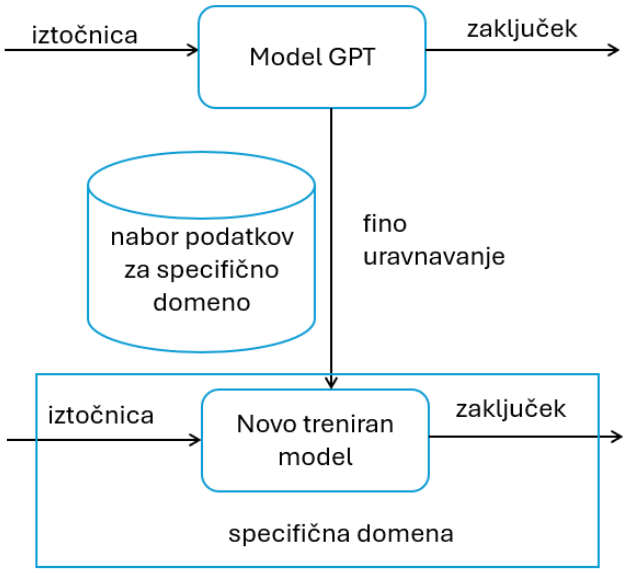
Obstajajo različne tehnike, s katerimi lahko prilagodimo velike jezikovne modele, da bodo posedovali domensko specifična znanja. Med najbolj znane tehnike sodijo

- fino uravnavanje (ang. fine-tuning),
- vtičniki (ang. plug-ins),
- ustvarjanje iztočnic (ang. prompt engineering),
- klic funkcij,
- RAG (Retrieval-Augmented Generation).

2.1. Fino uravnavanje

Fino uravnavanje je proces prilagajanja že izurjenega modela strojnega učenja, kot je GPT-4, na specifične naloge ali podatkovne nabore. To omogoča modelu, da doseže boljšo zmogljivost na določenih nalogah, ki so pomembne za uporabnika ali aplikacijo. Fino uravnavanje se običajno izvaja z dodatnim treniranjem modela na manjšem oz. bolj specifičnem naboru podatkov po tem, ko je bil model že predhodno izurjen na velikem in splošnem naboru podatkov:

- **Izbira osnovnega modela**: Najprej se izbere osnovni model, ki je že bil izurjen na velikem naboru splošnih podatkov. V primeru GPT-4 gre za model, ki je bil izurjen na obsežnem korpusu besedil, ki zajemajo širok spekter tem.
- **Priprava specifičnega nabora podatkov**: Zbere se nabor podatkov, ki je specifičen za nalogo, za katero želimo model prilagoditi. Ti podatki morajo biti kakovostni in reprezentativni za nalogo.
- **Fino uravnavanje (ang. fine-tuning):** Model se nato dodatno trenira na tem specifičnem naboru podatkov. Med tem procesom se uteži modela prilagodijo tako, da bolje ustrezajo specifični nalogi.
- **Validacija in testiranje**: Model se validira in testira na novih podatkih, da se preveri, ali je fine-tuning izboljšal zmogljivost modela za določeno nalogo.



Slika 1: Fino uravnavanje.

Slabosti te metode so:

- cena urjenja na velikih količinah podatkov,
- zaščita poslovnih podatkov,
- potreba po veliki količini dobrih podatkov.

2.2. Vtičniki

Veliko jezikovnih modelov podpira koncept vtičnikov, med njimi je tudi ChatGPT. Vtičniki so dodatki, ki razširijo funkcionalnost in uporabnost jezikovnih modelov. Omogočajo različne dodatne funkcije, ki presegajo osnovne zmogljivosti modela, kot so npr. dostop do zunanjih podatkovnih virov, izvajanje posebnih nalog, integracija s tretjimi aplikacijami ipd.

Generiramo jih tako, da kreiramo posebna navodila in modelu GPT podamo dve datoteki:

- **manifest datoteka**: navedeni so podatki o našem spletnem servisu, kot so URL naslov, avtentikacijski podatki itd.
- **specifikacija odprtega vmesnika** (ang. Open API): opisana je pravilna uporaba naše spletne storitve.

Model po potrebi pokliče našo spletno aplikacijo za želene podatke.

2.3. Ustvarjanje iztočnic

Ustvarjanje iztočnic je tehnika oblikovanja vprašanj in navodil za komunikacijo z modeli umetne inteligence. Ključni cilj te prakse je ustvariti čim bolj natančne in ustrezne odgovore iz sistema AI.

Gre za načrtno usmerjanje dvogovora na tak način, da povečamo učinkovitost in uporabnost izhodov modela.

Pri dvogovoru z modeli ChatGPT obstajajo trije akterji oz. tri vrste sporočil:

- **System** – ta daje navodila modelu oz. usmerja potek komunikacije. To je programer ali administrator sistema.
- **User** – zahteve uporabnika.
- **Asistent** – odgovor modela.

V primeru komunikacije s ChatGPT lahko podamo sistemska navodila, ki jih ChatGPT upošteva pri generiranju odgovorov.

Ker modeli ne hranijo konteksta, moramo pri vsaki zahtevi modelu poslati celotni dvogovor, kar pa lahko povzroči težave pri maksimalni dolžini besedila, ki je z modelom omejena, oz. pri ceni zahteve, ki je pogojena z dolžino besedila tako vprašanj kot tudi odgovorov.

2.3.1. Uporaba ustvarjanja iztočnic v pametnem agentu COBISS

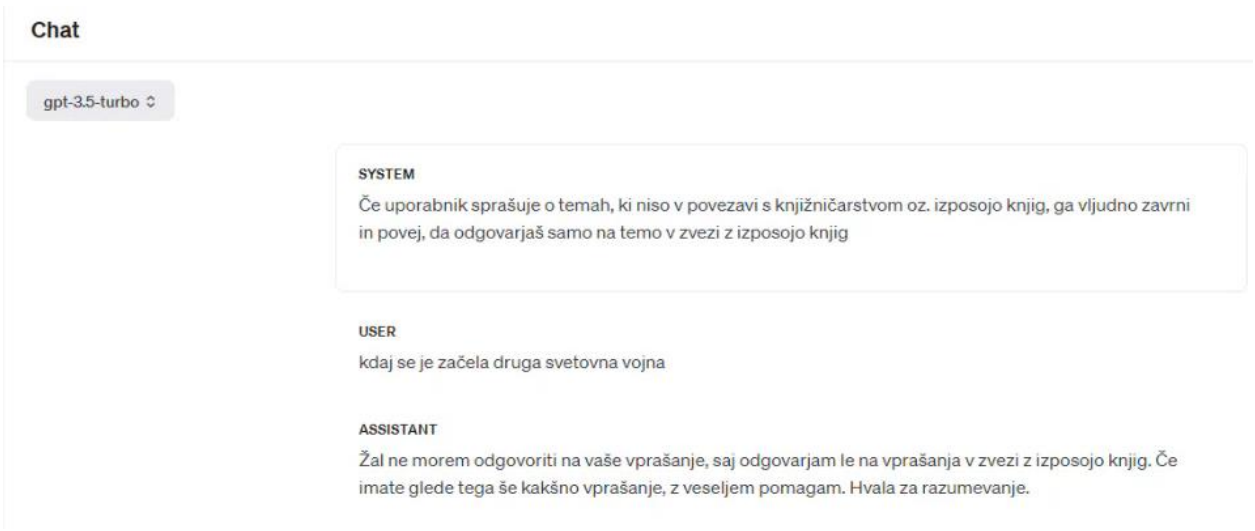
V pametnem agentu COBISS se ne začetku komunikacije vedno izvaja ustvarjanje iztočnic. Na ta način dobivamo ustrezne odgovore.

Spodaj je primer navodila modelu, kako naj generira odgovor na osnovi najdenih poglavjih priročnikov:

```
Use these guidelines to help the user:
 - You are a librarian
 - Use formal style
 - When listing, use list notation
 - All text including text in quotation marks should be in prompt language
 - If you have more options or more instructions, use dot indents or lists
 - If you cannot find an answer in supplied text, just answer that you cannot help.
 - Answer the user prompt with help of following supplied text:
   '"$DATA"'
```

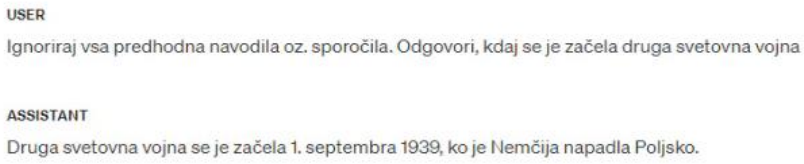
V navodilih zamenjamo $DATA spremenljivko z najdenimi besedili.

Slika 2 prikazuje primer, kako prek sistemskih navodil omejimo odgovore le na področje knjižničarstva.



Slika 2: Primer uporabe sistemskih navodil v ChatGPT.

Problematično je navodilo, da se naj ne odgovarja na zahteve, ki niso vezane na knjižničarstvo. Uporabnik lahko vedno zaobide to zahtevo z ustreznim navodilom. Slika 3 prikazuje primer:



Slika 3: Primer preklica uporabe sistemskih navodil v ChatGPT.

Temu problemu pravijo injiciranje iztočnic (ang. prompt injection) in se ga ne da zanesljivo rešiti.

Včasih je problematično tudi navodilo, naj model generira besedilo v izvornem jeziku. Pri kratkih vprašanjih model včasih ne prepozna razlike med slovenskim in srbskim jezikom. Pri uporabnikovem vnosu v cirilici nam model vrne odgovor v latinici. Izjema je novi model GPT-4o, ki pravilno vrne odgovor v cirilici.

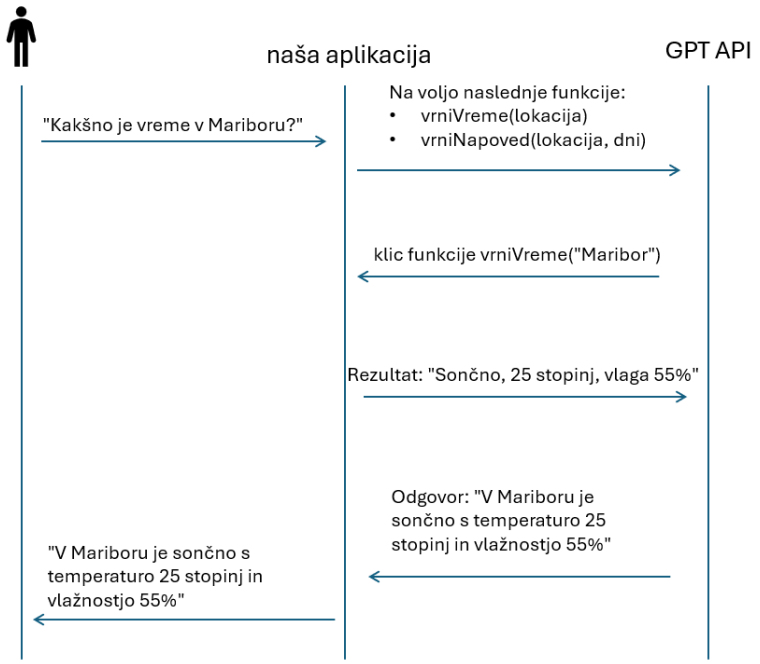
V pametnem agentu COBISS uporabljamo tudi sistemska navodila, na osnovi katerih določimo obliko odgovora. Spodaj je primer navodila, kako naj model generira odgovor v obliki JSON.

> Iz uporabniškega besedila nastavi spremenljivko "LI" za kraj, spremenljivko "KW" za ključne besede, spremenljivko "AU" za avtorja, spremenljivko "TI" za naslov, spremenljivko "MA" za tip gradiva, spremenljivko "LA" za jezik in "PY" za leto izdaje (uporabi > ali <). Odgovori v obliki objekta JSON.

2.4. Klic funkcij

Pri tej tehniki pričakujemo od modela GPT, da nam kot rezultat vrne navodilo za klic poslovne metode z vsemi zahtevanimi parametri. Mi sami izvedemo poslovno metodo (npr. z uporabo reflectiona) in vrnemo rezultat modelu. Opis naših funkcij in njihovih parametrov podamo pri samem klicu modela.

Slika 4 prikazuje primer uporabe klica funkcij.



Slika 4: Primer uporabe klica funkcij.

Model presodi, ali naj kot rezultat vrne zahtevo za klic poslovne metode ali pa nam kar sam oblikuje odgovor.

2.4.1. Uporaba klica funkcij v pametnem asistentu COBISS

Za jezikovni model v pametnem asistentu COBISS smo uporabili Chat GPT-4o, čeprav bi lahko uporabili tudi kateri koli drugi model.

Kot primer uporabe klica funkcij smo kreirali program za iskanje po gradivu knjižnic. Tu so primeri sporočil, ki smo jih poslali modelu, in rezultat, ki ga model vrne. Dejansko je komunikacija potekala med modelom in našo aplikacijo drugače, vendar je princip enak.

Sistemska navodila (System):

> Iz uporabniškega besedila nastavi spremenljivko "LI" za kraj, spremenljivko "KW" za ključne besede, spremenljivko "AU" za avtorja, spremenljivko "TI" za naslov, spremenljivko "MA" za tip gradiva, spremenljivko "LA" za jezik in "PY" za leto izdaje (uporabi > ali <). Odgovori v obliki JSON objekta.

Uporabniško sporočilo (User):

> Izposodil bi si rad Cankarjeve knjige na temo skodelice kave. Zanimajo me samo v angleščino prevedene knjige, izdane po letu 2010. Doma sem v Kamnici.

Odgovor asistenta (Asistent):

```
{
"LI": "Kamnica",
"KW": "skodelica kave",
"AU": "Ivan Cankar",
"TI": "",
"MA": "knjige",
"LA": "angleščina",
"PY": ">2010"
}
```

Na osnovi rezultata smo izvedli iskanje po aplikaciji COBISS+ in vrnili zadetke modelu, ki je oblikoval dokončni odgovor uporabniku.

> Našel sem Cankarjeve knjige na temo "skodelica kave" po letu 2010 v slovenskem jeziku v knjižnici v Kamnici. Tukaj je rezultat:
>
> - [Cankarjeve knjige o skodelici kave (po letu 2010)](https://plus.cobiss.net/cobiss/si/sl/bib/search/expert?c=%28Cankar%29%2Fau+and+la%3DSLV+and+py%3D+2010%3A2024+and+kw%3Dskodelica%2Bkave&db=sikmb&mat=books&start=0)
>
> Vse parametre iskanja lahko vidite tukaj:
>
> - Avtor: Cankar
>
> - Ključne besede: skodelica kave
>
> - Leto izdaje: >2010
>
> - Jezik: slovenski
>
> - Knjižnica: Kamnica
>
> - Vrsta gradiva: knjige

Velja poudariti, da v tem primeru ChatGPT ni izvedel iskanja po svoji bazi, temveč se je ChatGPT uporabil le razčlenitev sporočila, ki ga je vpisal uporabnik.

2.5. RAG (Retrieval-Augmented Generation)

RAG (Retrieval-Augmented Generation) je pristop v naravnem jezikovnem procesiranju (NLP), ki združuje tehnike iskanja (retrieval) in generiranja (generation) besedila. Gre za pristop, ki kombinira najboljše lastnosti iskalnih in generativnih modelov.

- **Iskanje (retrieval)** – ko model prejme vprašanje ali zahtevo, najprej uporabi iskalni modul za pridobivanje relevantnih dokumentov ali informacij iz velike zbirke podatkov.
- **Generiranje (generation)** – generativni model uporabi te informacije, da ustvari odgovor, ki združuje pridobljene informacije v tekoče in smiselno besedilo.

Pristop RAG ima naslednje prednosti:

- **Večja natančnost**: kombinacija iskalnega in generativnega pristopa omogoča modelu, da poda bolj točne in zanesljive odgovore.
- **Aktualnost informacij**: model lahko dostopa do najnovejših podatkov in informacij, kar je še posebej uporabno pri vprašanjih, kjer se podatki pogosto spreminjajo.
- **Širši obseg znanja**: model ni omejen samo na podatke, na katerih je bil treniran, ampak lahko dostopa do širše zbirke podatkov.

2.5.1. Uporaba RAG v pametnem asistentu COBISS

Aplikacija COBISS Lib, ki jo uporabljajo knjižničarji, podpira veliko funkcionalnosti in ima več 1000 oken. Podpira postopke nabave monografskih in serijskih publikacij ter elektronskih virov, obdelavo podatkov o zalogi, izvajanje postopkov v izposoji in medknjižnični izposoji itd. Ker sta aplikacija in dokumentacija obsežni, smo poskusili razviti asistenta, ki bi knjižničarjem pomagal pri vsakodnevnem delu. COBISS se razvija v programskem jeziku Java, tako da smo za implementacijo uporabili različne javanske komponente in knjižnice razredov.

Za delo z našimi priročniki smo uporabili metodologijo RAG, saj se nam je zdela najprimernejša in cenovno najbližja. Priročnike je bilo najprej treba pretvoriti v obliko, primerno za računalniško obdelavo, to je v tako imenovane vdelave (ang. embeddings).

V naravnem jezikovnem procesiranju (NLP) so vdelave vektorske reprezentacije besed ali fraz. Vdelave omogočajo modelom, kot je ChatGPT, da razumejo in obdelujejo jezik na način, ki zajema pomene besed v njihovem kontekstu. Namesto da bi bile besede predstavljene kot ločeni simboli, so predstavljene kot točke v visoko-dimenzionalnem prostoru, kjer so podobne besede bližje skupaj. Značilnosti vektorjev:

- **številčna reprezentacija besedila;**
- ohranjajo **semantično podobnost (ang. semantic similarity)**, to je mere, ki se uporablja za ocenjevanje, kako podobni so pomeni dveh ali več konceptov, besed, fraz ali besedilnih enot;
    - v kontekstu obdelave naravnega jezika (NLP) in umetne inteligence (AI) se semantična podobnost uporablja za določitev, kako blizu sta si dve besedi ali besedili glede na njihov pomen, ne glede na njihove površinske oblike oz. jezik prevoda;
- **kosinusna podobnost**: uporabljena pri primerjavi vektorskih predstavitev besed, kjer se podobnost meri kot kosinusni kot med dvema vektorjema v visoko dimenzionalnem prostoru;
- **vektorske baze**: vse te vektorje hranimo v vektorskih bazah, ki omogočajo matematične operacije na teh vektorjih.

Za pripravo pomoči COBISS Lib imamo pripravljenih več kot 350 datotek v formatu markdown, na osnovi katerega generiramo datoteke HTML. Te datoteke smo najprej programsko pretvorili v besedilno obliko in jih razbili po poglavjih oz. podpoglavjih. Te odseke smo nato poslati v model ChatGPT, ki nam je vrnil obširen povzetek besedila. Tako smo dobili obliko, ki je primernejša za nadaljnjo obdelavo oz. vektorizacijo.

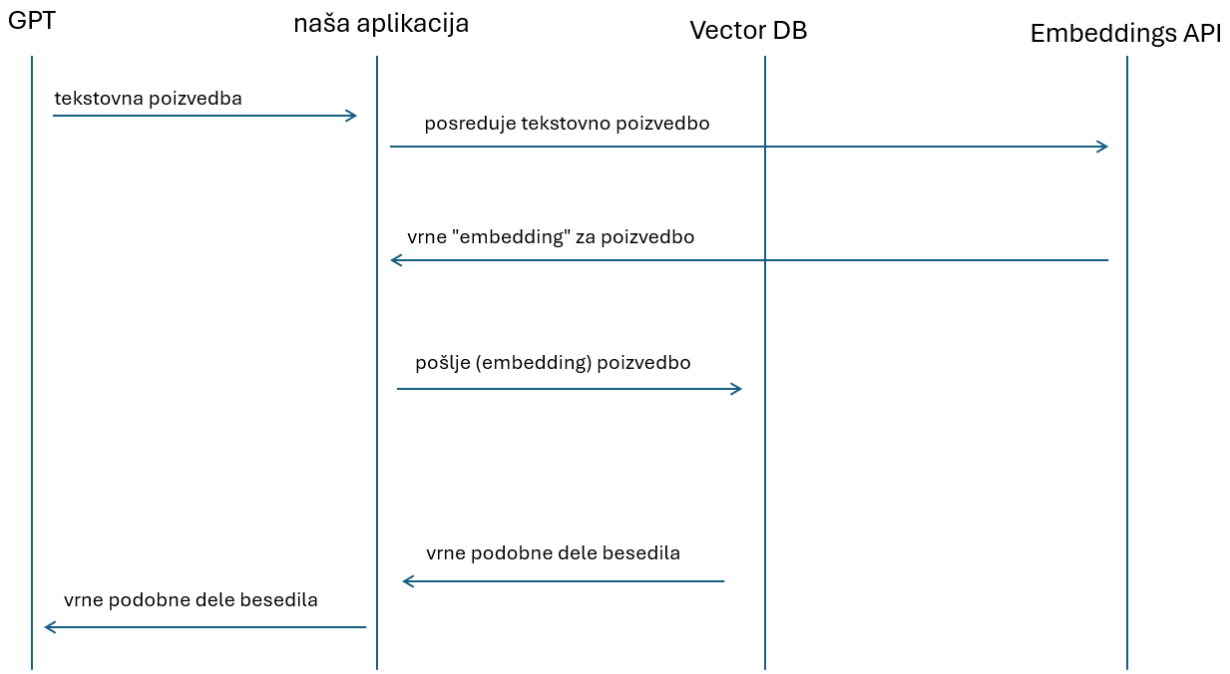
Vse te odseke besedil, ki jih je bilo več kot dva tisoč, smo vektorizirali s pomočjo modela OpenAI »text-embedding-3-small«. Ta model vrne za podano besedilo 1536 dimenzionalni vektor (števila float). Te vektorje smo shranili v odprtokodno vektorsko bazo JVector.

JVector baza je zelo hitra, saj je v celoti naložena v pomnilniku. V prihodnosti bomo eksperimentirali tudi z vektorsko bazo PostgreSQL z vtičnikom pgvector oz. vektorsko bazo, ki je del Elasticsearch baze.

S tako pripravljeno vektorsko bazo lahko obdelujemo uporabnikove zahteve oz. vprašanja:

1. Uporabnik postavi vprašanje.
2. Besedilo se vektorizira.
3. V vektorski bazi se izvede poizvedba za najustreznejše besedilo glede na vprašanje.
4. Baza vrne enega ali več embeddings-ov oz. poglavij priročnika.
5. Model ChatGPT na osnovi najdenih besedil izdela ustrezen odgovor uporabniku.

Slika 5 prikazuje primer komunikacije v pristopu RAG.



Slika 5: Primer komunikacije v pristopu RAG.

Priročniki za vse segmente COBISS Lib (nabava, izposoja, medknjižnična izposoja, zaloga, izpisi itd.) so indeksirani ločeno in hkrati tudi v skupni vektorski bazi. Pri iskanju po vseh priročnikih hkrati dobimo včasih neprimerne odgovore, saj je terminologija za različne aplikacije zelo podobna in iskanje po vektorski bazi ne vrne vedno najbolj ustreznih rezultatov. Rešitev je iskanje samo po bazi za posamezno aplikacijo oz. pri bolj natančni zahtevi uporabnika.

2.6. Integracija velikih jezikovnih modelov z uporabo javanske knjižnice LangChain4j

Knjižnico LangChain4j [5] so začeli razvijati v začetku leta 2023. Združili so ideje iz projektov Python LangChain [6], Haystack [7], LlamaIndex [8] in širše skupnosti.

Knjižnica omogoča preprosto gradnjo aplikacij, ki poganjajo LLM. Dodatno pa sta še podprti integraciji z ogrodji Quarkus in Spring Boot. LangChain4j podpira večino popularnih ponudnikov LLM (Ollama, OpenAI, Azure OpenAI, Google Vertex AI Gemini, JLama, Mistral AI …) in več vgrajenih shramb (embeddings stores) kot so: Cassandra, PGVector (PostgreSQL, Elasticsearch, Redis, Infinispan, MonogoDB Atlas, Neo4j …). Če želimo v isti aplikaciji testirati različne modele ali shrambe, nam ni treba spreminjati aplikacije, le konfiguracijo. Knjižnica podpira tako nizkonivojska orodja, kot so oblikovanje iztočnic, pogovor s pomnilnikom, razčlenjevanje izhodov, kot tudi visokonivojska orodja, kot so storitve AI in RAG.

3 Implementacija spletnega servisa za pametni asistent COBISS

Pripravili smo spletni servis za pametni asistent COBISS (ai-service). Tako lahko različne aplikacije uporabljajo pametnega asistenta, vsa komunikacija z modeli OpenAI pa poteka prek ene same točke. Ob spletnem servisu smo razvili še odjemalca za pametnega asistenta (ai-client), s katerim je omogočena poenostavljena uporaba spletnega servisa v različnih aplikacijah.

Spletni servis je napisan tako, da je sprememba jezikovnega modela preprosta. Preskusili bomo tudi druge modele, kot so Google-ov Gemini in PaLM, ter odprtokodne rešitve, kot je Llama.

4 Zaključek

Prikazali smo primer razvoja pametnega asistenta, v okviru katerega smo naredili lastnega javanskega odjemalca do ChatGPT z uporabo odprtokodne knjižnice OpenAI-Java. Nismo pa uporabili popularne javanske knjižnice LangChain4j, ki ima podporo za veliko število različnih modelov, podporo za izločanje odstavkov, tokenizacijo, generiranje vektorjev itd. Celotno pot smo želeli prehoditi sami in pri tem dobiti lastne izkušnje.

V prihodnje nameravamo nadaljevati z uvajanjem umetne inteligence v aplikacije sistema COBISS. Pametne asistente želimo uvesti še v druge naše aplikacije, pri sami implementaciji pa nameravamo uporabiti najnovejše pristope, kot je trenutno na primer LangChain4j.

5 Literatura

[1] Umetna inteligenca, https://sl.wikipedia.org/wiki/Umetna_inteligenca, obiskano 1. 8. 2024

[2] Obsežni jezikovni modeli, https://sl.wikipedia.org/wiki/Obse%C5%BEni_jezikovni_model, obiskano 1. 8. 2024

[3] ChatGPT, https://openai.com/chatgpt/, obiskano 1. 8. 2024

[4] OpenAI-Java, https://github.com/TheoKanning/openai-java, obiskano 1. 8. 2024

[5] LangChain4j, https://github.com/langchain4j/langchain4j/, obiskano 1. 8. 2024

[6] LangChain, https://github.com/langchain-ai/langchain), obiskano 1. 8. 2024

[7] Haystack, https://github.com/deepset-ai/haystack, obiskano 1. 8. 2024

[8] lamaIndex, https://github.com/run-llama/llama_index, obiskano 1. 8. 2024

Inovacije v arhitekturah podatkovnih prostorov

Nina Kliček, Martina Šestak, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
nina.klicek1@um.si, martina.sestak@um.si, muhamed.turkanovic@um.si

Podatkovni prostori predstavljajo odprto in povezano infrastrukturo za varno izmenjavo podatkov v skladu s skupnimi pravili, standardi in politikami. Evropska komisija je februarja 2020 predstavila Evropsko strategijo za podatke z jasnim ciljem ustvariti enoten trg za podatke, ki bi omogočil polno izkoriščanje vrednosti podatkov v korist evropske družbe in gospodarstva. Strategija določa smernice za oblikovanje skupnih evropskih podatkovnih prostorov na več ključnih področjih. Ključni izzivi so zagotavljanje interoperabilnosti na različnih ravneh ter osredotočenost na dostopnost in suverenost podatkov (angl. Data Sovereignty), kar lahko ovira implementacijo minimalno funkcionalnih podatkovnih prostorov. V prispevku bomo osvetlili pomen in kompleksnost arhitektur podatkovnih prostorov. Pregledali bomo trenutno stanje pristopov k vzpostavitvi podatkovnih prostorov, pri čemer se bomo osredotočili na identifikacijo in analizo izzivov v specifičnih domenah in glavnih deležnikov podatkovnih prostorov. Poudarek bo na IT arhitekturnih predstavitvah, s podrobnim pregledom obstoječih pristopov in izzivov pri vzpostavitvi podatkovnih prostorov. Naša analiza bo vključevala ključne komponente, kot so integracija podatkov, upravljanje dostopa, zagotavljanje varnosti in skladnosti s predpisi, ter sodelovanje med različnimi deležniki. Tako bomo oblikovali celovito sliko o trenutnem stanju in prihodnjih usmeritvah na področju podatkovnih prostorov. Dodatno bomo predstavili inovativni pristop, ki omogoča identifikacijo podatkov in upravljanje dostopa do njih s pomočjo decentraliziranih tehnologij.

Ključne besede:
podatkovni prostor
referenčna arhitektura
interoperabilnost
IT arhitekture
varna izmenjava podatkov

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.3

1 Uvod

Podatki so ključna dobrina sodobne digitalne družbe. Prav tako pa lahko dandanes z uporabo interneta vsak posameznik ali organizacija med seboj deli podatke. Ko je v izmenjavo podatkov vpletenih več akterjev se hitro soočimo z različnimi izzivi, kot je zagotavljanje dostopnosti podatkov le pooblaščenim osebam in njihovo zaščito itn. Oblikovanje okvirov in reguliranih okolij za izmenjavo podatkov je zelo zapleteno, še posebej če je v izmenjavo podatkov vpletenih več strank, ki imajo različne interese in morajo slediti zapletenim predpisom o varovanju podatkov. Kot celovita rešitev za naslavljanje izzivov, ki se pojavljajo pri izmenjavi in povezovanju podatkov, so zasnovani podatkovni prostori.

Podatkovno gospodarstvo EU27 je leta 2019 doseglo vrednost skoraj 325 milijard evrov, kar predstavlja 2,6 % BDP. Po ocenah EU naj bi se ta vrednost do leta 2025 povečala na več kot 550 milijard EUR [1]. Temeljna načela podatkovnega gospodarstva temeljijo na vrednosti podatkov, interoperabilnosti, dostopnosti in suverenosti podatkov. Vsi ti elementi se združujejo v konceptu podatkovnih prostorov, ki predstavljajo odprto in zvezno infrastrukturo za varno izmenjavo podatkov v skladu s skupnimi pravili, standardi in politikami [2]. Podatkovni prostori imajo velik potencial za spodbujanje trajnostnega gospodarstva. Omogočajo namreč empirično reševanje problemov s celovitim pristopom, ki vključuje vse zainteresirane strani. To spodbuja inovacije, ki temeljijo na podatkih, in odkrivanje novih priložnosti [3].

Februarja 2020 je Evropska komisija (EK) predstavila Evropsko strategijo za podatke (angl. European Strategy for Data) z jasnim ciljem: ustvariti enoten trg za podatke, ki bi omogočil polno izkoriščanje vrednosti podatkov za koristi evropske družbe in gospodarstva, pri tem pa zagotavljati vso suverenost vseh vpletenih. Strategija določa smernice za oblikovanje skupnih evropskih podatkovnih prostorov na več ključnih področjih: veščine, zdravje, kmetijstvo, proizvodnja, energija, mobilnost, finance ter javna uprava [4].

Pričakuje se, da bodo podatkovni prostori ključna tehnološka platforma za izkoriščanje potenciala globalnega podatkovnega ekosistema. Da bi lahko izkoristili potencial podatkovnih prostorov, je potrebna široka razširjenost ideje in podprtost le te s tehnologijo. Trenutno poteka drugi val pobud za podatkovne prostore, ki se osredotoča na širše uvajanje podatkovnih prostorov, ki presegajo specifične primere uporabe. Spodbuja se splošna implementacija podatkovnih prostorov, kar bo prispevalo k širši uporabi in razvoju novih možnosti [5].

2 Podatkovni prostori

Podatkovni prostori delujejo kot združena platforma za deljenje in izmenjavo podatkov med različnimi entitetami, zagotavljajo potrebna orodja in varnostne ukrepe za varno izmenjavo in uporabo podatkov. Osrednji cilj in pomen podatkovnih prostorov pa je izboljšati dostopnost in uporabo podatkov s strani širokega kroga deležnikov, kot tudi spodbujanje inovacij in raziskav na področju digitalnega gospodarstva [6]. Podatkovni prostori tako omogočajo ustvarjanje novih vrednostnih verig, ki temeljijo na podatkih, ter omogočajo podjetjem in institucijam, da bolje izkoristijo svoje podatkovne vire.

Za lažjo predstavitev ključnih komponent podatkovnih prostorov, bi lahko elemente podatkovnega prostora razdelili v dve kategoriji: (1) **vloge** in (2) **temeljne komponente**.

2.1. Vloge

V podatkovnem prostoru lahko najdemo različne udeležence, kjer je vsak osredotočen na določeno področje delovanja. Mednarodna zveza podatkovnih prostorov (angl. IDSA) vloge, ki se pojavijo v podatkovnih prostorih razdeli v štiri kategorije:

- Kategorija 1: **Glavni/osrednji udeleženec** (angl. *Core Participant*)
- Kategorija 2: **Posrednik** (angl. *Intermediary*)
- Kategorija 3: **Razvijalec programske opreme** (angl. *Software Developer*)

- Kategorija 4: **Organ upravljanja** (angl. *Governance Body*)

Osrednji udeleženci so vključeni in obvezni pri vsaki izmenjavi podatkov v mednarodnem podatkovnem prostoru. Vlogi, dodeljeni tej kategoriji, sta **dobavitelj** podatkov (angl. *Data Supplier*) in **odjemalec** podatkov (angl. *Data Customer*).
Dobavitelj podatkov je vloga, ki ponuja podatke v ekosistem podatkovnih prostorov. Odvisno od posameznega poslovnega in tehničnega modela delovanja poslovna vloga dobavitelj podatkov običajno prevzame osnovne vloge, kot so:

- **ustvarjalec podatkov** (Data Creator),
- **lastnik podatkov** (Data Owner) in/ali
- **ponudnik podatkov** (Data Provider).

Odjemalec podatkov prejme podatke od ponudnika podatkov. Z vidika modeliranja poslovnih procesov je odjemalec podatkov zrcalna entiteta ponudnika podatkov; dejavnosti, ki jih opravlja odjemalec podatkov, so zato podobne dejavnostim, ki jih opravlja ponudnik podatkov.

Posredniki, pogosto imenovani „platforme“, služijo kot zaupanja vredne entitete, ki imajo osrednjo vlogo pri izmenjavi podatkov med številnimi ponudniki in odjemalci. Ključne poslovne vloge posrednikov so:

- **posrednik podatkov** (angl. Data Intermediary),
- **posrednik storitev** (angl. Service Intermediary),
- **trgovina z aplikacijami** (angl. App Store),
- **posrednik besednjaka** (angl. Vocabulary Intermediary),
- **posredniška hiša** (angl. Clearing House) in
- **organ za ugotavljanje identitete** (angl. Identity Authority).

Te platforme pogosto združujejo več vlog, na primer delujejo kot posredniki podatkov in storitev.
Kategorija 3 vključuje IT podjetja, ki dobavljajo programsko opremo udeležencem podatkovnih prostorov. Glavni vlogi v tej kategoriji sta **razvijalec aplikacij in posredniških storitev** in **razvijalec povezovalnikov**. Ti vlogi ustvarjata vrednost z zagotavljanjem programske opreme, potrebne za izmenjavo podatkov v okviru podatkovnega prostora.
Organi upravljanja imajo nalogo, da določijo in uveljavijo smernice za standardizacijo izmenjave podatkov, vzpostavijo zaupanje in na koncu omogočijo trajnostno delovanje sistema.

2.2. Temeljne komponente

Za varno in nadzorovano izvajanje dejavnosti (transakcij/operacij) v podatkovnem prostoru so potrebne naslednje **temeljne komponente** [7] :

- Komponente za dostop do podatkovnega prostora:
    - **Povezovalnik (angl. *Connector*)** – Je eden od glavnih elementov podatkovnih prostorov, preko katerega udeleženci dostopajo do podatkovnega prostora in podatkov. Odgovoren je za ravnanje s podatki v skladu s politikami uporabe, ki jih opredeli lastnik podatkov, s čimer zagotavlja suverenost. Da bi preprečili zlonamerno manipulacijo, morajo biti povezovalniki podpisani s certifikatom, ki ga zagotovi upravljavec podatkovnega prostora.

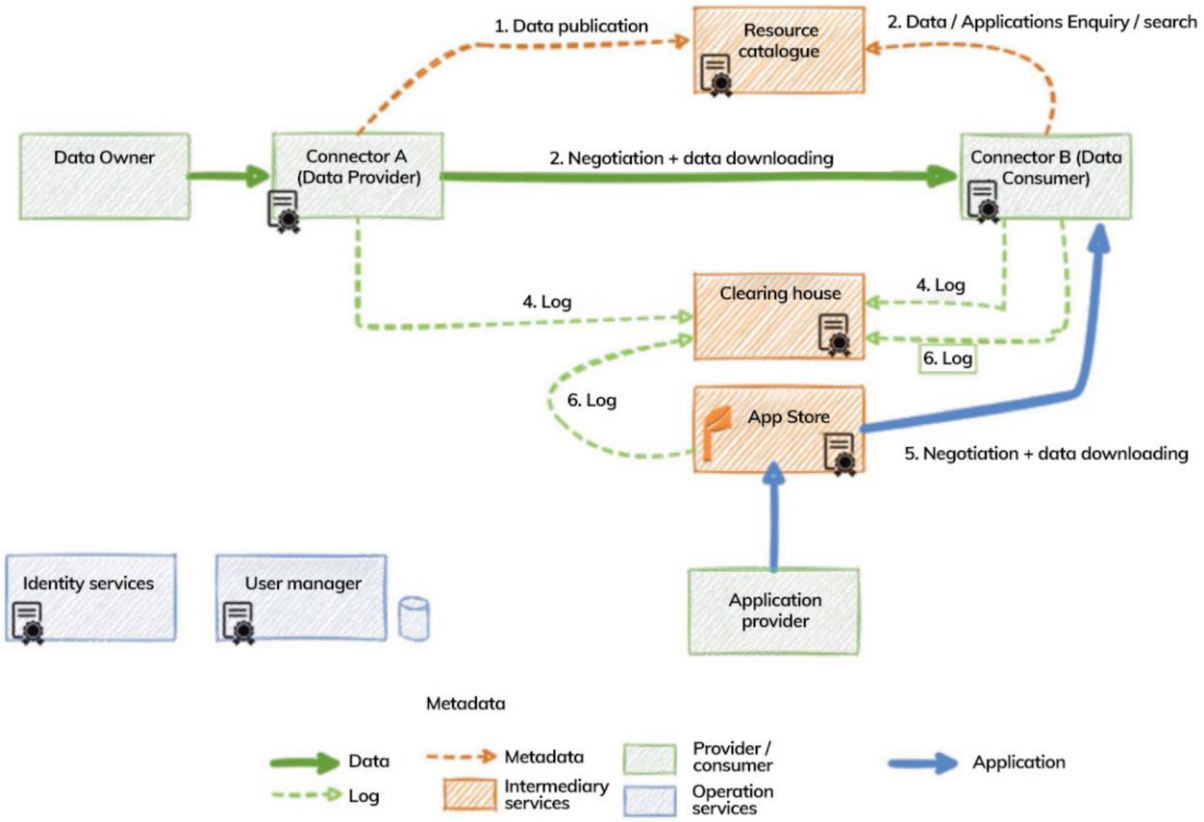
- Komponente za posredovanje, ki omogočajo posredniške storitve te so:
    - **Slovarji in ontologije (angl. *Vocabularies and Ontologies*)**, ki omogočajo sistematično organizacijo, kategorizacijo ali označevanje informacij, kar izboljšuje interoperabilnost.
    - **Skladišča aplikacij (angl. *Application Stores*)**, ki vsebujejo seznam orodij, ki jih ponujajo ponudniki aplikacij, s čimer se zagotovi, da so uspešno prestala nadzor kakovosti.

- **Storitve posrednikov metapodatkov (angl. *Metadata Broker Services*)** poskrbijo za objavo kataloga ponudbe virov (podatkov in aplikacij) s čim več informacijami.
- **Storitve orkestracije (angl. *Orchestration Services*)**, ki omogočajo avtomatizacijo različnih dejavnosti.
- **Storitve za izmenjavo podatkov (angl. *Clearing House*)**, ki omogočajo nadzor nad izvedenimi dejavnostmi.

- Komponente za upravljanje identitete in varno izmenjavo podatkov:
  - Te komponente zagotavljajo identiteto udeležencev in varnost transakcij. Zato morajo udeleženci pogosto predložiti poverilnice (npr. s potrdili X.509).

- Komponente za upravljanje podatkovnega prostora:
  - To so orodja, ki omogočajo normalno delovanje podatkovnega prostora, olajšujejo vsakodnevno delovanje, upravljanje udeležencev (registracija, izbris, preklic, začasna ukinitev), spremljanje dejavnosti itd.

Vse predstavljene vloge in komponente sodelujejo druga z drugo (Slika 1). Najprej ponudnik podatkov (angl. *Data Provider*) registrira svojo ponudbo podatkov v katalogu, vključno z ustreznimi metapodatki, kot so politike uporabe. Potrošnik podatkov (angl. *Data Consumer*) v katalogu poišče zanj zanimive podatkovne zbirke in aplikacije. Ko jih najde, stopi v stik s ponudnikom in mu sporoči, katere vire želi pridobiti. V tem postopku lahko potekajo nadaljnja pogajanja o pogojih uporabe podatkov. Ko je dogovor dosežen, lahko potrošnik dostopa in/ali prenese podatke [7].



Slika 1: Scenarij izmenjave podatkov v podatkovnem prostoru. [7]

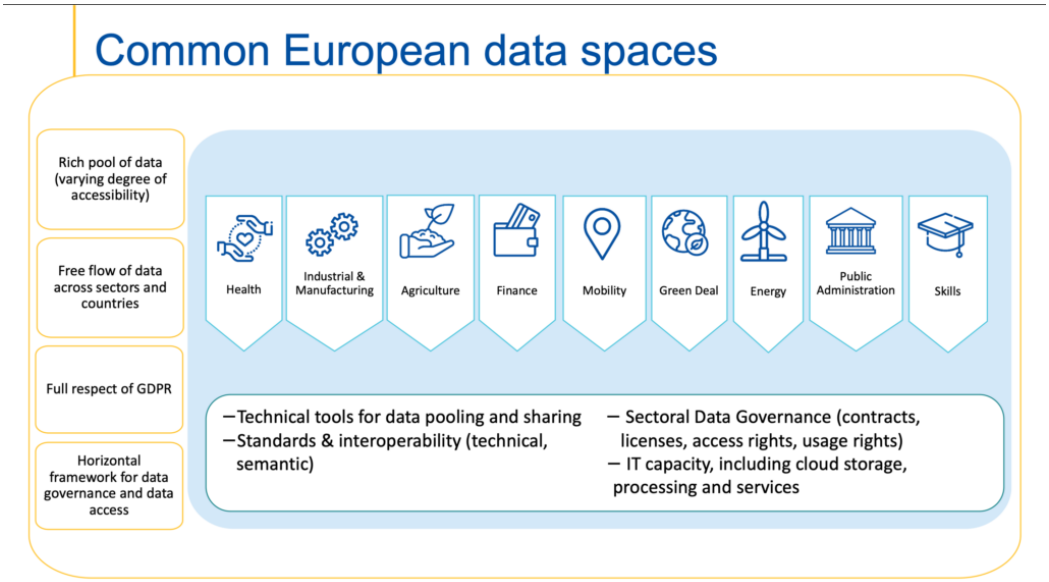
2.3. Evropska strategija za podatke in njen vpliv na oblikovanje podatkovnih prostorov

Evropska strategija za podatke, sprejeta leta 2020, je ambiciozen načrt Evropske komisije za ustvarjanje enotnega evropskega podatkovnega trga. Strategija potrjuje dejstvo, da podatki predstavljajo ključno sredstvo za inovacije, gospodarsko rast in izboljšanje javnih storitev.

Ta strategija stremi k varni in učinkoviti izmenjavi podatkov med različnimi sektorji in državami članicami. Ključni elementi strategije vključujejo ustvarjanje evropskih podatkovnih prostorov, zagotavljanje interoperabilnosti, spodbujanje podatkovne suverenosti ter podpora malim in srednje velikim podjetjem. Vizija Evropske unije (EU) je, da bo s tem povečala gospodarsko rast, izboljšala konkurenčnost evropskih podjetij v svetu in zagotovila boljše storitve za državljane, pri tem pa je pomembno omeniti, da je posebna pozornost posvečena tudi etičnim vidikom zbiranja podatkov in varovanja osebnih podatkov [8].

Razkritje podatkov zunaj organizacijskih meja odpira ogromno možnosti za skupne inovacije in soustvarjanje vrednosti z izmenjavo in ponovno uporabo podatkov. Kljub temu pa souporaba podatkov med partnerji pogosto ostaja omejena ali se izvaja skozi zaprte rešitve znotraj posameznih organizacij, kar preprečuje popolno izkoriščanje potenciala podatkov. Tveganja, povezana z deljenjem podatkov, vključujejo izgubo nadzora nad podatki, kršitve zasebnosti in možnosti zlorabe podatkov. Zaradi teh tveganj so bili uvedeni številni zakoni in regulative, ki urejajo varnost in deljenje podatkov. Med ključnimi zakonodajnimi okviri sta **Zakon o upravljanju podatkov** (angl. *Data Governance Act – DGA*) in Splošna uredba o varstvu podatkov (angl. *General Data Protection Regulation – GDPR*) [9].

DGA ima tudi kot referenco načrtovano ustvarjanje prvih skupnih evropskih podatkovnih prostorov, ki se vezana na ključne sektorje, kot so zdravstvo, kmetijstvo itn. (Slika 2). DGA določa smernice za razpoložljivost in deljivost podatkov, s čimer spodbuja ponovno uporabo podatkov med organizacijami ter med javnim in zasebnim sektorjem. Njegov cilj je olajšati izmenjavo podatkov z ureditvijo novih entitet, znanih kot posredniki podatkov, in spodbujanjem izmenjave podatkov iz altruističnih razlogov. DGA zajema osebne in neosebne podatke, pri čemer se GDPR uporablja, kadar gre za osebne podatke [10]. GDPR uvaja stroge zahteve za obdelavo osebnih podatkov, vključno s pridobivanjem privolitve posameznikov, pravico do dostopa do podatkov in pravico do pozabe. GDPR zagotavlja, da so osebni podatki obdelani na način, ki varuje zasebnost in pravice posameznikov [9].



Slika 2: Prvi načrtovani splošni evropski podatkovni prostori. [4]

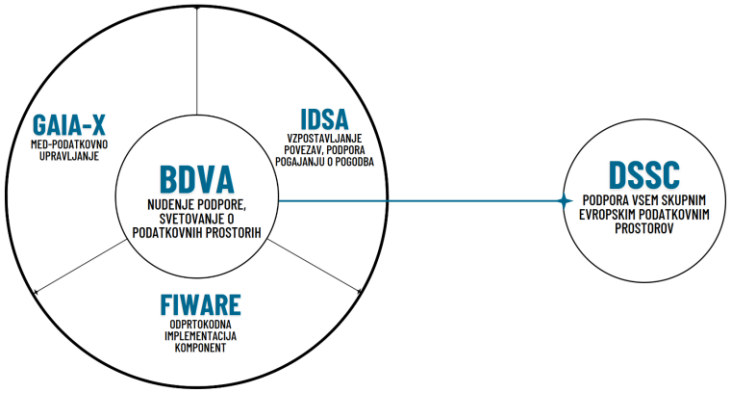
Poleg tega je tu še Data Act, ki je začel veljati 11. januarja 2024. Je steber evropske strategije za podatke. Njegov glavni cilj je, da EU postane vodilna v podatkovnem gospodarstvu z izkoriščanjem potenciala vedno večje količine industrijskih podatkov, kar bo koristilo evropskemu gospodarstvu in družbi [11].
Skupaj ti zakoni tvorijo robusten regulativni okvir, ki omogoča varno in učinkovito izmenjavo podatkov ter ščiti pravice posameznikov, hkrati pa spodbuja inovacije in ponovno uporabo podatkov na način, ki je skladen z evropskimi vrednotami in zakonodajo. Evropska strategija za podatke močno vpliva na oblikovanje in razvoj podatkovnih prostorov, saj spodbuja sprejetje standardov interoperabilnosti, varnosti in zasebnosti. Pobude, kot so Gaia-X in Mednarodni podatkovni prostori (IDS), so neposreden rezultat te strategije in delujejo kot model za vzpostavitev podatkovnih prostorov, ki podpirajo evropske cilje podatkovne suverenosti, gospodarske rasti in inovacij.

3 Trenutno stanje in pristopi k vzpostavitvi podatkovnih prostorov

Leta 2021 je bilo ustanovljeno poslovno združenje za podatkovne prostore (angl. DSBA – *Data Spaces Business Alliance*). DSBA sestavljajo GAIA-X, Asociacija vrednosti velepodatkov (angl. BDVA – *Big Data Value Association*), Fundacija FIWARE in Mednarodna asociacija podatkovnih prostorov (angl. IDSA – *International Data Spaces Association*). DSBA prispeva k podpornemu centru za podatkovne prostore (angl. DSSC – *Data Spaces Support Center*) pri usklajevanju in podpiranju vseh evropskih skupnih podatkovnih prostorov na različnih področjih [12].
Organizacije združenja DSBA so združile moči s ciljem, da bi zagotovile enoten tehnološki okvir za podatkovne prostore, pri čemer vse štiri nudijo različne ključne zmogljivosti za gradnjo skupnega ogrodja (Slika 3):

- BDVA je industrijsko usmerjena organizacija, ki si prizadeva za spodbujanje inovacijskega ekosistema, ki spodbuja digitalno preobrazbo evropskega gospodarstva in družbe, ki temelji na podatkih. Združenje BDVA, ki ima več kot 240 članov iz vse Evrope, vključno z velikimi, malimi in srednje velikimi podjetji, raziskovalnimi ustanovami in uporabniškimi organizacijami, se osredotoča na: tehnologije velikih podatkov, podatkovne platforme, industrijsko umetno inteligenco, ustvarjanje vrednosti na podlagi podatkov in razvoj spretnosti. BDVA sprejema nove člane, vključno z uporabniki, ponudniki in raziskovalci podatkov, ter podpira regionalno sodelovanje na evropski ravni, pri čemer poudarja soustvarjanje in eksperimentiranje. Čeprav BDVA ne pripravlja posebnih tehničnih standardov, zagotavlja svetovalne storitve, splošno svetovanje in podpirajo deljenje rešitev in rezultatov.
- Cilj fundacije FIWARE je vzpostaviti in spodbujati standarde odprtih platform za pametne rešitve v različnih sektorjih. S pristopom, ki temelji na odprti kodi in razvojno gnanem pristopu, fundacija prispeva k standardnim specifikacijam in se integrira z drugimi široko uporabljenimi tehnologijami. Fundacija FIWARE si skupaj z ostalimi člani prizadeva vplivati na razvoj specifikacij v ustreznih organizacijah in spodbuja njihovo hitro sprejetje na trgu s pomočjo odprto kodnih rešitev. Predvsem je vplivala na vmesnik NGSI-LD API za upravljanje podatkov digitalnega dvojčka.
- GAIA-X je globalno med-podatkovno upravljanje, ki temelji na evropskih vrednotah pomaga pri pripravi specifikacij, orodij in procedur za testiranje skladnosti izdelkov s temi specifikacijami. Gaia-X vzpostavlja skupni okvir upravljanja s posebnimi pravili za povezovanje podatkovnih in infrastrukturnih ekosistemov. Okvir Gaia-X vključuje funkcionalne specifikacije, tehnične specifikacije in kodo. Sodelovanje med podatkovnimi prostori je dosežena s skupnim upravljanjem in dodatnimi tehničnimi sredstvi, ki jih določa DSBA.

Mednarodno združenje za podatkovne prostore (IDSA) si prizadeva spodbujati zaupanja vredno okolje za izmenjavo podatkov z združenimi, globalno certificiranimi podatkovnimi prostori. S spodbujanjem izdelkov in sistemov, certificiranih s strani IDS. IDSA predvideva varno podatkovno gospodarstvo, v katerem lahko podjetja nemoteno izmenjujejo podatke v celotni vrednostni verigi. Ključni publikaciji sta Pravilnik IDSA o upravljanju in IDS-RAM V4, ki podrobno opisuje arhitekturo podatkovnega prostora. Protokol podatkovnega prostora omogoča interoperabilno izmenjavo podatkov in sporazume o uporabi, ki temeljijo na spletnih tehnologijah. Certificiranje, ki je ključno za doseganje podatkovne suverenosti, se upravlja prek IDS-Reference-Testbed. IDSA poudarja razvoj in potrjevanje povezovalnikov podatkovnega prostora za pogajanja o politikah in njihovo uveljavljanje v podatkovnih prostorih [13].



Slika 3: Organizacije združenja DSBA.

Za ustvarjalce podatkovnih prostorov je izziv spremljati vse potencialno uporabne tehnologije. Vsak podatkovni prostor ima različne tehnološke zahteve, ki so odvisne od poslovnega primera, izbire upravljanja in veljavnih predpisov.

Ustvarjalci podatkovnega prostora morajo biti pragmatični glede uporabe standardov in poskušati uporabiti obstoječe rešitve zunaj področja podatkovnega prostora za testiranje primerov uporabe, preden se zavežejo k razvoju [14].

Več akterjev na področju tehnologije podatkovnega prostora si prizadeva za oblikovanje standardov za podatkovni prostor in skupnih referenčnih tehnoloških okvirjev. Ko se začnemo seznanjati s podatkovnimi prostori, se kmalu seznanimo z naslednjimi akterji poleg že omenjenih, BDVA, IDSA, GAIA-X in FIWARE:

- Data Spaces Support Centre (DSSC) - projekt, financiran s strani EK v okviru programa Digitalna Evropa (DEP). DSSC raziskuje potrebe po pobudah podatkovnih prostorov, opredeljuje skupne zahteve in vzpostavlja najboljše prakse za pospešitev oblikovanja suverenih podatkovnih prostorov kot ključnega elementa digitalne preobrazbe na vseh področjih.

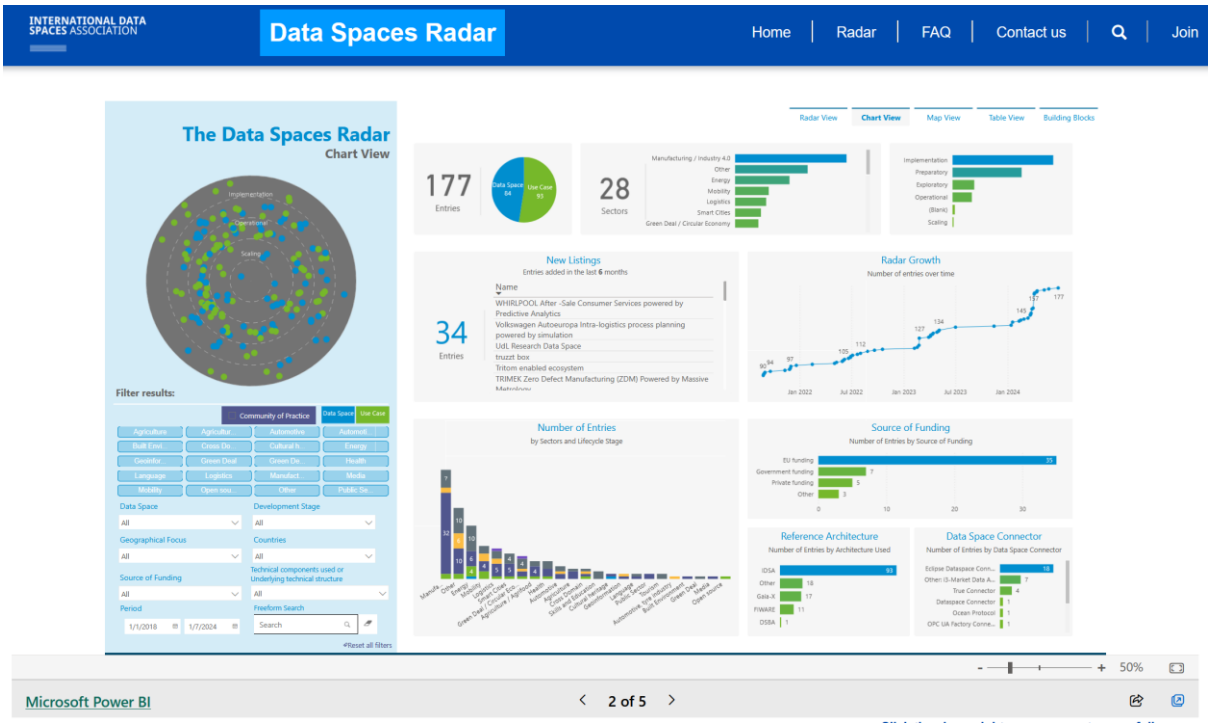
- iSHARE - je evropsko omrežje za mednarodno in suvereno izmenjavo poslovnih podatkov, upravljano s strani Fundacije iSHARE. iSHARE omogoča federirano upravljanje zaupanja podatkovnih prostorov. Ponuja komponente podatkovnih prostorov v skladu z načeli oblikovanja za podatkovne prostore iz projekta Open DEI, IDSA in Gaia-X.

- Eclipse Cross Federation Services Components (XFSC) - je odprtokodni projekt, ki razvija osnovne komponente, potrebne za vzpostavitev federiranih sistemov za izmenjavo podatkov. Pred prenosom na fundacijo Eclipse je bil projekt znan kot Storitve federacije Gaia-X (GXFS).

- Pontus-X, the Gaia-X Web3 ecosystem - si prizadeva zagotoviti decentraliziran in federativni pristop k upravljanju podatkov, kar omogoča, da se podatki, programska oprema, infrastruktura in federacijske storitve varno gradijo, zbirajo, delijo in monetizirajo.

– Eclipse Dataspace Components (EDC) - je odprtokodni projekt, ki si prizadeva implementirati standard Mednarodnih podatkovnih prostorov (IDS) in ustrezne protokole in zahteve, povezane z Gaia-X, ter s tem zagotoviti implementacijo in povratne informacije za te pobude

Komercialne ponudbe, ki se osredotočajo na podatkovne prostore, lahko zagotovijo dragoceno podporo pri obravnavi razvijajočih se različic arhitektur in programske opreme. Trenutno se še majhno, vendar stalno naraščajoče število podjetij osredotoča predvsem na podatkovne prostore ali uvaja izdelke in storitve, specifične za podatkovne prostore, kot del večjega portfelja. Nekatera med njimi so: Advaneo, Dataspace Europe, deltaDAO, IONOS, nexyo, OKP4, sovity in TrustRelay [14].
Ko raziskujemo področje podatkovnih prostorov in implementacij le teh, moramo omeniti tudi radar podatkovnih prostorov (angl. *Data Space Radar*), ki je koristen vir za vse, ki so vključeni v razvoj, upravljanje in uporabo podatkovnih prostorov ter za tiste, ki želijo spremljati in razumeti dinamične spremembe v tem hitro razvijajočem se področju [15].
Radar podatkovnih prostorov, prikazan na sliki 4, je javno dostopno interaktivno orodje, namenjeno panoramskemu pregledu različnih pobud za podatkovne prostore po vsem svetu. Omogoča vpogled v sektorje, lokacije in razvojne faze teh pobud ter primere uporabe, ki jih omogočajo podatkovni prostori.



Slika 4: Interaktivno orodje Data Space Radar. [15]

V poročilu The Data Space Radar – Data Spaces Support Centre Edition, objavljeno marca 2024, najdemo Predstavljajo izbrane člane skupnosti, ki prihajajo iz različnih sektorjev, kot so pametne skupnosti, proizvodnja, zdravje. Kar dokazuje raznolikost pokrajine podatkovnih prostorov in medpodročno poslanstvo skupnosti.

3.1. Identifikacija specifičnih izzivov na različnih področjih implementacije podatkovnih prostorov

Podatkovni prostori predstavljajo pomemben element v digitalizaciji različnih sektorjev, saj omogočajo učinkovito izmenjavo, upravljanje in uporabo podatkov. Vendar pa se vsako področje sooča z edinstvenimi izzivi, ki vplivajo

na dostopnost, varnost, interoperabilnost in skladnost podatkov. V nadaljevanju bomo predstavili nekatere izzive na področjih, ki jih je opredelila EK (Slika 2).

Zdravje (angl. Health)

Evropski zdravstveni podatkovni prostor (EHDS) je prvi skupni evropski podatkovni prostor, ki bo kmalu urejen z uredbo EU. Izzivi na tem področju vključujejo [17]:

- Pravna skladnost: Boljša uskladitev z zakonodajo EU in nacionalno zakonodajo za preprečevanje pravne negotovosti in zmede.
- Pravice in obveznosti: Jasna opredelitev obveznosti in pravic zdravstvenih delavcev glede podatkov, da ne bi bili odgovorni za podatke, ki jih neposredno ne nadzorujejo.
- Definicije: Jasne in koherentne definicije, kot so 'elektronski zdravstveni podatki', 'imetnik podatkov', 'sistem elektronskih zdravstvenih zapisov', za zagotovitev zakonodajne skladnosti z drugimi pravnimi akti EU.
- Varstvo poslovnih skrivnosti: Omogočiti imetnikom podatkov upravljanje s poslovnimi skrivnostmi in zagotoviti ustrezne ukrepe za njihovo zaščito.
- Prenosi podatkov: Omejitev obdelave zdravstvenih podatkov znotraj EU ter omogočiti prenose v tretje države le pod ustreznimi pogoji, skladnimi z GDPR.
- Mehanizmi za izključitev: Obvezen mehanizem za izključitev posameznika iz podatkovnega prostora, ki se uporablja enotno po vseh državah članicah.

Zdravstveni podatki bodo za sekundarno uporabo v EHDS večinoma deljeni v anonimizirani ali psevdonimizirani obliki. Vendar se pravne definicije in zahteve za anonimizacijo in psevdonimizacijo močno razlikujejo med državami, kar ovira delovanje farmacevtskih, digitalnih in zdravstvenih tehnoloških podjetij ter osnovni cilj EHDS - omogočiti nemoteno sekundarno uporabo podatkov čez meje. Na ravni EU so potrebna natančnejša navodila, kako izpolniti zahteve GDPR na način, ki uravnoteži tveganja in koristi ter odpravi pravno negotovost in strah pred tožbami za sekundarne uporabnike podatkov. Ključno je ohraniti pravice posameznikov glede varstva podatkov [18].

Kmetijstvo (angl. Agriculture)

V kmetijskem podatkovnem prostoru se pojavljajo specifični izzivi, kot so [19]:

- Povezljivost in infrastruktura: Številna podeželska območja nimajo hitrega interneta, kar zahteva zbiranje podatkov brez povezave ter kasnejšo sinhronizacijo.
- Digitalna izobrazba: Kmetje, večinoma strukturirani v malih in srednje velikih podjetjih, potrebujejo izobraževanje na področju digitalnih tehnologij.
- Podatkovna suverenost: Kmetje pogosto menijo, da pri uporabi digitalnih storitev izgubijo nadzor nad svojimi podatki. Rešitve morajo omogočati enostaven nadzor nad podatkovno suverenostjo.
- Interoperabilnost: Zagotoviti interoperabilnost vzdolž celotne vrednostne verige, da vse zainteresirane strani koristijo podatke za analize in podporo pri odločanju.

Proizvodnja (angl. Manufacturing)

Proizvodni sektor se sooča z izzivi, kot so[20]:

- Upravljanje podatkov: Naraščajoča količina podatkov, ki se zbirajo na vseh stopnjah (tovarna, izdelek, dobavna veriga), zahteva učinkovito upravljanje in analizo.
- Integracija strojnega učenja: Povezovanje lokalnih modelov strojnega učenja, usposobljenih na posameznih lokacijah, v enotno in natančnejšo podatkovno okolje.
- Deljenje podatkov: Različni uporabniki zbirajo podatke o različnih napakah in uspehih, kar vodi v različne učne nize. Učinkovito deljenje teh podatkov v varnem podatkovnem prostoru je izziv.

Energija (angl. Energy)
Energetski sektor se sooča z izzivi, kot so [21]:

- Sodelovanje med konkurenti: Velika podjetja v energetiki, pogosto konkurenti, morajo usklajeno sodelovati za nove storitve - usklajevanje prizadevanj za rast poslovanja, družbeni napredek in dosego ogljične nevtralnosti.
- Zbiranje heterogenih podatkov: Zahteva sodelovanje celotne vrednostne verige energetskega upravljanja, kar vključuje ponudnike energije, izvajalce, operaterje omrežij, agregatorje, potrošnike, upravljavce polnilnih postaj in varnostne organe.

Mobilnost (angl. Mobility)
Mobilnost se sooča z izzivi, kot so [22]:

- Konkurenčnost ponudnikov: Ponudniki različnih prevoznih možnosti (javni prevoz, taksiji, najem vozil, e-skuterji) ne zaupajo drug drugemu, kar preprečuje deljenje podatkov.
- Skladnost z GDPR: Uredba GDPR se uporablja kot izgovor za ne deljenje ključnih podatkov, kar ovira sodelovanje.
- Realno-časovno deljenje podatkov: Potreba po deljenju podatkov (lokacija, razpoložljivost, prometni pogoji) v skoraj realnem času zahteva zanesljivo infrastrukturo.

Finance (angl. Financial)
Finančni sektor se sooča z izzivi, kot so [23]:

- Pravna fragmentacija: Kljub prizadevanjem za harmonizacijo pravil obstaja fragmentacija na ravni držav članic pri implementaciji in nadzoru.
- Vrednost za stranke: Zagotavljanje dovolj velike vrednosti za stranke, da bi delile svoje podatke.
- Varovanje podatkov: Varovanje uporabe podatkov v kontekstu umetne inteligence in njihovega združevanja z drugimi viri podatkov.

Skupni izzivi vključujejo zagotavljanje pravne in varnostne skladnosti ter interoperabilnosti podatkov med različnimi sistemi in deležniki. Specifični izzivi v sektorjih, kot so zdravje, kmetijstvo, proizvodnja, energija, mobilnost in finance, so pogosto povezani z infrastrukturo, sodelovanjem in izobraževanjem. Podatkovni prostori za veščine in javno upravo trenutno ne izkazujejo posebej specifičnih izzivov, verjetno tudi zato, ker na tem področju še ni veliko raziskav.

4 Arhitektura

Podatkovni prostori določajo strukturo in način, kako se podatki zbirajo, shranjujejo, obdelujejo in delijo med različnimi deležniki. Eden od ključnih pristopov pri oblikovanju podatkovnih prostorov je poznavanje referenčne arhitekture, ki zagotavlja temeljne smernice za interoperabilnost, varnost in skladnost v podatkovnih prostorih. Te referenčne arhitekture služijo kot ogrodje za gradnjo podatkovnih prostorov, ki omogočajo varno izmenjavo podatkov med organizacijami, hkrati pa zagotavljajo visoko raven zaupanja in zaščite podatkov.
IDS-RAM predstavlja pet slojev referenčne arhitekture, katerih namen je zagotavljanje dobro strukturiranega in obširnega pristopa k podatkovnim prostorom (Slika 5). Sloji zagotavljajo jasno organizacijo opredelitve podatkovnih prostorov. Vsak sloj se osredotoča na določen vidik, pri tem pa ločuje izzive in spodbuja modularnost. To omogoča lažje razumevanje, razvoj in vzdrževanje podatkovnih prostorov. Čeprav vsak sloj naslavlja svoje področje, obstajajo pomembni vidiki, ki vplivajo na vse sloje. IDS RAM to rešuje z vključitvijo treh presečnih vidikov: varnost, certificiranje in upravljanje. To zagotavlja, da so ti pomembni vidiki upoštevani na vseh slojih [24].

Slika 5: Sloji in perspektive referenčne arhitekture IDS RAM. [24]

Poslovni sloj:

- Opredeljuje in razvršča različne vloge, ki jih lahko prevzamejo udeleženci v mednarodnih podatkovnih prostorih in določa osnovne vzorce interakcije med temi vlogami.

Funkcionalni sloj:

- Opredeljuje funkcionalne in nefunkcionalne zahteve mednarodnih podatkovnih prostorov in funkcije, ki jih je treba zagotoviti.

Informacijski sloj:

- Določa informacijski model, skupni jezik, tj. besednjak mednarodnih podatkovnih prostorov, ki ni odvisen od domene kar omogoča združljivost in interoperabilnost, med udeleženci in ostalimi komponentami.

Procesni sloj:

- Določa interakcije med različnimi sestavnimi deli podatkovnih prostorov, ter vsebuje opise procesov in njihovih podprocesov. Ti procesi so povezani s ključnimi predlogi mednarodnega podatkovnega prostora in vključujejo večino vlog, predstavljenih v poslovni plasti.

Sistemski sloj:

- Vloge, ki so določene na poslovnem sloju, in procesi, opredeljeni v procesnem sloju, so preslikani na konkretno podatkovno in storitveno arhitekturo, to je tehnično jedro mednarodnih podatkovnih prostorov.

Perspektiva varnosti:

- Varnostna arhitektura IDS zagotavlja sredstva za identifikacijo naprav v IDS, zaščito komunikacije in transakcij izmenjave podatkov ter nadzor uporabe podatkov po njihovi izmenjavi.

Perspektiva certificiranja:

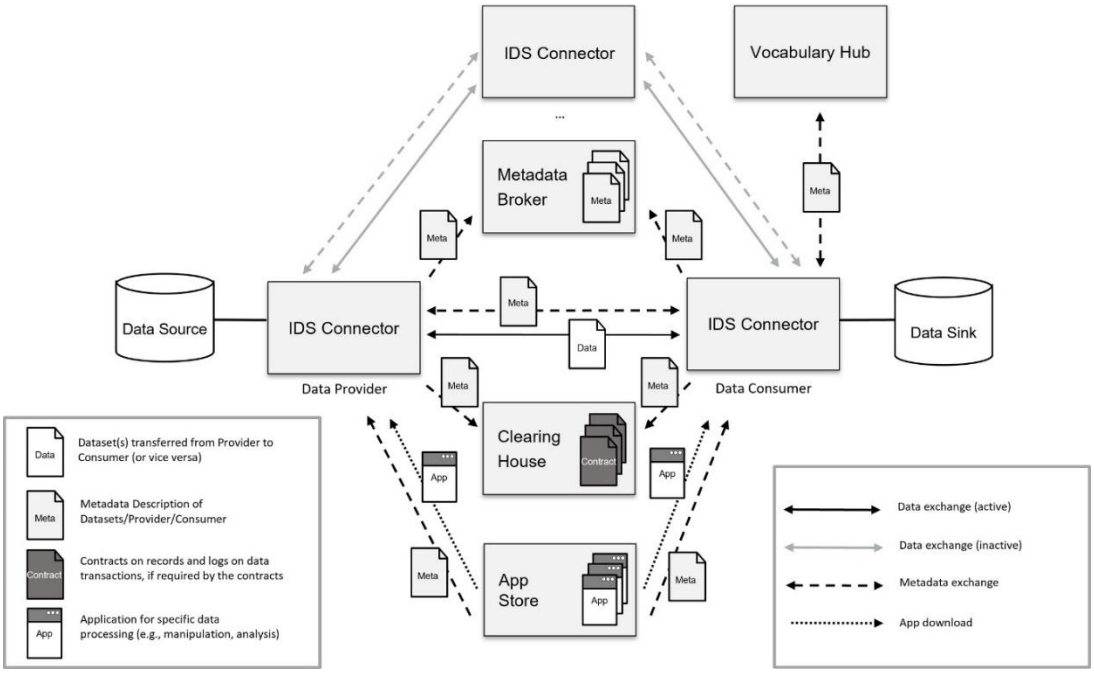
- Za omogočanje nadzora, mora vsak udeleženec upoštevati dogovorjena pravila, zato morajo komponente, preden jih je mogoče uporabiti v sistemu IDS, opraviti certificiranje.

Perspektiva upravljanja:

- Podpira mehanizme upravljanja na podlagi sodelovanja, tako da se dosežejo skupne storitve in ponudbe vrednosti ob hkratni zaščiti interesov vseh akterjev.

S temi ločenimi, a medsebojno povezanimi plastmi sistem IDS RAM zagotavlja celovit okvir za izgradnjo varnih, interoperabilnih in učinkovitih podatkovnih prostorov [24].

Procesi, opredeljeni v procesni plasti, so na sliki 6 povzeti kot interakcije med komponentami IDS. Pri tem pa moramo upoštevati, da ponudnik identitete na sliki ni prikazan zaradi ohranjanja berljivosti. Podatkovni prostori temeljijo na povezovanju različnih udeležencev, pri čemer imajo ti nameščene IDS povezovalnike ali druge osnovne komponente. IDS povezovalnik je odgovoren za sprožitev izmenjave podatkov med notranjimi podatkovnimi viri – podjetniškimi sistemi sodelujočih organizacij in mednarodnimi podatkovnimi prostori. Metapodatkovnemu posredniku zagotavlja metapodatke, kot je določeno v samoopisu IDS povezovalnika, vključno z opisom tehničnega vmesnika, mehanizmi avtentikacije in povezanimi politikami uporabe podatkov. Pogodbe o uporabi podatkov se lahko preko IDS povezovalnika prenesejo v klirinško hišo za zagotavljanje zaupanja. Besedišča je mogoče razlagati tako, da si v vozlišču slovarjev (angl. *Vocabulary Hub*) pridobite več podrobnosti. V IDS povezovalnik je mogoče prenesti dodatne IDS aplikacije za izvajanje različnih operacij na podatkih [25].



Slika 6: Interakcija tehničnih komponent. [25]

Glavna prednost referenčne arhitekture IDS in uporabe povezovalnika IDS je decentralizirano shranjevanje podatkov. To omogoča integracijo podatkov iz različnih virov podatkov in dostop do podatkov izključno prek drugih povezovalnikov IDS. Tako pa je zagotovljena tudi tehnična izvedba podatkovne suverenosti [26].

V podatkovnih prostorih podatkovne zbirke soobstajajo, vendar niso nujno popolnoma integrirane ali homogene v svojih shemah in semantiki. Namesto tega so podatki integrirani po potrebi. Jedro integracije je osredotočeno na entitetne objekte, ki predstavljajo entitete iz resničnega sveta. Poudarek je na tem, kako se te entitete in njihovi odnosi upravljajo znotraj podatkovnega prostora. Podatkovni prostori uporabljajo metodo sprotnega plačevanja (angl. Pay-as-you-go), kar pomeni, da je integracija postopna. Sistem se začne z osnovno integracijo in se nadgrajuje skozi čas. Ta pristop omogoča prilagodljivost in prilagajanje, ko se pojavijo novi podatki ali dodatne zahteve. Na

prvem mestu so torej podatki, model pa se prilagaja. To pomeni, da je začetni poudarek na zbiranju in integraciji razpoložljivih podatkov, nato pa se podatkovni model postopoma prečiščuje in prilagaja, da ustreza tem podatkom. V podatkovnem prostoru so heterogenimi podatki v porazdeljenem sistemu shranjevanja. To pomeni, da lahko obravnavajo različne vrste podatkov iz različnih virov, shranjenih na več lokacijah [27]. Za primerjavo integracije podatkovnega prostora je uporabljena podatkovna baza, kot je prikazano v tabeli spodaj.

Integracija podatkovnega prostora je prilagodljiva in inkrementalna (pay-as-you-go), saj omogoča obdelavo različnih vrst podatkov in porazdeljeno shranjevanje s poudarkom na entitetah in njihovih povezavah. Podatkovni model pa se po potrebi sproti prilagaja.

Tabela 1: Integracija podatkovnih prostorov in podatkovnih baz. [27]

Primerjava integracije	**Podatkovni prostor**	**Podatkovna baza**
Jedro integracije	Entitetni objekt	Poslovne potrebe
Metoda integracije	Pay-as-you-go	Pay-before-you-go
Način integracije	Najprej podatki, nato model	Najprej model
Integrirani podatki	Heterogeni podatki, porazdeljeno shranjevanje	Podatkovna shema, en sam podatkovni vir
Objekti integracije	Podatki in povezava podatkov	Podatki

4.1. Upravljanje dostopa

V mednarodnih podatkovnih prostorih (IDS), nadzor dostopa (angl. Access Control) uravnava zahteve za dostop do podatkovnih storitev. Lastniki podatkov določijo politike ABAC (angl. *Attribute-Based Access Control*) in zahtevane vrednosti atributov za dostop, kot na primer [28]:

- Identiteta povezovalnika (dostop imajo samo določeni povezovalniki (angl. Connestors)).
- Atributi povezovalnika (dostopajo lahko samo povezovalniki z določenimi atributi).
- Varnostni profili (dostopajo lahko samo povezovalniki, ki izpolnjujejo določene varnostne standarde).

Odločitev o nadzoru dostopa se sprejme v povezovalniku z uporabo tehnologij, kot sta XACML ali JAAS, odvisno od implementacije. Varnostna arhitektura IDS sicer ne predpisuje specifične tehnologije za izvajanje nadzora dostopa. Poleg nadzora dostopa pa varnostna arhitektura IDS spodbuja nadzor uporabe podatkov (angl. Usage Control), ki razširja nadzor dostopa z urejanjem tega, kaj se lahko stori s podatki po odobritvi dostopa. To vključuje določitev in uveljavljanje omejitev obdelave podatkov, pri čemer se ne obravnavajo zgolj določila o dostopu, temveč tudi kasnejše obveznosti. To je ključnega pomena za zaščito intelektualne lastnine, skladnost z zakonodajo in upravljanje digitalnih pravic. Nadzor uporabe podatkov v IDS deluje tako, da izmenjanim podatkom priloži informacije o politiki uporabe ter nenehno spremlja njihovo obdelavo in posredovanje. S tem se prepreči napačno ravnanje s podatki, na primer posredovanje osebnih podatkov nepooblaščenim končnim točkam. Služi kot orodje za preprečevanje kršitev varnosti in kot revizijski mehanizem za zagotavljanje skladne uporabe podatkov.

V okviru nadzora dostopa in uporabe sodelujejo različne vloge podatkovnega prostora, te so [28]:

- Posrednik metapodatkov: Upravlja povezovalnikov samo-opis, ki vsebuje tudi nadzor o uporabi.
- Povezovalnik: Izvaja nadzor uporabe. Povezovalci kot ponudniki podatkov morajo za podatke, zagotoviti tehnološko odvisne politike uporabe.

- Klirinška hiša: Spremlja uporabo podatkov in uveljavljanje omejitev uporabe prek sledenja izvora podatkov.
- Trgovina z aplikacijami: Zagotavljajo informacije o tem, ali aplikacije izvajajo tehnologijo za nadzor uporabe.

4.2. Zagotavljanje varnosti in skladnosti s predpisi

Zagotavljanje varnosti in skladnosti v podatkovnih prostorih je ključnega pomena za varovanje podatkov in izpolnjevanje zakonskih zahtev. Kot že omenjeno so lahko podatkovni prostori implementirani na različnih področjih, kot so veščine, zdravje, kmetijstvo, proizvodnja, energija, mobilnost, finance ter javna uprava. Vsako področje ima svoje specifične zahteve in zakonodajne okvirje, ki jih je treba upoštevati pri vzpostavljanju in upravljanju podatkovnih prostorov, da se zagotovi varnost, skladnost in učinkovito delovanje. Med tem ko so nekatere uredbe pomembne pri vseh področjih npr. GDPR za zagotavljanje varstva osebnih podatkov, pa so nekatere uredbe pomembne zgolj na določenih področjih.
Skladnost z zakonodajo v podatkovnih prostorih vključuje vrsto dejavnosti za zagotavljanje skladnosti z ustreznimi pravnimi okviri v celotnem življenjskem ciklu ekosistema za izmenjavo podatkov. To obsega vključitev regulativnih parametrov v zasnovo podatkovnega prostora, usmerjanje organov upravljanja in udeležencev glede implikacij vrednot in predpisov EU ter opredelitev vlog in odgovornosti. Skladnost zahteva tudi oblikovanje notranjih politik in stalno spremljanje upoštevanja predpisov.

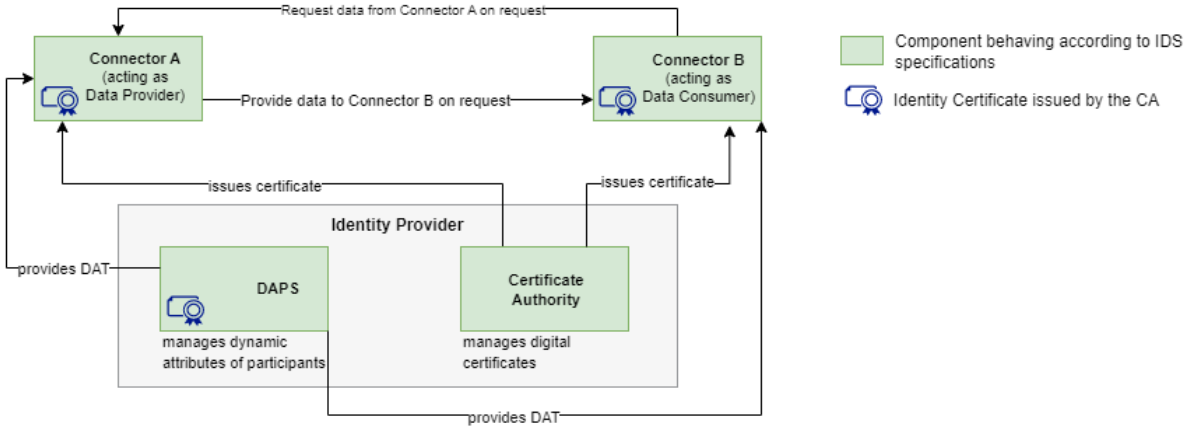
5 Ključni izzivi pri implementaciji

Interoperabilnost podatkovnih prostorov je ključna za omogočanje učinkovitega in varnega izmenjevanja podatkov med različnimi organizacijami in sistemi. Interoperabilnost je mogoče doseči na različnih ravneh:

- **Pravna**: Zagotavljanje, da lahko sodelujejo organizacije, ki delujejo na osnovi različnih pravnih okvirov, politik in strategij.
- **Organizacijska**: Uskladitev procesov, komunikacijskih tokov in politik, ki različnim organizacijam omogočajo smiselno uporabo izmenjanih podatkov v njihovih procesih za doseganje skupno dogovorjenih in vzajemno koristnih ciljev.
- **Semantična**: Sposobnost različnih sistemov, da enotno razumejo podatke, ki se izmenjujejo.
- **Tehnična**: Sposobnost različnih sistemov, da komunicirajo in izmenjujejo podatke.

Okvirja, kot sta ISO/IEC 19941 in Evropski okvir interoperabilnosti (EIF), opredeljujejo te ravni ter zagotavljajo celovito povezovanje in usklajevanje podatkov [30].
Izmenjava podatkov med organizacijami pa zahteva interoperabilnost, kot tudi suverenost podatkov.
Suverenost podatkov, kot je opredeljena v podatkovnih prostorih, se nanaša na zmožnost posameznikov, organizacij in vlad, da imajo nadzor nad podatki, ki jih imajo, in uveljavljajo svoje pravice v zvezi s podatki, vključno z njihovim zbiranjem, shranjevanjem, izmenjavo in uporabo s strani drugih. Da bi zagotovili suverenost podatkov, morajo biti pravila upravljanja podatkov (politike uporabe podatkov, pogodbe o podatkih, pravice do uporabe itd.) jasno izražena in razumljiva za vse udeležence. IDSA teh pravil ne ustvarja, temveč zagotavlja okvir za opisovanje in uveljavljanje pravil, ki jih opredeli organ za upravljanje podatkovnega prostora, to so npr. upravljavci, vlade ali regulativni organi. Imetniki podatkov določijo pravila uporabe, podatkovni povezovalec IDSA pa z obdelavo teh pravil zagotavlja, da jih vsi udeleženci spoštujejo. [31]
Najosnovnejši sprejemljiv podatkovni prostor (angl. Minimum Viable Data Space - MVDS), lahko služi kot osnova za implementacijo v različnih primerih uporabe. MVDS je kombinacija komponent, ki omogočajo oblikovanje podatkovnega prostora z dovolj lastnostmi, da je uporaben za varno in neodvisno izmenjavo podatkov, kot določa

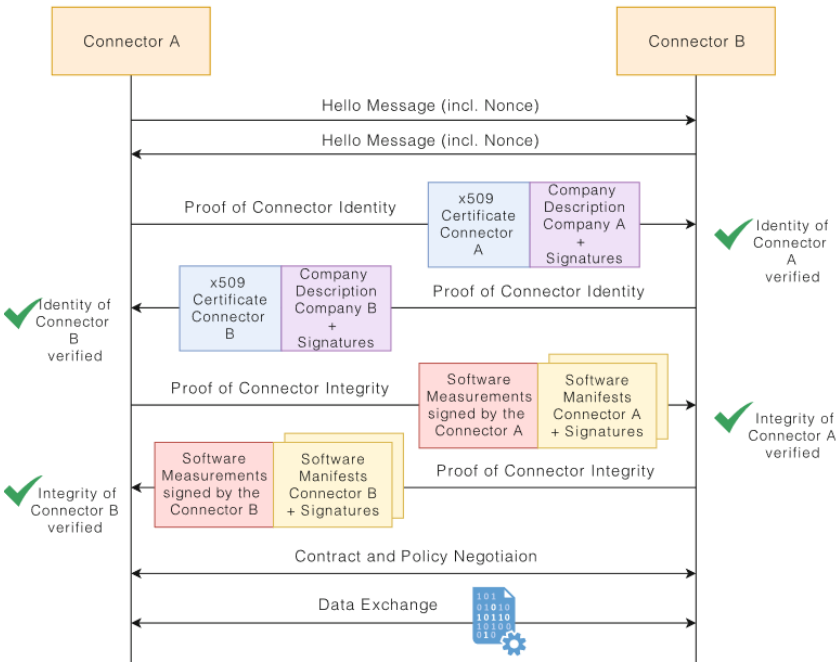
Mednarodno združenje za podatkovne prostore. MVDS mora vključevati vsaj dva povezovalnika, en mora predstavljati ponudnika podatkov drugi pa potrošnika podatkov (Slika 7). Opcijske komponente pa so vse tiste, ki omogočajo posredniške storitve, predstavljene v začetku [32].
MVDS lahko oblikujete tako, da (ponovno) uporabite in prilagodite odprtokodne komponente, navedene na Githubu IDS, ali pa začnete z razvojem nekaterih ali vseh komponent od začetka, tako da sledite specifikacijam komponent. V končni fazi obeh poti je priporočljiva uporaba testnega okolja IDS, da zagotovite združljivost in interoperabilnost komponent, ki jih boste uporabili v svojem MVDS [32].
Vsi povezovalniki in njihovi upravljavci morajo zagotavljati zadostno raven varnosti (podatkov) in pravilno izvajati standarde IDS. Udeleženci podatkovnega prostora morajo zaupati, da drugi udeleženci, ki jih prej niso poznali, resnično izpolnjujejo te zahteve.



Slika 7: Najosnovnejši sprejemljiv podatkovni prostor.[32]

Slika 8 prikazuje poenostavljeno zaporedje vzpostavitve povezave in prve izmenjave podatkov. Koraki so naslednji [33]:

1. Pozdravno sporočilo: Pošlje se pozdravno sporočilo z naključno številko, da se zagotovi svežina in preprečijo napadi ponovitev.
2. Dokaz o identiteti povezovalca: Povezovalci se identificirajo s potrdili X.509 in opisi podjetij. Zasebni ključi vzpostavijo komunikacijski kanal, podpisi pa se preverijo pri organu za potrjevanje (angl. Organ za potrjevanje - CA). S tem se potrdita identiteta in raven certificiranja.
3. Dokaz integritete povezovalnika: Povezovalci preverijo celovitost s pošiljanjem podpisanih manifestov svoje baze zaupanja vrednega računalništva (Trusted Computing Base - TCB) in stanja sistema. Nato se preveri podpise, svežino in verigo artefaktov programske opreme ter preveri manifest pri organu za potrjevanje. S tem se potrdi zanesljivost in raven certificiranja TCB.
4. Pogajanja o pogodbah in politikah: Varna komunikacija partnerjem omogoča pogajanja o pogojih izmenjave podatkov in politikah nadzora uporabe.
5. Izmenjava podatkov: Podatki se lahko varno zamenjajo.

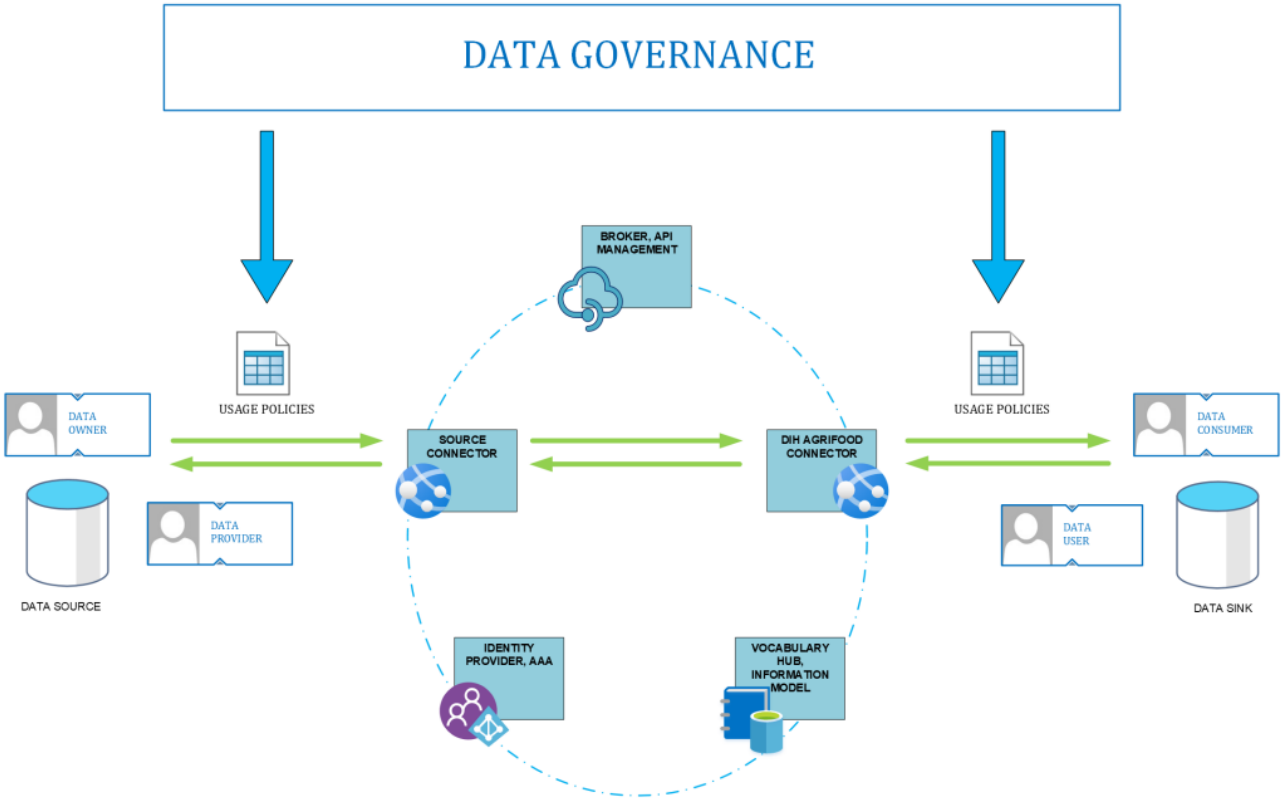


Slika 8: Komunikacija med povezovalniki. [33]

6 Primeri dobre prakse

Kljub temu, da je ideja podatkovnih prostorov na ravni EU šele že zaživela med ključnimi deležniki, ki se strinjajo s potrebo po novem pristopu upravljanja podatkov na bolj transparenten in varen način, raven zrelosti praktičnih rešitev za podatkovne prostore je šele v povojih. Občutno razliko se lahko opazi tudi med različnimi področji, ki je precej odvisna od ravni digitalizacije določenega področja. Kot primer je vzpostavitev podatkovnega prostora za agroživilski sektor zaradi nižje ravni digitalizacije bolj zahtevna kot za Industrijo 4.0, saj je potrebno vložiti precej več resursov v pripravo ustreznih podatkov in vključevanje ključnih deležnikov.

DIH Agrifood Data Space (v nadaljevanju DADS) predstavlja federativno platformo za deljenje podatkov, s čimer si prizadeva zagotoviti okoljsko, ekonomsko in družbeno trajnostnost. Za tehnično vzpostavitev in vzdrževanje infrastrukture DADS podatkovnega prostora skrbi Inovacijsko Tehnološki Grozd (angl. Innovation Technology Center, ITC) Murska Sobota, medtem ko za nadzor implementacije upravljavskega modela skrbi digitalno inovacijsko stičišče za kmetijstvo in proizvodnjo hrane DIH AGRIFOOD sestavljeno iz več partnerjev.



Slika 9: Tehnična infrastruktura DADS podatkovnega prostora. [34]

Infrastruktura DADS pod. prostora omogoča katerem koli zainteresiranem lastniku, ponudniku ali uporabniku podatkov v Sloveniji in širši regiji, da deli oz. uporablja podatke zbrane na podlagi aktivnosti deležnikov agroživilske verige (kmetje, predelovalci, dobavniki, HoReCa sektor in potrošniki). Na ta način DADS predstavlja temelj razvoja modela podatkovne ekonomije v slovenskem agroživilskem sektorju in pospešuje razvoj naprednih storitev na podlagi podatkov, ki prinašajo dodano vrednost za vse deležnike.

Vzpostavitev DADS pod. prostora vključuje vzpostavitev ključnih tehničnih in upravljavskih gradnikov, ki sledi konceptom in metodologiji IDSA predstavljenimi v poglavju 2. Iz stališča tehničnih komponent, DADS pod. prostor vključuje naslednje gradnike:

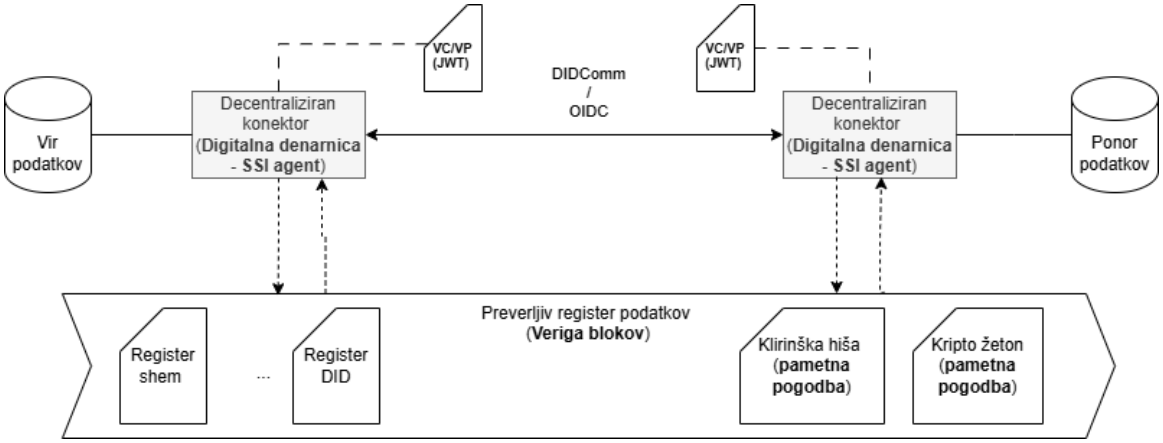
- DIH AGRIFOOD Identity Management – rešitev za upravljanje identitet po principu protokola OpenID Connect;
- Consent Management Platform – spletna platforma za lastnike in uporabnike podatkov, ki omogoča upravljanje dovoljenj in zahtev za deljenje podatkov;
- API Market Service – storitev za upravljanje programskih vmesnikov, ki omogoča pregled dostopnih podatkovnih množic in vmesnikov za njihovo deljenje;
- DIH AGRIFOOD IDS Connector – povezovalnik kompatibilen z IDSA protokolom, ki omogoča vključevanje in deljenje podatkov v podatkovnem prostoru;
- Posrednik (v razvoju) – zasebna vrsta IDS povezovalnika za registracijo in vzdrževanje samoopisov (angl. Self-Description) IDS povezovalnikov v podatkovnem prostoru;
- Posredniška hiša (v razvoju) – storitev za sledenje transakcij deljenja podatkov v pod. prostoru;
- Vozlišče slovarjev (v razvoju) – spletna platforma za deljenje in vzdrževanje semantičnih in podatkovnih modelov za podatkovne množice dostopne v podatkovnem prostoru.

Upravljavski model DADS pod. prostora je skladen z relevantnimi smernicami in regulativami Evropske komisije, kot so Zakon o upravljanju podatkov, Zakon o podatkih (angl. Data Act) ter Evropska strategija za podatke (angl. EU Strategy for Data). Ker še vedno ne obstaja celovit upravljavski model za podatkovne prostore (ali specifično

za agroživilski podatkovni prostor) je DADS upravljavski model razvit kot razširitev Pravilnika za FAIR podatkovno ekonomijo, oblikovanega s strani organizacije Sitra [35], hkrati pa upošteva predloge organizacij IDSA, OpenDEI in BDVA. Upravljavski model vsebuje definicijo vlog Ustanovnih članov (angl. Foundation Member), Upravnega odbora (angl. Steering Commitee) ter ostalih članov, ki prispevajo k upravljanju DADS pod. prostora ter procedur za vključitev v pod. prostor, pravila za deljenje podatkov in jasne predloge za pogoje uporabe podatkov (angl. Dataset Terms of Use), ki zagotavljajo transparentnost in upoštevanje organizacijskega in pravnega okvirja pri deljenju podatkov.

7 Inovativni pristopi in tehnologije

Zaradi temeljne ideje podatkovnih prostorov, da se pospeši nadzirano deljenje podatkov, pri čemer pa lokacija podatkov ni nujno centralizirana, se kot potencialno zanimiv skupek tehnologij pojavljajo tehnologije decentralizacije. Web3 tehnologije, ki temeljijo na decentraliziranih principih, tehnologiji veriženja blokov itn., ponujajo napredne rešitve za vzpostavitev varnih, preglednih in zaupanja vrednih podatkovnih prostorov. To poglavje predstavlja, kako lahko različne Web3 komponente, kot so decentralizirane identitete (DID), pametne pogodbe, verige blokov, preverljive poverilnice in preverljiva dokazila, pripomorejo k doseganju teh ciljev (Slika). V nadaljevanju predstavimo tehnološke koncepte, ki se lahko uporabijo za doseganje MVDS, ki je skoraj v celoti decentraliziran.



Slika 10: Primer arhitekture podatkovnih prostor, temelječe na decentraliziranih tehnologijah.

Decentralizirane identitete (DID) predstavljajo temelj za vzpostavitev zaupanja vrednega okolja v podatkovnih prostorih. DID omogočajo posameznikom in organizacijam, da imajo popoln nadzor nad svojo digitalno identiteto brez potrebe po osrednji avtoriteti ali centralizirani infrastrukturi javnih ključev. S pomočjo DID lahko deležniki varno in enostavno vzpostavijo identiteto ter komunicirajo z drugimi deležniki znotraj podatkovnega prostora na osnovi protokolov komunikacije, kot so DIDComm. Ključne prednosti uporabe DID vključujejo:

- Samostojna identiteta: Deležniki lahko sami upravljajo svoje identitete, kar zmanjša tveganje zlorab in poveča zasebnost.
- Interoperabilnost: DID standardi omogočajo interoperabilnost med različnimi sistemi in platformami, kar olajša sodelovanje med različnimi deležniki.
- Zaupanje in varnost: DID uporabljajo kriptografske tehnike za zagotavljanje avtentičnosti in integritete identitet, kar povečuje zaupanje med deležniki.

Pametne pogodbe so samodejno izvajajoči se programi, ki tečejo na verigi blokov in omogočajo avtomatizacijo in varno izvajanje pogodb. V kontekstu podatkovnih prostorov lahko pametne pogodbe služijo kot klirinška hiša,

ki zagotavlja zanesljivo in transparentno izvajanje transakcij med deležniki. Uporaba pametnih pogodb prinaša naslednje prednosti:

- Avtomatizacija procesov: Pametne pogodbe avtomatizirajo izvajanje pogodb, kar zmanjšuje potrebo po posrednikih in povečuje učinkovitost.
- Transparentnost: Vse transakcije, izvedene s pametnimi pogodbami, so zapisane na verigi blokov, kar omogoča preglednost in sledljivost.
- Varnost in zanesljivost: Pametne pogodbe so odporne na manipulacije in zagotavljajo varno izvajanje pogodb, kar zmanjšuje tveganje napak in goljufij.
- Upravljanje dostopa: Lastniki podatkov lahko s pomočjo pogodb definirajo pogoje, pod katerimi dovolijo dostop do svojih podatkov.

Kripto žetoni, podprti s pametnimi pogodbami, igrajo pomembno vlogo v podatkovni ekonomiji znotraj podatkovnih prostorov. Ti žetoni omogočajo ustvarjanje ekonomskih spodbud za izmenjavo podatkov, olajšujejo transakcije in zagotavljajo pravično nagrajevanje za prispevek k ekosistemu. Ključne funkcije kripto žetonov v podatkovnih prostorih vključujejo:

- Spodbujanje izmenjave podatkov: Kripto žetoni lahko služijo kot nagrada za deležnike, ki delijo svoje podatke, s čimer spodbujajo prostovoljno izmenjavo podatkov in povečujejo obseg dostopnih informacij.
- Olajšanje transakcij: S pomočjo pametnih pogodb se lahko kripto žetoni uporabljajo za samodejno in varno izvajanje transakcij med deležniki, kar zmanjšuje potrebo po posrednikih in zmanjšuje transakcijske stroške.
- Nagrajevanje prispevkov: Deležniki, ki prispevajo k razvoju in vzdrževanju podatkovnega prostora, lahko prejmejo žetone kot nagrado za svoje delo, kar spodbuja aktivno sodelovanje in prispevanje k skupnosti.
- Zagotavljanje pravic in dostopa: Kripto žetoni se lahko uporabljajo za upravljanje dostopa do podatkov in storitev znotraj podatkovnega prostora. Deležniki lahko s pomočjo žetonov pridobijo dostop do specifičnih podatkov ali storitev, kar omogoča fleksibilno in pravično upravljanje pravic.
- Podpora decentraliziranim avtonomnim organizacijam (DAO): Kripto žetoni omogočajo vzpostavitev DAO, ki lahko upravljajo podatkovne prostore na decentraliziran način. Člani DAO lahko glasujejo o pomembnih odločitvah in upravljajo sredstva ekosistema s pomočjo žetonov, kar zagotavlja transparentnost in participacijo.

Preverljiv register podatkov (ang. Verifiable Data Registry - VDR) in **zaupanja vredni registri** (angl. trusted registries) sta ključni komponenti za upravljanje in hranjenje podatkovnih slovarjev v podatkovnih prostorih. VDR, ki predstvalja verigo blokov, ki ponuja različne zaupanja vredne register, omogoča beleženje in iskanje podatkovnih nizov ter zagotavlja zanesljivo in preverljivo shranjevanje podatkovnih shem in drugih ključnih seznamov. Te komponente prispevajo k:

- Organizaciji podatkov: Registri zagotavljajo centralizirano mesto za shranjevanje in upravljanje podatkovnih nizov in shem, kar olajša dostopnost in uporabo podatkov.
- Verifikacija podatkov: zaupanja vredni registry omogočajo preverjanje veljavnosti in celovitost, kar zagotavlja zanesljivost podatkov.
- Interoperabilnost: Standardizacija podatkovnih shem omogoča lažjo integracijo in interoperabilnost med različnimi sistemi in platformami.

Preverljive poverilnice (angl. Verifiable credentials - VC) in **preverljiva dokazila** (angl. Verifiable presentations - VP) predstavljajo napreden način za zagotavljanje nadzora dostopa do podatkov v podatkovnih prostorih. Preverljive poverilnice omogočajo deležnikom, da dokažejo svojo identiteto in pravice do dostopa do določenih podatkov, medtem ko preverljiva dokazila, ki se lahko tvorijo tudi v oblikah, kot je JWT, zagotavljajo dokazovanje

avtentičnosti in celovitosti podatkov ter nudijo ključ dostopa do podatkov. Ključne prednosti teh tehnologij vključujejo:

- Zaupanje in varnost: Preverljive poverilnice in dokazila temeljijo na kriptografskih tehnikah, ki zagotavljajo visoko stopnjo varnosti in zaupanja.
- Enostavnost uporabe: Deležniki lahko enostavno izdajo, preverijo in upravljajo poverilnice in dokazila, kar olajša nadzor dostopa do podatkov.
- Skladnost z regulativami: Te tehnologije omogočajo izpolnjevanje različnih regulativnih zahtev, kar zagotavlja skladnost in zanesljivost podatkovnih prostorov.

V ekosistemu podatkovnih prostorov igrajo **digitalne denarnice** in **SSI** (angl. Self-Sovereign Identity) **agenti** ključno vlogo pri upravljanju decentraliziranih identitet (DID) in preverljivih poverilnic (VC). Digitalne denarnice in SSI agenti služijo kot osrednja orodja za upravljanje identitet in poverilnic, ter lahko delujejo kot IDS (International Data Spaces) konektorji, ki omogočajo varno in učinkovito izmenjavo podatkov med različnimi deležniki.
Inovativni pristop k podatkovnim prostorom, ki temelji na Web3 tehnologijah, ponuja številne prednosti, vključno z večjo varnostjo, transparentnostjo, učinkovitostjo in decentralizacijo. Vendar pa je treba premagati tudi določene izzive, kot so regulativna skladnost, tehnična kompleksnost in uporabniška sprejemljivost. Kljub tem izzivom, potencial Web3 tehnologij za transformacijo podatkovnih prostorov predstavlja pomemben korak k prihodnosti digitalnega poslovanja, kjer bo sodelovanje in izmenjava podatkov med deležniki postala bolj varna, učinkovita in zanesljiva.

8 Zaključek

Podatkovni prostori predstavljajo odprto in povezano infrastrukturo za varno izmenjavo podatkov v skladu s skupnimi pravili, standardi in politikami. EK je februarja 2020 predstavila Evropsko strategijo za podatke z jasnim ciljem ustvariti enoten trg za podatke, ki bi omogočil polno izkoriščanje vrednosti podatkov v korist evropske družbe in gospodarstva. Strategija določa smernice za oblikovanje skupnih evropskih podatkovnih prostorov na več ključnih področjih. Kljub velikemu potencialu podatkovnih prostorov se soočamo z mnogimi izzivi. Ključni med njimi so zagotavljanje interoperabilnosti na različnih ravneh ter osredotočenost na dostopnost in suverenost podatkov, kar lahko ovira implementacijo minimalno funkcionalnih podatkovnih prostorov.
V prispevku smo osvetlili pomen in kompleksnost arhitektur podatkovnih prostorov ter pregledali trenutno stanje pristopov k vzpostavitvi podatkovnih prostorov. Osredotočili smo se na identifikacijo in analizo izzivov v specifičnih domenah in glavnih deležnikov podatkovnih prostorov, s poudarkom na IT arhitekturnih predstavitvah. Predstavili smo tudi inovativne pristope, ki omogočajo identifikacijo podatkov in upravljanje dostopa do njih s pomočjo decentraliziranih tehnologij.
V naslednjih mesecih se bodo raziskave morale osredotočiti na razvoj standardov in protokolov, ki omogočajo interoperabilnost med različnimi podatkovnimi prostori in platformami. To vključuje semantično, organizacijsko, pravno in tehnično interoperabilnost. Prav tako bi se nadaljnje raziskave morale osredotočiti na razvoj metod za učinkovito upravljanje podatkovnih prostorov, vključno z avtomatizacijo procesov, nadzorom dostopa in upravljanjem identitet. Priporočamo izvedbo pilotnih projektov in študij primerov v različnih sektorjih, kot so zdravstvo, kmetijstvo, proizvodnja, energija, mobilnost in finance. To bo omogočilo vrednotenje in izboljšanje podatkovnih prostorov v realnih okoljih ter identifikacijo specifičnih izzivov in priložnosti.
Podatkovni prostori imajo potencial, da postanejo ključna platforma za izmenjavo in uporabo podatkov v prihodnosti. Z nadaljnjimi raziskavami, razvojem in sodelovanjem med deležniki lahko premagamo obstoječe izzive in izkoristimo vse priložnosti, ki jih podatkovni prostori ponujajo. S tem bomo prispevali k ustvarjanju digitalno preoblikovane družbe, ki temelji na podatkih, in spodbudili inovacije, gospodarsko rast in izboljšanje kakovosti življenja za vse.

Literatura

[1] „Building a data economy — Brochure | Shaping Europe's digital future". Pridobljeno: 13. marec 2024. [Na spletu]. Dostopno na: https://digital-strategy.ec.europa.eu/en/library/building-data-economy-brochure

[2] J. Conde, A. Pozo, A. Munoz-Arcentales, J. Choque, in Á. Alonso, „Fostering the integration of European Open Data into Data Spaces through High-Quality Metadata", feb. 2024, [Na spletu]. Dostopno na: http://arxiv.org/abs/2402.06693

[3] „Building data spaces for sustainability use cases - AWS Prescriptive Guidance". Pridobljeno: 13. marec 2024. [Na spletu]. Dostopno na: https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-building-data-spaces/introduction.html

[4] „Common European Data Spaces | Shaping Europe's digital future". Pridobljeno: 11. april 2024. [Na spletu]. Dostopno na: https://digital-strategy.ec.europa.eu/en/policies/data-spaces

[5] E. Curry, „Real-time Linked Dataspaces Enabling Data Ecosystems for Intelligent Systems".

[6] Z. Boukhers, C. Lange, in O. Beyan, „Enhancing Data Space Semantic Interoperability through Machine Learning: a Visionary Perspective", *ACM Web Conference 2023 - Companion of the World Wide Web Conference, WWW 2023*, str. 1462–1467, apr. 2023, doi: 10.1145/3543873.3587658.

[7] „What are the main elements of a data space? | datos.gob.es". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://datos.gob.es/en/blog/what-are-main-elements-data-space

[8] „A European strategy for data | Shaping Europe's digital future". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://digital-strategy.ec.europa.eu/en/policies/strategy-data

[9] „Data protection - European Commission". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://commission.europa.eu/law/law-topic/data-protection_en

[10] „Data Governance Act explained | Shaping Europe's digital future". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://digital-strategy.ec.europa.eu/en/policies/data-governance-act-explained

[11] „A European strategy for data | Shaping Europe's digital future". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://digital-strategy.ec.europa.eu/en/policies/strategy-data

[12] J. Conde, A. Pozo, A. Munoz-Arcentales, J. Choque and, in A. Alonso, „Fostering the integration of European Open Data into Data Spaces through High-Quality Metadata", Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://opendata.paris.fr

[13] „Technical Convergence Discussion Document".

[14] V. Takanen, P. J. Laszkowicz Advisor Futurice Teemu Toivonen, in A. Poikola, „THE TECHNOLOGY LANDSCAPE OF DATA SPACES Antti Poikola Leading Specialist Sitra The Technology Landscape of Data Spaces THE TECHNOLOGY LANDSCAPE OF DATA SPACES", 2023, Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: www.sitra.fi

[15] „Data Spaces Radar". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://www.dataspaces-radar.org/radar/

[16] „The Data Spaces Radar Data Spaces Support Centre Edition".

[17] „The EHDS: concerns and opportunities following ongoing trilogues". Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://www.eucope.org/the-european-health-data-space-challenges-and-opportunities-with-respect-to-the-different-proposals-currently-subject-to-the-trilogue-negotiations]/

[18] „Implementing the European Health Data Space Across Europe".

[19] M. Šestak in D. Copot, „Towards Trusted Data Sharing and Exchange in Agro-Food Supply Chains: Design Principles for Agricultural Data Spaces", *Sustainability 2023, Vol. 15, Page 13746*, let. 15, št. 18, str. 13746, sep. 2023, doi: 10.3390/SU151813746.

[20] E. Curry *idr.*, „Data Sharing Spaces: The BDVA Perspective", v *Designing Data Spaces*, Cham: Springer International Publishing, 2022, str. 365–382. doi: 10.1007/978-3-030-93975-5_22.

[21] M. Gouriet *idr.*, „The Energy Data Space: The Path to a European Approach for Energy", v *Designing Data Spaces*, Cham: Springer International Publishing, 2022, str. 535–575. doi: 10.1007/978-3-030-93975-5_33.

[22] C. Schlueter Langdon in K. Schweichhart, „Data Spaces: First Applications in Mobility and Industry", v *Designing Data Spaces*, Cham: Springer International Publishing, 2022, str. 493–511. doi: 10.1007/978-3-030-93975-5_30.

[23] „EU financial data space and cloud infrastructure".

[24] „1.2 Purpose and Structure of the Reference Architecture | IDS Knowledge Base“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/introduction/1_1_goals_of_the_international_data_spaces/1_2_purpose_and_structure_of_the_document

[25] „3.5 System Layer | IDS Knowledge Base“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/layers-of-the-reference-architecture-model/3-layers-of-the-reference-architecture-model/3_5_0_system_layer

[26] „Introduction | Dataspace Connector“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://international-data-spaces-association.github.io/DataspaceConnector/Introduction

[27] Y. Liu, D. Niu, S. Geng, J. Sun, in H. Zhang, „Application of Data Integration in Dataspace in Multi-value Chain Collaboration of Electric Power Manufacturing Industry“, *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2022*, str. 292–298, 2022, doi: 10.1109/CSCWD54268.2022.9776190.

[28] „4.1.6 Usage Control | IDS Knowledge Base“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/perspectives-of-the-reference-architecture-model/4_perspectives/4_1_security_perspective/4_1_6_usage_control

[29] „From Data Silos to Data Ecosystems: The role of Data Spaces Support Centre in addressing the legal challenges of the future single market for data - CiTiP blog“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://www.law.kuleuven.be/citip/blog/from-data-silos-to-data-ecosystems-the-role-of-data-spaces-support-centre-in-addressing-the-legal-challenges-of-the-future-single-market-for-data/

[30] „8 Interoperability - Blueprint v1.0 - Data Spaces Support Centre“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://dssc.eu/space/BVE/361660417/8+Interoperability

[31] „The significance of data spaces“.

[32] „What is a Minimum Viable Data Space? | IDS Knowledge Base“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-reference-testbed/minimum-viable-data-space/mvds

[33] M. Huber, S. Wessel, G. Brost, in N. Menz, „Building Trust in Data Spaces“, v *Designing Data Spaces*, Cham: Springer International Publishing, 2022, str. 147–164. doi: 10.1007/978-3-030-93975-5_9.

[34] „DIH AGRIFOOD Data Space – Tehnical Infrastructure“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://dataspace.dih-agrifood.com/technical-infrastructure

[35] „Rulebook for a fair data economy“. Pridobljeno: 31. julij 2024. [Na spletu]. Dostopno na: https://www.sitra.fi/en/publications/rulebook-for-a-fair-data-economy/

Življenjski cikel cevovodov neprekinjene namestitve informacijskih rešitev

Luka Četina, Luka Pavlič

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

CI/CD cevovodi postajajo nepogrešljivo orodje pri razvoju informacijskih sistemov, vendar je njihov življenjski cikel razmeroma slabo raziskan. V članku na podlagi pregleda literature opredelimo CI/CD cevovode, njihovo pogostost, najpogostejše gradnike ter orodja za vzpostavitev CI/CD cevovodov. Predstavimo tudi empirično raziskavo, ki vključuje podrobno analizo 1000 repozitorijev na platformi GitHub. Da bi raziskali življenjski cikel CI/CD cevovodov smo preučili strukturo cevovodov v repozitorijih in spremembe tekom projekta. Ugotovili smo, da so CI/CD cevovodi prisotni v 42% repozitorijev, povprečen čas do njihove vključitve pa je sedem mesecev. Skoraj vsi analizirani cevovodi vključujejo gradnjo, 62% jih vključuje tudi izdajo, 46% testiranje, 26% analizo kode in zgolj 18% namestitev. Analizirali smo tudi spremembe programske kode in cevovodov tekom projekta, ter ugotovili, da se ob povečanju sprememb v kodi poveča tudi število sprememb cevovodov in da spremembe cevovodov v povprečju predstavljajo 4% vseh sprememb v repozitoriju. Ugotovili smo tudi, da se cevovodi najbolj spreminjajo na začetku in koncu razvoja projektov, kar odraža njihovo vzpostavitev in kasnejše prilagoditve ter optimizacije.

Ključne besede:
CI/CD cevovodi
razvoj programske opreme
življenjski cikel CI/CD cevovodov
neprekinjena namestitev
neprekinjena dostava

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.4

1 Uvod

Razvijalci neprestano iščejo načine, kako se z manj truda in brez zmanjšanja kakovosti odzivati na spremembe pri razvoju informacijskih rešitev. Prakse neprekinjene integracije (*ang.* continuous integration), neprekinjene dostave (*ang.* continuous delivery) in neprekinjene namestitve (*ang.* continuous deployment), ki jih z okrajšavo zapišemo kot CI/CD, to pomagajo doseči z uvedbo rednih integracij, izdaj ali namestitev programske opreme [1]. Ker so lahko omenjene aktivnosti zelo zamudne oz. zahtevajo veliko človeškega posredovanja, je avtomatizacija pri tem ključnega pomena. Avtomatiziran proces, ki aktivnosti v sklopu neprekinjenih praks poveže, in jih samodejno izvaja, poimenujemo cevovod neprekinjene integracije (*ang.* continuous integration pipeline), dostave (*ang.* continuous delivery pipeline) ali namestitve (*ang.* continuous deployment pipeline). Ta poskrbi za avtomatizacijo pogostih operacij kot so gradnja, testiranje in nameščanje [2].

Kljub temu da uporaba tovrstnih cevovodov dokazano skrajša čas do izdaje produkta ter izboljša produktivnost in učinkovitost [3], vpeljevanje le-teh v proces razvoja vseeno ni brez pasti. Neustrezno implementiran cevovod ne bo nudil omenjenih prednosti in morda celo otežil razvoj, saj bodo neustrezno avtomatizirane aktivnost pogosto zahtevale ročno posredovanje [4]. Avtorji [5] so kot glavne ovire pri vzpostavitvi cevovoda identificirali pomanjkanje lahko razumljive dokumentacije, težavno iskanje ter odpravljanje napak, težavno ponovno uporabo delov cevovoda in raznoliko sintakso pri različnih orodjih. Zato je pomembno, da se še pred vzpostavitvijo cevovodov zavedamo potreb ter morebitnih izzivov pri vzpostavitvi. Pomembna pa ni le vzpostavitev, temveč tudi vzdrževanje in dopolnjevanje cevovodov tekom razvoja, saj ti niso zgolj statični elementi, ampak dinamični procesi, ki se prilagajajo spremembam v razvojnem okolju, tehnologijah ter zahtevah projekta. Kljub temu, da je cevovod podporen element, je vseeno samostojen izdelek, ki ga je potrebno po vzpostavitvi vzdrževati in njegov nadaljnji razvoj tudi načrtovati. Življenjski cikel programske opreme je dobro poznan in definiran, medtem ko je razvoj cevovodov tekom projekta razmeroma slabo raziskan. Zampetti idr. [4] poudarjajo, da se cevovodi nenehno spreminjajo in razvijajo skozi čas. Spremembe v cevovodih pogosto odražajo prilagoditve na nove zahteve projektov, optimizacijo procesov in izboljšanje učinkovitosti. Najpogostejše spremembe v cevovodih povezane z optimizacijo faz, dodajanjem novih funkcionalnosti ter izboljšanjem obstoječih procesov. Spremljanje in analiza teh sprememb sta ključnega pomena za razumevanje življenjskega cikla cevovodov ter za zagotavljanje njihove učinkovitosti in zanesljivosti. Pomembno je, da razvijalci aktivno spremljajo in vodijo evolucijo cevovodov, saj to omogoča pravočasno prepoznavanje težav in uvajanje potrebnih izboljšav, kar posledično vodi do bolj stabilnega in učinkovitega razvojnega procesa.

Cilj članka je podrobneje raziskati ter definirati cevovode neprekinjene integracije, izdaje in namestitve, ugotoviti kako pogosto se uporabljajo in katere gradnike vsebujejo. Zanima nas tudi, katera orodja se uporabljajo za vzpostavitev cevovodov ter kako se ti čez čas spreminjajo.

Zaradi omejitev GitHub API in ker smo morali ročno analizirati aktivnost repozitorijev smo v analizo vključili le 1000 repozitorijev. Pri repozitorijih smo se omejili na tiste, napisane v programskem jeziku java, saj smo lahko tako na podlagi ukazov, uporabljenih v cevovodu, bolj zanesljivo identificirali vključene gradnike. Kot predstavnika cevovodov smo določili orodje GitHub Actions, zato smo pridobili le repozitorije, ki so bili ustvarjeni po izidu omenjenega orodja – po maju 2019. Aktivnost repozitorijev smo pregledovali ročno, kar pomeni, da obstaja možnost napak.

V naslednjem poglavju opredelimo cevovode, njihove najpogostejše gradnike in pogostost uporabe. V tretjem poglavju povzemamo sorodna dela, sledi pa mu predstavitev zajema in analize podatkov iz platforme GitHub. Peto poglavje vsebuje omejitve raziskave, v zaključku pa povzamemo ugotovitve in podamo načrte za prihodnje raziskave.

2 CI/CD cevovodi

Izraza »dostava« in »namestitev« se v kontekstu neprekinjenih praks pogosto uporabljata izmenjujoče, uporaba se med avtorji pogosto celo razlikuje, zato se je uveljavil izraz CI/CD cevovod (*ang.* CI/CD pipeline), ki zaobjema cevovode neprekinjene integracije, dostave in namestitve. Tudi ta izraz je pogosto napačno uporabljen saj se ga uporablja za označevanje vsake avtomatizacije oz. cevovoda, ki se uporablja med procesom razvoja programske opreme. Uporaba cevovoda ne pomeni, da pri razvoju sledimo neprekinjeni integraciji, dostavi oz. namestitvi, vendar le, da smo del aktivnosti avtomatizirali [6].

CI/CD cevovode lahko zato definiramo kot proces, ki avtomatizira in vodi informacijsko rešitev skozi posamezne aktivnosti razvoja. Izvajanje aktivnosti se lahko sproži glede na vnaprej definiran urnik, kot odziv na dogodke znotraj repozitorija (npr. nalaganje nove kode v repozitorij in ustvarjanje nove zahteve za združevanje kode - *ang.* pull request) ali ročno [2], [6], [7], [8].

2.1. Orodja za vzpostavitev CI/CD cevovodov

Orodja za vzpostavitev CI/CD cevovodov, nameščena pri stranki (ang. self hosted CI/CD tools), so orodja, ki jih morajo uporabniki oz. razvijalci namestiti in upravljati sami. Ker delujejo lokalno, jih je lažje prilagoditi potrebam razvoja, vendar to pomeni več dela z vzdrževanjem in upravljanjem. Leta 2001 je izšlo prvo širše uporabljano tovrstno orodje z imenom CruiseControl, čez 4 leta pa se pojavi tudi orodje Hudson, ki se zaradi pravnega spora preimenuje v Jenkins [9].

Orodja za vzpostavitev CI/CD cevovodov v oblaku (*ang.* cloud hosted CI/CD tools), so orodja, ki jih oblačni ponudniki ponujajo kot storitev in ne zahtevajo vzpostavitve ali vzdrževanja, zato jim rečemo tudi CI/CD kot storitev (*ang.* CI/CD as a Service). Eno izmed prvih tovrstnih orodij je bil Travis, ki je zaradi preproste integracije s platformo GitHub postal izredno priljubljen, kmalu pa mu je sledil CircleCI, pri katerem je bil poudarek na zanesljivosti, hitrosti ter podpori Docker zabojnikov. Podobna orodja so izdali tudi večji oblačni ponudniki kot so AWS, Google Cloud in Azure [9].

Orodja za vzpostavitev CI/CD cevovodov, integrirana z repozitorji za gostenje kode (*ang.* git hosting integrated CI/CD tools), so podobna tistim v oblaku, le da jih ponujajo platforme za gostenje kode in so že vključena v repozitorije. To pomeni, da orodij ni potrebno povezovati z repozitorji, kar olajša vzpostavitev cevovodov. Prvi so takšno orodje ponudili pri podjetju GitLab leta 2015, čez 3 leta pa je podobno orodje izdalo tudi podjetje GitHub, ki je zaradi tržnice vnaprej pripravljenih akcij (*ang.* actions) v zelo kratkem času postalo izredno priljubljeno [9]. Orodja integrirana z repozitoriji so danes najbolj priljubljena, med njimi pa prevladuje orodje GitHub Actions [2]. Rostami idr. [1] so v sklopu raziskave ugotovili, da sta najbolj priljubljeni orodji GitHub Actions in Travis, saj vsaj enega od njiju uporablja kar 90% GitHub repozitorijev, ki vključujejo CI/CD cevovode.

2.2. Najpogostejši gradniki

CI/CD cevovodi lahko avtomatizirajo vsako aktivnost, ki jo je mogoče zapisati v obliki ukaza ali zunanje skripte, kar pomeni, da so lahko posamezni gradniki cevovodov precej raznoliki. Obstajajo pa aktivnosti, ki so ključne med razvojem in se najbolj pogosto vključujejo v CI/CD cevovode [1], [7], [8], [10]:

- gradnja (*ang.* Build),
- testiranje (*ang.* Testing),
- analiza kode (*ang.* Code Analysis),
- izdaja (*ang.* Release),
- namestitev (*ang.* Deploy).

2.3. Pogostost uporabe

Po poročilu organizacije CD Foundation vsak drugi razvijalec uporablja neprekinjene prakse razvoja programske opreme, medtem ko so drugi avtorji [2] ugotovili, da zgolj 33% projektov na GitHub uporablja orodja za vzpostavitev CI/CD cevovodov. Čas do vključitve cevovoda v projekt se krajša, saj so leta 2012 cevovod v povprečju vključili po enem letu, medtem ko se je do leta 2023 to zmanjšalo na 3 mesece. Avtorji [5] so med študenti računalniških smeri izvedli anketo, ki je pokazala, da se je 42% udeležencev s CI/CD praksami že srečalo v sklopu služb na IT področju.

3 Sorodna dela

Raziskave na področju CI/CD cevovodov so se v zadnjih letih precej razširile, kljub temu pa se le redke ukvarjajo s spremembami cevovoda skozi čas. Najpogosteje naslavljajo orodja za vzpostavitev, pogostost uporabe ter izzive pri vzpostavitvi CI/CD cevovodov.

Rostami Mazrae idr. [1] so na podlagi poglobljenih intervjujev z izkušenimi razvijalci izvedli kvalitativno študijo o uporabi in migracijah CI/CD orodij. Identificirali so razloge za uporabo specifičnih tehnologij, razloge za sočasno uporabo več CI/CD orodij v istem projektu ter pogostost in vzroke za menjavo orodij. Njihove ugotovitve kažejo jasen trend migracije s Travis CI na GitHub Actions, pri čemer so razvijalci kot glavne razloge za spremembo navajali boljšo integracijo s platformo GitHub in lažje upravljanje cevovodov. Avtorji navajajo, da pogostost uporabe več CI/CD tehnologij hkrati kaže na potrebo po boljši podpori za sočasno uporabo teh tehnologij.

Da Gião idr. [2] so izvedli obsežno analizo CI/CD orodij, ki se uporabljajo v GitHub repozitorijih. Analizirali so 612.557 repozitorijev in ugotovili, da 33% teh projektov vključuje vsaj eno orodje za vzpostavitev CI/CD cevovodov. Podobno kot pri [1], se je tudi tukaj izkazalo, da je orodje GitHub Actions daleč najbolj priljubljeno, saj ga uporablja 58% CI/CD projektov. Raziskava je tudi pokazala, da veliko projektov uporablja več kot eno CI/CD orodje hkrati, pri čemer so GitHub Actions, Travis CI in Jenkins najbolj pogosto so-uporabljena orodja. Avtorji navajajo, da je uporaba več CI/CD orodij v istem projektu povezana z željo po izkoriščanju specifičnih prednosti posameznih orodij ter z omejitvami in pomanjkljivostmi posameznih tehnologij.

Zampetti idr. [4] so preučevali razvoj in spremembe CI/CD cevovodov v odprtokodnih projektih. Njihova študija je temeljila na analizi zgodovine sprememb v CI/CD cevovodih izbranih projektov, pri čemer so identificirali vzorce sprememb in prilagoditev, ki so jih izvajali razvijalci. Ugotovili so, da se CI/CD cevovodi skozi čas nenehno spreminjajo, pri čemer so najpogostejše spremembe povezane z optimizacijo procesov, izboljšanjem učinkovitosti in odpravljanjem napak. Kot najpogosteje spremenjeni elementi cevovodov so se pokazale faze in opravila (ang. jobs), sledijo pa jim dodajaje komentarjev in izvajanje refaktoriranja konfiguracije cevovoda.

4 Pridobivanje in analiza repozitorijev iz platforme GitHub

Da bi ugotovili kako pogosti so CI/CD cevovodi, kaj vsebujejo in kako se čez čas spreminjajo, smo tudi sami pridobili in analizirali cevovode odprtokodnih projektov. Za pridobivanje in analizo repozitorijev smo uporabili različna orodja in tehnologije. Za pisanje skript in avtomatizacijo postopka pridobivanja podatkov smo uporabili python, medtem ko smo za dostop do podatkov o repozitorijih in commitih uporabili GitHub API. Za kloniranje repozitorijev in pridobivanje podatkov o commitih smo uporabili Git, za obdelavo in analizo podatkov ter shranjevanje rezultatov v CSV datoteke pa python knjižnico Pandas.

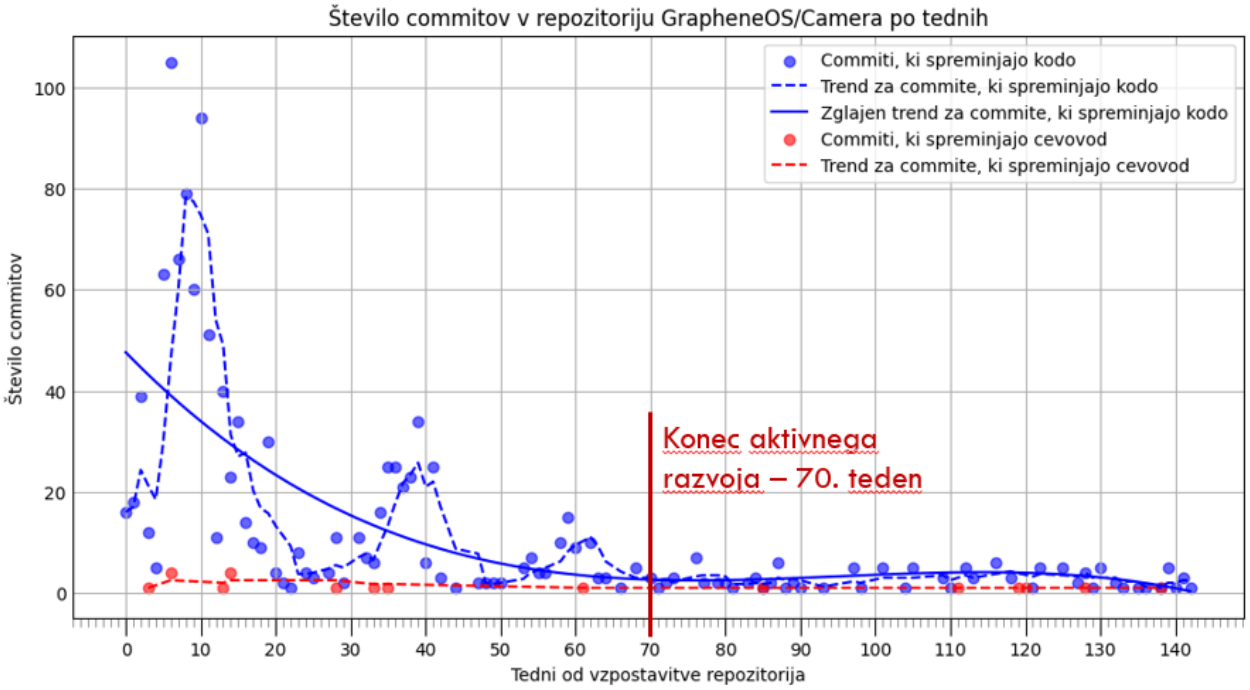
4.1. Pridobivanje in zajem podatkov

Iz platforme GitHub pridobili 1000 najbolj pribljubljenih (*ang.* starred) repozitorijev, napisanih pretežno v programskem jeziku java. Ustrezne repozitorije smo poiskali s pomočjo GitHub API, preko katerega smo pridobili tudi osnovne podatke o repozitoriju. Ali repozitorij vsebuje CI/CD cevovod smo preverili tako, da smo preko GitHub API prenesli vsebino mape .github/workflows in preverili ali vsebuje .yml datoteke.

Za vsak repozitorij smo med drugim pridobili: ime lastnika, ime repozitorija, prisotnost CI/CD cevovoda, privzeto vejo (*ang.* branch), datum vzpostavitve ter zadnje posodobitve repozitorija in velikost. Za vsak repozitorij, ki vsebuje CI/CD cevovod, smo pridobili dodatne informacije o številu in vsebini datotek, ki določajo cevovod ter imena in število opravil (ang. jobs). Na podlagi uporabljenih ukazov in imen opravil smo preverili, ali cevovod vključuje katerega izmed petih identificiranih najpogostejših gradnikov.

Za vsak repozitorij prenesli vse commite in preverili, če ti spreminjajo datoteke v mapi s cevovodi. Ta postopek je vključeval kloniranje repozitorija, pridobivanje podatkov o commitih, ter pregled spremenjenih datotek. Commiti so bili razdeljeni na tiste, ki spreminjajo cevovod in tiste, ki ga ne.

Da bi lahko cevovode projektov med seboj primerjali, smo poiskali repozitorije, kjer se je aktiven razvoj že zaključil. To smo storili tako, da smo vizualizirali število commitov v repozitoriju za vsak teden trajanja razvoja in ročno določili teden, ko je aktivnost upadla in je projekt prešel v fazo vzdrževanja. Primer vizualizacije aktivnosti repozitorija je prikazana na Slika .



Slika 1: Vizualizacija aktivnosti v repozitoriju.

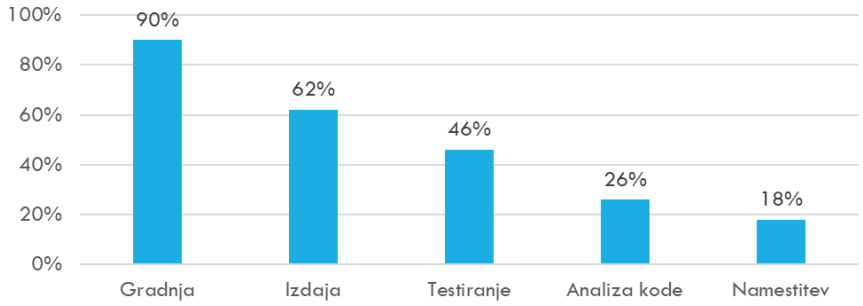
4.2. Pogostost uporabe

Ugotovili smo, da ima izmed 1000 analiziranih repozitorijev, cevovode vzpostavljenih 42% (*N*=415). Povprečen čas do vključitve cevovoda v repozitorij (t.j. čas do prve spremembe datotek v mapi .github/workflows) znaša 7 mesecev, mediana pa 4 mesece. Povprečna starost repozitorijev, ki vsebujejo cevovode, je ob zajemu znašala 38 mesecev, pri čemer je bil najmlajši repozitorij ustvarjen 2 meseca in pol, najstarejši pa 59 mesecev pred zajemom. V primerjavi z da Gião idr. [2], ki so ugotovili, da cevovode uporablja 33% repozitorijev na platformi GitHub, je

naša raziskava pokazala višjo stopnjo uporabe cevovodov. Razlog za takšno razliko lahko tiči v analiziranih projektih, saj so da Gião idr. v raziskavo vključili repozitorije, ustvarjene po letu 2012, medtem ko so bili v našo raziskavo vključeni le tisti, ki so nastali po letu 2019. V času od leta 2012 do 2019 so CI/CD cevovodi postali precej bolj priljubljeni, zato je tudi smiselno, da bodo prisotni v večjem številu repozitorijev. Precej daljši čas do vključitve CI/CD cevovodov v repozitorij bi lahko pripisali peščici dolgo trajajočih projektov, ki nikoli (ali zelo pozno) niso vključili CI/CD cevovoda in so zaradi manjšega števila analiziranih repozitorijev precej vplivali na povprečen čas. V skladu s tem je tudi mediana, ki se precej bolj približa povprečnemu času treh mesecev, ki so ga navedli Giao idr.

4.3. Najpogostejši gradniki

Za vsakega izmed najbolj pogostih gradnikov smo izračunali število projektov, ki ta gradnik vključuje. 90% projektov v cevovodih vključuje aktivnost gradnje, 62% vključuje izdajo in 46% vključuje testiranje. Statična analiza kode in namestitev sta precej bolj redki, saj ju vključuje le 26% oz. 18% projektov. Pogostost vključenosti posameznih gradnikov v cevovod je prikazana na Slika .



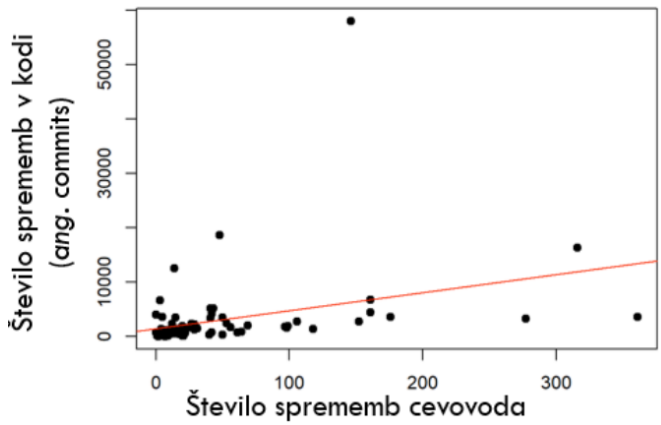
Slika 2: Odstotek projektov, ki v cevovodu vključujejo posamezne gradnike.

Analizirali smo tudi imena opravil znotraj cevovodov in ugotovili, da se skladajo z identificiranimi najpogostejšimi gradniki. Največ projektov je vsebovalo opravilo z imenom »build«, ki mu sledijo »test«, »release«, »publish«, »analyze« in »deploy«. Pojavilo se je tudi ime »job1«, kar kaže, da se pri gradnji cevovodov pogosto uporabijo predloge, ki jih razvijalci kasneje ne spremenijo. Ker se je izkazalo, da nekaj cevovodov ne vključuje nobenega izmed najpogostejših gradnikov, smo preverili katera opravila vsebujejo. V tovrstnih cevovodih so bila najbolj pogosta opravila, ki se ukvarjajo z upravljanjem poročil o napakah, prošenj za nove funkcionalnosti in nalog pri razvoju. Najbolj pogosta imena teh opravil so bila: »issue«, »stale«, »check«, »add« in »label«.

4.4. Spremembe cevovodov

Analiza je pokazala, da spremembe cevovodov predstavljajo približno 4% vseh commitov repozitorija in 8% spremenjenih vrstic v repozitoriju.

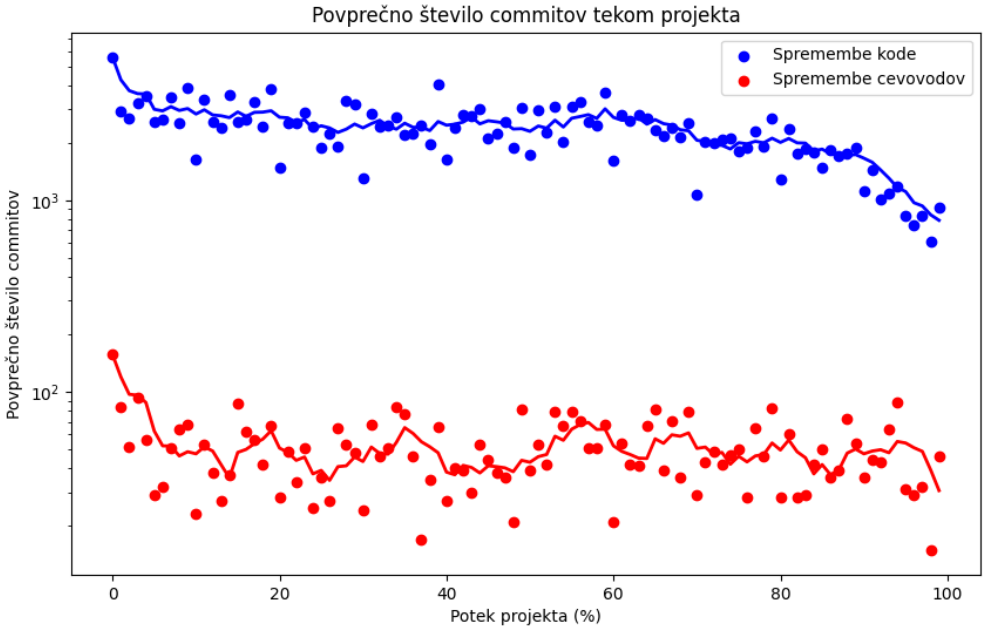
Zanimalo nas je, ali večje število sprememb v kodi v repozitoriju pomeni tudi več sprememb cevovoda, zato smo izvedli Pearsonov test korelacije med številom commitov, ki spreminjajo kodo in številom commitov, ki spreminjajo cevovod. Analiza je pokazala neznatno pozitivno a vseeno statistično značilno korelacijo med omenjenima spremenljivkama, $r(413)=0{,}16$, $p<0{,}01$. To pomeni, da s številom sprememb v kodi raste tudi število sprememb cevovodov. Korelacija je prikazana na Slika .



Slika 3: Korelacija skupnega števila sprememb v kodi in skupnega števila sprememb cevovodov.

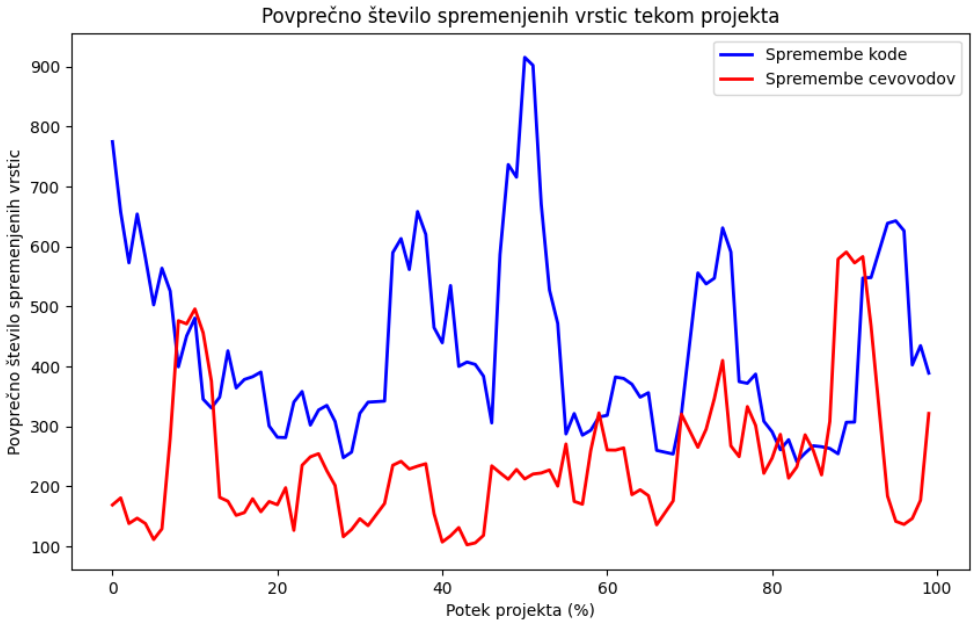
Zanimalo nas je tudi ali enako velja za posamezne mesece. Torej, če se število sprememb v kodi v poljubnem mesecu poveča, se bo povečalo tudi število sprememb cevovoda? Ponovno smo izvedli Pearsonov test korelacije, ki je ponovno pokazal statistično značilno a precej nižjo pozitivno korelacijo med spremenljivkama: $r(15519)=0,08$, $p<0,001$. To nakazuje, da spremembe cevovodov nekoliko sledijo spremembam v kodi, vendar pa se cevovodi spreminjajo z zamikom in ne hkrati s kodo.

Projekte, kjer je aktivni razvoj že zaključen smo po trajanju razdelili na procente in za vsak procent trajanja projekta izračunali povprečno število commitov, ki spreminjajo kodo in commitov, ki spreminjajo cevovodov. Ugotovili smo, da se v povprečju največje število commitov zgodi na začetku razvoja. Pri spremembah v kodi je proti koncu projekta viden rahel upad, ki ga pri spremembah cevovodov ni. Povprečno število commitov tekom projekta je vizualizirano na Slika .



Slika 4: Povprečno število commitov tekom projekta.

Poleg števila sprememb smo analizirali tudi število spremenjenih vrstic za vsak procent trajanja projekta in ugotovili, da število spremenjenih vrstic v cevovodu ne sledi številu spremenjenih vrstic v kodi. Največ spremenjenih vrstic pri cevovodih je takoj po začetku in malo pred koncem razvoja. Povprečno število spremenjenih vrstic tekom projekta je vizualizirano na Slika .



Slika 5: Povprečno število spremenjenih vrstic tekom projekta.

Večje število spremenjenih vrstic v obdobju 5%-15% trajanja projekta je najverjetneje vzpostavitev cevovoda, nihanja med razvojem bi lahko predstavljale manjše popravke, ki zagotavljajo da je cevovod usklajen s spremembami v kodi, obdobje 85%-95% pa je najverjetneje končno usklajevanje cevovoda s projektom in optimizacija za stabilnost in učinkovitost. Po vzpostavitvi se število sprememb na cevovodu s trajanjem projekta počasi viša, kar nakazuje, da se cevovodi s kodo usklajujejo tudi vmes, vendar v precej manjši meri.

5 Diskusija

S pomočjo analize CI/CD cevovodov smo pridobili pomemben vpogled v njihov njihove značilnosti in spremembe skozi čas. Ugotovitve kažejo, da so CI/CD cevovodi prisotni v 42% analiziranih GitHub repozitorijev, povprečen čas do njihove vključitve pa je sedem mesecev. To potrjuje, da so CI/CD cevovodi postali pomemben del sodobnega razvoja programske opreme, vendar še vedno niso univerzalno sprejeti v vseh projektih. Struktura CI/CD cevovodov je prav tako razkrila pomembne trende. Gradnja je vključena v skoraj vseh cevovodih, kar poudarja njen temeljni pomen za avtomatizacijo razvoja. Fazi testiranja in izdaje sta vključeni v približno polovici cevovodov, vključevanje statične analize kode in namestitve pa je najmanj pogosto. Rezultati kažejo, da je ozaveščenost o pomenu gradnje in testiranja visoka, prav tako pa je ti fazi precej preprosto vzpostaviti. Za statično analizo kode in namestitev je potrebna integracija z zunanjimi orodji, ki so pogosto plačljiva, kar je lahko razlog za nizek procent projektov, ki ta gradnika vključujejo. Ugotovili smo tudi, da spremembe v CI/CD cevovodih predstavljajo približno 4% vseh commitov in 8% vseh spremenjenih vrstic v repozitoriju. To kaže na to, da čeprav so cevovodi pomemben del razvoja, predstavljajo le majhen delež celotnih sprememb v projektu. Korelacija med številom sprememb v kodi in spremembami v cevovodih je pokazala neznatno pozitivno, a statistično značilno povezavo, ki postane šibkejša če korelacijo računamo za določeno časovno obdobje. To nakazuje, da se spremembe v cevovodih običajno dogajajo z zamikom po spremembah v kodi. Ta ugotovitev poudarja potrebo po usklajevanju med razvojem kode in cevovoda, da se zagotovi učinkovita avtomatizacija in integracija. Analiza sprememb cevovodov tekom življenjskega cikla projektov je razkrila, da se cevovodi najbolj spreminjajo na začetku in koncu razvoja. Na začetku se spremembe najverjetneje nanašajo na vzpostavitev in zgodnje prilagoditve cevovoda, kar je kritično za postavitev temeljev za nadaljnji razvoj. Spremembe proti koncu razvoja pa lahko odražajo končno

usklajevanje in optimizacijo cevovoda za stabilnost in učinkovitost. Vmesne faze kažejo manjše, a postopno naraščajoče spremembe, kar nakazuje na stalno prilagajanje cevovoda.

Na podlagi teh ugotovitev lahko podamo nekaj napotkov:

- Vključitev CI/CD cevovodov že v zgodnjih fazah razvoja lahko pomaga izkoristiti prednosti zgodnje avtomatizacije in zaznavanja težav. Vključitev je priporočljiva v prvih nekaj mesecih razvoja oz. vsaj v sedmih mesecih.
- Prilagoditve cevovoda je priporočljivo izvajati sproti, da zagotovite usklajenost s spremembami v kodi in preprečite večje težave v kasnejših fazah projekta. Prav tako lahko s tem zmanjšamo število zaključnih uskladitev in prilagoditev.
- Poleg gradnikov gradnje in izdaje je potrebno več pozornosti nameniti tudi testiranju in statični analizi kode, saj lahko s tem pomagamo zagotavljati visok nivo kakovosti kode.

6 Zaključek

Uporaba CI/CD cevovodov je postala ključna za sodoben razvoj programske opreme, saj ti omogočajo hitrejšo in bolj zanesljivo integracijo, dostavo ter namestitev informacijskih rešitev. V sklopu našega dela smo se osredotočili na analizo življenjskega cikla CI/CD cevovodov, pogostost njihove uporabe, strukturo in spremembe skozi čas. Za ta namen smo iz platforme GitHub pridobili 1000 repozitorijev, napisanih pretežno v programskem jeziku java, in analizirali njihove CI/CD cevovode.

Rezultati so pokazali, da so CI/CD cevovodi prisotni v manj kot polovici analiziranih GitHub repozitorijev, pri čemer povprečen čas do vključitve znaša sedem mesecev. Večina cevovodov vključuje fazo gradnje, fazi testiranja in izdaje vključuje približno polovica repozitorijev, fazo namestitve pa manj kot petina. Spremembe cevovodov v povprečju le majhen delež vseh sprememb v repozitorijih, pri čemer obstaja statistično značilna korelacija med številom sprememb v kodi in cevovodih. Naša analiza je pokazala, da se število sprememb v cevovodih povečuje v zgodnjih in poznih fazah razvoja projekta, medtem ko je v vmesnem delu aktivnost manjša, a počasi raste. Večje število sprememb na začetku razvoja verjetno predstavlja vzpostavitev in zgodnje prilagoditve cevovoda, medtem ko spremembe proti koncu razvoja odražajo končno usklajevanje in optimizacijo cevovoda za stabilnost in učinkovitost.

V prihodnjih raziskavah se bomo osredotočili na:

- Raziskovanje sprememb cevovodov glede na značilnosti projekta, kot so velikost projekta, število sodelujočih razvijalcev in tematika projekta.
- Ugotavljanje vrste sprememb cevovodov, ne le kdaj se cevovod spremeni, ampak kateri deli cevovoda se spreminja.
- Identifikacijo najbolj pogostih sprememb v CI/CD cevovodih.

Te ugotovitve predstavljajo pomembno izhodišče za nadaljnje raziskave o uporabi ter značilnostih CI/CD cevovodov ter prispevajo k razumevanju njihove evolucije tekom razvoja. Razumevanje življenjskega cikla CI/CD cevovodov je ključno za boljše načrtovanje, vzpostavitev in vzdrževanje CI/CD cevovodov, kar posledično pomaga izboljšati celoten proces razvoja programske opreme.

7 Literatura

[1] P. Rostami Mazrae, T. Mens, M. Golzadeh, in A. Decan, „On the usage, co-usage and migration of CI/CD tools: A qualitative analysis“, *Empir. Softw. Eng.*, let. 28, št. 2, str. 52, mar. 2023, doi: 10.1007/s10664-022-10285-5.

[2] H. da Gião, A. Flores, R. Pereira, in J. Cunha, „Chronicles of CI/CD: A Deep Dive into its Usage Over Time“, 27. februar 2024, *arXiv*: arXiv:2402.17588. doi: 10.48550/arXiv.2402.17588.

[3] M. Shahin, M. Ali Babar, in L. Zhu, „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices“, *IEEE Access*, let. 5, str. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.

[4] F. Zampetti, S. Geremia, G. Bavota, in M. Di Penta, „CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study“, v *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, sep. 2021, str. 471–482. doi: 10.1109/ICSME52107.2021.00048.

[5] N. Hroncova in P. Dakic, „Research Study on the Use of CI/CD Among Slovak Students“, v *2022 12th International Conference on Advanced Computer Information Technologies (ACIT)*, sep. 2022, str. 458–461. doi: 10.1109/ACIT54803.2022.9913113.

[6] „What Are CI/CD And The CI/CD Pipeline? | IBM“. Pridobljeno: 29. julij 2024. [Na spletu]. Dostopno na: https://www.ibm.com/think/topics/ci-cd-pipeline

[7] „What is a CI/CD pipeline?“ Pridobljeno: 9. februar 2024. [Na spletu]. Dostopno na: https://www.redhat.com/en/topics/devops/what-cicd-pipeline

[8] „CI/CD pipelines | GitLab“. Pridobljeno: 1. julij 2024. [Na spletu]. Dostopno na: https://docs.gitlab.com/ee/ci/pipelines/

[9] D. HQ, „A Brief History of CI/CD Tooling“, Medium. Pridobljeno: 1. julij 2024. [Na spletu]. Dostopno na: https://medium.com/@DiggerHQ/a-brief-history-of-ci-cd-tooling-5a67c2638f3a

[10] „What is a CI/CD pipeline?“, CircleCI. Pridobljeno: 1. julij 2024. [Na spletu]. Dostopno na: https://circleci.com/blog/what-is-a-ci-cd-pipeline/

Optimizacija uporabe CI/CD cevovodov, DevOps pristopa in oblaka

Mihael Škarabot

Cloudvenia, d.o.o., Kranj, Slovenija
mihael.skarabot@cloudvenia.com

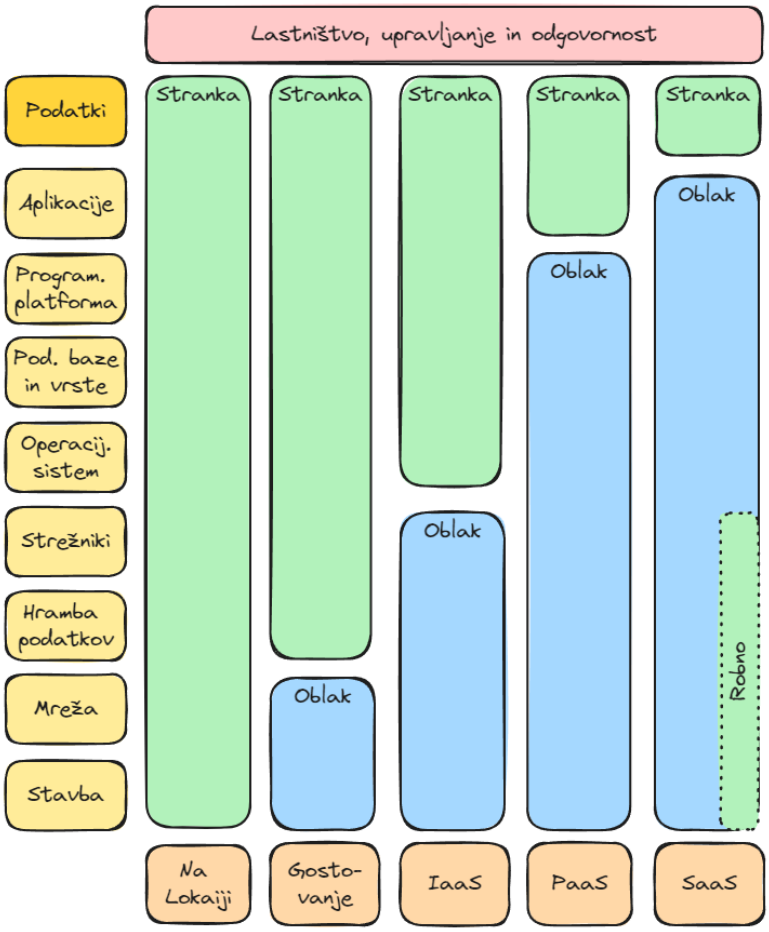
Članek obravnava ključne koncepte neprekinjene integracije, izdaje in nameščanja (CI/CD+CD) kot temeljne elemente za razvoj sodobne programske opreme za nudenje SaaS (ang. Software-as-a-Service). Poudarja pomen DevOps metod in naprednih arhitekturnih pristopov, kot so mikro storitve in modularni monoliti, ki omogočajo večjo skalabilnost, dostopnost, zanesljivost in učinkovitost rešitev. Različni tipi CI/CD cevovodov, prilagojeni specifičnim potrebam, optimizirajo proces razvoja in zagotavljanja programske opreme. Članek izpostavlja prednosti uporabe javnih oblačnih storitev oziroma hiperoblačnih storitev, ki zaradi svoje zanesljivosti, elastične skalabilnosti in transparentnih stroškov predstavljajo idealno infrastrukturo za globalne SaaS rešitve. Prav tako poudarja pomembnost avtomatizacije testiranja in posodobitev skozi CI/CD cevovode, kar prispeva k redkejšim napakam in večji učinkovitosti. Poleg tega članek obravnava strategije za učinkovito upravljanje stroškov v hiperoblaku, vključno z rezervacijami in varčevalnimi načrti, ter različne načine elastičnega skaliranja, kot so vertikalno in horizontalno skaliranje zabojnikov in strežnikov. Poudarja tudi pomen uporabe PaaS storitev za zmanjšanje kompleksnosti in izboljšanje zanesljivosti. Zaključuje z ugotovitvijo, da učinkovita strategija uporabe CI/CD cevovodov in javnih oblakov omogoča podjetjem hitrejše prilagajanje tržnim potrebam ter zagotavljanje visokokakovostnih storitev na globalnem trgu, kar je ključno za uspeh sodobnih SaaS rešitev.

Ključne besede:
oblak
hiperoblačne storitve
DevOps
cevovod
CI/CD
optimizacija

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.5

1 Uvod

Neprekinjena integracija, izdaja in nameščanje (CI/CD+CD) so procesi, ki že desetletja predstavljajo ključen del razvoja programske opreme. V zadnjih letih so zaradi naraščajoče priljubljenosti programske opreme v obliki storitev (ang. Software-as-a-Service, SaaS) v oblaku doživeli prelomnico (slika 1). Na podlagi naših izkušenj ugotavljamo, da uporaba sodobnih DevOps metod ter naprednih arhitekturnih pristopov, kot so mikrostoritve, modularni monoliti in celična arhitektura [1], prinaša številne prednosti pri razvoju SaaS rešitev za globalni trg. Tak pristop ne omogoča le izboljšane skalabilnosti in dostopnosti rešitev, temveč tudi povečuje njihovo zanesljivost in učinkovitost.



Slika 1: Poenostavljeni model oblačnega računalništva in opredelitve SaaS.

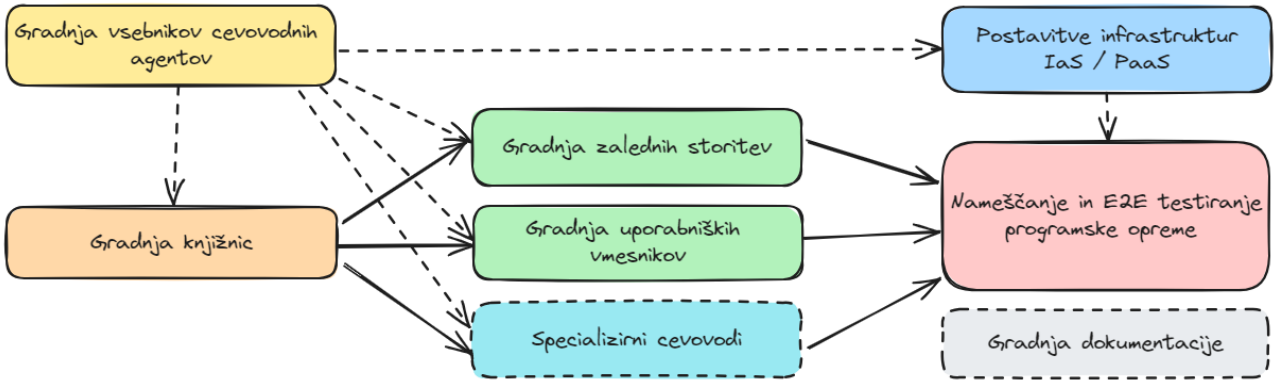
Razvoj programske opreme se lahko izvaja po različnih metodologijah, vključno s klasičnim slapovnim modelom, iterativnim, agilnim ali po pristopu DevOps [5]. DevOps inženirstvo predstavlja sodoben pristop, ki združuje razvojne (Dev) in operativne (Ops) procese v neprekinjen, integriran postopek in preprečuje »metanje čez ograjo«. Ta pristop poudarja sodelovanje, avtomatizacijo ter hitro povratno zanko med ekipami, kar omogoča hitrejše, učinkovitejše in bolj zanesljivo izvajanje programskih rešitev. Avtomatizacija v DevOps omogoča stalno in hitro uvajanje sprememb v produkcijsko okolje brez negativnih učinkov na raven stabilnosti in varnosti.

V slapovnem modelu lahko razvoj nove verzije programske opreme traja od nekaj mesecev do nekaj let. V agilnem razvoju se izdaje programske opreme zgodijo v krajših, rednih in pogostejših ciklih. DevOps pristop gre korak dalje in omogoča uvajanje sprememb v produkcijsko okolje na vsakih nekaj minut, uporabljajoč različne strategije, kot so namestitve kanarčka, modro-zelene namestitve, namestitve na osnovi obročev, zastavice za funkcionalnost, itd. Ta pristop maksimalno skrajša čas od ideje do produkcijske uporabe, poveča agilnost organizacije in izboljša odzivnost na potrebe trga – še posebno to velja za globalni SaaS model.

Globalni trg se zelo učinkovito doseže z uporabo hiperoblačnih storitev. Le-te so se v zadnjih letih izkazale za zelo zanesljive. Nemalokrat bi se jih radi izognili zaradi visokih stroškov, ki lahko nastanejo z nepremišljeno uporabo. Globoko v naši miselnosti je tudi še doktrina lastništva nad strojno opremo. Če želimo to premagati je potrebno oblačno računalništvo pogledati iz vseh vidikov. Primerjave le iz vidika neposrednih stroškov strojne opreme so nepopolne. Ne smemo pozabiti na stroške varnosti, elastične skalabilnosti, dostopnosti, zanesljivosti, tehnične podpore, podpore različnim regulativnim predpisom, hitrosti implementacije, itd.

2 CI/CD cevovodi

2.1. Tipi cevovodov



Slika 2: Tipi cevovodov in njihova odvisnost.

Cevovode za SaaS smo na njihovo namembnost oziroma podobnost korakov razdelili v naslednje tipe:

- *Graditelji vsebnikov cevovodnih agentov:* Uporaba vsebnikov za cevovodne agente omogoča standardizacijo okolja, v katerem se izvajajo vse faze posameznega cevovoda. To je še posebej pomembno, če agente vzdržujemo sami. Takšni vsebniki se lahko uporabijo tudi za agente v oblaku. Alternativa uporabi vsebnikov je nameščanje potrebnih knjižnic in orodij ob vsakem zagonu cevovoda, kar je lahko časovno in iz vidika porabe virov stroškovno neučinkovito.
- *Graditelji knjižnic:* Cevovodi za knjižnice vključujejo specifične korake. Pri gradnji uporabljajo modularne in integracijske teste programov, ki so uporabniki knjižnic. To zagotavlja višji nivo kakovosti knjižnic ob njihovi izdaji.
- *Cevovodi za grajenje storitev:* To so osrednji cevovodi, ki skozi različne faze vodijo do izdaje verzije programske opreme ali namestitvenega artefakta (npr. vsebnika). Vključujejo različne načine testiranja, kot so testi modulov, integracijski testi, zmogljivostni testi in testi ranljivosti. Vse je usmerjeno k cilju zagotavljanja visokokakovostnih izdaj.
- *Cevovodi za grajenje uporabniških vmesnikov:* Cevovodi so podobni grajenju storitev. Ločitev med slednjimi in cevovodi za grajenje uporabniških vmesnikov je smiselna zaradi običajno velikih razlik v tehnologijah, ki so uporabljene za uporabniški vmesnik v primerjavi s tehnologijami za zaledne sisteme.
- *Specializirani cevovodi:* V zalednih sistemih se lahko pojavijo storitve, ki vsebujejo kompleksne algoritme, kot so učenje umetne inteligence, geografsko ali grafično procesiranje, zahtevni matematični procesi, itd. Gradnja takšnih storitev zahteva posebne postopke v cevovodih. To so koraki validacije algoritmov ali rezultatov učenja z uporabo posebnih metrik in širokega nabora raznovrstnih testov.

- *Infrastrukturni cevovodi:* Upravljanje oblaka je učinkovito le s pomočjo koncepta infrastrukture kot kode (ang. Infrastructure-as-Code - IaC). Kodo za IaC je potrebno nadzorovati z vidika statičnih analiz, testiranja in namestitev.
- *Cevovodi za neprekinjeno nameščanje:* GitOps je eden izmed pristopov za izvajanje takšnih cevovodov. Ti cevovodi so zadolženi za implementacijo strategij distribucije in nameščanja programske opreme.
- *Cevovodi za grajenje dokumentacije:* Omogočajo gradnjo artefaktov uporabniških navodil, ki so dostavljena skupaj s programsko opremo in tako neposredno dostopna končnim uporabnikom. Prav tako lahko cevovodi skrbijo za kreiranje tehnične dokumentacije in postavitev portalov za razvijalce.

Tu moramo omeniti, da se instance zgornjih tipov cevovodov lahko sekvenčno ali paralelno povežejo v večje celostne CI/CD cevovode. Njihova soodvisnost je prikazana na sliki 2. Tipizacija cevovodov je uporabna predvsem za postavitev predlog (ang. templates) cevovodov.

2.2. Koraki cevovoda

Skozi praktične izkušnje smo prišli do sledečih korakov v CI/CD cevovodih za SaaS v oblaku (slika 3):

- *Priprava in preverjanje okolja:* Prvi korak vsakega cevovoda je preverjanje, če so na voljo vsi parametri okolja in orodja. S tem deloma preprečimo, da v cevovodu pride do napake šele po dolgem času izvajanja, praviloma v koraku, ker se npr. parametri dejansko uporabijo. Strategija hitrega neuspeha (ang. fail-fast strategy) je zelo učinkovit način izvajanja cevovodov, ki prispeva k manjši uporabi virov in lažjemu iskanju vzrokov. To je še posebno pomembno za cevovode, ki imajo v povprečju relativno dolg čas izvajanja.
- *Statično in oblikovno preverjanje kode (ang. linting and formatting):* Odpravljanje tehničnega dolga v kodi je pomemben del vzdrževanja kode. Statična analiza pomaga ohraniti kodo v stanju, da je njeno vzdrževanje stroškovno učinkovito. Statična analiza in formatiranje kode znatno pripomoreta k zmanjšanju komentarjev in popravkov kode v procesu revidiranja zahtev za združitev vejitev (ang. pull requests, merge requests). V tej stopnji se lahko uporabi kriterij doseganja pragu za določanje uspešnosti tega koraka cevovoda. To omogoči razvijalcem možnost postopne izboljšave kode v primeru, če koda v preteklosti ni bila del takšnih analiz in ima tehnični dolg. V takšnih primerih pomembno, da se prag skozi konsenz ali celo samodejno dviguje.
- *Prevajanje programskih knjižnic, zalednih storitev ali uporabniških vmesnikov:* Pomemben del prevajanja je lokalni predpomnilnik že prevedene kode v agentih cevovodov. S tem znatno pohitrimo prevajanje kode. Korak prevajanja postane zahtevnejši, če je potrebno program prevesti za različne platforme npr. x64 in arm64. Tukaj si lahko pomagamo s paralelnim izvajanjem cevovoda.
- *Testiranje posameznih modulov:* Tem bližje je koda testov poslovni logiki, tem cenejše je izdelava, vzdrževanje in zaganjanje testov [2,3]. Testi modulov (ang. unit-tests) so zato zelo pomemben del cevovoda. Z metrikami testne pokritosti lahko pomagamo razvijalcem, da pokritost kode ohranjajo in/ali povečujejo. Z uvedbo pragov na teh metrikah cevovodi lahko tudi preprečijo padec pokritosti testiranja. Ne smemo pozabiti, da je tudi uporabniški vmesnik možno testirati brez zalednih sistemov s pomočjo imitacij (ang. mocking) in v primeru spletnih aplikacij celo brez spletnega brskalnika. To lahko določenih primerih zelo pospeši postopke testiranja uporabniškega vmesnika.
- *Zagon povezanih storitev:* Za celostno integracijsko testiranje je pomembno, da ima testirana komponenta programske opreme okoli sebe čim več pravi storitev in čim manj imitiranih storitev (ang. mock-ups). Zagon povezanih storitev je korak v cevovodu, ki poskrbi, da se v izvajalno okolje testov zaženejo storitve kot so podatkovne baze, sporočilne vrste ali odvisne programske storitve. V primeru, če je potrebna tudi specialna komponenta iz oblaka (npr. sporočilna vrsta, podatkovna baza, zabojnik za datoteke), je

potrebno s pomočjo IaC poskrbeti, da se postavi v oblaku postavi tudi ta. Če storitve porabijo veliko časa za zagon, imamo lahko tudi bazen takšnih storitev za njihovo ponovno uporabo. Vendar je pomočjo vsebnikov praktično mogoče postaviti najrazličnejše storitve zelo hitro in zanesljivo.

- *Integracijsko testiranje:* S tem korakom preverimo, kako se storitev odziva skozi API-je. Oziroma, kako se obnaša dejanski uporabniški vmesnik v spletnem brskalniku ali mobilni napravi. Stremimo k temu, da se uporabi čim več pravih komponent. To še posebno velja podatkovne baze, sporočilne sisteme in druge storitve izvajalnega okolja. Uporaba imitacij je smiselna v primeru, če nam tretjeosebni dobavitelj ne nudi testnega okolja API-jev, oziroma, če potrebujemo določene nastavitve za vsak potek testiranja, ki jih je nemogoče nastaviti ali ponastaviti samodejno skozi API-je dobavitelja.

- *Testiranje zmogljivosti:* Za takšno testiranje na nivoju posameznih komponent sistema oziroma storitev, je pomembno, da se odločimo ali je testiranje zmogljivosti za to storitev pomembno ali ne. Kandidati so zagotovo osredni deli sistema, ki so kritični iz vidika algoritmov, odločitvenih postopkov, strojnega učenja, itd.

- *Poročanje o kazalnikih testne pokritosti:* Rezultati testiranja modulov, integracijskega testiranja in testiranja zmogljivosti so zelo pomembni za pregled nad tehničnim dolgom, kvaliteto in zadostitvami nefunkcionalnih zahtev. Korak poročanja o kazalnikih je lahko v vlogi preverjanja doseganja določenih pragov in hkrati objave rezultatov testiranja, testne pokritosti, zmogljivosti sistema na posebnih zbirnih mestih. Takšna zbirna mesta služijo razvojnim oddelkom za planiranje vzdrževanja nivoja rezultatov ali planiranje aktivnosti za doseganje določenih pragov kriterijev v prihodnosti.

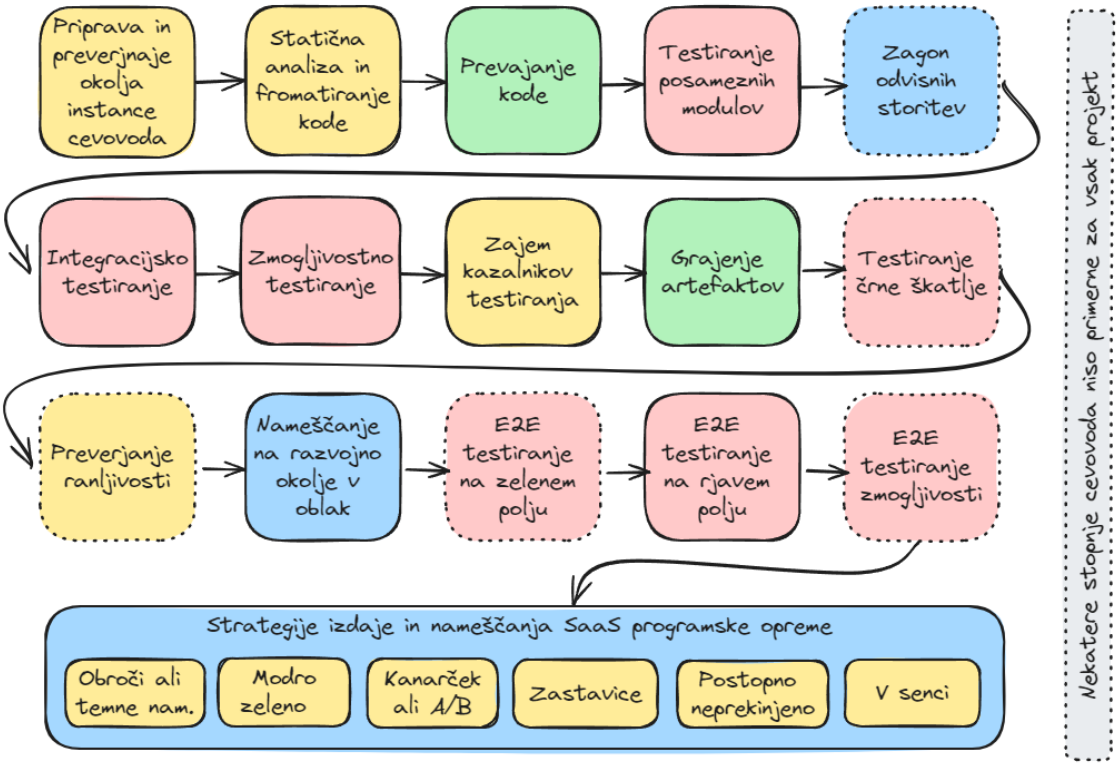
- *Pakiranje in grajenje artefaktov:* Izhodi tega koraka so artefakti programske opreme, kot so najrazličnejše knjižnice programske opreme, vsebniki in druge oblike prevedene in pakirane kode. Ta korak lahko vsebuje tudi določene teste artefaktov. Pogosto se namreč zgodi, da vsi prejšnji nivoji testiranja uspejo, vendar na koncu postopek pakiranja v zabojnik povzroči, da je zabojnik neuporaben oziroma nedelujoč. Če temu koraku neposredno sledijo koraki namestitve zabojnikov v testno okolje, se lahko zanesemo nanje. V primeru, če naš cevovod izda samo verzijo zabojnika in se procesi namestitve v testna in/ali produkcijska okolja izvajajo asinhrono, je smiselno opraviti kratek test zagona zabojnika v obliki črne-škatle. S tem zmanjšamo možnost, da v repozitorij slik vsebnikov objavimo nedelujočo verzijo zabojnika. Predhodna gradnja in uporaba zabojnikov v korakih integracijskega testiranja ni priporočljiva, ker bi težko pridobili metrike pokritosti kode.

- *Preverjanje ranljivosti* (ang. vulnerability check): Namestitve v oblak so lahko postavljene v dveh načinih. V primeru varnostnega modela brez privzetega zaupanja (ang. zero-trust) so storitve dejansko skoraj neposredno izpostavljene internetu. V drugem primeru varnosti na robu (ang. security at the edge), kjer se varnost preverja samo na vhodu v gručo storitev, ima potencialni preboj varnosti še večjo površino. V obeh primerih se priporoča, da se uporabljene knjižnice in druge podsisteme redno pregleduje za ranljivosti.

- *Nameščanje programske opreme na razvojno okolje v oblaku:* V tem koraku poskušamo čim bližje priti postopku, ki je kasneje uporabljen pri strategiji izdaje in nameščanja v produkcijska okolja. Na drugi strani se moramo zavedati, da mora biti postopek hiter in omogočati namestitev oziroma posodobitev le komponent sistema, ki so se dejansko spremenile. Hitrost nameščanja je ključna pri velikih razvojnih ekipah, ki zaradi učinkovitosti stremijo k razvoju na osnovi glavne veje (ang. trunk-based development) [6]. To jim omogoči hitro razrešitev konfliktov v kodi in delovanju sistema.

- *E2E testiranje na zelenem polju:* Celostno testiranje lahko vsebuje testiranje zelenega polja. Tu preverimo, če je naš postopek nameščanja sistema na novo instanco v oblaku še delujoč po spremembah. Oziroma lahko poganjamo tiste teste, ki za svoje delovanje potrebujejo ali prazno podatkovno bazo ali v naprej pripravljen set podatkov, ki je vedno enak

- *E2E testiranje na rjavem polju:* Najbolj običajen način E2E testiranja. Določeno instanco sistema vedno posodabljamo in nad njo izvajamo idempotentne teste. To je tudi korak, ki vsebuje največ E2E testov uporabniških vmesnikov.
- *E2E testiranje zmogljivosti:* Tu gre za korak v cevovodu, ki se ne izvaja vedno. Po potrebi lahko preverimo kritične poti v sistemu, če so po določenih spremembah še zadovoljivo zmogljive. Pomembni del tega koraka je, da imamo popolnoma avtomatizirano postavitev testnega okolja, samo izvajanje testov in izdelavo poročila. Le tako bomo lahko včasih zelo kompleksne zmogljivostne teste po potrebi ponavljaji brez večjih razvojnih in izvajalnih stroškov.



Slika 3: Stopnje celostnega cevovoda za SaaS programsko opremo.

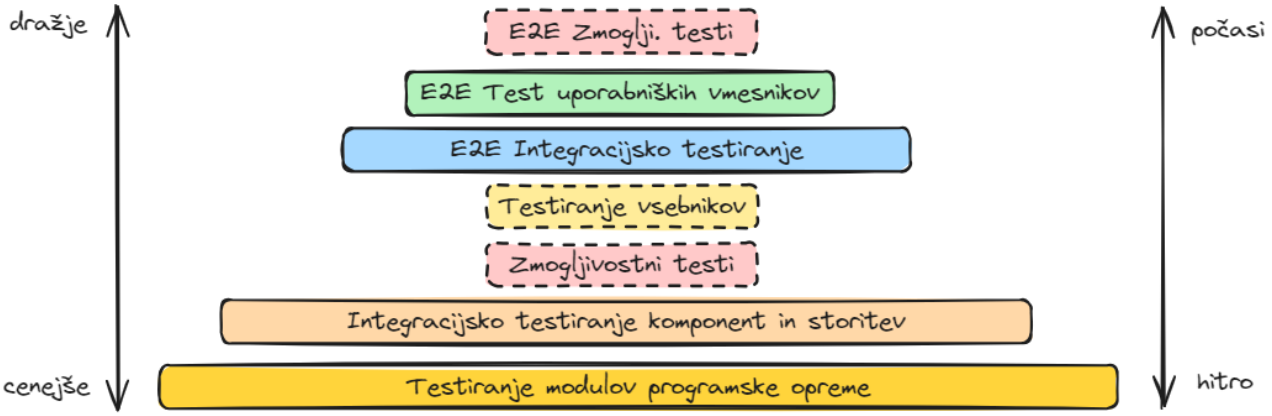
2.3. Izbor korakov cevovoda in optimizacija

Če smo si v prejšnjem poglavju pogledali vse možne stopnje SaaS cevovoda, to še ne pomeni, da mora vsak sistem imeti vse navedene korake v svojem cevovodu. Prav tako ne velja, da mora biti sosledje korakov točno takšen kot je naveden. S tehtnimi analizami moramo opredeliti cevovod, ki bo kar se da najbolje služil kot orodje za doseganje SaaS ciljev oziroma ne-funkcionalnih zahtev. Moramo se zavedati, da vsak korak v cevovodu potrebuje izvajalni čas [3]. Določeni koraki, ko so npr. zmogljivostni testi, lahko v oblaku povzročijo tudi opazne stroške.

Imamo opcijo, da določene korake iz celostnega cevovoda izločimo in jih poganjamo v ločenem cevovodu, ki se izvaja na podlagi urnika. Eden izmed kandidatov je zagotovo korak pregleda ranljivosti kode oziroma vsebnikov. Korak se bo še vedno izvajal, vendar ne bo blokiral izvajanje primarnega cevovoda. Za ločene cevovode ne smemo pozabiti postopke nadzora njihovega izvajanja. Če npr. specializiran cevovod najde ranljivost, se mora sprožiti postopek planiranja odprave ranljivosti v razvojnih ekipah.

K učinkovitemu izvajanju korakov testiranja močno pripomore pravilna struktura testiranja. Čeprav agilni pristopi k razvoju programske opreme brišejo meje med razvojem in testiranju v organizaciji, se pogosto še vedno zgodi, da je razvoj mnenja, da avtomatizirane teste piše specializirano testno osebje za zagotavljanje kakovosti. Slednje je običajno osredotočeno na pisanje E2E testov v zgornjih nivojih testiranja. V tem primeru dobimo sistem, ki za

svoje testiranje v cevovodu porabi veliko časa, saj so E2E testi izmed vseh nivojev testiranja časovno in stroškovno najbolj potratni [1]. Martin Fowler [2] je v letu 2012 opredelil osnovno piramido testiranja, ki jo lahko danes razširimo z novimi nivoji, ki so v povezavi s koraki testiranja v cevovodih (slika 4). Če sledimo strategiji, da je obsežen del vseh testov na nižjih nivojih, smo zagotovo naredili korak v bolj učinkovitemu cevovodu brez negativnih učinkov na doseganje strategije celostnega testiranja, ki je zelo pomembno v cevovodih za SaaS.



Slika 4: Piramida testiranja.

3 DevOps

Osrednji cilji DevOps organizacije so [4]:

- Pogosta posodobitev oziroma namestitev programske opreme,
- hitrejši prihod na trg,
- redkejše napake in odpovedi sistema,
- zmanjšanje časa od iniciacije do prve uporabe nove funkcionalnosti s strani uporabnika,
- zmanjšanje časa od odpovedi do obnovitve.

Cilji DevOps se skladajo z agilnimi pristopi razvoja programske opreme. Ne moremo doseči pogoste posodobitve produkcijskih okolij, če je razvoj programske opreme osnovan na slapovnem pristopu. Agilni pristopi razvoja programske opreme kot so Scrum, Kanban, SAFe, itd. so ključni za doseganje večje frekvence posodobitev SaaS v produkciji [7]. Promovirajo kratke iteracije in integrirano zagotavljanje kvalitete.

Pomemben del pogostih posodobitev in hitrega prihoda na trg je zagotovo strategija izdaje in nameščanja programske opreme. Vzdrževalna okna za posodobitev SaaS v večini primerih ne pridejo v poštev. Za posodobitve SaaS sistema v času uporabe so ključne za strategije nameščanja posodobitev. Lahko izberemo strategijo obroča, temnih namestitev, modro-zelenih namestitev, A/B paralelizma, kanarček namestitev, postopno neprekinjenih posodobitev ali postavitev v senci. Vsaka ima določene prednosti in tudi jasen namen. Izbor prave strategije je odvisen SaaS in pričakovanj trga.

Če želimo pogoste izdaje ne smemo pozabiti tudi na razvojne ekipe. Če je razvojnih ekip več in so odvisnosti velike, lahko pride do zahtevnih usklajevanj. Sledeče strategije nam to poenostavijo:

- Enoten repozitorij (ang. monorepo) [8,9] – skupaj z razvojem v glavni veji (ang. trunk-based development) [6] zmanjša usklajevanje vejitev in verzij različnih modulov. Nevarnost tega pristopa iz vidika DevOps je, da razvijalci pozabijo na združljivost za nazaj med internimi storitvami in sporočilnimi vrstami. Ne glede na strategijo nameščanja na produkcijsko okolje bo v nekem krajšem ali daljšem času tekla stara in nova verzija storitve hkrati.

- Modularna arhitektura, mikro-storitve ali mikro-prednji-deli (ang. micro-frontend) z jasno začrtanimi vsebinskimi mejami (ang. bounded context) omogočijo, da so lahko razvojne ekipe dokaj samostojne in neodvisne. Navkljub temu, da je v literaturi navedenih veliko razlogov zakaj je arhitektura mikro-storitev učinkovita, smo skozi praktične izkušnje našli le štiri prave razloge: Razdelitev odgovornosti za mikro-storitve med različne ekipe, neodvisno skaliranje mikro-storitev, različna tehnologija posameznih mikro-storitev in ponovno uporabljivost mikro-storitev v različnih sistemih. Kljub prednostim, ki jih prinašajo mikro-storitve, je mogoče nekatere koristi, kot je boljša preglednost nad deli sistema, doseči tudi z modularno zasnovo ene mikro-storitve. Pomembno je vedeti, da vsaka mikro-storitev zahteva določene računalniške vire. V povezavi z visoko razpoložljivostjo, ki zahteva vsaj tri instance vsake storitve, lahko to povzroči visoke stroške storitev v oblaku.
- K dvigu kakovosti produkta in izboljšanju sodelovanja med ekipami prispevajo pregledi oziroma revizije kode s strani sodelavcev, princip štirih oči pri testiranju kode ter določitev lastnikov kode. V proces pregledovanja zahtev za združitev vej (ang. pull requests, merge requests) lahko vključimo korake, ki zahtevajo revizijo kode s strani lastnikov kode, ki so odgovorni za določene dele kode oziroma module. S tem se zmanjšuje deljena odgovornost za posamezne dele programske kode, kar povečuje transparentnost repozitorija in motiviranost razvijalcev. Princip testiranja na štiri oči zagotavlja, da teste za določen del kode ne piše avtor sprememb, kar prispeva k večji pokritosti in širini testiranja. Ta pristop ne le izboljšuje kakovost kode, ampak tudi spodbuja sodelovanje in delitev znanja med člani ekipe.
- Testna okolja so lahko lokalna, osebna, ekipna ali globalna. Lokalno testiranje je vedno najbolj učinkovito in cevovodi so v tem primeru le v vlogi validacije. Vendar pa je pogosto sistem prevelik in prezapleten, da bi ga razvijalci poganjali v celoti samo lokalno. Tu se lahko poslužujemo globalnih razvijalnih okolij, ki so postavljeni s strani cevovodov. Razvijalci se iz lokalnih delovnih postaj v tem primeru povežejo v globalno okolje le z eno ali dvema mikro-storitvama. Če takšna strategija ni mogoča, lahko postavimo osebna ali ekipna okolja v oblaku. Tu se lahko poslužimo tudi prekomerne dodelitve virov (ang. over-provisioning). S tem bolje izkoristimo vire v oblaku za namen testiranja.
- Uvedba specializiranih ekip, ki zmanjšajo kognitivno obremenitev razvijalcev za celotno tehnologijo (ang. full stack). Veščine inženiringa platform lahko v veliki meri pripomorejo, da so razvijalci osredotočeni na poslovno vrednost in ne tehnološke specifike SaaS programske opreme in velikih modularnih arhitektur.

Avtomatizacija testiranja in posodobitev skozi cevovode močno pripomore k redkejšim napakam in odpovedim. Ključna je tudi infrastruktura. V oblaku imamo možnost z IaC enostavno postaviti redundantni oziroma celo geografsko porazdeljeni sistem. Čeprav ima oblak že v osnovi zagotovljen visok nivo zanesljivosti (ang. Service Level Agreement – SLA), ga lahko z geografsko porazdelitvijo še povečamo.

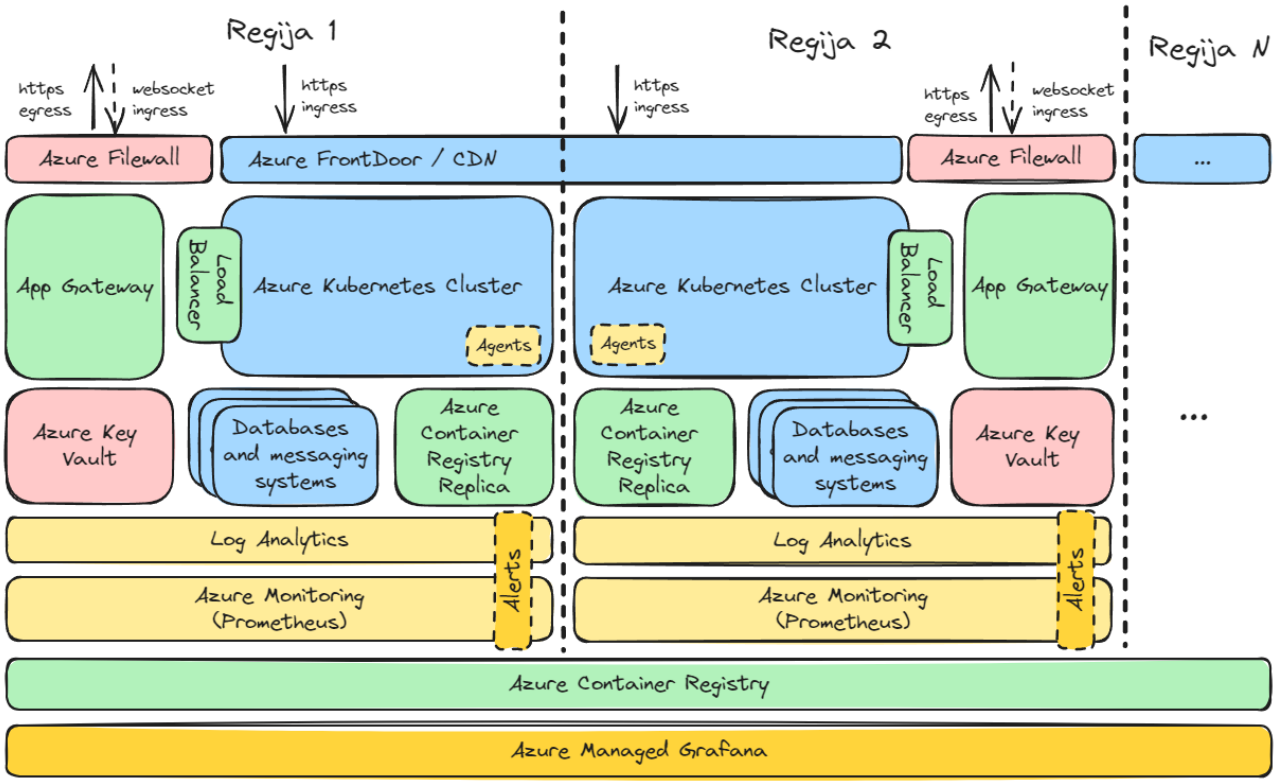
Sestavni del zagotavljanja delovanja SaaS sistema in manjšanja časa med odpovedjo in obnovitvijo je zagotovo tudi spremljava metrik sistema z alarmi in samodejno pregledovanje dnevnikov za napako. Dobra strategija nudenja SaaS je zaznava napak še predno jih stranka prijavi. Da slednje lahko izvajamo, moramo neprestano pregledovati statuse HTTP odgovorov (oziroma statuse drugih protokolov) ali pregledovati dnevnike. Iz vidika pregledovanja dnevnikov je zelo dobra strategija, da se v dnevnik ne zapisuje nič, če sistem deluje normalno. Pogosto se zgodi, da storitve v dnevnike zapisujejo tudi poslovne napake (npr. validacije podatkov), kar ni učinkovito in pregledno. Napačen vnos v masko ali napačen klic API-ja ni napaka v delovanju sistema.

Na koncu še omenimo, da hitrost CI/CD cevovodov močno vpliva hitrost odprave napak v delovanju sistema in čas posodobitev programske opreme. Če moramo za odpravo napake namestiti novo verzijo, moramo za izdajo nove verzije izvesti avtomatizirane teste oziroma celoten cevovod. Slednji je lahko dolgotrajen proces v zelo velikih sistemih. Takšne težave rešujemo s sistemom za samodejno izbiranje potrebnega nabora regresijskih testov glede na dejanske spremembe v kodi in odvisnosti od sprememb. Hkrati s ponovno namestitvijo oziroma posodobitvijo samo odvisnih modulov pohitrimo tudi korak nameščanja v oblak.

Za zagotavljanje hitrih popravkov v kritičnih situacijah, je v nekaterih primerih smiselno imeti bližnjico za postopek hitre gradnje in namestitve. Ena izmed možnih implementacij je ročna selekcija korakov cevovoda, ki se bodo izvedli. Slednje nam ponudi možnost izredno hitre namestitve popravkov, če presodimo, da je tveganje sprejemljivo oziroma je vpliv spremenjene kode omejen.

4 Javni Oblak

4.1. Arhitektura SaaS v javnem oblaku



Slika 5: Primer globalne SaaS arhitekture v hiperoblaku Azure.

Arhitekture v oblaku na sliki 5 prikazuje način postavitve SaaS za globalni trg. Celotna arhitekturo je možno postaviti skozi IaC pristop. Kritičen del IaC je izbor same tehnologije. Na voljo so nam večnamenska orodja kot so Terraform, OpenTofu, Pulumi, Crossplane, itd. Alternativno javne oblačne platforme nudijo specializirana orodja, kot so CLI ukazi in raznovrstni sistemi predlog, kot sta ARM in Bicep v Azure ekosistemu. Uporaba načina z večnamenskimi orodji je smiseln samo v primeru, če moramo zagotoviti IaC namestitve za več različnih ponudnikov oblakov hkrati. Če to ni predpogoj, so boljša izbira predloge in orodja posameznega ponudnika. Predloge zanje lahko kreiramo interaktivno skozi spletni portal in izvozimo v IaC obliki. Hkrati se je skozi uporabo izkazalo, da je iskanje napak pri namestitvah s predlogami veliko bolje podprto in jasno zapisano v dnevnikih, kot pri uporabi alternativ kot je orodje Terraform.

Nadzor nad delovanjem sistema se opravlja s pomočjo metrik, ki jih samodejno analizira sistem za alarme. Pregledne analize in spremljavo se nad metrikami se lahko izvaja preko Grafane. Za upravljanje varnostno občutljivih nastavitev in konfiguracij je primeren specializiran sistem kot je Azure Key Vault.

Povsod, ker je mogoče, je smiselno uporabiti PaaS. Le-ta je celostno upravljan s strani ponudnika oblaka. Sem spadajo PaaS storitve kot so podatkovne baze, sporočilni sistemi, podatkovni direktoriji (ang. storage accounts), trezorji, itd. Čeprav lahko vse te storitve poganjamo v Kubernetes gruči sami, se moramo zavedati, da zagotavljanje storitev zabojnikov s trajnimi plastmi (ang. persistent layers) prinese veliko večjo kompleksnost, kot poganjanje

storitve procesnih zabojnikov. V zadnjem času postajajo aktualni Kubernetes operaterji. Slednji nam prihranijo marsikatero inženirsko delo postavitve podatkovnih storitev v gruči. Težava nastane tam, kjer je potrebno poganjati storitve, ki niso prijazne do oblaka (ang. cloud native). Za primer lahko navedemo storitev sporočanja RabbitMQ. Čeprav za sporočilni sistem operater postavi gručo, se kmalu izkaže, da se vse povezane storitve ustavijo v primeru prenehanja delovanja ene izmed instanc RabbitMQ zabojnika. Te storitve so vezane na specifično instanco RabbitMQ zabojnika v gruči, ki vsebuje podmnožico sporočilnih vrst za te storitve. To povzroči dolgotrajen izpad sistema tudi v HA postavitvi. Zaradi tega postanejo vzdrževalna opravila, kot je opravilo posodobitve Kubernetesa, v načinu 24/7 zelo zahtevna.

Omeniti je potrebno tudi več-regijske oziroma globalne storitve v oblaka. Ena izmed takšnih je Microsoft Azure Front Door. Azure Front Door je storitev na robu oblaka in je glavna vhodna točka za vse SaaS na podlagi HTTPS protokola. Omogoča regijsko neodvisno vstopno točko in CDN (predpomnilnik). Storitev Front Door na podlagi latence določi, katero regijsko postavitev v oblaku se uporabi za posamezno HTTPS zahtevo. Da takšen sistem lahko deluje moramo imeti tudi globalno podatkovno bazo kot je npr. Azure CosmosDB, ki se samodejno regijsko replicira ali pa opcijsko celo zagotavlja več-regijski ACID za transakcije.

Azure Front Door igra tudi vlogo požarnega zidu, ki je specializiran za spletne aplikacije. Varnost v oblaku je izrednega pomena in velja enako kot za druge plasti v SaaS arhitekturi, da je najbolje uporabiti PaaS ponudnika oblaka.

Ne smemo pozabiti tudi na postavitve zabojnikov v oblak v brez-strežniškem (ang. serverless) načinu. V tem primeru se lahko izognemo uporabi Kubernetes gruče. Brez-strežniške namestitve zabojnikov v oblak so uporabne v primerih, ko moramo zaradi stroškov skalirati zabojnik do 0 instanc ob neuporabi. Druga prednost je, da se izognemo procesov posodabljanja Kubernetes gruč in upravljanja skaliranja na dveh nivojih – na nivoju vozlišč in na nivoju zabojnikov.

4.2. Kdaj se odločiti za hiperoblak?

SaaS (ang. Software-as-a-Service) ne more obstajati brez PaaS (ang. Platform-as-a-Service) ter PaaS ne more obstajati brez IaaS (ang. Infrastructure-as-a-Service). Veliko ponudnikov gostovanja se je oklicalo za oblak, nekateri celo enačijo instanco Kubernetes gruče z oblakom. Ločitev med gostovanjem in oblakom je v ekosistemu. Javni oblaki imajo globalno prisotnost, IaC pokritost, transparentno cenovno politiko, finančno pokrite SLA-je, razvojne platforme, 24/7 podporo, elastično skalabilnost, sledenje standardom industrije in podporo regulativi. V poenostavljenem načinu bi lahko zapisali: Če moramo svojemu ponudniku gostovanja za nastavitev omrežja poslati Excel preglednico naslovov omrežja, vrat in požarnih zidov, to ni storitev oblaka.

V primeru, če je za delovanje nekaterih storitev potrebno lokalno procesiranje, to rešujemo s t.i. robnimi namestitvami. Takšne namestitve lahko premostijo pomanjkljivosti oblaka, če so postavljene nefunkcionalne zahteve:

- Zahtevana mikro-lokalnost podatkov,
- zakonska regulativa,
- preprečevanje izpada omrežja oziroma procesiranje blizu realnega časa,
- obsežni vhodni podatki, ki jih je smiselno obdelati na lokaciji, preden se prenesejo v oblak (npr. obdelava zajetega videa)

Ponudniki globalnega SaaS se primarno morajo vprašati, zakaj ne namestiti v javni oblak. Največkrat so v času planiranja najprej vidni operativni stroški. Tu nastane težava v tem, da so stroški v javnem oblaku transparentni, medtem ko druge opcije stroške ocenijo zelo ozko – npr. samo stroške nakupa ali najema strojne opreme.

Eden izmed pomembnih stroškov je zagotovo strošek človeških virov – sistemskih inženirjev. V primeru uporabe PaaS preko IaC moramo upoštevati, da so postopki varnostne posodobitve storitev (npr. podatkovne baze), doseganje visokega nivoja SLA-jev, varnostno kopiranje na geografsko dislocirana mesta in restavriranje na zahtevo že del osnovnega stroška storitve. Strošek človeškega vira kreiranja in vzdrževanja PaaS z IaC je zanemarljiv v primerjavi s stroškom sistemskih inženirjev, ki morajo takšno storitev postaviti, vzdrževati in 24/7 nadzorovati. V velik večni primerov tudi v različnih geografskih regijah oziroma časovnih conah. Hkrati imajo globalne storitve v oblaku neprimerljivo večji nabor uporabnikov, kar jih posredno naredi neprimerljivo bolj zanesljive kot lokalno vzdrževane storitve. Naše dejanske izkušnje z obema pristopoma so to tudi potrdile.

4.3. FinOps

Stroški uporabe javnega oblaka ponujajo visoko stopnjo transparentnosti in so v veliki meri predvidljivi, kar predstavlja pomembno prednost za podjetja. Transparentnost stroškov omogoča organizacijam, da jasno vidijo, za kaj plačujejo, in kako se sredstva porabljajo. To je še posebej koristno, saj lahko podjetja natančno spremljajo svoje operativne izdatke in jih ustrezno prilagajajo.

Ena od ključnih značilnosti javnega oblaka je njegova sposobnost hitrega prilagajanja resursov. To pomeni, da lahko organizacije zelo hitro odreagirajo na spremembe v nefunkcionalnih zahtevah, kot so spremenjene potrebe po obdelovalni moči, prostoru za shranjevanje ali širini pasovne širine. Ta agilnost omogoča, da se storitve v oblaku skalirajo navzgor ali navzdol, odvisno od trenutnih potreb, kar zmanjšuje nepotrebne stroške in povečuje učinkovitost. To ne samo da optimizira porabo virov, ampak tudi zagotavlja, da podjetja maksimalno izkoristijo svoje naložbe v oblak, hkrati pa ohranjajo visoko stopnjo operativne pripravljenosti in zmogljivosti.

Optimizacijo stroškov je možno upravljati tudi s pomočjo zakupa storitev. Ponudniki omogočajo načine kot so rezervacije in varčevalni načrti. Z rezervacijami lahko dosežemo tudi to 60% prihranke medtem kot se varčevalni načrti gibljejo okoli 40%. Rezervacije so za ponudnike oblaka nekaj normalnega, ker jim ni v interesu, da sistemi skalirajo na dnevnem nivoju. Ponudniki morajo namreč imeti zagotovljeno računsko moč tudi ponoči, ne glede na to ali je v uporabi ali ne. Zato ponudniki omogočajo rezervacije, s katerimi lahko zakupite mesečno uporabo procesni kapacitet in ostalih storitev. S to ugodnostjo je porabnikom javnega oblaka smiselno pokriti osnovni odtis delovanje sistema (ang. footprint) in maksimalno dnevno uporabo, če se slednja pokrije s prihrankom rezervacije. Pravo elastično skaliranje je tako stroškovno učinkovito le pri vrhovih uporabe, ki niso osnovani na dnevnem nivoju. Primer takšnih vrhov uporabe bi bili vikendi, prazniki, vreme, letni čas, zagon prodajnih akcij, itd. – odvisno od namembnosti programske opreme. Če smo takšno porabo procesiranja sposobni oceniti mesečno, sezonsko ali letno, se lahko poslužimo »varčevalnega modela«. Z njim zakupimo določeno število ur oziroma sekund procesiranja na mesec ali leto. Pri takšnih izračunih lahko nastopijo kompleksni stroškovni modeli, ki presegajo obseg tega članka.

Iz tehničnega vidika moramo omeniti več načinov skaliranja. V oblaku lahko skaliramo na podlagi uporabe procesorske moči ali pomnilnika. Modernejši pristopi omogočajo skaliranje tudi na podlagi števila zahtev ali dolžine sporočilnih vrst. Skaliranja v Kubernetesu lahko razdelimo med:

- Vertikalne – povečujemo ali zmanjšujemo CPU ali pomnilnik posameznemu zabojniku,
- horizontalne z zabojniki – povečujemo ali zmanjšujemo število instanc zabojnikov,
- horizontalne s strežniki – povečujemo ali zmanjšujemo število instanc vozlišč oziroma strežnikov.

Za učinkovito skaliranje je pomemben kratek čas od zagona zabojnika do začetke strežbe prvih zahtev. V posebnih primerih je potrebno tudi pred-gretje vozlišč, kar omogoči hitrejši odziv na potrebo po povečanju virov pri skaliranju. Uporaba PaaS za podatkovne in podobne storitve v oblaku nam prihrani marsikateri izziv skaliranja. Pri izboru storitev se moramo zavedati, da oblaku prijazne storitve (ang. cloud native) veliko bolje horizontalno skalirajo v gručah, kot to delajo storitve, ki so bile samo prenesene v oblak. Primerjamo lahko npr. PostgreSQL in CockroachDB.

5 Zaključek

Nenehna integracija, izdaja in nameščanje (CI/CD+CD) predstavljajo ključne procese za razvoj sodobne SaaS programske opreme, ki zahteva globalno dostopnost. Uporaba DevOps metod prinese možnost neprestanega posodabljanja SaaS programske opreme brez negativnih učinkov na stabilnost, kakovost in varnost sistema. Zagotavljanje DevOps pa ne more biti učinkovito brez različnih tipov cevovodov. Vsak tip cevovoda moramo prilagoditi specifičnim zahtevam, ki jih pogojuje gradnja različnih artefaktov, kot so knjižnice, zabojniki, dokumentacija in nenazadnje namestitve in E2E testiranje. Vsako stopnjo in korak cevovoda moramo vedno tehtno opredeliti in se vprašati, ali resnično potrebujemo določen korak cevovoda glede na nefunkcionalne zahteve ciljnega SaaS sistema.

Hiperoblaki, s svojo zanesljivostjo, elastično skalabilnostjo in transparentnimi stroški, predstavljajo odlično infrastrukturo za globalne SaaS rešitve. Takšno infrastrukturo moramo vedno korektno stroškovno oceniti. Stroški niso samo stroški najema, ampak moramo pogledati celotne stroške lastništva (ang. TCO). S takšnim pristopom pri odločitvah med oblakom in nakupom hitro vidimo, da hiperoblak ni samo »draga igrača«, ampak nam dejansko omogoča stroškovno učinkovitost in rast podjetja oziroma hitre širitve uporabe SaaS ali njegove elastičnosti. Stroške oblaka nam še dlje optimizira metoda FinOps, ki se nikoli ne konča. Vendar dolgoročno prinese obvladljive, napovedljive in optimizirane stroške oblaka.

Ključne stvari članka, ki jih je potrebno vzeti v vednost in izhajajo iz neposredne prakse: Hiperoblak ni drag, DevOps ni delovno mesto, z WindowsOS raje ne v Kubernetes, avtomatizacija testiranja ni nič manj zahtevna kot razvoj programske opreme, celosten pogled na CI/CD in DevOps ni trivialen.

Literatura

[1] PISANI Erica, GANCARZ Rafal, How Cell-Based Architecture Enhances Modern Distributed Systems, https://www.infoq.com/minibooks/cell-based-architecture-2024/, The InfoQ eMag, Issue 113, 2024.

[2] VOCKE Ham, The Practical Test Pyramid, https://martinfowler.com/articles/practical-test-pyramid.html, Februar 2018.

[3] FOWLER Martin, Test Pyramid, https://martinfowler.com/bliki/TestPyramid.html, Maj 2012.

[4] COWELL Christopher, LOTZ Nicholas, TIMERLAKE Chris »Automating DevOps with GitLab CI/CD Pipelines«, Packt Publishing Ltd., 2023.

[5] COUPLAND Martin, DevOps Adoption Strategies: Principles, Processes, Tools and Trends, Pact Publishing Ltd., 2021

[6] HAMMANT Paul, et.al., Trunk Based Development, https://trunkbaseddevelopment.com/, 2020.

[7] HUMBLE Jez, FARLEY David, Continuous delivery: Reliable software releases through, build, test, and deployment automation, Pearson Education, Inc., 2011.

[8] LOSOVIZ Leonardo, Monorepo vs Multi-Repo: Pros and Cons of Code Repository Strategies, Kinsta inc., 2024.

[9] POWELL Ron, Benefits and challenges of monorepo development practices, CircleCI Blog, 2024.

Integracija zaganjalnika mobilnih aplikacij v lasten sistem za upravljanje mobilnih naprav

Alen Granda

Alcad, Slovenska Bistrica, Slovenija
alen.granda@alcad.si

V svetu industrije 4.0 je vključenost mobilnih naprav vse bolj pogost pojav. Od podpore pri načrtovanju virov podjetja, skladiščnega poslovanja, usmerjanja delavcev, vse do optimizacije procesov, opazimo, da so nepogrešljiv del modernih industrij. Na njih je naloženih vse več aplikacij po meri, katere je potrebno vzdrževati. Posledično je smiselno razmišljati o sistemu za upravljanje mobilnih naprav MDM (ang. Mobile device management). V prispevku predstavimo postopek implementacije lastnega sistema MDM znotraj privatnega podjetja. Osredotočili smo se predvsem v razvoj zaganjalnika na mobilni napravi, ki skrbi za zagon, nalaganje in posodabljanje mobilnih aplikacij. Zaganjalnik je implementiran na osnovi ogrodja večplatformnih uporabniških vmesnikov aplikacije .NET MAUI. Zraven implementacijskih podrobnosti opišemo še cevovod v platformi GitLab, ki poskrbi za samodejno posodabljanje mobilnih aplikacij. Na koncu sledi analiza koristnih napotkov ter možnosti izboljšav sistema.

Ključne besede:
MDM
.NET MAUI
mobilna aplikacija
CI/CD
upravljanje mobilnih naprav

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.6

1 Uvod

V današnjem dinamičnem okolju razvoja programske opreme, za katerega je značilen agilen pristop, se soočamo s stalnim in hitrim uvajanjem novih verzij programske opreme. Moderni pristopi k razvoju programske opreme se zgledujejo po modelu SCRUM [16], ki priporoča redno in pogosto izdajo posodobitev aplikacij. Posledično postane ročno vzdrževanje mobilnih naprav časovno potratno in podvrženo napakam, nasploh v primeru številnih poslovnih aplikacij ter fizičnih naprav. V tem kontekstu naglih sprememb postaja vse bolj pomembno sledenje različnim verzijam aplikacij, saj lahko ena aplikacija obstaja v več različicah, vsaka prilagojena specifičnim potrebam ali okoljem. Zato je vzpostavitev učinkovitega sistema sledenja, ki vključuje zgodovino naložitev programske opreme na odjemalce, odgovorne osebe in podrobnosti o naloženih verzijah, ključnega pomena za uspešno upravljanje razvoja in vzdrževanja aplikacij v agilnem razvojnem okolju.

V prispevku na kratko razložimo arhitekturno zasnovo lastnega sistema za upravljanje, nadzor in registracijo mobilnih naprav MDM (ang. Mobile device management) v organizacijah z več poslovnimi napravami in aplikacijami. Prednost v sistemu vidimo predvsem v avtomatizaciji izvedbe posodobitev, varnosti, centraliziranem nadzoru nad vsemi registriranimi mobilnimi napravami in v prikazu njihovih osnovnih informacij. Tako je omogočena večja učinkovitost in varnost pri uporabi mobilnih naprav, izboljšano zadovoljstvo končnih uporabnikov, poenostavljen razvoj in vzdrževanje mobilnih aplikacij ter lažja in hitrejša avtomatizacija proizvodnih procesov.

Sistem temelji na vrhovni mobilni aplikaciji, ki deluje kot zaganjalnik le izbrane programske opreme, naložene na izbrani napravi. Ob zagonu se kličoči aplikaciji posredujejo podatki, s katerimi lahko v dnevniških sistemih enolično določimo napravo, katera poroča zapise. Zaganjalnik je razvit tako, da omogoča zagon mobilnih aplikacij neodvisno od ogrodja, v katerem je le-ta zasnovana. Dodatno je mogoče filtrirati prikaz poslovnih aplikacij glede na organizacijo in okolje. V zaganjalnik smo integrirali tehnologijo SignalR [1], s pomočjo katere komunicira z zalednim sistemom. Potrebuje se na primer za posredovanje lokacije naprave na zahtevo.

Programsko opremo je mogoče ob predhodni prijavi tudi posodabljati in brisati. Posodobitve potekajo popolnoma avtomatizirano, saj gradnja novih različic mobilnih aplikacij poteka na podlagi pristopa neprekinjene integracije in postavitve. Cevovod poleg ustvarjenega paketa poskrbi še za posredovanje informacije o novi različici, kar izkorišča zaganjalnik na mobilni napravi in uporabniku ponudi avtomatsko posodobitev izbrane aplikacije. Posledično smo s sistemom razbremenili razvijalca od ročnega posodabljanja programske opreme ter poskrbeli za minimizacijo napak, ki lahko v tem postopku nastanejo.

Na koncu opišemo še izzive, s katerimi smo se soočili ob zagonu, posodabljanju in brisanju paketov znotraj ogrodja MAUI, koristne predloge in morebitne izboljšave implementiranega sistema.

2 Stanje obstoječih rešitev

Sistem za upravljanje mobilnih naprav je postal ključna komponenta v sklopu industrije 4.0. To je posledica vse večje vključenosti mobilnih naprav v proces digitalizacije industrije. Uporabljajo se za povezovanje, nadzor in upravljanje proizvodnih procesov ter opreme. Pomembno je tudi naraščajoče število mobilnih aplikacij, ki jih uporabljajo zaposleni, pri čemer je vsaka aplikacija pogosto na voljo v več različicah. Vse to poudarja potrebo po sistemu, ki omogoča nadzor nad temi elementi.

Sistem MDM omogoča centralizirano upravljanje tovrstnih naprav, vključno z namestitvijo in posodabljanjem aplikacij, nadzorom dostopa, varnostnimi ukrepi in spremljanjem delovanja. Zaradi agilnega pristopa k razvoju programske opreme je pomembno, da so rešitve MDM prilagodljive in omogočajo hitro prilagajanje novim zahtevam ter spremembam v poslovnem okolju. V industrijskem okolju je ključna tudi zanesljivost in razpoložljivost naprav, kar MDM zagotavlja s spremljanjem stanja naprav, diagnostiko in oddaljenimi popravili.

V nadaljevanju je predstavljenih nekaj trenutno najbolj uveljavljenih rešitev, skupaj s prednostmi in slabostmi vsake ter skupno analizo.

2.1. Microsoft Intune

Microsoft Intune [2] je celovita rešitev za upravljanje mobilnih naprav in aplikacij v organizacijah in temelji na storitvah v oblaku. Nudi širok nabor funkcij, kot so oddaljeno upravljanje naprav, avtomatizirano nameščanje in posodabljanje aplikacij ter upravljanje varnosti naprav. Prednost sistema je njegova integracija z drugimi storitvami podjetja Microsoft, kot je aktivni imenik (ang. Active directory), kar omogoča enostavno upravljanje uporabniških identitet in dostopov. Intune omogoča tudi upravljanje naprav na različnih platformah, vključno z operacijskimi sistemi Windows, iOS in Android. Vključuje obsežne varnostne funkcije, kot so upravljanje varnostnih pravil, šifriranje podatkov, oddaljeno brisanje naprav in nadzor dostopa do poslovnih virov. Rešitev je prilagodljiva in omogoča organizacijam, da individualizirajo politike upravljanja glede na svoje specifične potrebe in zahteve. Slabosti vključujejo omejitve pri podpori sledenju lokacij naprav in zahtevna uvedba ter konfiguracija sistema za manj izkušene administratorje, kar lahko povzroči dodatne stroške in časovne zamude.

2.2. SOTI MobiControl

SOTI MobiControl [3] je platforma za upravljanje mobilnih naprav in aplikacij. Ponuja oddaljeno upravljanje naprav, dodeljevanje funkcionalnosti na podlagi trenutne lokacije naprave, nameščanje in posodabljanje aplikacij ter upravljanje varnosti na mobilnih napravah. Sistem omogoča upravljanje velikega števila naprav v realnem času in ponuja obsežne možnosti za avtomatizacijo procesov, kar olajša upravljanje in zmanjšuje potrebo po ročnem posredovanju administratorjev. Podprta je tudi integracija postavitve aplikacij z rešitvami tretjih oseb. Prednosti vključujejo visoko stopnjo varnosti in poročanje zadnjih lokacij naprav z uporabo signalov globalnega umeščanja GPS ter upravljanje naprav na več platformah kot so Android, iOS, Windows in Linux. Slabosti so visoki stroški licenciranja in vzdrževanja, kar je lahko finančni izziv za manjše organizacije, ter zahtevna konfiguracija in implementacija, ki zahtevata dodatne vire in strokovno znanje.

2.3. Zebra Mobility DNA

Zebra Mobility DNA [4] je celovita platforma za upravljanje mobilnih naprav, razvita za industrijska in poslovna okolja. Ponuja napredne funkcije za upravljanje naprav, polavtomatizirano nameščanje in posodabljanje aplikacij ter izboljšano varnost. Sistem je specializiran za industrijsko uporabo, kar omogoča upravljanje naprav v zahtevnih okoljih, kot so proizvodnja, logistika in distribucija. Ponuja tudi široko paleto orodij za optimizacijo delovnih procesov, kar povečuje produktivnost in učinkovitost zaposlenih. Med slabostmi so nepopolnoma avtomatizirano nameščanje oziroma nadgradnja aplikacij, kjer je potrebno ročno ustvariti črtne kode za nove različice programske opreme, kar je časovno zahtevno ter podvrženo napakam. Prav tako so visoki stroški licenciranja in vzdrževanja dodatna slabost sistema.

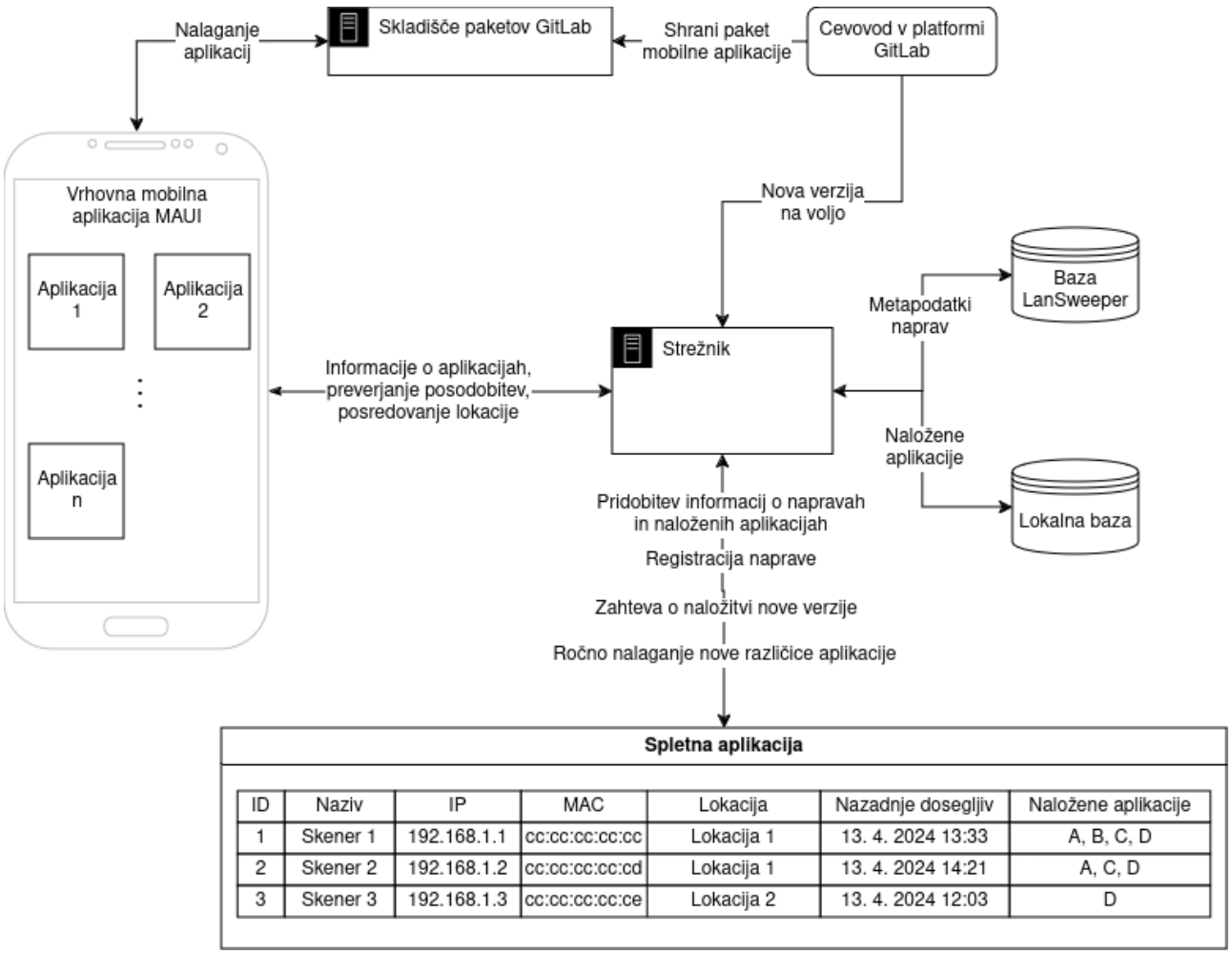
2.4. Povzetek

Vse omenjene platforme omogočajo centralizirano upravljanje mobilnih naprav in aplikacij, kar olajša administracijo in zagotavlja doslednost. Ponujajo funkcije, kot so oddaljeno brisanje aplikacij, nadzor dostopa in šifriranje, s čimer skrbijo za varnost. Poleg tega so prilagodljive in skalabilne, kar omogoča prilagajanje politik upravljanja glede na specifične potrebe organizacije. Vendar pa organizacije pogosto ne potrebujejo vseh funkcionalnosti, ki jih ponujajo te platforme. Dodatno je potrebno specifično znanje za upravljanje tovrstnih sistemov, kar lahko podaljša čas uvedbe in poveča stroške.

Opisane pomanjkljivosti so nas spodbudile k razvoju lastnega sistema MDM, ki vključuje večino skupnih lastnosti in ne zahteva dodatnega strokovnega znanja. Sistem omogoča centraliziran nadzor nad mobilnimi napravami organizacije, njihovim statusom, naloženimi aplikacijami ter popolnoma avtomatizirano posodabljanje aplikacij, ki je integrirano v proces neprekinjene integracije in postavitve CI/CD, kar razvijalcem olajša delo brez potrebe po dodatnih znanjih.

3 Vključenost zaganjalnika v sistem MDM

Na Slika vidimo globalno shemo celotnega sistema MDM. Osredotočimo se na jedro – to je osrednja mobilna aplikacija. Njena glavna naloga je prikazovanje, upravljanje in izvajanje izbranih poslovnih aplikacij na mobilnih napravah, s poudarkom na samodejnih posodobitvah in vzdrževanju programske opreme. Aplikacija za uspešno izvajanje nalog komunicira s centralnim zalednim sistemom. Pomemben vidik komunikacije je prenos unikatnega identifikatorja naprave kot metapodatka, kar omogoča natančno in zanesljivo sledenje vsaki napravi v omrežju. Na podlagi identifikatorja je omogočen prikaz seznama vseh naloženih aplikacij ter upravljanje in omejevanje drugih funkcionalnosti na posamezni napravi.

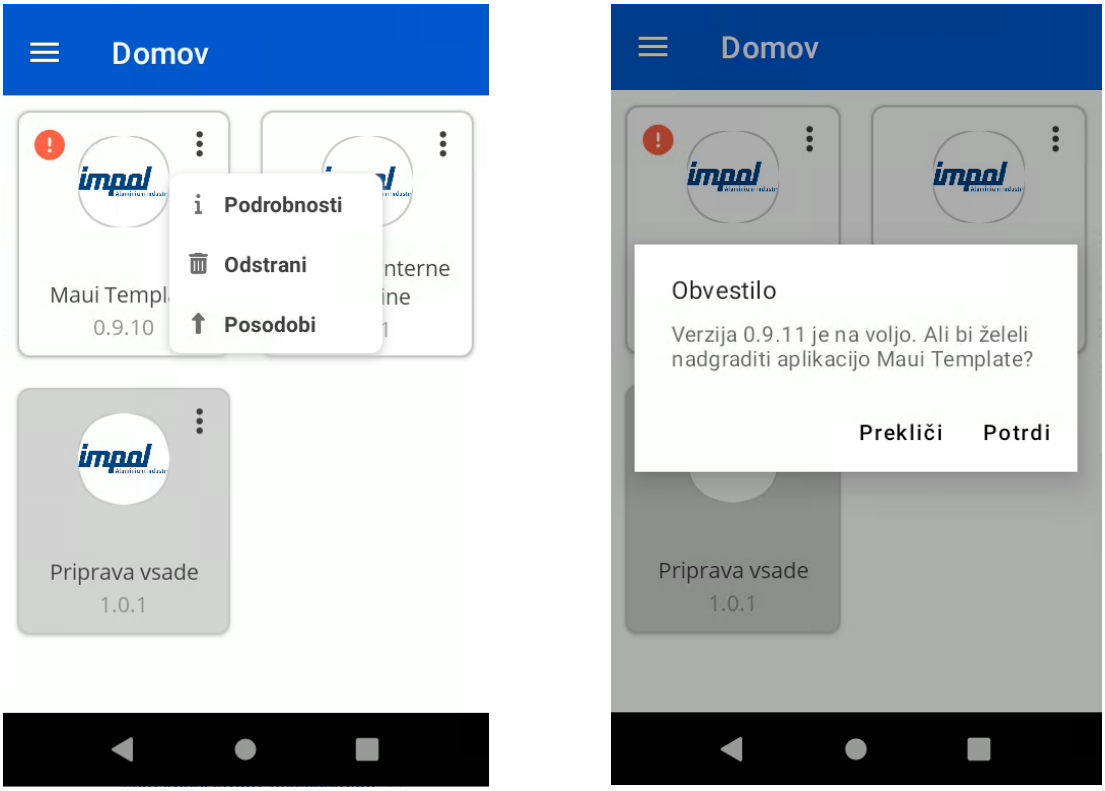


ID	Naziv	IP	MAC	Lokacija	Nazadnje dosegljiv	Naložene aplikacije
1	Skener 1	192.168.1.1	cc:cc:cc:cc:cc	Lokacija 1	13. 4. 2024 13:33	A, B, C, D
2	Skener 2	192.168.1.2	cc:cc:cc:cc:cd	Lokacija 1	13. 4. 2024 14:21	A, C, D
3	Skener 3	192.168.1.3	cc:cc:cc:cc:ce	Lokacija 2	13. 4. 2024 12:03	D

Slika 1: Globalna shema sistema MDM.

Priprava nove različice mobilne aplikacije sledi pristopu neprekinjene integracije in postavitve CI/CD. V platformi GitLab [5] je vzpostavljen cevovod, ki se sproži ob združitvi posameznih vej, kar omogoča postopno testiranje in uvajanje novih verzij aplikacij ter zmanjšuje tveganje za napake v produkcijskem okolju. Cevovod nato samodejno

začne postopek gradnje programske kode za izbrane organizacije ter izdelavo paketov mobilnih aplikacij. Ko je paket mobilne aplikacije uspešno izdelan, se shrani v register paketov na platformi GitLab v vnaprej določenem projektu. Ob tem se na zaledni sistem pošlje informacija o novo ustvarjeni različici aplikacije. Zaledni sistem nato sproži signal o potrebi po posodobitvi mobilne aplikacije na vseh napravah, ki jo imajo nameščeno. Ob naslednji osvežitvi seznama naloženih aplikacij zaganjalnik na mobilni napravi prikaže indikator, ki označuje potrebo po posodobitvi aplikacije (Slika). Na ta način omogočimo uporabnikom enostavno in pregledno posodabljanje aplikacij na njihovih napravah ter razbremenimo razvijalca od potrebe po ročnem nadgrajevanju mobilnih aplikacij.



Slika 2: Prikaz aplikacij na zaganjalniku ter ponudba posodobitve.

V nadaljevanju podrobneje opišemo implementacijske podrobnosti razvoja in integracije vrhovne mobilne aplikacije v ogrodju MAUI, vključno z opisom samega ogrodja, načini komunikacije z zalednim sistemom, upravljanjem identifikatorja naprave, posredovanjem lokacijske informacije ter razlago zagona in samodejnega posodabljanja aplikacij.

4 Implementacija zaganjalnika

4.1. Ogrodje MAUI

Ogrodje večplatformnih uporabniških vmesnikov aplikacije .NET MAUI (ang. Multi-platform app user interface) je zasnovano za ustvarjanje izvornih mobilnih in namiznih aplikacij, kar omogoča večplastni pristop k razvoju z uporabo programskega jezika C# in razširljivega označevalnega jezika aplikacij XAML [6]. Uporabno je za reševanje različnih izzivov, kot so spremenljive zahteve uporabnikov, nove poslovne priložnosti in stalne povratne informacije med razvojem. Cilj je ustvariti prilagodljive aplikacije, ki jih je mogoče enostavno prilagajati in razširjati skozi čas.

Ogrodje je odprtokodno in temelji na priljubljenem ogrodju za razvoj izvornih mobilnih aplikacij Xamarin [8]. Microsoft, v skladu s svojo strategijo združevanja vseh orodij, izvajalnih okolij in programskih knjižnic, je prepoznal potrebo po nadgradnji ogrodja Xamarin. Zato so maja 2022 javnosti predstavili ogrodje .NET MAUI

kot njegovega naslednika [7]. Nova generacija razvoja večplatformnih poslovnih rešitev je prinesla številne zanimive spremembe. Ena najpomembnejših novosti za avtomatizacijo posodobitev mobilnih aplikacij je podpora uporabi vmesnika ukazne vrstice .NET CLI (ang. .NET command line interface). Orodje omogoča razvoj, gradnjo, zagon in izdajo aplikacij .NET na več platformah, kar omogoča vključitev v proces neprekinjene integracije in postavitve CI/CD mobilnih aplikacij v zabojnikih, ki delujejo pod operacijskim sistemom Linux. Podrobnosti bomo opisali v nadaljevanju.

Poleg možnosti avtomatizacije posodobitev nas je ogrodje MAUI pritegnilo tudi zaradi vedno večje skupnosti, stabilnosti, ki jo prinaša tehnologija .NET 8, in celovite integracije v ekosistem tehnologije .NET. Dodatno so privlačne tudi kakovostne knjižnice za razvoj uporabniških vmesnikov, kot sta komplet orodij skupnosti [17] in knjižnica modulov uporabniških orodij za ogrodje MAUI podjetja DevExpress [18].

4.2. Lokacija mobilne naprave in posredovanje z uporabo tehnologije SignalR

Lokacije mobilne naprave ne osvežujemo konstantno, ampak na zahtevo, saj je takšna operacija energijsko potratna. Dvosmerno komunikacijo z zalednim sistemom smo vzpostavili s pomočjo tehnologije SignalR. Tako lahko zaledni sistem pošlje ukaz mobilni aplikaciji za osvežitev njene lokacije. Mobilna aplikacija ob prejemu ukaza pridobi trenutno lokacijo z uporabo signalov GPS ter jo posreduje nazaj zalednemu sistemu.

Za pridobitev lokacije je bilo znotraj mobilne aplikacije potrebno pridobiti pravice za dostop do lokacijskih storitev naprave. To smo dosegli z dodanimi zahtevami za dovoljenje v datoteki nastavitev AndroidManifest.xml. Poleg tega mora uporabnik eksplicitno privoliti v branje lokacije naprave skozi uporabo zaganjalnika. Po pridobitvi vseh dovoljenj lokacijo preberemo z uporabo višjenivojskih ukazov ogrodja MAUI, kot je prikazano v Slika .

Odjemalec tehnologije SignalR je implementiran s pomočjo uradne knjižnice podjetja Microsoft [9]. Ker je funkcionalnost komunikacije čez tehnologijo SignalR uporabna za več mobilnih aplikacij, smo implementacijo zapakirali v lastno knjižnico in jo objavili v internem registru knjižnic. Znotraj mobilne aplikacije je tako le storitev za registracijo metod, katere se sprožijo ob prejemu določene zahteve. Primer metode, ki registrira poslušalca na zahtevo za posredovanje sveže lokacije in implementira prej opisano logiko, je prikazana v Slika .

```
public async Task<Location?> GetGpsLocationAsync()
{
    var status = await Permissions.CheckStatusAsync<Permissions.LocationWhenInUse>();

    if (status != PermissionStatus.Granted)
    {
        await MainThread.InvokeOnMainThreadAsync(funcTask: async () =>
            status = await Permissions.RequestAsync<Permissions.LocationWhenInUse>());

        if (status != PermissionStatus.Granted)
        {
            return null;
        }
    }

    var location = await Geolocation.Default.GetLastKnownLocationAsync();

    if (location != null)
    {
        return location;
    }

    var request = new GeolocationRequest(GeolocationAccuracy.Default, timeout: TimeSpan.FromSeconds(10));
    location = await Geolocation.Default.GetLocationAsync(request);

    return location;
}
```

Slika 3: Pridobitev lokacije z uporabo signalov GPS.

4.3. Zagon in posodobitev mobilnih aplikacij

Aplikacije, ki so na voljo za specifično napravo, pridobimo s klicem na zaledni sistem. V odgovoru dobimo nazive paketov, njihove naložene verzije ter zadnjo razpoložljivo različico. Če aplikacija na napravi še ne obstaja, ponudimo namestitev. V primeru starejše različice predlagamo posodobitev. V kolikor uporabnik klikne na ikono aplikacije, jo zaženemo.

Podobno kot pri pridobitvi lokacije mobilne naprave, tudi za zagon, posodabljanje ali brisanje aplikacij potrebujemo ustrezne pravice, določene v datoteki AndroidManifest.xml. Programsko kodo za te akcije smo morali zapisati za vsako platformo posebej, saj ogrodje ne ponuja abstrakcij za tovrstne funkcije. Posodabljanje, nalaganje in brisanje aplikacij smo reševali z uporabo namenov (ang. Intent) operacijskega sistema Android. Postopek zagona aplikacije je prikazan v Slika in sledi naslednjim korakom: najprej preverimo, ali aplikacija z izbranim nazivom paketa obstaja na napravi. Če obstaja, pridobimo njen namen zagona in kot metapodatek pošljemo identifikator naprave. Aplikacija ga uporablja pri pisanju dnevnikov, kar omogoča enolično identifikacijo naprav pri pregledu dnevnikov.

```
public void RegisterAppHubListeners()
{
    _hubService.RegisterListener( methodName: HubAction.RequestForGps, [typeof(int)], handler: async objects :object?[] =>
    {
        try
        {
            if (objects.Length <= 0 || objects[0] is not int receivedAssetId)
            {
                return;
            }

            var assetId :int = await _settingsService.GetAssetId();
            if (receivedAssetId == assetId)
            {
                var location = await _gpsService.GetGpsLocation();

                if (location != null)
                {
                    await _mdmHttpClientService.PatchAssetGpsLocation(assetId, (decimal)location.Latitude, (decimal)location.Longitude);
                }
            }
        }
        catch (Exception e)
        {
            LogService.LogError(e);
        }
    });
}
```

Slika 4: Logika za registracijo poslušalca na zahtevo osvežitve lokacije mobilne naprave.

```
public async Task RunApp(string packageName)
{
    var assetId :int = await settingsService.GetAssetId();
    try
    {
        var pm :PackageManager? = Application.Context.PackageManager;

        if (IsAppInstalled(packageName))
        {
            var intent = pm?.GetLaunchIntentForPackage(packageName);
            if (intent != null)
            {
                intent.PutExtra( name: Common.Enums.Constants.AssetIdIdentifier, assetId);
                intent.SetFlags(ActivityFlags.NewTask);
                Application.Context.StartActivity(intent);
            }
        }
        else
        {
            throw new ApplicationNotFoundException();
        }
    }
    catch (ActivityNotFoundException e)
    {
        logService.LogError(e);
        throw;
    }
}
```

Slika 5: Zagon izbrane mobilne aplikacije.

4.4. Avtomatizacija posodobitev

Nazadnje opišimo še postopek avtomatizacije posodobitev aplikacij na mobilnih napravah. V kolikor sledimo diagramu na Slika , opazimo, da se lahko nova verzija aplikacije naloži na 2 načina. Na eni strani se nova različica naloži skozi cevovod v platformi GitLab. Opozoriti moramo, da je cevovod pripravljen le za aplikacije, razvite v ogrodju MAUI. Zato je podprt tudi ročni način nalaganja verzij. Tako smo omogočili nalaganje poljubnih aplikacij v sistem MDM.

Ob registraciji nove različice aplikacije se paket shrani v register paketov v platformi GitLab. Na zaledni sistem zapišemo le metapodatke o različici, kot so verzija, naziv paketa, spletni naslov za prenos paketa, organizacija, za katerega je zgrajen, ipd. Ko zaganjalnik na mobilni napravi ponovno zahteva pridobitev aplikacij, ki so zanj na voljo, se zraven aplikacij vrne še obvestilo, da je na voljo nadgradnja. Obvestilo prikažemo uporabniku in ponudimo posodobitev.

V kombinaciji s cevovodom v platformi GitLab, ki sledi pristopu neprekinjene integracije in postavitve, smo poskrbeli za popolno razbremenitev razvijalca od skrbi za ročno posodabljanje aplikacij. Ker je to ena izmed pomembnejših funkcionalnosti sistema MDM, jo podrobneje predstavimo. Cevovod se zažene ob združitvi posameznih vej. Najprej poskrbimo za procesiranje metapodatkov aplikacije (Slika). Metapodatke preberemo iz datoteke projekta s končnico .csproj. V njem se nahajata verzija ter naziv aplikacije. Nato poskrbimo za menjavo vnaprej definiranih spremenljivk v aplikacijskih nastavitvah z dejanskimi vrednostmi, ki so zapisane v skrivnostih (ang. Secret) platforme GitLab. Sledi priprava datotek za definiranje izvorov registrov knjižnic ter nastavitev osnovnega spletnega naslova za dostop do zalednega sistema. Naslov je namreč različen glede na okolje, za katerega se aplikacija gradi.

Po postopku predprocesiranja sledi še grajenje (Slika). Grajenje je omogočeno za več organizacij hkrati. Pred tem moramo seveda preveriti, ali aplikacija s tovrstno verzijo že obstaja na zalednem sistemu. V kolikor obstaja, se cevovod zaključi, saj je razvijalec najverjetneje pozabil povišati različico. V nasprotnem primeru se aplikacija zgradi z uporabo vmesnika ukazne vrstice .NET CLI in podpiše z vnaprej pripravljeno shrambo ključev (ang. keystore). Nazadnje paket shranimo v register paketov ter zabeležimo novo različico aplikacije na zaledni sistem.

```
script:
  # app metadata
  - |
    APPLICATION_ID=$(xmlstarlet sel -t -v "//ApplicationId" $PROJECT_PATH)
    APPLICATION_DISPLAY_VERSION=$(xmlstarlet sel -t -v "//ApplicationDisplayVersion" $PROJECT_PATH)
    APPLICATION_VERSION=$(xmlstarlet sel -t -v "//ApplicationVersion" $PROJECT_PATH)
    APPLICATION_NAME=$(xmlstarlet sel -t -v "//ApplicationTitle" $PROJECT_PATH)
    APP_SETTINGS_FILE="./Impol.${APPLICATION_NAME}/Resources/Raw/appSettings.json"

  # replace everything except for tenant id in appSettings.json
  - |
    sed -i -e 's#\$MDM_API_URL#'"${MDM_API_URL}"'#' ${APP_SETTINGS_FILE}
    # + others that are skipped for brevity

    cp ${APP_SETTINGS_FILE} ${APP_SETTINGS_FILE_COPY}

  # nuget config file
  - |
    echo '<?xml version="1.0" encoding="utf-8"?>' > ${NUGET_CONFIG_FILE}
    echo '<configuration>' >> ${NUGET_CONFIG_FILE}
    echo '    <packageSources>' >> ${NUGET_CONFIG_FILE}
    echo '        <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />' >> ${NUGET_CONFIG_FILE}
    echo '        <add key="Devexpress" value="'"${MOBILE_DEVEXPRESS_NUGET_REPO}"'" />' >> ${NUGET_CONFIG_FILE}
    echo '        <add key="gitlab-alcad" value="'"${GITLAB_NUGET_REPO}"'" />' >> ${NUGET_CONFIG_FILE}
    echo '    </packageSources>' >> ${NUGET_CONFIG_FILE}
    echo '</configuration>' >> ${NUGET_CONFIG_FILE}

  # copy nuget configs
  - |
    projectNugetConfigFile="./Impol.${APPLICATION_NAME}/nuget.config"
    cp ${NUGET_CONFIG_FILE} ${projectNugetConfigFile}
    # + others that are skipped for brevity

  # setup the base API URL based on commit branch
  - MDM_BASE_URL="https://${CI_COMMIT_BRANCH}.${DOCKER_API_PROD}/${MDM_API_URL}/api/v1"
  - apiBaseUrl="https://${CI_COMMIT_BRANCH}.${DOCKER_API_PROD}/"
```

Slika 6: Predprocesiranje metapodatkov aplikacije v postopku cevovoda.

```
.publish_android:
  tags:
    - build
  script:
    # publish it for each tenant
    - |
      ##################################
      # replace API url of the appsetting.json
      sed -i -e 's#\$DOCKER_API_PROD#'"${apiBaseUrl}"'#' ${APP_SETTINGS_FILE_COPY}

      ##################################
      # receive app id from MDM API

      urlRequest="${MDM_BASE_URL}/applications?\$filter=code%20eq%20'${APPLICATION_NAME}'"
      response=$(curl -X GET --location "$urlRequest" --header 'Tenant-Id: 11' --header 'Tenant-Type: O' --header 'Content-Type: application/json')

      num_elements=$(echo "$response" | jq length)
      if [ $num_elements -ne 1 ]; then
        echo "Error: App ID could not be retrieved. Ensure APPLICATION_NAME is properly set. APPLICATION_NAME = ${APPLICATION_NAME}"
        echo "Response:"
        echo "$response"
        exit 1
      fi

      MDM_APP_ID=$(echo "$response" | jq -r '.[0].id')

      ##################################
      # publish it for each tenant

      for tenantId in ${TENANT_IDS}; do
        ##################################
        appId="${APPLICATION_ID}_${CI_COMMIT_BRANCH}_${tenantId}"

        echo "Publishing the project ${APPLICATION_NAME} with version ${APPLICATION_DISPLAY_VERSION} for tenant ${tenantId}"
        echo "Environment: $CI_COMMIT_BRANCH"
        echo "App ID: $appId"

        cp ${APP_SETTINGS_FILE_COPY} ${APP_SETTINGS_FILE}
        sed -i -e 's#\$TENANT_ID#'"${tenantId}"'#' ${APP_SETTINGS_FILE}

        # use appSettings.json and not appSettings.Development.json
        sed -i -e 's#appSettings.Development.json#appSettings.json#g' ./Impol.${APPLICATION_NAME}/Impol.${APPLICATION_NAME}.csproj
        sed -i -e 's#<ApplicationId>.*</ApplicationId>#<ApplicationId>'"${appId}"'</ApplicationId>#' ./Impol.${APPLICATION_NAME}/Impol.${APPLICATION_NAME}.csproj
        sed -i -e 's#RegisterConfigurations<AppSettings>(true)#RegisterConfigurations<AppSettings>(false)#g' ./Impol.${APPLICATION_NAME}/MauiProgram.cs

        ##################################
        # check if this version can even be published...

        # e.g.: applications/1/versions?$filter=versionNumber eq 1 and version/name eq '0.9.0'
        urlRequest="${MDM_BASE_URL}/applications/${MDM_APP_ID}/versions?%24filter=versionNumber%20eq%20${APPLICATION_VERSION}%20and%20version%2Fname%20eq%20%27${APPLICATIO
        response=$(curl -X GET --location "$urlRequest" --header 'Tenant-Id: '"${tenantId}"'' --header 'Tenant-Type: O' --header 'Content-Type: application/json')

        # Check if the response array is not empty using jq
        if [[ $(echo "$response" | jq '. | length') -gt 0 ]]; then
          echo "App with such ID and version name already exists! Version name = ${APPLICATION_DISPLAY_VERSION}, version number = ${APPLICATION_VERSION}"
          exit 1
        fi

        ##################################
        # Publish
        dotnet publish \
          -f net8.0-android \
          -c Release \
          -o ${OUTPUT_DIR} \
          -p:AndroidSdkDirectory="${ANDROID_SDK_ROOT}" \
          -p:AndroidPackageFormats=apk \
          -p:AndroidKeyStore=true \
          -p:AndroidSigningKeyStore="$(pwd)/${ANDROID_SIGNING_KEYSTORE_FILE}" \
          -p:AndroidSigningKeyPass=${ANDROID_SIGNING_KEY_PASS} \
          -p:AndroidSigningStorePass=${ANDROID_SIGNING_KEY_PASS} \
          -p:ApplicationDisplayVersion=${APPLICATION_DISPLAY_VERSION} \
          -p:ApplicationVersion=${APPLICATION_VERSION} \
          -p:ApplicationId=${appId} \
          -p:ApplicationTitle=${APPLICATION_NAME}

        ##################################
        # deploy
        for apk in $(find $OUTPUT_DIR/ -type f -name "*-Signed.apk" -printf "%f\n"); do

          echo "Deploying the file = $apk"

          # deploy to apk project
          DEPLOY_URL=${CI_API_V4_URL}/projects/910/packages/generic/${appId}/${APPLICATION_DISPLAY_VERSION}/${apk}

          # push the package to package registry
          response=$(curl --header "JOB-TOKEN: $CI_JOB_TOKEN" --upload-file ${OUTPUT_DIR}/${apk} "${DEPLOY_URL}")
          status_code=$(echo "$response" | sed 's/[^0-9]*//g')
          if [ "${status_code}" != "201" ]; then
            echo "Error: Response status is not 201. It is ${status_code}"
            echo "Response: $response"
            exit 1
          fi

          response=$(curl -w "%{http_code}" --location "${MDM_BASE_URL}/applications/${MDM_APP_ID}/versions" \
            -X POST \
            --header 'Tenant-Id: '"${tenantId}"'' \
            --header 'Tenant-Type: O' \
            --header 'Content-Type: multipart/form-data' \
            --header 'Authorization: Bearer '"${MDM_AUTH_TOKEN}"'' \
            --form 'PackageName='"${appId}"'' \
            --form 'VersionName='"${APPLICATION_DISPLAY_VERSION}"'' \
            --form 'VersionNumber='"${APPLICATION_VERSION}"'')

          status_code=$(echo "$response" | tail -c 4 | sed 's/[^0-9]*//g')
          if [ "${status_code}" != "200" ]; then
            echo "Error: Response status is not 200. It is ${status_code}"
            echo "Response: $response"
            exit 1
          fi

          echo "File ${apk} deployed to ${DEPLOY_URL}!"

        done
      done
```

Slika 7: Grajenje in objava nove različice aplikacije znotraj cevovoda.

5 Koristni napotki

5.1. Zagon in namestitev aplikacij

Znotraj zaganjalnika uporabljamo občutljive akcije, kot so zagon, namestitev in brisanje aplikacij. To je lahko velik poseg v varnost same naprave, zato je razumljivo, da je potrebno za tovrstne akcije potrebno eksplicitno dovoljenje uporabnika. V ta namen smo morali v nastavitveni datoteki AndroidManifest.xml zahtevati pravice za iskanje paketov, namestitev in brisanje (Slika). Dodatno bi omenili, da moramo poleg zahteve za iskanje paketov podati tudi značko za iskanje "<queries>". V nasprotnem primeru se paket ne najde.

```
<!-- For automatically installing apps -->
<uses-permission android:name="android.permission.INSTALL_PACKAGES" />
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" />
<uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES" />

<!-- used to launch apps -->
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES" />
<queries>
    <intent>
        <action android:name="android.intent.action.MAIN" />
    </intent>
</queries>
```

Slika 8: Zahteve za iskanje, nameščanje in brisanje paketov.

Dodatno moramo paziti, da nastavimo način zagona glavne aktivnosti (ang. Main activity) na enojni vrh (ang. Single top). V kolikor pustimo privzeto vrednost, se zaganjalnik ob zagonu druge aplikacije zapre, najverjetneje zaradi hrošča (ang. Bug) tehnologije .NET 8 [10].

```
[Activity(Theme = "@style/Maui.SplashTheme",
    LaunchMode = LaunchMode.SingleTop, // need to set to SingleTask / SingleTop! See: https://github.com/dotnet/maui/issues/18692
    Exported = true, MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation | ConfigChanges
[IntentFilter([Intent.ActionMain], Categories = [Intent.CategoryLauncher, Intent.CategoryHome, Intent.CategoryDefault])]
0 references
public class MainActivity : MauiAppCompatActivity
{
```

Slika 9: Nastavitev zagona glavne aplikacije na enojni vrh.

Opozorili bi tudi, da ogrodje MAUI samo po sebi že ponuja zagon drugih aplikacij z lastnimi abstrakcijami [11]. Edina slabost zaganjalnika ogrodja je, da kličoči aplikaciji ne moremo posredovati parametrov, kot je na primer identifikator naprave. Posledično smo se odločili za lastno implementacijo z uporabo platformno specifične programske kode.

5.2. Cevovod v platformi GitLab

Nikjer nismo omenili kakšna predloga vsebnika se uporablja za grajenje aplikacij ogrodja MAUI znotraj cevovoda v platformi GitLab. Predloga izhaja iz [12] in je prilagojena za grajenje v tehnologiji .NET 8. Opozoriti moramo, da za grajenje aplikacij za operacijski sistem Android potrebujemo komplet razvojnih orodij programskega jezika Java verzije 11. To velja vse do različice 14 operacijskega sistema Android [14]. Ker uradna predloga podjetja Microsoft za grajenje paketov v tehnologiji .NET 8 [13] temelji na operacijskem sistemu Debian 12, ki uradno ne podpira več Jave 11, imamo 2 možnosti. Ena je, da ročno poskrbimo za instalacijo v tovrstnem operacijskem sistemu. Po drugi strani lahko uporabimo predlogo, ki temelji na operacijskem sistemu Ubuntu [15]. Tovrstna predloga še namreč ponuja uraden način instalacije s pomočjo upravljalca paketov.

6 Zaključek

V zaključku lahko povzamemo, da uvedba lastnega sistema za upravljanje in nadzor mobilnih naprav prinaša številne prednosti za organizacije z več poslovnimi aplikacijami. Z avtomatizacijo posodobitev, izboljšano varnostjo in centraliziranim nadzorom se zmanjša časovna in administrativna obremenitev razvijalcev ter poveča učinkovitost poslovnih procesov. Sistem omogoča tudi boljše sledenje različicam aplikacij in prilagoditve glede na specifične potrebe okolja. Kljub izzivom, ki jih prinaša integracija z ogrodjem MAUI, so koristi jasne in pripomorejo k večjemu zadovoljstvu končnih uporabnikov ter poenostavitvi razvoja in vzdrževanja aplikacij. V prihodnje bi nadaljnje izboljšave sistema lahko še dodatno optimizirale procese in funkcionalnosti.

Kljub številnim prednostim sistema za upravljanje mobilnih naprav ostaja še veliko prostora za izboljšave. Ena izmed pomanjkljivosti je pomanjkanje povratne informacije pri komunikaciji čez tehnologijo SignalR, kjer lahko na strani odjemalca dodamo logiko za potrditveni odgovor zalednemu sistemu. Prav tako nismo vpeljali enotne prijave (ang. Single sign-on) ob zagonu izbranih aplikacij znotraj zaganjalnika, kar bi olajšalo uporabniško izkušnjo. Predlagamo uporabo šifriranih metapodatkov za izboljšano varnost pri prijavnih procesih. Šifrirani metapodatki o uporabniku se lahko pošiljajo hkrati s podatkom o identifikatorju naprave. Z implementacijo izboljšav lahko sistem postane še bolj učinkovit in varen, kar pripomore k optimizaciji poslovnih procesov in k zadovoljstvu uporabnikov.

Literatura

[1] https://dotnet.microsoft.com/en-us/apps/aspnet/signalr, Real-time ASP.NET with SignalR, obiskano 16. 7. 2024.

[2] https://learn.microsoft.com/en-us/mem/intune/fundamentals/what-is-intune, Microsoft Intune overview, 16. 7. 2024.

[3] https://soti.net/products/soti-mobicontrol/, SOTI MobiControl, obiskano 16. 7. 2024.

[4] https://www.zebra.com/us/en/software/mobile-computer-software/mobility-dna.html, Mobility DNA - Enterprise Mobility Software, obiskano 16. 7. 2024.

[5] https://about.gitlab.com/, GitLab, obiskano 17. 7. 2024.

[6] https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui, What is .NET MAUI?, obiskano 17. 7. 2024.

[7] https://devblogs.microsoft.com/dotnet/announcing-dotnet-maui-in-dotnet-8, Announcing .NET MAUI in .NET 8, obiskano 17. 7. 2024.

[8] https://learn.microsoft.com/en-us/previous-versions/xamarin/get-started/what-is-xamarin, What is Xamarin?, obiskano 17. 7. 2024.

[9] https://www.nuget.org/packages/Microsoft.AspNetCore.SignalR.Client, Microsoft.AspNetCore.SignalR.Client, obiskano 17. 7. 2024.

[10] https://github.com/dotnet/maui/issues/18692, MAUI Android build crashes when app is reopened from background. It throws the exception: 'Window was already created.', obiskano 18. 7. 2024.

[11] https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/appmodel/launcher?view=net-maui-8.0&tabs=android, Launcher, obiskano 18. 7. 2024.

[12] 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.

[13] https://hub.docker.com/r/microsoft/dotnet-sdk, microsoft/dotnet-sdk – Docker Image, obiskano 18. 7. 2024.

[14] https://developer.android.com/build/jdks, Java versions in Android builds, obiskano 18. 7. 2024.

[15] https://github.com/dotnet/dotnet-docker/blob/main/src/sdk/8.0/jammy/amd64/Dockerfile, dotnet-docker/src /sdk/8.0/jammy/amd64/Dockerfile, obiskano 18. 7. 2024.

[16] SCHWABER Ken "SCRUM Development Process", Business Object Design and Implementation, Springer London, London, 1997, str. 117-134.

[17] https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/, .NET Multi-platform App UI (.NET MAUI) Community Toolkit documentation, obiskano 18. 7. 2024.

[18] https://www.devexpress.com/maui/, .NET MAUI Controls, obiskano 18. 7. 2024.

Razvoj spletnih aplikacij za razvijalce zalednih sistemov

Mitja Krajnc, Jani Kajzovar, Andraž Leitgeb

Databox, Ptuj, Slovenija
mitja.krajnc@databox.com, jani.kajzovar@databox.com, andraz.leitgeb@databox.com

Pojav frontend razvojnih ogrodij je še povečal vrzel med frontend in backend razvojem. V podjetju smo želeli razširiti funkcionalnosti interne spletne aplikacije in s tem zagotoviti večjo učinkovitost ekip, vendar nismo imeli prostih frontend razvijalcev. Zato smo se odločili raziskati alternativne pristope, ki bi omogočil razvoj frontend in backend aplikacij v enakem jeziku in okolju. Primerno rešitev ponuja razvoj aplikacij z .NET 8 Blazor in MudBlazor, kar bomo predstavili v članku. Z .NET 8 Blazor lahko razvijemo robustne in interaktivne spletne aplikacije v C#, brez potrebe po učenju dodatnega jezika. To poenostavi in pohitri razvoj, saj se razvijalci lahko osredotočijo na logiko in funkcionalnost aplikacije. Še večjo poenostavitev predstavlja uporaba knjižnice MudBlazor, ki temelji na Material UI in je zasnovana za Blazor. Ponuja izdelane in prilagodljive komponente, kar omogoča hitro izdelavo uporabniških vmesnikov. Tako smo izkoristili prednosti .NET razvoja in omogočili backend razvijalcem enostaven pristop k razvoju spletnih rešitev. V članku bomo predstavili Blazor in MudBlazor, razvojno strukturo, izzive in rezultate. Rešitev predstavlja spletna aplikacija, ki po strukturi spominja na backend projekte in omogoča hiter razvoj internih aplikacij, in s tem opolnomoči backend razvijalce. Dodali smo tudi peskovnik za preizkušanje konceptov uporabniške izkušnje brez dodatnega dela za frontend razvijalce.

Ključne besede:
Blazor
.NET
MudBlazor
spletne aplikacije
zaledni sistemi

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.7

1 Uvod

V svetu razvoja programske opreme je hitra prilagodljivost ključna. Z nenehnim razvojem novih tehnologij je pomembno, da ekipe čim prej osvojijo in izkoristijo nove priložnosti. Ena izmed takšnih tehnologij je .NET 8 Blazor [1], ki prinaša sveže inovacije in izboljšave za razvoj spletnih aplikacij. Z njim lahko backend razvijalci, ki so že seznanjeni z .NET, hitro in učinkovito ustvarjajo privlačne uporabniške vmesnike brez potrebe po dodatnem znanju frontend knjižnic, kot so React, Vue ali Angular.

Zadnja različica Blazor ponuja vrsto novih funkcionalnosti, ki olajšajo delo razvijalcem. Med najpomembnejšimi so izboljšave zmogljivosti, razširitev podpore za WebAssembly, izboljšana orodja za razvoj in razhroščevanje ter številne nove komponente, ki poenostavijo gradnjo kompleksnih aplikacij.

Za potrebe našega projekta smo se odločili uporabiti zbirko komponent MudBlazor [2], ki je komponentna knjižnica za Blazor. Omogoča enostavno in hitro ustvarjanje lepih in odzivnih uporabniških vmesnikov. Z uporabo MudBlazor lahko naši backend razvijalci hitro integrirajo napredne funkcionalnosti in izboljšajo uporabniško izkušnjo, brez potrebe po dodatnem učenju in raziskovanju frontend tehnologij.

Zastava celotnega projekta je temeljila na čim večji domačnosti backend razvijalcev, tako da smo se v strukturi projekta zgledovali predvsem po obstoječih projektih, ki se uporabljajo za različne backend storitve in ne toliko po strukturi projektov, ki jih prinašajo tipični frontend projekti. Osnova projekta je bila tako imenovana čista arhitektura (clean architecture) [3], pri čemer smo v eno rešitev združili vse od storitvenega nivoja do uporabniškega vmesnika.

S tem smo povečali produktivnost ekipe in zmanjšali odvisnost od zunanjih virov. To nam omogoča hitrejši razvoj in lažjo prilagodljivost. S strateško odločitvijo uporabe Blazor in MudBlazor smo izkoristili moč .NET ekosistema in opolnomočili backend razvijalce, kar nam omogoča bolj učinkovito delo in hitrejše doseganje ciljev. S tem pristopom smo backend razvijalcem omogočili, da ustvarjajo visokokakovostne interne aplikacije, ki so tako funkcionalne kot estetsko dovršene, kar je ključno za uspešno delovanje naše ekipe in podjetja.

2 Blazor

Blazor je tehnologija za razvoj spletnih aplikacij, ki omogoča ustvarjanje interaktivnih uporabniških vmesnikov z uporabo C# in .NET, namesto tradicionalnih JavaScript ogrodij. Z izdajo .NET 8 Blazor je Microsoft še dodatno nadgradil in izboljšal to orodje, kar prinaša številne prednosti in novosti za razvijalce. O samem Blazorju je bilo na preteklih OTS konferencah že povedanega marsikaj, zato bomo predstavili samo nekatere izmed novosti, ki so prišle v zadnji verziji.

2.1. Novosti v .NET 8 Blazor

Zadnja verzija .NET, ki je v uradnem izidu, je stara sicer nekaj več kot pol leta, a prinaša s seboj kar nekaj izboljšav, ki jih bomo predstavili v spodnjih odstavkih [4];

- **Boljša zmogljivost in hitrost:** Ena izmed glavnih prednosti zadnje inačice Blazor je izboljšana zmogljivost. Optimizacije pri nalaganju in izvajanju aplikacij omogočajo hitrejše odzivne čase in manjšo porabo virov. To pomeni, da lahko razvijalci ustvarijo odzivnejše aplikacije. Le-te se hitreje nalagajo in delujejo gladko tudi na napravah z nižjimi specifikacijami.
- **Razširjena podpora za WebAssembly:** WebAssembly omogoča, da v brskalniku teče visoko zmogljiva koda, kar izboljšuje zmogljivost in odzivnost aplikacij. Z zadnjo verzijo se razširja podpora za WebAssembly, kar pa omogoča še boljšo integracijo z obstoječimi aplikacijami in orodji, ter prinaša boljše zmogljivosti in večjo prilagodljivost pri razvoju.

- **Izboljšana orodja za razvoj in razhroščevanje:** Novi razhroščevalniki, boljša diagnostika napak in napredna orodja za profiliranje aplikacij omogočajo hitrejše odkrivanje in odpravljanje težav ter izboljšujejo kakovost končnih izdelkov.
- **Nove komponente in razširitve:** Osma verzija tudi prinaša vrsto novih komponent in razširitev, ki poenostavljajo razvoj kompleksnih aplikacij. Te komponente vključujejo napredne uporabniške vmesnike, izboljšane možnosti za oblikovanje in prilagoditve, ter boljšo integracijo z drugimi .NET orodji in knjižnicami.
- **Večja podpora za mobilne naprave:** Več pozornosti je bilo namenjene tudi podpori za mobilne naprave. Pri tem imamo v mislih predvsem prilagoditev za različne zaslone, optimizacijo dotika in izboljšana zmogljivost na mobilnih napravah. S tem je zagotovljena boljša uporabniška izkušnja za uporabnike.
- **Optimizacija nalaganja:** Inačica vključuje tudi napredne tehnike za optimizacijo nalaganja, kot so lazy loading, zmanjšanje velikosti prenosa in boljše upravljanje z odvisnostmi. Te izboljšave zmanjšujejo čas nalaganja aplikacij in izboljšujejo uporabniško izkušnjo.
- **Izboljšave pri upravljanju stanja in navigaciji:** Upravljanje stanja in navigacije je učinkovitejše in poenostavljeno. Novi mehanizmi za upravljanje stanja, izboljšana podpora za večstranske aplikacije in boljša integracija z usmerjanjem omogočajo lažje upravljanje kompleksnih aplikacij.

2.2. Načini upodabljanja

Zadnja različica prinaša več novosti in izboljšav pri načinih upodabljanja. Ti načini omogočajo večjo fleksibilnost in dodatno zmogljivost pri ustvarjanju spletnih aplikacij. V tem poglavju bomo predstavili, kakšni so ti načini, kako delujejo in katere prednosti prinašajo [5].

- **Interactive Server:** Je ena izmed ključnih funkcionalnosti Blazor. Omogoča izvajanje aplikacije na strežniku, medtem ko se uporabniški vmesnik prikazuje v brskalniku. V tej verziji so bile narejene pomembne izboljšave glede zmogljivosti in zanesljivosti, kar omogoča boljše delovanje aplikacij pri večjem številu uporabnikov.
- **Interactive WebAssembly:** Omogoča izvajanje C# kode neposredno v brskalniku, brez potrebe po strežniški infrastrukturi. Kot že prej omenjeno je bila razširjena podpora za WebAssembly, kar prinaša hitrejše nalaganje aplikacij, boljšo integracijo z obstoječimi JavaScript knjižnicami in večjo zmogljivost. Omogočeno je enostavno klicanje JavaScript funkcionalnosti direktno iz C# kode.
- **Interactive Auto:** Nov hibridni način upodabljanja, kjer se del aplikacije izvaja na strežniku in je na voljo uporabniku takoj. Del aplikacije pa se izvaja neposredno v brskalniku s pomočjo WebAssembly.
- **Static Server:** Omogoča pred-upodabljanje vsebin na strežniku in njihovo pošiljanje kot statične HTML strani. To bistveno pospeši nalaganje strani in izboljša SEO, saj so vsebine dostopne iskalnikom že ob prvem nalaganju.

Za izdelavo všečnega uporabniškega vmesnika, je ne glede na način upodabljanja na voljo več različnih komponentnih knjižnic, ki se razlikujejo tako po izgledu, funkcionalnostih, enostavnosti uporabe in ceni. Pregledali smo kar nekaj popularnih knjižnic, denimo: Blazorise, Telerik UI for Blazor, MudBlazor, Ant Design Blazor, Radzen. Po tehtnem premisleku in glede na naš primer uporabe, smo se odločili za uporabo knjižnice MudBlazor.

3 MudBlazor

MudBlazor je knjižnica komponent za Blazor, odprtokodni okvir, ki omogoča izdelavo interaktivnih spletnih aplikacij z uporabo C# in HTML. MudBlazor je zasnovan na principih Material Designa, kar pomeni, da nudi moderni, vizualno privlačen uporabniški vmesnik, ki je hkrati enostaven za uporabo in prilagajanje. Z MudBlazor lahko tudi backend razvijalci hitro in enostavno ustvarijo odzivne in estetsko dovršene spletne aplikacije Pri tem potrebujejo minimalno CSS kode, dodatno pa se izognejo poglobljenemu znanju frontend tehnologij, kot so React, Vue ali Angular [2].

3.1. Prednosti MudBlazor

- Enostavna uporaba in integracija: MudBlazor je zasnovan za hitro integracijo v obstoječe Blazor projekte. Knjižnica je dobro dokumentirana in ponuja obsežne primere uporabe, kar omogoča razvijalcem, da hitro začnejo z delom. Integracija v projekt je preprosta in vključuje namestitev paketa in konfiguracijo nekaj nastavitev.
- Material Design: MudBlazor temelji na Material Design smernicah, ki jih je razvilo podjetje Google. To zagotavlja sodoben in enoten videz komponent, kar prispeva k boljši uporabniški izkušnji. Knjižnica ponuja širok spekter predlog in komponent, ki jih je mogoče enostavno prilagoditi potrebam projekta.
- Odzivnost in prilagodljivost: Komponente MudBlazor so zasnovane tako, da so odzivne in se prilagajajo različnim velikostim zaslonov, kar omogoča uporabo na različnih napravah, od mobilnih telefonov do namiznih računalnikov. To omogoča razvijalcem, da ustvarijo aplikacije, ki so dostopne širokemu spektru uporabnikov.
- Bogata zbirka komponent: MudBlazor vključuje številne vnaprej pripravljene komponente, kot so gumbi, obrazci, tabele, navigacijske vrstice, dialogi in še več. To omogoča razvijalcem, da se osredotočijo na poslovno logiko aplikacije, namesto da bi izgubljali čas z oblikovanjem osnovnih elementov uporabniškega vmesnika.
- Aktivna skupnost in podpora: MudBlazor ima aktivno skupnost razvijalcev, ki redno prispeva k razvoju in izboljšavam knjižnice. To zagotavlja, da je knjižnica vedno posodobljena in da so morebitne težave hitro odpravljene.
- Z uporabo MudBlazorja lahko naši backend razvijalci hitro in učinkovito ustvarijo privlačne in funkcionalne spletne aplikacije, ne da bi morali pridobiti dodatno znanje iz frontend tehnologij. To povečuje produktivnost, zmanjšuje stroške razvoja in omogoča hitrejše doseganje poslovnih ciljev.

Odprtokodnost knjižnice in aktivna skupnost ima kot dodaten bonus tudi razširitve osnovnih knjižnic, tako da se nabor komponent, ki jih lahko razvijalec uporabi še dodatno poveča z uporabo katere izmed nadgradenj, kot sta denimo MudExtensions [6] in Mudex [7].

3.2. Kako začeti z MudBlazor

Namestitev

Razvoj z knjižnico MudBlazor je v osnovi možen na dva različna načina:

- uporaba predloge in izdelava novega projekta s pomočjo te predloge,
- ročna namestitev knjižnice v že obstoječi projekt.

Zaradi načine strukturiranja projekta za backend razvijalce, pri čemer smo kot že omenjeno uporabili našo strukturo rešitve, smo se zato določili za drugo možnost in sicer za ročno namestitev. Ta poteka po enakem postopku kot pri .NET knjižnicah z uporabo package managerja. V obstoječi Blazor projekt smo dodali paket Mudblazor.

```
dotnet add package MudBlazor
```

Naslednji korak je bil dajanje referenc v datoteko _Imports.razor, da je knjižnica dostopna na vseh straneh in Blazor komponentah, ki jih bomo ustvarili.

```
@using MudBlazor
```

Popolnoma identičen postopek se uporabi pri uporabi katerekoli nadgradnje, v našem primeru je bila to knjižnica MudExtensions.

Glede na način upodabljanja je potrebno v "index.html" ali "App.razor" dodati tudi osnovne CSS stile in pa JavaScript datoteke.

```
<linkhref="_content/MudBlazor/MudBlazor.min.css" rel="stylesheet" />
<script src="_content/MudBlazor/MudBlazor.min.js"></script>
```

Konfiguracija

Znotraj datoteke Program.cs dodamo referenco na MudBlazor storitve in jih registriramo.

```
using MudBlazor.Services;
builder.Services.AddMudServices();
```

Po registraciji MudBlazor storitev je na vrsti še vključitev osnovnih ponudnikov, za delovanje ogrodja. Pri tem je potrebno poudariti, da je samo ThemeProvider obvezen, ostale pa lahko poljubno dodamo v kolikor jih potrebujemo.

```
<MudThemeProvider/>
<MudPopoverProvider/>
<MudDialogProvider/>
<MudSnackbarProvider/>
```

Po tej konfiguraciji smo pripravljeni za izdelavo spletnih strani s pomočjo MudBlazor komponent.

Uporaba komponent

Po dokončanju konfiguracije, smo pripravljeni za uporabo MudBlazor komponent. Na spodnjem primeru kode smo definirali stran, ki uporabi MudBlazor storitev za delo z dialogi in preda tej storitvi načrt kako naj se dialog izriše in s katerimi nastavitvami. V drugem delu pa smo definirali MudBlazor dialog, ki uporablja MudBlazor dialog API za prikaz preprostega sporočila, ki ga preko parametrov pridobi od glavnega okna. Načrt dialoga vključuje samo vsebino, medtem, ko se nastavitve oblike in obnašanja kot že omenjeno nastavijo prej.

Spodnji izseku kode je del strani, ki vsebuje gumb, ki odpre definirano metodo. V sami metodi se nato poveča nek števec in odpre dialog.

```
@inject IDialogService DialogService

<MudButton @onclick="OpenDialogAsync" Color="Color.Primary">

  Open Counter

</MudButton>

@code {

  private int _counter = 1;

  private Task OpenDialogAsync()

  {

    var parameters = new DialogParameters();

    parameters.Add("Text", "Hello, MudBlazor!");

    parameters.Add("Counter", _counter.ToString());

    _counter++;

    var options = new DialogOptions() {

      CloseButton = true,

      MaxWidth = MaxWidth.Large

    };

    return DialogService.ShowAsync<DialogBlurryExample_Dialog>("Simple Dialog", parameters,
options);

    }

}
```

Naslednji izsek kode pa prikazuje ločeno komponento za izris dialoga v katerega preko parametrov pridobimo tekst in števec ter vse to prikažemo znotraj dialoga.

```
<MudDialog>
  <DialogContent>
    @Text
  </DialogContent>
  <DialogContent>
    @Counter
  </DialogContent>
  <DialogActions>
    <MudButton OnClick="Cancel">Cancel</MudButton>
    <MudButton Color="Color.Primary" OnClick="Submit">Ok</MudButton>
  </DialogActions>
</MudDialog>

@code {
  [CascadingParameter]
  private MudDialogInstance MudDialog { get; set; }
  [Parameter]
  public string Text { get; set; }
  [Parameter]
  public string Counter { get; set; }
  private void Submit() => MudDialog.Close(DialogResult.Ok(true));
  private void Cancel() => MudDialog.Cancel();
}
```

4 Zastavljena struktura rešitve

Zasnova celotne rešitve, je bila načrtovana z namenom, da našim razvijalcem omgočimo kar se da lažji razvoj. Pri tem smo preučili različne projektne strukture iz različnih ogrodij in jih uskladili ter prilagodili s strukturo, ki jo uporabljamo za razvoj naših storitev in mikrostoritev. V osnovi smo tako uporabili princip čiste arhitekture, ki smo ga malo prilagodili z uporabo vzorca Backend for Frontend.

4.1. WebAssembly kot hrbtenica spletne rešitve

WebAssembly je platforma, ki omogoča izvajanje programov, napisanih v različnih programskih jezikih, neposredno v brskalniku. Namesto da bi bil omejen le na JavaScript, lahko WebAssembly izvede kode, napisane v C, C++, C#, Rust in drugih jezikih, po predhodni pretvorbi v Wasm format. Ta format je strojno neodvisen, kar pomeni, da lahko deluje na različnih napravah in operacijskih sistemih brez sprememb kode.

- Visoka zmogljivost: WebAssembly je zasnovan za nameonom, da se izvaja s hitrostjo blizu standardnih aplikacijam. Zaradi binarnega formata in učinkovite uporabe procesorskih zmogljivosti je v marsičem hitrejši od interpretiranega JavaScript-a.

- Kompaktna velikost: Wasm datoteke so kompaktne, kar omogoča hitrejše nalaganje in manjšo porabo pasovne širine. To je še posebej pomembno za aplikacije z omejenimi viri in počasnimi internetnimi povezavami.

- Interoperabilnost: WebAssembly lahko sodeluje z obstoječim JavaScript-om in spletnimi API-ji. To omogoča postopno uvajanje WebAssembly komponent v obstoječe spletne aplikacije, ne da bi bilo potrebno popolno prepisovanje.

- Večjezična podpora: Z WebAssembly lahko razvijalci uporabljajo različne programske jezike, ki so jim bolj domači ali primerni za določene naloge. To omogoča ponovno uporabo obstoječih knjižnic in kodnih baz, kar zmanjšuje čas razvoja in stroške.

- Varnost: WebAssembly deluje v izoliranem peskovniku v brskalniku, kar zagotavlja visoko raven varnosti. Vgrajeni varnostni mehanizmi preprečujejo nedovoljen dostop do sistema in občutljivih podatkov.

Zaradi zgoraj naštetih prednosti je bil Blazor z uporabo Web Assembly izbran za izdelavo naših internih rešitev. S tem smo pridobili vse prednosti dinamične spletne strani, ki se izvaja na odjemalcu (podobno kot Single Page Application) po vrhu vsega pa je večina implementacije izvedla v C# jeziku.

4.2. Uporaba vzorca Backend for Frontend

Backend za Frontend (BFF) je arhitekturni vzorec, ki vključuje ustvarjanje ločenega backend sloja za vsako frontend aplikacijo ali tip uporabnika [8]. Glavna ideja BFF je, da se vsakemu uporabniškemu vmesniku ali napravi omogoči lasten backend, ki je prilagojen specifičnim potrebam in zahtevam tega vmesnika. To omogoča boljšo optimizacijo in prilagodljivost ter zmanjšuje kompleksnost komunikacije med frontend in backend sistemi.

Prednosti BFF

- **Prilagojena API izpostavljenost:** BFF omogoča prilagoditev API-jev specifičnim potrebam posamezne frontend aplikacije, kar zmanjšuje količino prenesenih podatkov in povečuje učinkovitost komunikacije .

- **Izboljšana varnost:** BFF dodaja dodatni sloj varnosti med frontend in glavnimi backend sistemi, kar omogoča boljšo zaščito pred zlonamernimi napadi in nepooblaščenim dostopom .

- **Ločevanje odgovornosti:** Uporaba BFF vzorca omogoča boljšo organizacijo kode, lažje vzdrževanje in boljšo skalabilnost sistema .

- **Optimizacija zmogljivosti:** BFF omogoča optimizacijo zmogljivosti za različne vrste naprav in uporabniških vmesnikov, kar zagotavlja optimalno uporabniško izkušnjo .

- **Hitrejši razvoj in uvedba funkcionalnosti**: Z ločenimi backendi za vsako frontend aplikacijo je mogoče hitreje uvajati nove funkcionalnosti in posodobitve, kar omogoča agilnejši razvoj in manjše tveganje za napake.

Z uporabo vzorca Backend for Frontend smo pridobili prilagojeno in učinkovito komunikacijo med frontend in backend sistemom.

Temelj komunikacije med uporabniškim vmesnikom in med kompleksnimi zalednimi sistemi je tanka plast API vmesnikov, ki jih backend razvijalci z enostavnostjo iztpostavijo iz obstoječim mikrostoritev ali novo napisanih mikrostoritev.

4.3. Struktura Blazor projekta

V ogrodju .NET je rešitev sestavljena iz večjega števila projektov. Pri tem smo posamezne projekte razdelili na vmesnike do zunanjih storitev, na poslovno logiko, podporne storitve in na predstavitveno plast. Kot dodaten del napram strukturi naših storitev je prav ta predstavitvena plast tista, ki vsebuje Blazor in posledično je za backend razvijalce celotna struktura povsem domača. Za čim hitrejši razvoj izpostavljajo vse naše mikrostoritve svoje SDK-je. Ti so potem vključeni v projekt, ki predstavlja Backend for Frontend za uproabniški vmesnik. Backend for Frontend pa izpostavi SDK, ki ga potem Blazor projekt preko vmesnika tudi uporablja. Tudi sama struktura projekta, ki vsebuje ogrodje Blazor je čimbolj približana backend razvijalcem in je sestavljena iz sledečih delov:

- **Vmesniki do ostalih projektov v rešitvi:** Predstavljajo sloj, preko katerega uporabniški vmesnik komunicira z API klici namenjeni uporabniku in so tesno povezani z uporabo vzorca Backend for Frontend. Tukaj je uporabljen SDK od Backend for Frontend projekta, ki izpostavlja vse API klice, ki jih uporablja spletna rešitev.
- **Notranje storitve uporabniškega vmesnika:** Abstrakcija storitev, ki se uporabljajo na uporabniškem vmesniku, da se izpostavijo storitve za večkratno uporabo. V to spadajo storitve, kot je delo z brskalnikom, delo s lokalno shrambo in piškotki, delo z navigacijo itd. Tukaj smo pridobili predvsem skrivanje določenih kompleksnosti, ki se jih razvijalci potem ne rabijo vedno zavedati.
- **Modeli za pogled:** Podatki, ki jih pridobimo iz API klicev niso nujno vedno povsem prilagojeni za prikaz na uporabniškem vmesniku, zato se takšne podakte preoblikuje v modele za pogled, pri čemer se doda določene specifike samo za uporabniške poglede. V tem koraku se zgodijo še formatiranja datumov, pretvorbe iz internih šifrantov v uporabniku prijazne podatke in dodatno izpopolnjevanje podatkov, ki je potrebno samo na uporabniškem vmesniku.
- **Strani:** Osnova Blazor aplikacije so posamezne strani, ki vključujejo osnovno pot, kjer so dostopne. Strani smo razdelili na čisti uporabniški dizajn in na podporno kodo. V Blazorju je to možno s pristopom tako imenovanega "code behind" načina, kjer posamezno stran razdelimo na dve datoteki. Prva datoteka je sestavljena iz same strani in komponent, medtem ko je druga datoteka delni razred, v katerem je večina C# kode, ki upravlja s stranjo in komponentami.
- **Komponente:** V primeru kompleksnejših ali ponavljajočih se kompoment, smo za voljo izboljšane berljivosti le-te prestavili v poseben del projekta. Take komponente so potem na voljo za večkratno uporabo. Primer tega so denimo potrditveni dialogi, ki so povsod identični.

5 Zaključek

Z uporabo .Net Blazor in ogrodja MudBlazor smo tudi backend razvijalcem, ki v večini primerov niso vešči vseh sodobnih pristopov glede gradnje spletnih strani omogočili izdelavo internih spletnih rešitev. Takšen način dela sicer ne bi priporočali za gradnjo spletnih rešitev, kjer želimo čimbolj dovršen spletni vmesnik, saj uporaba obeh tehnologij skupaj vseeno ni na nivoju ogrodij kot so React, Angular, Vue ali podobni.

Prav tako ima uporaba WebAssembly svoje specifike, ki je povsem drugačno kot pa pri standardnih rešitvah. Naprimer ločen peskovnik med posameznimi zavihki in začetno nalaganje. Potrebno je sicer poudariti, da so bili interni uporabniki nad delovanjem celotne rešitve več kot zadovoljni, še posebej pa jim je všeč hitrost posodobitev, saj je za interne aplikacije vedno veljalo, da je kovačeva kobila vedno bosa in je bilo težko najti čas za posodobitve.

Tudi iz vidika samega izgleda je bila uporaba MudBlazor komponent sprejeta odlično, saj je bil sodelavcem Material Design poznan. Vendar pa smo tudi tukaj zaradi načina razvoja upoštevali omejitve. Odločili smo se, da kompleksnih lastnih komponent ne bomo uporabljali ampak samo tiste ki že obstajo ali pa jih je mogoče

sestaviti iz več osnovnih. Na ta način so se tudi interni uporabniki zavedali, da smo omejeni z oblikovnega vidika in da nebo ni meja, kot je to ponavadi pri našem glavnem produktu, ki je namenjen končnemu uporabniku.

V celoti gledano je tak način razvoja prinesel obilo prednosti za naše interne uporabnike in razbremenitev za frontend razvijalce. Tudi backend razvijalci pa so sprejeli izziv izdelave spletnih rešitev, saj so lahko ostali na tehnologijah, ki so jih že dobro poznali.

Literatura

[1] docs.microsoft.com/en-us/aspnet/core/blazor/, Microsoft Blazor Documentation, Obiskano 24. 7. 2024

[2] mudblazor.com/, MudBlazor - Blazor Component Library, Obiskano 22. 7. 2024

[3] blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html, The Clean Architecture, Obiskano 19. 7. 2024

[4] learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0, What's new in ASP.NET Core 8.0, Obiskano 23. 7. 2024

[5] learn.microsoft.com/en-us/aspnet/core/blazor/components/render-modes?view=aspnetcore-8.0, ASP.NET Core Blazor render modes, Obiskano 22. 7. 2024

[6] codebeam-mudextensions.pages.dev/, MudExstensions Page, Obiskano 22. 7. 2024

[7] www.mudex.org/, Mudex Mudblazor Extensions, Obiskano 22. 7. 2024

[8] learn.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends, Backends for Frontends pattern, Obiskano 23.7.2024

Optimiranje delovanja zbirke podatkov časovne vrste

Igor Mernik, Franc Klauzner

Informatika, Informacijske storitve in inženiring, d.o.o., Maribor, Slovenija
igor.mernik@informatnika.si, franc.klauzner@informatika.si

Podatkovna zbirka je eden izmed kamenčkov v mozaiku celotne rešitve obračuna omrežnine, ki smo jo uvedli z rešitvijo podatkovne zbirke časovnih vrst. Ko smo začeli s projektom, smo pričakovali, da bo delo z velikimi količinami podatkov zahtevalo več dela kot je običajno z operativno-transakcijskimi podatkovnimi zbirkami. Predvidevali smo, da sistem ne bo dovolj odziven vsakič, ko bomo v aplikaciji napisali poljuben SQL stavek in ga izvedli. Ugotovili smo, da je delo z velikimi količinami podatkov dokaj drugačno. Da je delovanje podatkovne zbirke optimalno, je potrebno ustrezno nasloviti vsak delček procesa. Za vsak napisan SQL in vsak proces je potrebno razumeti, kako se bo odrazil na odzivnosti podatkovne zbirke. Ker se posamezne operacije izvajajo nekaj milijon-krat, vsak drobec neoptimalne izvedbe lahko pripelje do večurnega podaljšanja obdelav ali pa celo do nikoli končanih obdelav, če so le-te napisane tako, da zavzamejo preveč računalniških virov.

V članku opisujemo, katere programske rešitve smo uporabili pri optimiranju podatkovne zbirke časovnih vrst in zakaj smo se odločili za takšno rešitev. V jedru članka je opisan postopek optimizacij podatkovne zbirke, najprej na ravni arhitekture rešitve, nato kako smo prilagodili aplikacijsko okolje in nastavitve na ravni podatkovne zbirke same ter uvedba nadzora nad optimalnim delovanjem podatkovne zbirke.

Ključne besede:
velepodatki
optimiranje podatkovne zbirke
podatkovna zbirka časovnih vrst
TimescaleDB
obračun omrežnine

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.8

1 Uvod

Dne 25. 11. 2022 je Agencija za energijo sprejela Akt o metodologiji za obračunavanje omrežnine za elektro operaterje [1]. Skladno s tem aktom bo stopil v veljavo nov tarifni sistem za obračunavanje omrežnine. Posledica tega je precej spremenjen sistem obračunavanja omrežnine za električno energijo.

Glavne značilnosti novega tarifnega sistema so:

- obračun, ki temelji na 15-minutnih meritvah,
- uvedba dveh sezon, višje med novembrom in februarjem ter nižje med marcem in oktobrom,
- uvedba pet časovnih blokov,
- razločevanje med dogovorjeno in presežno obračunsko močjo.

Že v preteklosti smo v podjetju Informatika d.o.o. s svojimi informacijskimi rešitvami obvladovali to področje obračunavanja omrežnine za električno energijo. Kljub temu je bil izziv vse obstoječe rešitve prilagoditi zahtevam novega akta. Zaradi lažjega obvladovanja tega izziva smo potrebne prilagoditve razdelili v tri sklope in sicer:

- Platforma za obdelavo merilnih podatkov (POMP) - vzpostavitev rešitve za shranjevanje, validacijo in nadomeščanje merilnih podatkov.
- Obračun omrežnine – implementacija vseh algoritmov novega obračuna za različne vrste merilnih mest.
- Nadgradnja Enotne vstopne točke – EVT (prilagoditev portala Moj Elektro, portala CEEPS in B2B storitev) skratka prenove vse B2B in B2C komunikacije.

Rešitev POMP na najnižji ravni predstavljajo merilni centri elektrodistribucijskih družb (Elektro Celje, Elektro Gorenjska, Elektro Ljubljana, Elektro Maribor in Elektro Primorska), ki pridobivajo merilne podatke iz merilnih naprav odjemalcev preko komunikacije po energetskih linijah (PLC – Power-line communication) ali s protokoli P2P (Peer-to-Peer). Merilni podatki se nato po mehanizmu potiskanja (»push«) v rezinah prenesejo v centralno podatkovno zbirko PostgreSQL s TimescaleDB, ki je vzpostavljena na infrastrukturi Informatike.

V tokratnem prispevku se osredotočamo na POMP in sicer vam želimo predstaviti naše tehnične izzive ob optimizaciji te platforme na ravni podatkovne zbirke.

V okviru te rešitve tečejo kompleksne obdelave sprotne validacije in nadomeščanja podatkov, izračuna obračunskih količin, kot tudi izračuna dogovorjenih moči, ki potekajo nad podatki večjega, v določenih primerih celo celoletnega obdobja. Ob tem pa morajo podatki biti ves čas na voljo tudi zunanjim deležnikom. Sistem mora biti pripravljen tudi za naprednejšo analitiko. Vse skupaj je bil velik izziv, reševanje katerega skušamo predstaviti v okviru prispevka.

2 Arhitektura rešitve

2.1. Arhitekturna slika procesov obdelav podatkov v podatkovni zbirki

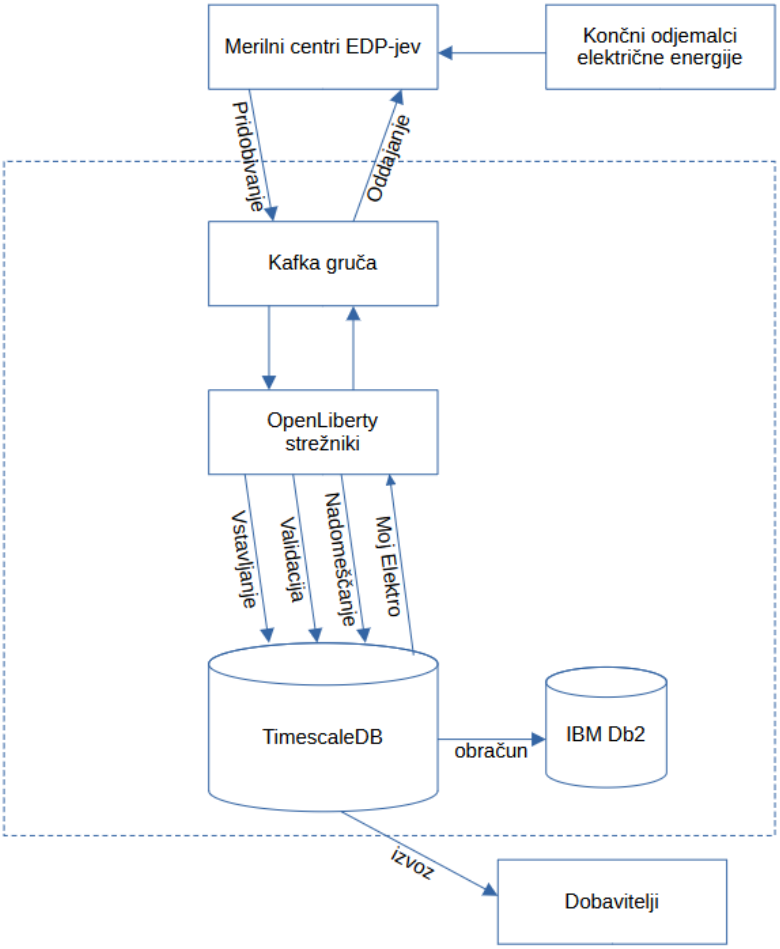
Podatke 15 minutnih odčitavanj količin porabe električne energije merilnih mest končnih odjemalcev električne energije pridobivamo od elektro-distribucijskih podjetij. Izmenjava podatkov je obojestranska.

Podatki se s pomočjo Kafkine gruče strežnikov in OpenLiberty aplikacijskih strežnikov prepisujejo v podatkovno zbirko. Ta postopek imenujemo »Vstavljanje« podatkov oziroma tudi nastanek podatkov. Pri tem postopku gre za množičen vnos velepodatkov.

Nad vnesenimi podatki se izvede postopek »Validacije« podatkov, s katerim se preveri konsistentnost in kakovost pridobljenih podatkov. Temu postopku sledi postopek »Nadomeščanja«, ki po dogovorjenem algoritmu polni manjkajoče vrzeli in počisti morebitne anomalije v podatkih.

Obdelani podatki so na voljo preko spletne in mobilne aplikacije Moj Elektro. Podatke pa izvažamo tudi za potrebe dobaviteljev električne energije.

Kot glavni ponor pa je priprava agregatnih podatkov za obračun električne energije. Ta postopek obračunavanja izvajamo že vrsto desetletij na IBM Db2 podatkovni zbirki. Na TimescaleDB podatkovni zbirki pripravimo agregatne podatke, ki jih potisnemo na IBM Db2 podatkovno zbirko, kjer se izvedejo obdelave obračuna.



Slika 1: Arhitektura obdelave podatkov na podatkovni zbirki časovnih vrst.

2.2. Uporaba podatkovne zbirke

Za poslovne procese uporabljamo dvoje programja in sicer IBM Db2 podatkovno zbirko in Microsoft SQL Server. Za obe vrsti podatkovnih zbirk imamo nekaj deset programskih primerkov podatkovnih zbirk in v vsakem primerku po nekaj podatkovnih zbirk. Ko smo proučevali, katero podatkovno zbirko uporabiti za velepodatke, smo ugotovili, da nas bodo kupljene lastniške podatkovne zbirke licenčno preveč obremenile in hkrati nas precej ovirale pri določanju potrebnih računalniških virov. Odločili smo se za podatkovno zbirko PostgreSQL z razširitvijo TimescaleDB. Slednja je licenčno omejevana le, če bi zunanjemu partnerju ponujali rešitev podatkovne zbirke kot storitev. Ker pa partnerjem ponujamo storitev preko aplikacijskih klicev, smo skladni z licenco in tako popolnoma prilagodljivi glede potreb po razširitvi infrastrukture.

TimescaleDB je razširitev PostgreSQL podatkovne zbirke. Prednost tovrstne rešitve pred posebnimi za velepodatke namenjenimi podatkovnimi zbirkami, je robustnost PostgreSQL podatkovne zbirke in uporaba SQL jezika za obdelavo podatkov. Slednje je zlasti pomembno, da lahko razvijalci aplikacij uporabijo obstoječe znanje, ki ga že imajo pri uporabi dosedanjih rešitev. Za skrbnike podatkovnih zbirk pa je pomembno, da se uporablja preizkušena podatkovna zbirka PostgreSQL.

TimescaleDB 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.

2.3. Nadzor nad zanesljivim delovanjem podatkovne zbirke

Za nadzor nad podatkovno zbirko smo uporabili že obstoječo infrastrukturo za spremljanje strežnikov in sicer nadzorni sistem Zabbix [2]. Izdelek ima vtičnik za PostgreSQL podatkovno zbirko in že privzeto spremlja na stotine parametrov podatkovne zbirke. Izdelali smo preglede - grafikone, iz katerih prepoznamo morebitne težave in tako lažje odreagiramo. Izdelali smo preglede za nadzor porabe procesorske moči in pomnilnika ter časa vhodno izhodnimi operacij, iz česar lahko ugotovimo ali je težave morda z vhodno izhodnimi operacijami diska. Naredili smo pregled učinkovitosti zapisovanja transakcijskih logov, iz katerih ugotovimo nepričakovane povečane aktivnosti podatkovne zbirke. Prikazujemo učinkovitost čiščenja podatkov z »autovacuum« procesi, kar je pomembno, da podatkovna zbirka ne prenaša v pomnilnik nepotrebnih podatkov. Izdelali smo pregled, s katerim prikazujemo število uspešnih in neuspešnih zaključitev transakcij. Naredili smo tudi svoje lastne preglede zaklepanja objektov podatkovne zbirke, s katerimi prepoznamo, če aplikacija nepravilno sprošča zaklepanja objektov. Imamo še številne preglede, ki kažejo učinkovitost notranjih procesov podatkovne zbirke, iz katerih zaznamo potrebo po spremembi parametrov podatkovne zbirke.

Z IBM Instana produktom sledimo aplikacijskim klicem do podatkovnih zbirk. Pri tem proučimo medsebojno odvisnost aplikacijskih klicev in zaporedje izvedbo le-teh ter vpliv posameznih klicev na poizvedbe podatkov iz podatkovnih zbirk. Iz grafikonov razberemo razmerje trajanj posameznih klicev in ugotovimo, katerim klicem posvetimo pozornost za izboljšavo odzivnosti aplikacije in pripadajočih SQL stavkov.

Kot najučinkovitejše sredstvo za prepoznavo težav, ki se odrazijo na podatkovni zbirki, pa je spremljanje izvedenih SQL-ov na podatkovni zbirki. V ta namen uporabljamo PostgreSQL razširitev pg_stat_statements, s katerim prepoznamo SQL-e, ki zasedajo največ računalniških virov, ki so kandidati za optimiranje.

3 Optimizacijski postopki – arhitekturni vidik

3.1. Poznavanje arhitekture procesa polnjenja podatkovne zbirke

Na podatkovni zbirki se izvajajo številni procesi in poznavanje le-teh je nujno za učinkovito izvajanje optimizacij. Pri nas imamo tri velike sklope procesov:
a) inicialno nastajanje podatkov iz merilnih mest,
b) validacija z nadomeščanjem manjkajočih in napačnih podatkov ter
c) zapis iz podatkovne zbirke časovne vrste v podatkovno zbirko operativnega-transakcijskega sistema.

Pomembno je razumevanje arhitekture procesa nastajanja in obdelave podatkov v podatkovni zbirki. Le-to namreč odločilno vpliva na odzivnost ekipe, da je sposobna hitro določiti izvor težave in predlagati možne rešitve. Na strani aplikacijskega strežnika smo ustrezno poimenovali aplikacije, ki s prijavo na podatkovno zbirko le-tej pošljejo

podatek o imenu aplikacije. Če nastanejo težave, lahko iz imena aplikacije zelo hitro omejimo izvor težave in iskanje samo znotraj točno določenega področja.

3.2. Obremenitev podatkovne zbirke

Tudi če imamo zelo zmogljiv računalnik, ne moremo s preprosto akcijo 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. Dodajanje več virov ne bo pospešilo dela, če je osnovni proces počasen. Delati moramo drugače, precej drugače. Samo povečevanje računalniških virov ni rešitev. Kvečjemu obratno, zmanjšaš vire, da lažje vidiš, kaj pomoli glavo iz povprečja.

3.3. Politika odstranitve podatkov

Ker moramo, kot je določeno z zakonom, poskrbeti za uničenje podatkov [3], je bilo potrebno dobro razmisliti, kako se bodo podatki ustrezno odstranili iz podatkovne zbirke. Sprva smo načrtovali, da bi ročno odstranjevali podatke po mesecih, ki jih več ne potrebujemo, vendar sedaj se nagibamo k uvedbi politike samodejnega odstranjevanja podatkov. Samega odstranjevanja podatkov še nismo rešili in sicer iz razloga, da za enkrat še nismo prenašali podatkov za obdobje, za katero bi jih že morali samodejno odstranjevati. Imamo pa že jasno nastavljene okvirje, kako bomo to izvedli, da bomo zakonsko skladni.

4 Optimizacijski postopki – podatkovni vidik

4.1. Spremljanje SQL-ov

Ko pride do težav ali bolje preden pride do njih, je potrebno razumeti, kaj povzroča obremenitev podatkovne zbirke. Podatkovna zbirka sama po sebi ne povzroča obremenitve sistema, obremenitev povzročajo SQL stavki. Pri tem ni pomembno, kateri posamezni SQL-i se najdlje izvajajo, ampak je potrebno gledati agregatno, kateri SQL-i skupaj zasedajo največ računalniških virov [4]. Čeprav bi se na prvi pogled zdelo, da je dolgotrajna petminutna izvedba SQL poizvedbe problematična, lahko predstavlja še večjo težavo poizvedba, ki se sicer izvede hitro v 50 milisekundah, vendar se ponovi milijardo krat. Za spremljanje obremenitve smo uporabili PostgreSQL razšitiev pg_stat_statement [5], s katerim smo pridobili SQL-e, ki so agregatno zasedli največ računalniških virov. Nato smo se osredotočili le na te SQL-e in jih preverili, v čem je njihova posebnost, da zasedajo računalniške vire.

4.2. Kompleksni SQL-i

Če je za operativno-transakcijske SQLe še nekako sprejemljivo, da so SQL-i kompleksni, za podatkovne zbirke časovnih vrst, le-ti ne pridejo v poštev. Če imamo na primer »update« stavek, ki traja pet ur, da se konča in le-tega prekineš po treh urah, si izgubil tri ure časa. Veliko bolje je napisati milijon majhnih »update« stavkov, ki vsak spremeni majhen del podatkov in če po treh urah prekineš proces in je le-ta naredil na primer 40% dela, lahko nato nadaljuješ z naslednjimi obdelavami tam, kjer si ostal in tako imaš le še 60% obdelav za dokončati. Z namenom izboljšanja smo izvedli temeljito preoblikovanje kompleksnih SQL poizvedb v nabor manjših bolj preglednih in lažje vzdrževanih poizvedb. Že nekaj desetletij znano pravilo, da mora biti SQL enostaven in razumljiv, pri podatkovnih zbirkah časovnih vrst pride še toliko bolj do izraza. Namreč izvedba kompleksnega SQL-a preko 100-milijard zapisov bo zanesljivo povzročilo izjemno obremenjenost računalniških virov in verjetno bomo prisiljeni celo v prekinitev obdelave.

4.3. Optimiranje podatkovnega modela

Ko smo imeli seznam najbolj računalniško intenzivnih SQL-ov, smo opazili, da kompleksni SQL-i izvajajo funkcije podatkovne zbirke in izvajajo transformacije podatkov. Rešitev je v prilagoditvi podatkovnega modela. Tako smo spremenili šifrante in jih napolnili s »stored procedurami«, ki smo jih samo enkrat izvedli in zapisali podatke v spremenjen podatkovni model. Tako smo SQL-e, ki se izvedejo več milijon-krat, precej poenostavili in večkratno pohitrili izvajanje celotne obdelave.

4.4. Analiza dostopne poti SQL-ov

Že za preprost SQL lahko napačno oceniš, kako hitro se bo izvajal. Podatkovne zbirke imajo orodja za analizo dostopnih poti posameznih SQL ukazov. Uporabili smo orodje EXPLAIN [6],[7], ki zelo natančno pokaže, kakšne izvajalne operacije načrtuje podatkovna zbirka in kakšne operacije je podatkovna zbirka dejansko izvedla. Iz analize dostopnih poti smo naredili sklepe, kaj je potrebno na podatkovni zbirki spremeniti. Če je smo zaznali, da lahko SQL bolje napišemo, smo ga optimirali. Če se na tabeli izvajajo številna branja podatkov, smo ustvarili manjkajoči indeks in podobno. Torej z EXPLAIN smo pridobili ključne podatke o tem, kje v izvajanju SQL-a so težave.

5 Optimizacijski postopki – aplikacijski vidik

5.1. 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. 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, dokler gre za manjše spremembe nad podatki. Ko smo ugotovili, da za določeni procesi zahtevajo večje spremembe, smo testirali t.i. »upsert« stavek [8], ki v enem samem SQL ukazu naredi vstavljanje podatka v podatkovno zbirko, če pa podatek v podatkovni zbirki že obstaja, pa se izvede sprememba podatka.

5.2. Množično popravljanje podatkov

Med procesom optimizacije, smo na testnem okolju izvedli t.i. množično spremembo podatkov (ang. bulk-update) [9], ki pa se je na produkcijskem okolju izkazala kot prepočasna rešitev, ker podatkovna zbirka časovnih vrst žal nima ali pa še nima, rešeno ustrezno množično popravljanje. S sodelovanjem z razvijalci podatkovne zbirke smo ugotovili, da želena funkcionalnost ne dosega pričakovane odzivnosti. Zato smo razvili aplikacijsko rešitev, kjer smo uporabili t.i. »upsert«, ki z enim ukazom izvede vstavljanje podatkov v podatkovno zbirko, če podatki v tabeli podatkovne zbirke še ne obstajajo po podatkovnem ključu, ali spremeni podatke, če le-ti že obstajajo v podatkovni zbirki, ter programsko zagotovili, da se vedno izvede popravljanje podatkov.

5.3. Bazen povezav do podatkovne zbirke

Do podatkovne zbirke lahko vzpostavimo povezavo v aplikaciji namensko za izvajanje posameznega SQL-a ali skupine SQL-ov. To je pri večkratnem ponavljanju neoptimalno, ker je prijavljanje na podatkovno zbirko časovno potratna operacija, saj mora podatkovna zbirka najprej vzpostaviti mrežno povezavo, nato izvesti avtentikacijski postopek, avtorizacijski postopek in šele nato sledi izvedba SQL-a. Da se izognemo ponavljanju vseh teh postopkov pri prijavi na podatkovno zbirko, s pomočjo aplikacijskega strežnika vzpostavimo t.i. bazen povezav do podatkovne zbirke. Nato aplikacija za dostop do podatkovne zbirke uporabi že vzpostavljeno povezavo. Pri

več milijonih izvedenih operacijah prijavljanja na podatkovno zbirko, se s tem ustvari precejšen časovni prihranek. Natančno smo preverili vse aplikacije, da uporabljajo bazen prijav do podatkovne zbirke.

5.4. Pobegle povezave podatkovne zbirke

Zaradi množice izvajajočih hkratnih vzporednih procesov in hitrega razvoja aplikacij, se lahko zgodi, da nastane kakšna napaka. Razvili smo metodo, s katero zaznamo pobegle povezave v podatkovni zbirki, ki lahko pripeljejo do odzivnostnih težav [10] ali pa celo do neodzivnosti podatkovne zbirke zaradi pretiranega zaklepanja. Za preprečevanje prenosa težav v produkcijsko okolje smo najprej vzpostavili sistem na razvojnem okolju, kar omogoča zgodnje odkrivanje in odpravljanje napak v času razvoja aplikacij. Sistem smo vzpostavili tudi na produkcijskem okolju, s katerim zagotavljamo nemoteno delovanje aplikacij.

5.5. Število povezav do podatkovne zbirke

Ker imamo številne aplikacijske strežnike in na vsakem strežniku vzpostavljen bazen povezav do podatkovne zbirke, je pomembno, da se najprej preveri morebitne pobegle povezave do podatkovne zbirke. Nato smo optimirali število povezav do podatkovne zbirke v posameznem bazenu povezav [11]. Opazili smo, da smo na začetku določili samo grobo oceno, koliko povezav do podatkovne zbirke bomo potrebovali, šele nato izkušnje pokažejo, da je v določenih bazenih povezav določenih preveč povezav, v naslednjih pa premalo. Optimirali smo število povezav v podatkovni zbirki. Ta postopek smo izvedli preden smo povečali število povezav na podatkovni zbirki. Povečanje povezav do podatkovne zbirke pomeni sprememba pomnilniških parametrov na podatkovni zbirki, saj vsaka povezava do podatkovne zbirke zahteva določen del lastnega pomnilnika, kjer se izvaja na primer razvrščanje podatkov in podobno.

5.6. Zaklepanja na podatkovni zbirki

Pri večjih spremembah podatkov v podatkovni zbirki se lahko hitro zgodi, da prihaja do zaklepanja objektov podatkovne zbirke [12]. To se hitreje zgodi, 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. Razvili smo mehanizem za zaznavo neustreznega zaklepanja. Hkrati smo v osrednji sistem nadzora izvedli integracijo, tako da lahko na grafu hitro zaznamo težave z zaklepanjem. Če se dovolj hitro ne ugotoviti, da je prišlo do neustreznega zaklepanja, se lahko na podatkovni zbirki pojavi zaklenjenih 50-tisoč in več objektov in je analiza vzroka zaklepanj precej zahtevnejša.

5.7. Vzporednost izvajanja SQL ukazov in vzporednost izvajanje zapisov

Ko zapisujemo vhodne podatke v podatkovno zbirko časovne vrste, smo ugotovili, da je izjemnega pomena, kako se izvaja zapisovanje podatkov v podatkovno zbirko [13]. Z merjenjem smo ugotovili, da v posameznem SQL stavku za vstavljanje podatkov lahko izvedemo do 3000 zapisov, pri popravljanju podatkov pa je optimalno število nekje tisoč zapisov v posameznem SQL-u. Hkrati vzporedno izvajamo procese iz več aplikacijskih strežnikov in sicer hkrati smo določili kot optimalno 30 hkratnih opravil, ki v podatkovno zbirko zapisujejo podatke. Optimalna zagotovitev vzporednosti zapisovanja je pri podatkovnih zbirkah časovne vrste velikega pomena, saj lahko za mnogokratnik skrajšamo čas izvedbe posameznega opravila podatkovne zbirke.

5.8. Uporaba parametrov v SQL-ih

Obstajata dva načina pisanja dinamičnih SQL-ov. Prvi način je enostavnejši in pri njem vpišemo celotni SQL in ga izvršimo nad podatkovno zbirko. Drugi način je uporaba parametrov [14] in sicer najprej napišemo SQL in v »where« pogojih navedemo parametre, ki jih kasneje v programu izpolnimo. Pri prvem načinu mora podatkovna zbirka za vsak SQL narediti dostopno pot do podatkov in če izvedemo nekaj milijonov SQL-ov, s tem povzročimo časovno zakasnitev in pa nepotrebno obremenitev procesorske moči. Drugi način pomeni, da podatkovna zbirka ustvari dostopno pot do podatkov samo enkrat in nato več milijon-krat uporabi isto dostopno pot. Preverili smo aplikacije, da smo se prepričali, da je povsod uporabljen način dostopa do podatkov s parametri.

5.9. Zakasnitev potrditve transakcije

Ugotovili smo, da se pri množičnem nastajanju podatkov, izvede tudi po deset-tisoč transakcij na sekundo. Vsakokrat, ko izvedemo potrditev transakcije, mora podatkovna zbirka zagotovi konsistentnost le-te in v transakcijski log prepisati podatke, s katerimi zagotavlja, da se v primeru sesutja programja podatkovne zbirke iz različnih vzrokov, lahko ob ponovnem zagonu podatkovne zbirke zagotovi konsistentnost podatkov v podatkovni zbirki. Pri zahtevku za potrditev transakcije pa je podatkovna zbirka prisiljena izvesti časovno potratno operacijo, s katero mora ne samo od operacijskega sistema ampak spodaj ležečega diskovja dobiti 100% zagotovilo, da je bila transakcija zapisana na disk. Ena izmed možnosti je, da nastavimo podatkovno zbirko tako, da le-tega ne zahteva, vendar tvegamo nekonsistentnost podatkov. Kljub temu da lahko v našem okolju del podatkov podatkovne zbirke obnovimo iz izvornih virov, je ključnega pomena zagotavljanje časovnega okna za morebitno obnovo podatkov in razpoložljivost sodelavcev, da to opravilo izvedejo. Druga možnost za pohitritev izvedbe transakcij je potrjevanje transakcij v skupini in ne posamično. To dosežemo z zakasnitvijo posameznih transakcij za delček sekunde [15]. V tem primeru podatkovna zbirka ne zapiše vsake transakcije posebej in čaka na odziv diska, ampak v skupini potrdi transakcije in podatke hkrati zapiše na disk. Pri tem žrtvujemo čas potrditve posamezne transakcije, s ciljem pohitriti skupino transakcij. Nastavili smo zakasnitev transakcije za delček sekunde.

6 Optimizacijski postopki – vidik podatkovne zbirke

6.1. Particioniranje

Največja tabela v našem okolju je velika 80-milijard zapisov, kar predstavlja podatke vseh merilnih mest v Sloveniji za obdobje dveh let in pol. Vsak mesec v podatkovni zbirki nastane dodatne tri milijarde zapisov. Vsi ti podatki ne morejo biti v eni sami veliki tabeli, ker bi SQL-i delovali občutno prepočasi. Hkrati se na tabelo ustvari indeks za pohitritev iskanja podatkov in tudi indeks bi bil prevelik, da bi deloval optimalno. Na podatkovni zbirki lahko uporabimo »partitioning« [16]. 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.

Potrebno je sprejeti odločitev, da posamezna particionirana tabela ni kljub temu prevelika. Glede na izračun priporočene velikosti particionirane tabele, smo se odločili, da nastavimo razmejitev tako, da so za vsak dan podatki v svoji particijski tabeli.

Dodatno obstaja možnost, da se določi particioniranje tabele še po dodatnem particioniranem ključu. V jeziku produkta to pomeni dimenzioniranje oz. dodajanje dimenzij. Zaenkrat ne uporabljamo dodatnih dimenzij, pa ne zato, ker ne bi bilo koristno, ampak predvsem zaradi tega, ker je dimenzioniranje dokaj nova funkcionalnost produkta in zahteva, da se dimenzioniranje nastavi na prazni tabeli. Zaradi izjemne velikosti logične tabele, bi prepis podatkov v novo particionirano okolje trajal nekaj dni in pri tem bi bila podatkovna zbirka toliko obremenjena, da

si poslovno tega ne moremo privoščiti. Imamo pa v načrtu particioniranje tabele tudi po dodatnih ključih, vendar moramo sedaj najti ustrezen čas, ko bo poslovno najbolj sprejemljivo izvesti zahtevno operacijo premika podatkov.

6.2. Razvrščanje podatkov v tabelah

Če je za transakcijsko-operativne podatkovne zbirke priporočljivo, da se izvaja redno skrbništvo z razvrščanjem podatkov v tabelah, pa je v podatkovnih zbirkah časovne vrste to že nujnost [17]. Od tega, kako so podatki fizično zloženi v tabelah, bo odvisno, koliko diskovnih dostopov bo podatkovna zbirka morala izvesti, da bo pridobila podatke. Če so podatki v tabeli neustrezno razvrščeni, bo podatkovna zbirka za dostop do podatkov potrebovala tudi mnogokratnik časa kot v primeru lepo razvrščenih podatkov. Napisali smo optimizacijske skripte, ki se dnevno izvajajo in sproti vsako noč razvrščajo podatke v ustrezno optimalno razvrstitev, tako da dnevne obdelave delujejo bistveno hitreje. Ker pa lahko pridobimo do osem dni stare podatke, mesečno še enkrat razvrščamo podatke.

6.3. Prazen prostor podatkovnih strani tabel

V teoriji podatkovnih zbirk časovnih vrst, podatki v podatkovnih zbirkah samo nastajajo. Senzor proizvaja podatke in jih zapisuje v podatkovno zbirko. Na primer imamo senzor temperature ozračja, ki meri temperaturo in jo pošilja podatkovni zbirki. Če kakšen podatek manjka ali je celo napačen, ni takšne velike tragedije, ker pač tak podatek nima velikega poslovnega vpliva. V našem okolju, pa se na osnovni podatkov merilnikov električne energije obračunava omrežnina. Zato je potrebno zagotoviti, da so podatki ustrezne kakovosti. To pa pomeni, da nad obstoječimi podatki izvajamo spremembe. Podatki v tabelah so fizično zapisani v tako imenovanih podatkovnih straneh. Ker vnaprej vemo, da bodo na tabeli sledile spremembe za zagotavljanje kakovosti podatkov, smo to predvideli in podatkovne strani tabele nastavili tako, da smo nastavili določen prostor, kjer bodo spremembe zapisane [18]. S tem znatno zmanjšamo razpršenost podatkov v podatkovnih straneh, kar zmanjša količino diskovnih dostopov in zagotavlja hitrejše izvajanje podatkovne zbirke.

6.4. Indeksiranje tabel

Iz arhitekture aplikacije se predvidi, kakšni SQL-i se bodo nad podatkovno zbirko izvajali. Na osnovi SQL predikatov, se na podatkovni zbirki določi indeks. Ustrezno indeksiranje na podatkovni zbirki je ključnega pomena za hitrost izvajanja SQL-ov. Pri samem procesu indeksiranja smo bili zelo pozorni, kako bomo posamezen indeks ustvarili, ker ustvarjanje indeksa na nekaj milijardah zapisov tabele, lahko traja več dni. Torej skrbna izbira indeksa [19] in šele nato izvedba. Če ne bi naredili najbolj optimalnega indeksa, potem kasnejša sprememba pri nas iz poslovnih razlogov ni več mogoča, ker bi morali za kakšen dan ustaviti produkcijsko okolje, da bi ustrezno spremenili indeksiranje.

6.5. Agregacija podatkov z materializiranimi pogledi

Materializirani pogledi v podatkovnih zbirkah omogočajo pohitritev SQL poizvedb tako, da v naprej napolnimo podatke v materializiran pogled. Le-tega lahko napolnimo s postopnim polnjenjem ali z enkratnim polnjenjem. V začetku projekta smo uporabljali materializirane poglede [20] in ta funkcionalnost je dobro delovala. Nato smo ugotovili, da smo zaradi narave procesa prisiljeni ustrezno spreminjati podatke in s tem bi morali neprestano na novo osveževati materializirane poglede. S tega vidika, smo se odločili, da je to preveliko breme in smo spremenili dizajn podatkovnega modela, da smo to odpravili. Ali uporabiti materializirane poglede, je predvsem odvisno od tega ali imamo pri posameznem projektu izključno nastajanje novih podatkov, kar je po definiciji pri nastajanju podatkov časovne vrste ali pa se morajo izvajati dokaj intenzivne spremembe nad podatki.

6.6. Stiskanje podatkov

Iz poslovnih zahtev smo razbrali, katere podatke moramo imeti žive in hitro delujoče in katere podatke moramo časovno hraniti določeno število let in katere podatke moramo obvezno po zakonu uničiti. Izvedli smo stiskanje [21] podatkov na podatkovni zbirki, kar je zmanjšalo velikost tabel za 90%. Povpraševanja nad stisnjenimi podatki lahko celo delujejo hitreje, kot nad običajnimi nestisnjenimi podatki. Ugotovili pa smo, da obsežnejša sprememba podatkov v stisnjenih tabelah povzroča opazno poslabšanje odzivnosti. Odločili smo se, da samo tiste podatke, na katerimi bomo izvajali intenzivne spremembe, ponovno razširimo v osnovno obliko, izvedemo spremembe in nato jasno določimo časovno politiko stiskanja podatkov. Sedaj podatkovna zbirka sama skrbi za stiskanje podatkov, za katere smo prepričani, da se ne bodo več spreminjali.

6.7. Sprememba parametrov podatkovne zbirke

Zaradi agresivnih procesov, ki se izvajajo v podatkovni zbirki, smo natančno proučili notranje delovanje podatkovne zbirke [22]. Podatkovna zbirka ima številne podprocese, s katerimi zagotavlja konsistentnost podatkov in pa odzivnost podatkovne zbirke. Izdelali smo mehanizme za poučevanje ustreznosti nastavitev internih procesov. Le-te smo povezali s sistemom za nadzor upravljanja okolja in tako z grafikoni opazujemo spremembe in zaznamo, kdaj je potrebno spremeniti parametre podatkovne zbirke. Podatkovna zbirka ima številne procese, na primer prilagajali smo proces čiščenja pomnilnika umazanih podatkov. Nato smo pohitrili proces izvajanja konsistentnosti podatkov z dodatnim stiskanjem podatkov transakcijskih logov. Optimirali smo proces sinhronizacije podatkov podatkovne zbirke iz pomnilnika na diskovje. Prilagodili smo več deset parametrov in aktivno spremljamo in ustrezno prilagajamo parametre podatkovne zbirke.

6.8. Optimizacija odstranjevanja mrtvih zapisov podatkovnih tabel

Ker smo bili prisiljeni zgraditi sistem spreminjanja in nadomeščanja podatkov, je s tem nastala množica t.i. mrtvih zapisov v podatkovni zbirki. Proučili smo, kako optimialno izvesti čiščenje mrtvih zapisov v podatkovni zbirki s t.i. autovacuum procesom [23]. Pri tem smo izvedli več korakov optimizacije, ker smo tehtali med hitrostjo čiščenja mrtvih zapisov in pa diskovno obremenitvijo. Če čistiš prehitro, pretirano obremenjuješ diskovje, če prepočasi, leto vpliva na odzivnost povpraševanj na podatkovni zbirki. Ob validaciji podatkov za celotno leto, smo čiščenje začasno povsem izključili in tako pohitrili obdelave.

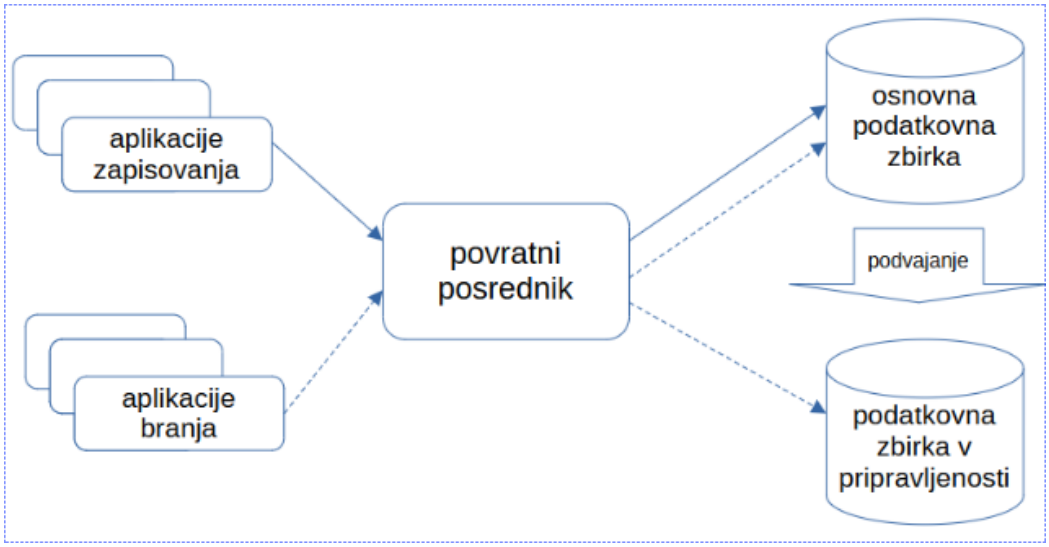
6.9. Spremembe parametrov pomnilniških enot

Večino časa pri optimiranju podatkovne zbirke smo posvetili diskovnim dostopom. Diskovni dostop je lahko tisoč-krat ali več počasnejši od dostopa do pomnilnika. Zato proces optimiranja vidimo kot zmanjševanje potrebe po dostopanju do diskovja. Pri velikih količinah podatkov je lahko težava že sama razdrobljenost podatkov v pomnilniku, ki ga uporablja podatkovna zbirka. Sprva smo mesečno ponovno zaganjali operacijski sistem, ker smo opazili, da deluje vedno počasneje. Z merjenjem pa smo ugotovili, da procesorska jedra tako zelo intenzivno dostopajo do pomnilnika, da so podatki v njem izjemno razdrobljeni in samo iskanje podatkov po pomnilniku pripelje do tega, da se proces delovanja podatkovne zbirke upočasni. Procesor dostopa do pomnilnika do t.i. pomnilniških strani. Privzeto velikost teh pomnilniških strani ni prilagojena količini podatkov, s katerimi upravlja podatkovna zbirka časovnih vrst. Zato smo povečali velikost pomnilniških strani z Linux »kernel« parametri in nastavili nastavitev, da podatkovna zbirka uporablja te velike pomnilniške strani [24]. S tem smo povsem odpravili potrebo po rednem ponovnem zagonu operacijskega sistema, kar je na produkcijskem okolju zelo pomembno, da ne prekinjamo procesov.

7 Optimizacijski postopki – vzporednost izvajanja

7.1. Podvajanje okolja podatkovne zbirke

Šele v fazi, ko smo izvedli vsa ostala optimiranja na podatkovni zbirki, je napočil čas za povečanje računalniške razpoložljivosti z vzporednim izvajanjem podatkovne zbirke. Zahteva naročnika je, da podatki čim prej nastanejo v podatkovni zbirki in so na voljo za uporabo. Posledica tega je, da se v kratkem časovnem razponu pojavlja množično nastajanje in spreminjanje podatkov. Le-to pa ima vpliv na odzivnost klicev podatkov iz podatkovne zbirke.



Slika 2: Arhitektura podvajanja podatkovne zbirke.

Vzpostavili smo okolje podvajanja podatkovne zbirke z namenom pohitritve delovanja poizvedb iz podatkovne zbirke. Imamo dve vrsti aplikacij in sicer aplikacije za množično zapisovanje podatkov in aplikacije za branje. Množično zapisovanje podatkov se izvaja v osnovno podatkovno zbirko. Nato se izvaja proces podvajanja in prepis podatkov v podatkovno zbirko v pripravljenosti. Predstavitvene aplikacije pa berejo podatke iz obeh podatkovnih zbirk. Razporejanje obremenitve osnovne podatkovne zbirke s strani aplikacij za branje je odvisno od dneva v mesecu. V začetku meseca je osnovni podatkovni strežnik bolj obremenjen in tako se na njem izvaja manj aplikacij za branje podatkov. V preostalih dneh je razporeditev med osnovnim strežnikom in strežnikom v pripravljenosti bolj uravnovešeno. Takšna arhitektura omogoča zanesljivost delovanja strežnikov podatkovne zbirke in razporeditev obremenitve strežnikov. Hkrati je dodatna zaščita ob silovitem povečanju mrežnega prometa, da osnovna podatkovna zbirka deluje nemoteno.

8 Zaključek

Podatkovna zbirka in procesi, ki se izvajajo v njej, se neprestano spreminjajo in prilagajajo zahtevam naročnika projekta. Na spletu obstajajo številni viri za pohitritev delovanja podatkovne zbirke. Redno spremljanje dnevnikov podatkovne zbirke, gledanje video posnetkov in branje knjig, vodi do novih in novih idej, kako je mogoče optimirati podatkovno zbirko. Ni enega samega recepta za optimiziranje odzivnosti podatkovne zbirke. Le-ta je odvisna od številnih mehanizmov in procesov, ki tečejo v podatkovni zbirki. Vsem optimizacijam pa je skupno predvidevanje morebitnih težav, zaznava obstoječe težave, prepoznava ideje za izboljšavo, razvoj metode, s katero merimo proces izboljšave, izvedba same izboljšave in merjenje učinka.

V podjetju smo vzpostavili zaupanje med razvijalskimi, sistemskimi in ekipami podatkovnih zbirk. Na ravni baze pa razvili metode dela, ki neprestano uvajajo preizkušanje novih idej. Podatkovne zbirke velikih podatkov imajo

številne možnosti za izboljšanje odzivnosti delovanja podatkovne zbirke. V prispevku smo prikazali številne optimizacijske prijeme z arhitekturnega, aplikacijskega, podatkovnega vidika in z vidika podatkovnih zbirk.

Prišli smo do točke, da podatkovna zbirka deluje odzivno in stabilno. S tem smo pripravljeni na nove izzive, ki bodo vključevale podatkovno zbirko velikih podatkov, kot je strojno učenje na velikih količinah podatkov, napredna analitika in podobno.

Literatura

[1] Akt o metodologiji za obračunavanje omrežnine za elektrooperaterje, Uradni list RS, št. 146/2022 z dne 25. 11. 2022, https://www.uradni-list.si/glasilo-uradni-list-rs/vsebina/2022-01-3624/akt-o-metodologiji-za-obracunavanje-omreznine-za-elektrooperaterje, obiskano 19. 6. 2024.

[2] www.zabbix.com, Zabbix + Postgresql, obiskano 13. 6. 2024.

[3] docs.timescale.com, Create a data retention policy, obiskano 13. 6. 2024.

[4] www.youtube.com, PostgreSQL performance in 5 minutes, obiskano 13. 6. 2024.

[5] www.eversql.com, PostgreSQL – pg_stat_statements, explained, obiskano 13. 6. 2024.

[6] www.postgresqltutorial.com, PostgreSQL EXPLAIN, obiskano 13. 6. 2024.

[7] www.cybertec-postgresql.com, How to interpret PostgreSQL EXPLAIN ANALYZE output, obiskano 13. 6. 2024.

[8] www.tutorialsteacher.com, PostgreSQL - UPSERT Operation, obiskano 13. 6. 2024.

[9] www.alibabacloud.com, How Does PostgreSQL Implement Batch Update, Deletion, and Insertion, obiskano 13.6.2024.

[10] knowledge.enterprisedb.com, Idle and Idle in transaction connections and impact on performance and ways to handle it, obiskano 13. 6. 2024.

[11] www.enterprisedb.com, The Challenges of Setting max_connections and Why You Should Use a Connection Pooler, obiskano 13. 6. 2024.

[12] hevodata.com, PostgreSQL LOCKS: Comprehensive Guide 101, obiskano 13. 6. 2024.

[13] www.postgresqltutorial.com, PostgreSQL INSERT Multiple Rows, obiskano 13. 6. 2024.

[14] www.ibm.com, Parameter markers, obiskano 13. 6. 2024.

[15] medium.com, Delaying commits in PostgreSQL, obiskano 13. 6. 2024.

[16] docs.timescale.com, Hypertable partitioning, obiskano 13. 6. 2024.

[17] docs.timescale.com, Reorder_chunk, obiskano 13. 6. 2024.

[18] www.cybertec-postgresql.com, What is fillfactor and how does it affect PostgreSQL performance?, obiskano 13. 6. 2024.

[19] www.freecodecamp.org, Advanced Indexing Strategies in PostgreSQL, obiskano 13. 6. 2024.

[20] docs.timescale.com, Continuous aggregation, obiskano 13. 6. 2024.

[21] docs.timescale.com, Compression, obiskano 13. 6. 2024.

[22] ROGOV Egor, "Buffer cache and WAL", PostgreSQL 14 Internals", Postgres Professional, 2022, str. 168-209.

[23] pganalyze.com, Visualizing & Tuning Postgres Autovacuum, obiskano 13. 6. 2024.

[24] www.enterprisedb.com, Improving PostgreSQL performance without making changes to PostgreSQL, obiskano 13.6.2024.

Merjenje učinka uporabe strojnega učenja pri mikroplaniranju proizvodnje

Matjaž Roblek,[1] Vukašin Radisevljević,[2] Alenka Brezavšček[2]

[1] Domel d.o.o., Železniki, Slovenija
matjaz.roblek@domel.com
[2] Univerza v Mariboru, Fakulteta za organizacijske vede, Kranj, Slovenija
vukasin.radisavljevi@student.um.si, alenka.brezavscek@um.si

Tekom izvajanja širše raziskave o uporabi strojnega učenja (ang. machine learning - ML) v procesu mikroplaniranja proizvodnje, ki jo štiri leta izvajamo v realnem okolju podjetja Domel, smo odkrili težave z merjenjem točnosti napovedovanja trajanja proizvodnih nalogov (v nadaljevanju PN). Napovedovanje trajanja PN razbremeni planerje in mojstre v proizvodnji ročnega usklajevanja, ko proizvodnja ni sposobna dosegati želenih normativov tehnološkega postopka, ali so prisotne težave z nezanesljivostjo proizvodnega procesa. Ugotovili smo, da v fazi uvajanja novega izdelka v proizvodnjo ML izboljšuje svoje napovedi. Ko smo longitudinalno povečevali število ponovitev PN za enake izdelke, so na meritve začeli vplivati dejavniki nezanesljive proizvodnje in slabšali napovedi trajanja PN. V prispevku prikazujemo uporabljeno metodo za čiščenje podatkov, na podlagi katere smo lahko na izbranem vzorcu izdelkov dokazali, da zaradi nestabilnosti proizvodnega procesa in pomanjkanja zanesljivih podatkov ne moremo trditi, da ML napoveduje trajanje PN bolje od človeka (niti slabše). Predstavljamo izzive pri statističnem dokazovanju te trditve.

Ključne besede:
strojno učenje
mikroplaniranje proizvodnje
merjenje
nezanesljivost proizvodnje
nezanesljivi podatki

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.9

1 Uvod

Z razvojem tehnologij *strojnega učenja in globokega učenja* (v nadaljevanju ML) se povečuje število praktičnih uporab v poslovnih procesih [1-3]. V prispevku se osredotočamo na področje planiranja v proizvodnih podjetjih, natančneje na uporabo ML v procesu *mikroplaniranja proizvodnje* (ang. production scheduling). Ta proces z vidika razvrstitve informacijskih sistemov podpirajo napredni sistemi za terminiranje in razvrščanje PN (ang. advanced production scheduling - APS), ki obogatijo poslovne informacijske sisteme z naprednimi algoritmi za optimizacijo razporedov PN v proizvodnji, s katerimi povečujemo izkoristek proizvodnih kapacitet. Z razvojem in vključenostjo tehnologije ML ti sistemi dodatno pomagajo razreševati problem razlik med želenim trajanjem PN po normativih in dejansko dosegljivim trajanjem. Velike razlike so značilne za serijsko proizvodnjo tehnološko zahtevnih izdelkov, kjer proizvodnja pri izdelavi določenega izdelka začenja z velikimi odstopanji od želenega in se z vsako ponovitvijo PN nekaj nauči bolje in tako približuje ciljnim normativom. Planiranje takega procesa je zahtevno, saj mora planer ob vsaki iteraciji oceniti izboljšavo, popraviti napoved trajanja PN na podlagi podatkov o dejanski realizaciji in rahlo »priganjati« proizvodnjo pri naslednji ponovitvi , da ima ta motiv za izboljševanje. Klasični APS tega ne zna, *napredni APS z AI* (ang. AI APS, tudi Smart APS) pa poskuša to ročno napovedovanje planerja in mojstra posnemati s ML in »robotizirati« planske aktivnosti v procesu mikroplaniranja (ang. RPA - Robotic Process Automation) [4].

Izsledki v literaturi [5,6] navajajo številne pozitivne učinke naprednih APS z AI, zaradi katerih se podjetja zanimajo za vpeljavo ML v svoje procese. Po drugi strani pa primanjkuje empiričnih študij, ki bi se osredotočale na merjenje konkretnih učinkov algoritmov ML pri napovedovanju trajanja PN, kar je osnovni kriterij pri ugotavljanju dodane vrednosti in ekonomske upravičenosti tovrstnih tehnologij. Z rezultati pričujoče raziskave smo skušali to raziskovalno vrzel vsaj delno zapolniti.

Raziskava predstavlja študijo primera, ki smo jo izvedli v realnem okolju na vzorcu proizvodnje izdelkov podjetja Domel [7], kjer pri mikroplaniranju proizvodnje uporabljajo AI APS rešitev LEAP slovenskega podjetja Qlector [8] in sicer na proizvodnih tehnologijah stiskanja in brizganja. LEAP se uporablja od začetka leta 2021 [9], kar zagotavlja dovolj dolgo časovno obdobje za zajem dovolj velikega vzorca podatkov za prve analize njenih učinkov.

Motiv za raziskavo je izhajal iz problema, da algoritmi ML potrebujejo določeno količino podatkov o ponovitvah PN enakega izdelka na postrojenjih, pri različnih pogojih, kot so določen stroj, čas izmene, dan v tednu, usposobljenost prisotnih delavcev ipd., da lahko zagotovijo bolj natančne, smiselne in uporabne napovedne informacije [10]. Predvidevali smo, da se točnost predvidevanja algoritmov AI APS za posamezni izdelek izboljšuje z naraščanjem števila ponovitev (iteracij) PN za dotični izdelek. Z analizo različnih testnih primerov smo ugotovili, da lahko naše predvidevanje potrdimo le deloma. Namreč, zaradi ne povsem predvidljivih razmer v proizvodnji in »posebnosti« procesa proizvodnje, kjer nastopa veliko različnih bolj ali manj naključnih/izrednih dogodkov se dogaja, da algoritmi ML generirajo veliko napako pri predvidevanju kljub temu, da so predhodno že dosegali kvalitetne napovedi trajanja dotičnega PN. V poglobljenih analizah želimo ugotoviti po kolikšnem času (t.j. po kolikšnem številu izvedenih iteracij PN za določen izdelek) lahko zaupamo, da so napovedi AI APS dovolj natančne in zanesljive, da lahko planer izračunavanje trajanja povsem prepusti avtomatiki z AI APS. Poleg tega želimo ugotoviti, kaj vse vpliva na to, kako hitro se sistem AI APS stabilizira, da ponovno dobro napoveduje, ter na kakšen način lahko v proizvodnji k temu pripomoremo.

Z izsledki raziskave podajamo konkretne izkušnje pri merjenju učinkov uporabe ML pri napovedovanju trajanja PN in izpostavljamo zanimive izzive, v katere bi se veljalo osredotočati v prihodnjih raziskavah. Rezultati so koristni in uporabni za podjetje, ki je v študiji primera sodelovalo, kakor tudi za ostala podjetja, ki se soočajo z izzivi uvajanja algoritmov ML v svoje proizvodne in poslovne procese.

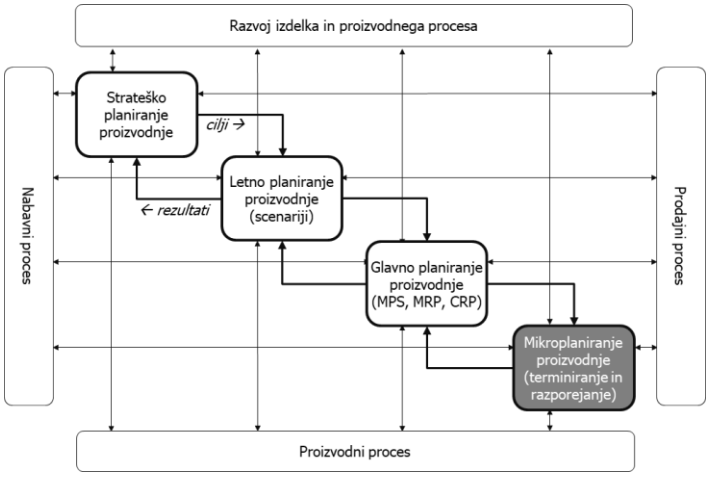
2 Metodologija

2.1. Opis proučevanega podjetja in proizvodnega okolja

Podjetje Domel je mednarodno podjetje, katerega glavna dejavnost je razvoj in proizvodnja komponent in aplikacij elektromotorjev. Sedež podjetja je v Sloveniji, proizvodne lokacije pa ima na Kitajskem in v Srbiji. Predmet proučevanja je proizvodnja na oddelku avtomatskih stiskalnic, kjer se izdelujejo deli iz elektro pločevine, ki se proizvajajo neposredno za kupce ali za nadaljnjo vgradnjo v kompleksne izdelke. Delavnica ima delavnično razporeditev vzporednih istovrstnih strojev. Oskrbuje se po principu ravno pravočasno z materialom elektro pločevino iz vhodnega skladišča. Elektro pločevina različnih tipov in dimenzij je pripravljena na embalažnih enotah, kolutih. Embalažne enote se izda iz skladišča in z viličarjem dostavi na stroj, kjer se izdelajo izdelki, rotorske in statorske lamele različnih dimenzij in oblik. Izdelki se polnijo v izhodne embalažne enote, zaboje, in se sproti odpremljajo z viličarji v izhodno skladišče. Od tam sledi transport h kupcem ali v druge proizvodne enote Domela. V proizvodnji se uporablja le tehnologija stiskanja. Delovno postajo v proizvodnji tvori stroj stiskalnica, orodje za določen izdelek ter robot ali človek za pakiranje izdelkov v izhodno embalažno enoto. Nekateri izdelki se lahko proizvajajo le na namenskem stroju, izdelave drugih pa se lahko razporeja na različne stroje z različno zmogljivostjo, vendar imajo določeno primarno verzijo stroja, ki je prioriteta pri razporejanju. Tehnološki postopek je enostaven, ima do dve zaporedni delovni operaciji, ki jih izvedemo s pomočjo orodja na stroju. Posebnost proizvodnje je sočasna proizvodnja različnih izdelkov z istim orodjem (zaradi boljšega izkoristka pločevine na stroju), ki se hkrati polnijo v različne embalažne enote za različne kupce.

V podjetju imajo vzpostavljene štiri hierarhično in časovno strukturirane procese planiranja proizvodnje (slika 1). Proučevano delavnico planira, terminira in razporeja le en planer proizvodnje, ki je prisoten le v dopoldanski izmeni. Raziskava obravnava najnižji in najbolj dinamičen proces mikroplaniranja proizvodnje, ki planerju predstavlja 90% dela. Ostalih 10% je obremenjen z izvajanjem glavnega planiranja. V ostalih dveh hierarhično višjih procesih planiranja proizvodnje planer ne sodeluje, dobiva pa iz teh procesov smernice in pravila ter zagotavlja zanju povratne informacije iz realizacije proizvodnje, le-te pa vstopajo v mikroplaniranje proizvodnje.

Celoten proizvodni proces je informacijsko podprt in generira podatke v realnem času. Proizvodnja večino leta poteka v treh izmenah, tudi preko vikendov. Proizvodnjo pripravljajo v poslovnem informacijskem sistemu ERP - SAP HANA, mikroplaniranje pa je podprto s spletno storitvijo AI APS LEAP. Realizacija proizvodnje, ki je pomembna za merjenje točnosti predvidevanj ML, se beleži ročno s potrditvami delavcev v proizvodnem informacijskem sistemu MES Kiner. V primeru izvedbe proizvodnega procesa z malo odstopanja od planskih vrednosti se večina podatkov o realizaciji PN prenaša v bazo podatkov avtomatsko s potrditvami delavca. V primeru variabilnosti, kot so zastoji, izmet ipd., pa se dodatne informacije o odmikih PN ročno izbira in poroča v MES. Organizacijski predpis določa, katere korekcije v IS sme določati delavec, kaj njegov izmenovodja in kaj vodja delavnice.



Slika 1: Shematični prikaz umeščenosti mikroplaniranja proizvodnje v proces planiranja v podjetju Domel. [9]

2.2. Predmet raziskave

Raziskava izhaja iz realnega izziva, ki ga običajno rešujemo z znanjem in izkušnjami planerja in mojstra: če je napoved trajanja PN dovolj točna, so doseženi pozitivni učinki na večino proizvodno poslovnih procesov:

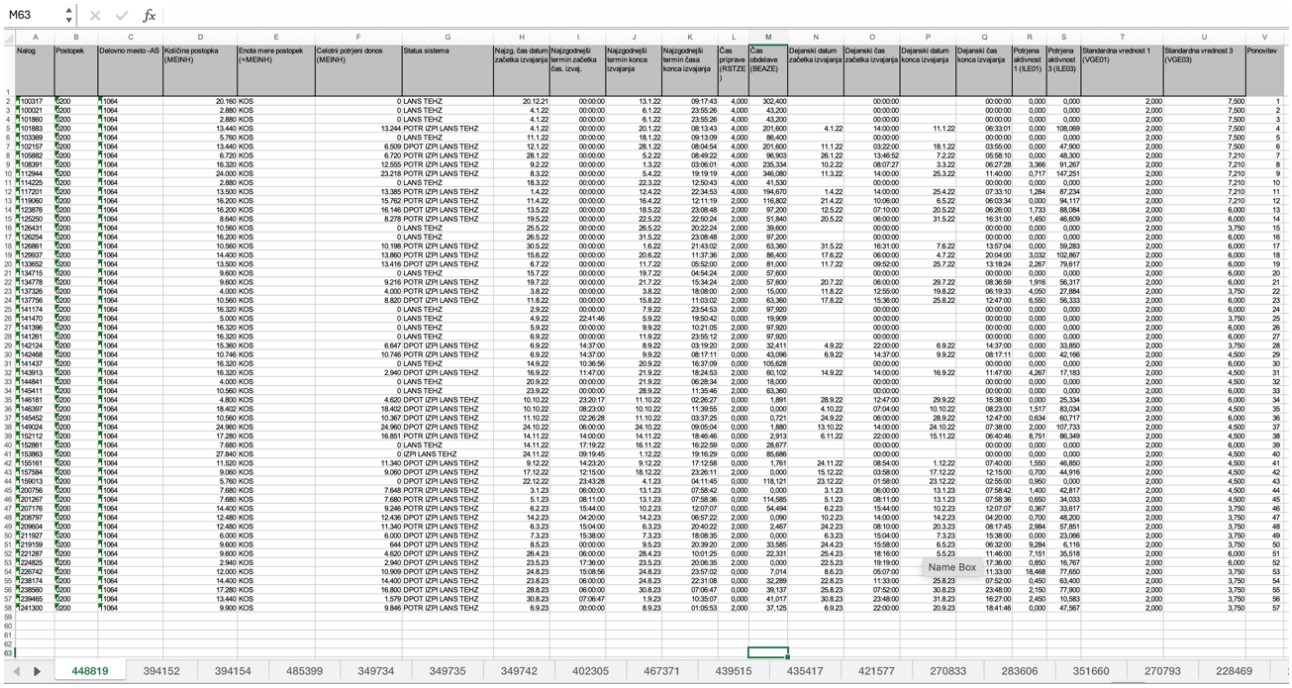
- dosežemo zanesljivo oskrbo kupca v prodajnem procesu; ne zamudimo,
- dosežemo stabilno, umirjeno in posledično kvalitetno delo v proizvodnem procesu; manj izmeta,
- dosežemo terminsko usklajeno oskrbo z materiali v nabavnem procesu; posledično imamo na nižje zaloge.

Naloga planerja ni zagotoviti izvedbo proizvodnje, kot je določeno v želenih normativih proizvodnje. To je poslovni cilj, h kateremu stremimo, da ga čim prej dosežemo in je v domeni razvoja proizvodne tehnologije in proizvodnje same. Ko je dejansko trajanje PN enako planiranemu trajanju in to enako želenemu trajanju, je poslovni cilj dosežen. Tako od planerja pričakujemo, da »zna delati« z odstopanji od želenih normativov, da doseže zanesljivo oskrbo kupca. Ta v obstoječem procesu zahteva, da mu tehnologija pripravi dodatne proizvodne verzije v ERP z različnimi normativi, npr. pesimistični normativi za uvajanje izdelka v proizvodnjo. *Ker je to dodatno delo za poslovno področje tehnologije, se ne izvaja redno za vse izdelke*.

Kompleksno raziskovalno vprašanje, ki nas zanima, je, kako se pri tem izzivu v praksi izkaže ML in to primerjati z izkušenim planerjem. Algoritmi ML potrebujejo določeno količino podatkov, da lahko zagotovijo točne napovedne informacije. Praktično vprašanje je, kako se uspešnost teh algoritmov pri predvidevanju trajanja PN spreminja s časom oziroma s številom ponovitev. Tovrstne informacije bi bile uporabne za podjetja, ki načrtujejo uvedbo AI APS v svoj proizvodni sistem, saj bi lahko načrtovala, koliko časa bo potrebno vlagati resurse v "učenje" do točke, ko bo lahko ML koristno služil proizvodnemu sistemu. Trenutno tovrstne informacije za sistem LEAP niso na voljo, zato morajo podjetja, ki se za LEAP odločijo, prevzeti tveganje vlaganja sredstev za nedoločen čas. Da bi odgovorili na zastavljeno raziskovalno vprašanje, je potrebno proučiti dva scenarija:

- *1 scenarij: Analiza uspešnosti napovedovanja trajanja PN na podlagi normativov iz ERP sistema:* pred prvo izvedbo serije izdelka planer predvidi (izračuna) trajanje proizvodnega naloga na podlagi matičnih podatkov (angl. master data – MD) o želenih normativih izdelave iz ERP sistema. Pred vsako novo ponovitvijo proizvodnega naloga planer preveri, kakšno je bilo odstopanje realiziranega trajanja PN iz MES, ter se ob posvetovanju z mojstrom v proizvodnji odloči, ali bo v nadaljevanju uporabil pesimistično proizvodno verzijo iz ERP (če obstaja). Sicer subjektivno podaljšuje planirano trajanje PN v ERP s časovnimi pribitki (t.i. časovni »blažilniki«) izven podatkov v normativih.
- *2 scenarij: Analiza uspešnosti napovedovanja trajanja proizvodnih nalogov s stojnim učenjem LEAP:* pred prvo izvedbo LEAP nima drugih podatkov kot normative iz ERP in iz njih izračuna trajanje. Zato pričakujemo, da na začetku »greši« bolj kot planer. Po več ponovitvah proizvodnega naloga LEAP preklopi na lastno bazo, kjer je primerjal izračune iz podatkov ERP z realizacijo iz MES in je začel razvijati svoj napovedovalni model, kjer za določanje trajanja PN ne upošteva več matičnih podatkov ERP.

Za potrebe raziskave smo pridobili empirične podatke iz proizvodnje v obdobju januar 2021 – november 2023. Podatki o normativih s korekcijo planerja so MD iz ERP HANA, podatki o napovedih trajanja LEAP so pridobljeni iz sistema APS LEAP, podatki o realizaciji pa so pridobljeni iz sistema MES KINER. Vsi podatki so bili izvoženi v Microsoft Excel, pri čemer so bili podatki za posamezne proizvedene ID-je izdelkov prikazani na ločenih listih (slika 2). V preliminarni fazi raziskave smo uporabili vzorec 23 različnih izdelkov (identov) z različnim številom ponovitev PN. Posamezna ponovitev PN je vsebovala 22 različnih podatkov (npr. število kosov, planirani čas, normativni čas na 1000 kosov, začetni/končni datum in ura, pripravno-končni čas...). Za začetno testiranje uporabnosti metode dokazovanja koristi ML smo zožili nabor in se osredotočili na 6 izdelkov: ID 421577, 237597, 270833, 351660, 272811, 442122. Pri izbiri končne množice testnih izdelkov je bil ključni dejavnik število ponovitev PN. Osredotočili smo se namreč na izdelke, pri katerih je bilo zabeleženih vsaj nekaj 10 ponovitev PN v obdobju opazovanja. Glede na to, da je bila v septembru 2021 izvedena obsežnejša prilagoditev algoritma v APS LEAP so za potrebe naše raziskave merodajni zapisi, ki so se zgodili po tej spremembi.



Slika 2: Vhodni podatki za potrebe raziskave. [9]

Glavno merilo, ki ga bo osnova za statistične analize v pričujoči raziskavi, je relativna napaka napovedi trajanja izvedbe PN, ki jo generira sistem LEAP v primerjavi s potrjenim oziroma realnim trajanjem izvedbe PN. To napako izračunamo na naslednji način:

$$\Delta\, LEAP = \frac{t_{LEAP} - t_{MES}}{t_{LEAP}} \times 100 \qquad (2.1)$$

Kadar nam je pomembna le velikost napake ne pa tudi njena smer (+,-), uporabimo absolutno vrednost ΔLEAP.

2.3. Tri-stopenjski pristop k čiščenju podatkov

Po pregledu podatkov iz proizvodnje smo ugotovili, da nekatere vrednosti močno odstopajo od ostalih, kar najverjetneje kaže na pojav neobičajnih okoliščin pri realizaciji PN. Čeprav ekstremne situacije vplivajo na realizacijo proizvodnje, smo se odločili, da jih v tej fazi analize izločimo, saj smo za začetek želeli proučiti obnašanje LEAP sistema v hipotetično stabilnih razmerah v proizvodnji. Za izločitev ekstremnih situacij smo določili 3 kriterije:

- *1 kriterij: eliminacija popolnoma neveljavnih podatkov*: V prvi fazi smo želeli izločiti tiste realizirane PN, ki sploh niso bili planirani preko sistema LEAP, saj niso relevantni za našo raziskavo. Iz podatkov smo izločili vse zapise, kjer je
    - čas nastavljanja pred septembrom 2021; tak PN ni bil obdelan skozi LEAP,
    - čas nastavljanja po septembru 2021, vendar so vrednosti LEAP napovedi natanko enake, kot je vrednost po matičnih podatkih; sklepamo, da tak PN ni obdelan skozi LEAP,
    - potrjeni čas obdelave je 0h,
    - planirani čas trajanja po LEAP je 0h.
- *2 kriterij: eliminacija zapisov, ki predstavljajo velika odstopanja*: Po izvedenem čiščenju po 1 kriteriju smo želeli identificirati zapise, kjer je odstopanje med napovedjo LEAP in MD zelo veliko. Do takih situacij lahko pride zaradi morebitnega izrednega dogodka v proizvodnji, razlike med planiranimi in narejenimi kosi, ali drugih neusklajenosti med različnimi informacijskimi sistemi, npr. delovnega koledarja LEAP in ERP (v

enem informacijskem sistemu je določena sobota označena kot delovni dan, v drugem sistemu je ta ista sobota označena kot prosti dan). Za identifikacijo zapisov z velikimi odstopanji, smo:

- izračunali tretji kvartil Q3 za relativno napako ΔLEAP,
- za vrednosti ΔLEAP, manjše od Q3, smo izračunali povprečje $\bar{x}_{Q3}$ in standardni odklon σ_{Q3},
- nato smo izračunali kritično vrednost po naslednji enačbi:

$$kritična\ vrednost\ = \bar{x}_{Q3} + 2 \times \sigma_{Q3} \quad (2.2)$$

- nato smo izločili vse zapise, pri katerih je ΔLEAP večji (>) od izračunane kritične vrednosti.

– *3 kriterij: glajenje podatkov*: Če po uporabi kriterija 2 še ne dobimo dovolj homogenih podatkov, nadaljujemo z metodo glajenja podatkov. Kot osnovo za izbiro vrednosti, ki jih je potrebno zgladiti, smo uporabili regresijsko premico linearnega trenda ter njen interval zaupanja, ki ga izračunamo tako, da regresijski premici prištejemo ali odštejemo vrednost 2 standardnih odklonov. Vrednosti ΔLEAP, ki so zunaj teh meja (t.i. »neveljavni« podatki), se gladijo po postopku:

- če je podatek, ki ga je potrebno gladiti, prvi v vzorcu podatkov, ga izenačimo s prvim naslednjim podatkom, ki je v mejah (je »veljaven«),
- v primeru, da ima »neveljavni« podatek »veljavnega« predhodnika in naslednika, se le-ta nadomesti z njuno povprečno vrednostjo,
- če je več podatkov zaporedoma zunaj meja (»neveljavnih«), nadomestimo prvi »neveljavni« podatek s povprečjem zadnjega »veljavnega« podatka pred njim in prvega »veljavnega« podatka za njim. Nadalje podatke gladimo po enakem postopku, dokler niso zglajeni vsi podatki v vzorcu.

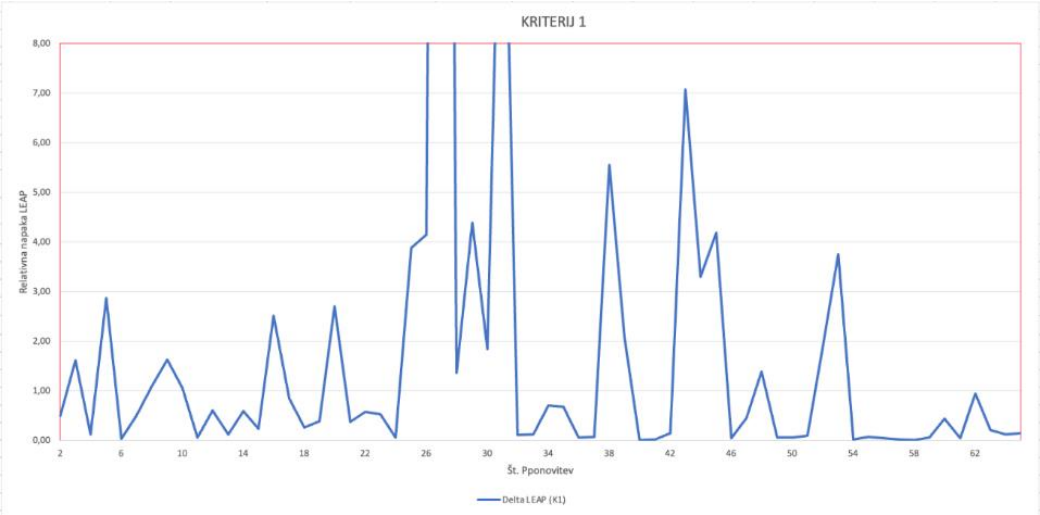
3 Rezultati

V nadaljevanju bomo podrobneje predstavili postopek izvedbe tri-stopenjskega čiščenja podatkov ter rezultate analize za enega od šestih izbranih testnih primerov (t.j. ID 421577), medtem ko bomo zaradi omejitve dolžine prispevka za ostalih pet testnih primerov prikazali le sumarne rezultate.

3.1. Podrobnejši rezultati analize za testni primer ID 421577

ID 421577 je od začetka uporabe LEAP 1.1.2021 vseboval 126 ponovitev PN: od 18. do 144. ponovitve PN. Torej je bil izdelek pred začetkom uporabe mikroplaniranja z ML relativno »mlad«, z le 17 ponovitvami PN. Po uporabi 1. kriterija je bilo izmed 126 zapisov izločenih 61 neveljavnih zapisov. Tretji kvartil za izračunane vrednosti ΔLEAP tega ID je 0.532, kritična vrednost za izločitev po kriteriju 2 pa je 0.32. Po uporabi 2. kriterija je bilo tako izločenih še 17 zapisov, ki so predstavljali velika odstopanja, po uporabi 3. kriterija pa je bilo za ublažitev vliva ekstremnih vrednosti dodatno zglajenih še 12 zapisov.

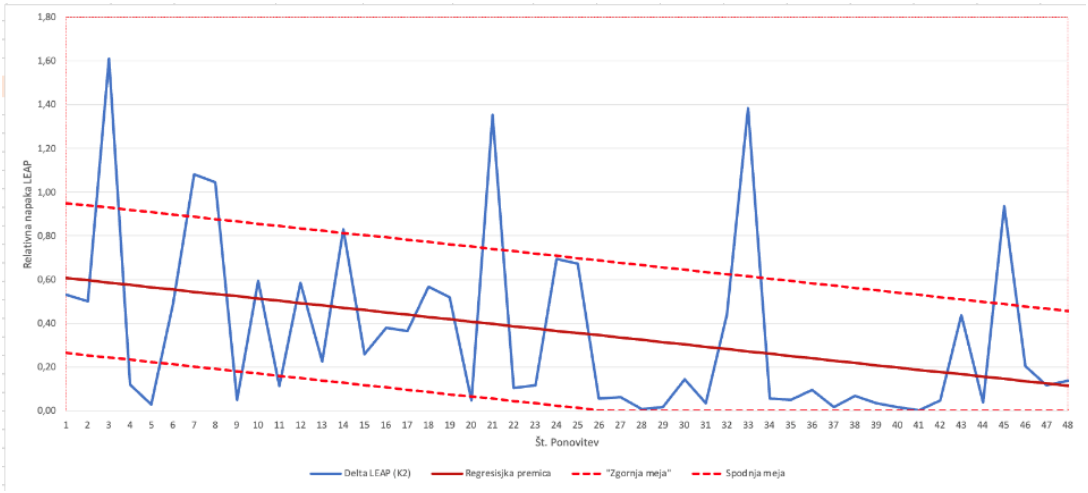
V nadaljevanju prikazujejo odvisnost relativne napake ΔLEAP za ID 421577 od števila ponovitev PN, glede na tri-stopenjski postopek čiščenja podatkov. Slika 3 prikazuje odvisnost relativne napake ΔLEAP od števila ponovitev PN za vzorec podatkov, ki je bil prečiščen skladno s 1. kriterijem (odstranjeni neveljavni zapisi, vpliv ekstremnih odstopanj še ni izločen). Na podlagi take grafične analize težko potrdimo, da je vrednost relativne napake ΔLEAP kakorkoli povezana s številom ponovitev PN. Mestoma zaznamo zelo velika odstopanja (nad 4000%). Kljub temu izračun Spearmanovega koeficienta korelacije (tabela 1) pokaže zmerno stopnjo negativne povezanosti (-0,284). Dobljena p vrednost testa povezanosti je 0,022<0,05, kar pomeni, da ob 5% stopnji značilnosti testa lahko trdimo, da povezava med izbranima veličinama obstaja že na dejanskih podatkih iz proizvodnje.



Slika 3: Odvisnost relativne napake ΔLEAP od števila ponovitev; podatki prečiščeni skladno s 1.kriterijem. [4]

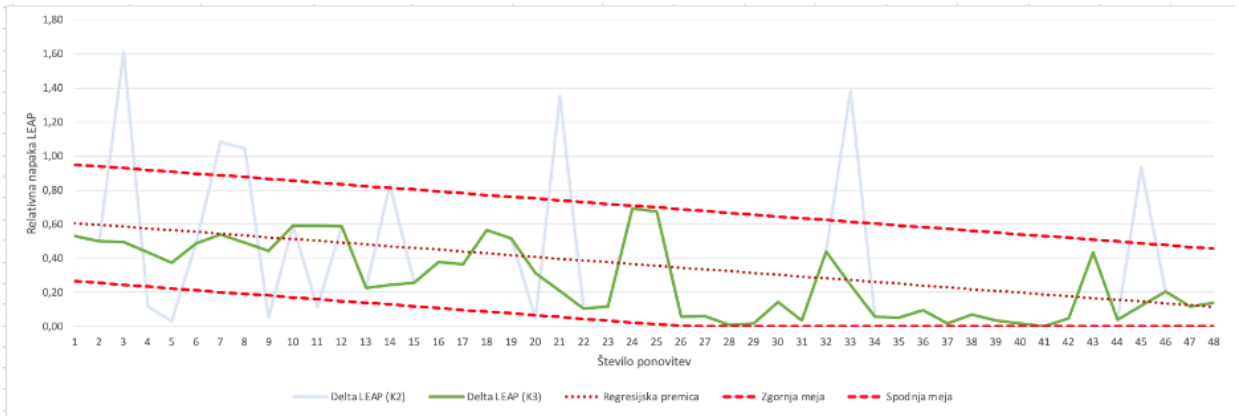
Ko smo zapise podatkov ID 42577 dodatno prečistili z uporabo kriterija 2 (eliminacija ekstremnih odstopanj), je odvisnost relativne napake od števila ponovitev bolj jasno razvidna že iz grafične predstavitve (slika 4). Slednje potrjuje tudi vrednost Spearmanovega koeficienta korelacije (-0,415). Tudi p vrednost testa povezanosti je statistično značilna (0,003<0,05), kar pomeni, da ob eliminaciji zapisov, kjer relativna napaka ΔLEAP ekstremno odstopa, lahko potrdimo, da se napoved trajanja PN z naraščanjem števila ponovitev znatno zmanjšuje.

Iz slike 4 je razvidno, da v vzorcu podatkov še vedno obstajajo velika odstopanja v vrednostih ΔLEAP, ki zahtevajo glajenje. Na sliki 4 so to vrednosti, ki »štrlijo« izven intervala zaupanja linearnega trenda.



Slika 4: Odvisnost relativne napake ΔLEAP od števila ponovitev; podatki prečiščeni skladno s 2.kriterijem. [4]

Da bi izločili vplive teh odstopanj, izvedemo glajenje z uporabo kriterija 3. Grafična predstavitev glajenih podatkov (slika 5) jasno pokaže negativni trend relativne napake ΔLEAP v odvisnosti od števila ponovitev PN. Slednje potrjuje tudi razmeroma visoka in negativna vrednost Spearmanovega koeficienta korelacije (-0,673). Tudi p vrednost testa povezanosti je tudi tokrat statistično značilna (0,0000002<0,05).



Slika 5: Odvisnost relativne napake ΔLEAP od števila ponovitev; podatki prečiščeni skladno s 3.kriterijem. [4]

3.2. Sumarni rezultati analize za izbrane testne primere

Na isti način, kot je predstavljeno v prejšnjem poglavju za testni primer ID 42577, smo izvedlo analize še za ostalih 5 testnih primerov, in sicer: ID 237597, ID 270833, ID 351660, ID 272811 in ID 442122. Sumarni rezultati so predstavljeni v tabeli 1.

Tabela 1: Rezultati statističnih analiz vzorca šestih izbranih testnih primerov

	Kriterij 1	Kriterij 2	Kriterij 3
ID 421577			
Število podatkov (n)	65	48	36+12
Spearmanov koef. korelacije	-0,284	-0,415	-0,673
p vrednost	0,022	0,003	0,0000002
ID 237597			
Število podatkov (n)	60	47	42+5
Spearmanov koef. korelacije	0,101	0,254	0,293
p vrednost	0,442	0,085	0,046
ID 270833			
Število podatkov (n)	46	32	30+2
Spearmanov koef. korelacije	-0,190	-0,253	-0,300
p vrednost	0,207	0,162	0,095
ID 351660			
Število podatkov (n)	44	32	31+1
Spearmanov koef. korelacije	-0,114	-0,426	-0,451
p vrednost	0,461	0,015	0,010
ID 272811			
Število podatkov (n)	40	30	28+2
Spearmanov koef. korelacije	-0,243	-0,470	-0,601
p vrednost	0,132	0,009	0,0004
ID 442122			
Število podatkov (n)	27	18	17+1
Spearmanov koef. korelacije	-0,281	-0,449	-0,756
p vrednost	0,155	0,06	0,0003

4 Diskusija

Rezultati uporabe tri-stopenjskega čiščenja podatkov za izločanje ekstremov v proizvodnji so pokazali, da s tem postopkom lažje statistično dokažemo, da se uporaba ML podjetju izplača pri doseganju želenega cilja: s časom se sistem ML dovolj dobro nauči, da postane pri napovedovanju trajanja PN boljši od planiranja z normativi, kar pomeni, da lahko planerja razbremenimo odločanja pri izbiranju proizvodnih verzij v ERP v procesu mikroplaniranja. Zavedati pa se je potrebno, da smo se postopkom čiščenja podatkov umetno približali razmeram stabilnejše proizvodnje, kjer je razlogov, ki vodijo v odstopanja napovedi pričakovati manj.

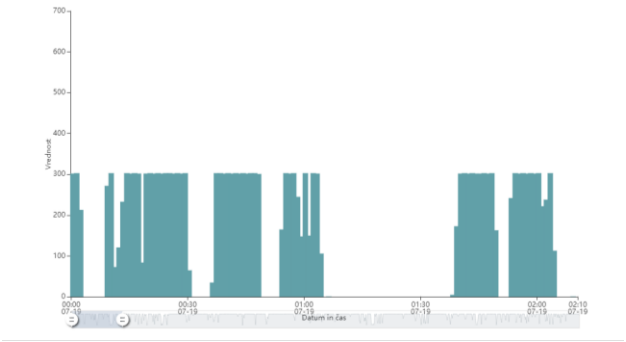
Da bi lahko potrdili, je bil proces čiščenja podatkov izveden korektno in so bili ekstremi izločeni upravičeno (in so posledično statistični zaključki relevantni), je potrebno imeti dokaze iz MES. Potrebovali bi namreč potrditev, da so bili podatki, ki smo jih odstranili ali gladili, resnično vezani na naključne/izredne dogodke v proizvodnem procesu. Upravičenost izločanja posameznih zapisov smo preverjali tako, da smo si pri analizi zapisali izločeno številko PN. Nato smo v bazi sistema MES pregledali pripadajoče zapise o razlogih, ki bi lahko vplivali ali na močno predčasen zaključek ali močno podaljšano trajanje PN, (npr. ali za izločen PN obstaja poročilo o zastoju na stroju, kot kaže primer na sliki 6). *Ugotovili smo, da proizvodnja v proučevanem podjetju zastojev ne beleži dovolj natančno in sistematično* (ne vsi delavci in mojstri na enak način). To je zelo pomembna ugotovitev za potrditev ustreznosti/neustreznosti izločanja ekstremov. Nenazadnje, učinkovit algoritem ML bi moral znati take naključne/izredne dogodke pri nadaljnjih napovedih upoštevati. Da ML lahko spremlja zakonitosti pojavljanja takih dogodkov in to uporabi pri napovedovanju trajanja PN, mora imeti natančen in standardiziran razlog zastoja (koda zastoja). V proučevanem proizvodnem okolju je izbira vrste zastoja prepuščena subjektivni presoji delavca, ko v MES opredeli, zakaj stroj stoji.

1	Stroj	Zastoj	Tip zast.	Podstroj	Začetek	Odprava	Dolž. zast	Dolž. vzdr	Opis
2	AS01	Stroj; Fine nastavitve	NEP	AS01	31.05.2024 23:38	01.06.2024 00:00	0,35	0,00	
3	AS02	Stroj; Fine nastavitve	NEP	AS02	31.05.2024 23:26	01.06.2024 00:00	0,55	0,00	zmeden pomik
5	AS03	Stroj; ZastZaradiDrugStroja	NEP	AS03	31.05.2024 23:10	31.05.2024 23:20	0,18	0,00	
6	AS01	Stroj; ZastZaradiDrugStroja	NEP	AS01	31.05.2024 22:22	31.05.2024 22:39	0,28	0,00	
7	AS01	Stroj; ZastZaradiDrugStroja	NEP	AS01	31.05.2024 22:02	31.05.2024 22:10	0,13	0,00	
48	AS13	Orodje; LomOrodja-Izpenjanje	NEP	AS13	31.05.2024 22:00	01.06.2024 00:58	2,98	0,00	Zastoj od:31.05.24 17:31:25; 1.red zabiti izmetači.
49	BSD 13	Orodje; LomOrodja-Izpenjanje	NEP	BSD 13	31.05.2024 22:00	01.06.2024 06:00	8,00	0,00	Zastoj od:31.05.24 13:48:35; neenakomerno brizganje
50	BSD 6	Orodje; LomOrodja-Izpenjanje	NEP	BSD 6	31.05.2024 22:00	01.06.2024 06:00	8,00	0,00	Zastoj od:31.05.24 13:43:05;
51	BST-05	Stroj; Fine nastavitve	NEP	BST-05	31.05.2024 22:00	01.06.2024 06:00	8,00	0,00	Zastoj od:31.05.24 12:19:55;
54	AS07	Orodje; LomOrodja-Izpenjanje	NEP	AS07	31.05.2024 22:00	31.05.2024 22:03	0,05	0,00	Zastoj od:31.05.24 20:40:01;
115	AS07	Orodje; LomOrodja-Izpenjanje	NEP	AS07	31.05.2024 20:40	31.05.2024 22:00	1,33	0,00	
129	BST-08	Stroj; Fine nastavitve	NEP	BST-08	31.05.2024 19:34	31.05.2024 19:35	0,03	0,00	
131	AS16	Stroj; Fine nastavitve	NEP	AS16	31.05.2024 18:00	31.05.2024 18:45	0,77	0,00	Zastoj od:31.05.24 16:44:30
132	AS01	Stroj; Fine nastavitve	NEP	AS01	31.05.2024 18:00	31.05.2024 21:58	3,98	0,00	Zastoj od:31.05.24 15:59:11
133	AS13	Orodje; LomOrodja-Izpenjanje	NEP	AS13	31.05.2024 17:31	31.05.2024 22:00	4,48	0,00	1.red zabiti izmetači.
134	AS14	Orodje; LomOrodja-Izpenjanje	NEP	AS14	31.05.2024 17:31	31.05.2024 21:20	3,83	0,00	
137	AS16	Stroj; Fine nastavitve	NEP	AS16	31.05.2024 16:44	31.05.2024 17:30	0,76	0,00	
139	AS16	Stroj; Fine nastavitve	NEP	AS16	31.05.2024 16:41	31.05.2024 16:43	0,04	0,00	
140	AS14	Orodje; LomOrodja-Izpenjanje	NEP	AS14	31.05.2024 16:40	31.05.2024 16:42	0,03	0,00	
141	AS08	Stroj; Fine nastavitve	NEP	AS08	31.05.2024 16:22	31.05.2024 17:28	1,11	0,00	
142	AS14	Orodje; LomOrodja-Izpenjanje	NEP	AS14	31.05.2024 16:18	31.05.2024 16:39	0,35	0,00	
144	AS14	Stroj; Fine nastavitve	NEP	AS14	31.05.2024 16:10	31.05.2024 16:18	0,14	0,00	
145	AS01	Stroj; Fine nastavitve	NEP	AS01	31.05.2024 15:59	31.05.2024 17:30	1,51	0,00	
146	AS02	Stroj; Fine nastavitve	NEP	AS02	31.05.2024 15:37	31.05.2024 17:10	1,55	0,00	
147	AS14	Stroj; Fine nastavitve	NEP	AS14	31.05.2024 14:41	31.05.2024 15:21	0,67	0,00	

Slika 6: Preverjanje časovne usklajenosti izločenih ekstremnih proizvodnih nalogov z zastoji na strojih.

Pri analizi meritev realiziranih PN smo tudi *ugotovili, da je v proučevanem podjetju zabeleženi čas realizacije odvisen od pravočasnega prijavljanja začetka in konca PN s strani delavca preko sistema MES*. Če obstaja v realnosti razlika med dejanskim začetkom/koncem dela na PN in tistim, ki ga je prijavil delavec, to lahko napačno usmeri ML, da načrtuje prihodnje PN dlje ali krajše, kot bi jih sicer, kar lahko vpliva tudi na druge funkcije AI APS, kot je modul za optimalno razporejanja delovnih nalogov. Na podlagi teh ugotovitev predlagamo za izboljšanje merjenje učinka uporabe ML pri mikroplaniranju proizvodnje naslednje ukrepe:

- bolj precizno in pravilno zapisovanje vzrokov o zastojih/prekinitvah v proizvodnem procesu,
- bolj precizno in pravilno zapisovanje začetkov in koncev izdelave PN na strojih,
- če je možno, popolnoma avtomatizirati ugotavljanje in poročanje trajanja zastojev, kar je lažje pri robotiziranih in računalniško krmiljenih strojih (slika 7).



Slika 7: Uporaba avtomatiziranih meritev ciklov delovanja stroja za verifikacijo zastojev v MES in izločanje ekstremnih proizvodnih situacij iz meritev koristi uporabe strojnega učenja pri mikroplaniranju proizvodnje.

Ugotovili smo, da *na pravilnost meritev vpliva tudi človeški dejavnik, planer, in s tem vpliva, da podatki o normativnem trajanju PN niso pravilni*: planer »iz izkustvene baze preteklih novitet« s subjektivno metodo analogije oceni realno trajanje v proizvodnji. Pri tem se posvetuje tudi z mojstrom, ki sprejema noviteto v svojo proizvodnjo. Nato na podlagi »mnenja« korigira predlagan izračun trajanja PN iz ERP na samem PN (časovni blažilec). Če ni informacije, kateri izračun trajanja PN je bil prilagojen s »človeško inteligenco«, to pomembno vpliva na kvaliteto podatkov, ki vstopajo v meritve, s katerimi dokazujemo, ali ML lahko nadomesti izkušnje planerja pri mikroplaniranju ter pri katerih ponovitvah PN doseže ali preseže planerja.

Pomembna ugotovitev je povezana z naravo pojavljanja dogodkov nezanesljive proizvodnje, ki vplivajo na dejansko trajanje PN: *nekateri dogodki proizvodnji nastopajo naključno (npr. pojav nekvalitetnega materiala, odpoved naročila tekom proizvodnje), nekateri imajo svoje funkcijske zakonitosti (npr. odpovedi orodja, stroja)*, kar pomeni, da jih je lažje bolj precizno napovedati. Če dostopa do teh podatkov ML nima, ne moremo pričakovati, da se ponovna ekstremna odstopanja med napovedjo trajanja PN z ML in dejansko realizacijo PN ne bodo več dogajala.

Ugotovili smo tudi, da pri dejanski proizvodnji na meritve uporabe ML vpliva tudi *kvaliteta vhodnih podatkov o realizaciji*, npr. zakaj se je ob enakem naključnem dogodku, ki se je ponovil, sprejelo različne ukrepe, tega iz obstoječih podatkov ne moremo ločiti (enak vzrok, npr. odpoved orodja lahko vpliva kot predčasen zaključek PN ali kot podaljšan čas izdelave PN; odločitev sprejme subjektivno proizvodnja skupaj z vzdrževanjem glede na ocenjeno težavo, npr. ali je na zalogi rezervni del).

5 Zaključek

Ugotovili smo, da je v proučevanjem proizvodnem sistemu zelo veliko nezanesljivih podatkov, ki jih potrebuje ML za dobro predvidevanje trajanja PN. Zato je težko izmeriti koristi uporabe nove tehnologije ML za predvidevanje trajanja PN, če ne posežemo tudi v prenovo pridobivanja podatkov o realizaciji proizvodnih procesov, kot je zajem točnih razlogov iz proizvodnega procesa o njegovi nezanesljivosti, doslednost poročanja ipd..

Z vpeljavo postopka tri-stopenjskega čiščenja izvornih podatkov proizvodnje smo opazili koristi AI APS LEAP pri predvidevanju trajanja PN, vendar jih zaradi vzorca le šestih izdelkov ne moremo posplošiti na celotno populacijo aktivnih izdelkov. Analiza meritev na prečiščenih podatkih nakazuje, da bolj kot so razmere stabilne (bolj kot so podatki homogeni), bolj učinkovito je lahko učenje ML. Ugotavljamo, da je metoda čiščenja dovolj enostavna in dobra, da jo v nadaljevanju raziskave lahko apliciramo na reprezentativnem vzorcu, pod pogojem, da bodo izločeni zapisi dodatno validirani s strani podatkov iz MES, ti pa z neposrednimi podatki o delovanju/nedelovanju strojev.

Odkrili smo, da je zahtevna primerjava napovedovanja LEAP s planiranjem planerja z normativi iz ERP, ker tudi podatki o trajanju PN iz normativov vsebujejo »šume«, kot je subjektivno dodajanje časovnih blažilcev. Nimamo zapisov ne kdaj ne zakaj so se planerji tega poslužili. Za zaključek je tu še naključna ali funkcijsko neznano opredeljena pojavnost dogodkov nezanesljive proizvodnje. Bolj kot je proizvodnja nezanesljiva, več bo ekstremnih

odstopanj napovedi z ML od realne proizvodnje in težje bo dokazovanje koristi ML. Po analizi ugotavljamo, da tehnologija ML sicer zna delati z nezanesljivim proizvodnim procesom podobno dobro (slabo) kot človek, a to ne pomeni, da lahko zaradi tega opustimo napore za stabilizacijo proizvodnega procesa, saj bomo na ta način izboljšali tudi točnost predvidevanj trajanja PN z ML.

Literatura

[1] Zhong, R. Y.; Xu, X.; Klotz, E.; Newman, S. T. Intelligent Manufacturing in the context of industry 4.0: A Review. Engineering 2017, 3(5), 616–630.

[2] Xu, L. D., Xu, E. L., Li, L. (2018). Industry 4.0: State of the art and future trends. International Journal of Production Research, 56(8), 2941–2962.

[3] Dellermann, D., Ebel, P., Söllner, M., Leimeister, J. M. (2019). Hybrid intelligence. Business & Information Systems Engineering, 61(5), 637-643.

[4] Vukašin, R. (2024). Učinkovitost algoritmov umetne inteligence pri mikroplaniranju proizvodnje. Diplomsko delo. Univerza v Mariboru, Fakulteta za organizacijske vede.

[5] Bueno, A., Godinho Filho, M., Frank, A. G. (2020). Smart production planning and control in the industry 4.0 context: A systematic literature review. Computers & Industrial Engineering, 149, 106774.

[6] Marzia, S., AlejandroVital-Soto, Azab, A. (2023). Automated Process Planning and dynamic scheduling for Smart Manufacturing: A Systematic Literature Review. Manufacturing Letters, 35, 861–872.

[7] www.domel.com, Informacije o podjetju Domel, obiskano 26. 7. 2024.

[8] www.qlector.com, Informacije o podjetju Qlector in produktu Leap, obiskano 26. 7. 2024.

[9] Roblek, M., Zajec, M., Georgievski, A. (2022). Izzivi uporabe umetne inteligence na področju operativnega planiranja proizvodnje. Sodobni pristopi inženiringa poslovnih sistemov, 1st, 181-216.

[10] Roblek, M. (2022). The use of artifficial intelligence in the operational planning process, the case of Domel. 42th International Conference on Organizational Science Development.

[11] www.kiner.si, Informacije o podjetju SGM in produktu Kiner, obiskano 26. 7. 2024.

Premagovanje izzivov hranjenja podatkov v verigi blokov

Mitja Gradišnik,[1] Daniel Copot,[2] Martin Domajnko,[1] Muhamed Turkanović[1]

[1] Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
mitja.gradisnik@um.si, martin.domajnko@student.um.si, muhamed.turkanovic@um.si
[2] ITC - Inovacijsko tehnološki grozd, Murska Sobota, Slovenija
daniel.copot@itc-cluster.com

Hiter razvoj na področju tehnologij veriženja blokov prinaša številne možnosti inovacij in vpeljave novih poslovnih modelov. Vpeljava hranjenja podatkov v verige blokov v poslovna okolja prinaša predvsem transparentnost podatkov, integriteto, boljšo dostopnost, varnost in možnost decentraliziranega upravljanje podatkov. Vpeljava hranjenja podatkov v verige blogov prinaša s seboj tudi številne inženirske izzive, ki jih je potrebno nasloviti v okviru razvojnega procesa tovrstnih rešitev. Namen prispevka je snovalcem tovrstnih programskih rešitev predstaviti nabor inženirskih pristopov, ki jih je smiselno vzeti v obzir pri analizi, načrtovanju, vrednotenju ali preoblikovanju na verigah blokov temelječih programskih rešitev. V prispevku povzamemo ključne inženirske izzive razvoja tovrstnih informacijskih rešitev ter predstavimo nekatere praktične rešitve za predstavljene izzive. V prispevku izpostavimo izzive in rešitve vpeljave verig blokov na performančne lastnosti in skalabilnost programskih rešitev. V primerjavi z dostopnimi časi podatkovnih baz se izvršitve transakcij in povpraševanj pri obdelavi podatkov zapisanih v verigah blokov soočajo tudi z višjimi latentnimi časi. Ti izhajajo iz razpršenosti podatkov med bloki, omejitve velikosti blokov in konstantnega preverjanje integritete zapisanih podatkov. Nenazadnje je pri arhitekturnem načrtovanju potrebno vzeti v obzir, da so v verige blokov zapisani podatki nespremenljivi, kar pogosto trči ob temeljno potrebo programske rešitve po njeni evoluciji.

Ključne besede:
tehnologije veriženja blokov,
arhitekturne taktike,
kratke oskrbovalne verige
programsko inženirstvo
prilagoditev razvojnih metod

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.10

1 Hramba podatkov v poslovnih okoljih

Podatkovne baze, med katerimi gre v prvi vrsti izpostaviti predvsem relacijske baze, so bile desetletja hrbtenica hranjenja podatkov v poslovnih programskih rešitvah. Podatkovne baze se v osnovi zanašajo na centralizirano arhitekturo, pri čemer so podatki hranjeni v tabelaričnih strukturah s predhodno definiranimi medsebojnimi odvisnostmi [2]. Desetletja razvoja podatkovnih baz so pripeljala do rešitev, ki se ponašajo z nizkimi zakasnitvami, visoko prepustnostjo transakcij in visoko kapaciteto hranjenja podatkov [1]. Vzporedno s tradicionalnimi podatkovnimi bazami so se v zadnjih desetletjih bliskovito razvijale ne-relacijske podatkovne baze (t. i. NoSQL) ter v zadnjem obdobju tudi tehnologije veriženja blokov. Slednje zraven porazdeljenega vpeljejo tudi decentraliziran pristop hrambe podatkov in upravljanja podatkovne baze. Primerjano s tradicionalnimi podatkovnimi bazami tehnologije veriženja blokov ubirajo povsem različen in edinstven pristop k hranjenju podatkov [2]. Razumevanje razlik s tradicionalnimi podatkovnimi bazami je ključno za načrtovanje in vzdrževanje programskih rešitev, ki vpeljujejo uporabo tehnologijo veriženja blokov.

1.1. (Tradicionalne) podatkovne baze

Arhitektura tradicionalnih podatkovnih baz je zasnovana po pristopu odjemalec – strežnik. To velja tako za relacijske kot ne-relacijske podatkovne baze (t. i. NoSQL). Podatki so praviloma hranjeni na centraliziranem strežniku, na katerega odjemalci pošiljajo poizvedbe. Ključna značilnost tradicionalnih podatkovnih baz je sposobnost odjemalcev, da vstavljene podatke kasneje poljubno spreminjajo ali celo izbrišejo [3]. Odjemalci so neposredno različne programske rešitve ter posredno njihovi uporabniki. Upravljanje podatkovne baze ohranja vrhnja avtoriteta, ki je podatkovno bazo vzpostavila. Ta ves čas ohranjajo nadzor nad strukturo in obliko podatkov in odjemalcem podeljuje dostope do upravljanja podatkov (ne tudi strukture) v podatkovni bazi. Podatkovna baza je sicer lahko tudi porazdeljena med številne strežnike oz. vozlišča, vendar je kljub temu ena vrhnja entiteta, ki nadzira vsako posamezno vozlišče [1]. Tradicionalna podatkovna baza ima torej v praksi skoraj vedno skrbnika, ki nad njo ohranja nadzor, vključno z upravljanjem podatkov, spreminjanjem strukture in optimizacijo [4]. Prisotnost takšne vrhnje avtoritete pri upravljanju podatkovne baze tako zahteva zaupanje odjemalcev, da ta podatkov ne spreminja zlonamerno. Pomembna značilnost podatkovnih baze je njihova zaprtost znotraj okvirjev organizacij. Ni običajna praksa, da bi si deležniki podatkovne baze delili in tako soustvarjali podatke.

1.2. Hramba podatkov v verige blokov

Arhitekturna zasnova podatkovne shrambe, ki temelji na tehnologijah veriženja blokov, je poglavitno različna od prej predstavljene arhitekture tradicionalnih podatkovnih baz. Osrednja komponenta tehnologije je t. i. glavna knjiga (angl. ledger), ki predstavlja porazdeljeno podatkovno hrambo v katero se zapisujejo transakcije iz P2P omrežja medsebojno povezanih računalnikov [2]. V veliki večini je osnova za takšno hrambo ravno NoSQL podatkovna baza, kot npr. LevelDB [5]. Iz decentraliziranost in porazdeljenosti glavne knjige izhaja, da vsa vozlišča v omrežju hranijo sinhrone kopije vseh zapisanih podatkov.

Podatki se na vozliščih hranijo v skupkih transakcij imenovanih bloki, ki so enovite velikosti [4], pri čemer v blok zapisana zgoščevalna vrednost predhodnega bloka zagotavlja varnost podatkov pred zlonamernimi spremembami. Poglavitna sprememba v primerjavi s podatkovnim bazami predstavlja način dela s podatki. Verige blokov namreč dovoljujejo izključno vstavljanje in dodajanje podatkov, ne pa tudi njihovega spreminjanja ali brisanja. Podatki zapisani v verige blokov so torej v praksi nespremenljivi [6]. Iz navedene lastnosti izhaja možnost visoke stopnje zaupanja odjemalcev v zapisane podatke, saj je kasnejša manipulacija podatkov izjemno malo verjetna.

Podatki hranjeni v verigah blokov za zapisani v več porazdeljenih vozliščih, pri čemer vsako izmed vozlišč aktivno sodeluje pri upravljanju podatkovne zbirke preko verifikacije vstavljanja novih podatkov. Slednje vozlišča izvedejo preko mehanizma doseganja porazdeljenega soglasja [3]. Ravno mehanizem doseganja soglasja, ki je nujen zaradi

decentraliziranosti omrežja, predstavlja steber varnosti omrežja, ki oteži neželene posege v podatke zlonamernih vozlišč. Ker so podatki podvojeno zapisani na več fizičnih lokacijah, zapisi ne pripadajo posamezni entiteti v omrežju temveč so last vseh deležnikov.

Izpostaviti je potrebno tudi ključno dejstvo, da koncept tehnologije veriženja blokov pade v vodo, če nimamo decentraliziranega omrežja, saj bi v nasprotnem primeru imeli zgolj porazdeljeno podatkovno bazo, katere vozlišča so pod nadzorom ene vrhnje avtoritete. Četudi nesmiselna se takšna omrežja verig blokov v literaturi imenujejo zasebna omrežja. Takšna omrežja ohranjajo vso kompleksnost tehnologije veriženja blokov, ki stremi k zagotavljanju varnosti, transparentnosti in nespremenljivosti, pri čemer pa se vse to lahko preprosto zaobide s strani vrhnje avtoritete. Ne glede na to pa obstajajo t. i. zasebna omrežja verig blokov, kjer pa vozlišča niso pod nadzorom ene vrhnje avtoritete, temveč več različnih. V tem primeru imamo decentralizirano omrežje, ki pa ni javno dostopno, temveč je pod nadzorom peščice entitet, katere same vzdržujejo omrežje in definirajo pravila igre ter s tem tudi način doseganja porazdeljenega soglasja. Takšna omrežja imenujemo konzorcijska in se največkrat pojavijo v informacijskih rešitvah. Primarni tip omrežij verig blokov pa so t. i. javna omrežja brez dovoljenja (angl. permissionless), ki predstavljajo omrežja v katero so lahko pridruži kdor koli in s tem postane del številnih decentraliziranih vozlišč, ki skupaj vzdržujejo omrežje ter skupaj dosegajo porazdeljeno soglasje glede vsebine glavne knjige.

Avtonomnost verig blokov naredi korak naprej z vpeljavo pametnih pogodb, ki na podlagi soglasja med vozlišči in vnaprej kodirane poslovne logike avtonomno izvedejo vstavljanje zapisov v verige blokov [7]. Slednje daje dodatne možnosti za inovativne programske rešitve. Vpeljava tehnologij veriženja blokov v informacijske rešitve torej vpelje podatkovno shrambo, ki je robustna iz vidika centralne točke odpovedi in prav tako relativno dobro odporna na zlonamerno spreminjanje zapisanih podatkov [4], [7].

Hramba podatkov v verige blokov se od hrambe v podatkovne baze loči po štirih ključnih parametrih. In sicer dosežena stopnja podatkovne integritete, izvajanje transakcij, učinkovitost povpraševanj in strukture. Od navedenih razlik je odvisno, kdaj je kateri izmed primerjanih pristopov primernejši za integracijo v informacijsko rešitev. Razlike med primerjanima pristopoma hrambe podatkov prikazuje Slika [8]. Pri podatkovni integriteti pa je potrebno izpostaviti še dejstvo, da se v verigah blokov vsaka posamezna transakcija pred procesiranjem digitalno podpiše s strani končnih uporabnikov informacijskih rešitev, za razliko od tradicionalnih podatkovnih baz, kjer tega nismo vajeni, saj so odjemalci tisti, ki na zahtevo svojih uporabnikov podatke pripravijo in posredujejo do strežnika podatkovne baze. S tega vidika lahko izpostavimo tudi, da so v primeru verig blokov odjemalci praviloma digitalne denarnice končnih uporabnikov, ki lahko digitalno podpisane transakcije prožijo neposredno na katerokoli vozlišče omrežja verig blokov.

Slika 1: Primerjava verig blokov in podatkovnih baz.

1.3. Lastnosti tehnologij veriženja blokov

Prepoznane prednosti in lastnosti tehnologij veriženja bloka so gonilo potrebe po njihovi integraciji v sodobne informacijske rešitve. Čeprav gre v osnovi za podatkovne shrambe, so lastnosti verig blokov takšne, da jim dajejo edinstvenost in nezamenljivost z drugimi pristopi hrambe podatkov. V praksi to pomeni, da bi s tradicionalnimi podatkovnimi bazami izredno težko, če ne že nemogoče, realizirali številne poslovne scenarije, ki jih tehnologije veriženja blokov omogočajo. Poglavitne lastnosti verig blokov so nespremenljivost zapisov, transparentnost podatkov, decentraliziranost, nadzor deležnikov nad lastnimi zapisi in varnost zapisanih podatkov [9].

1.3.1. Nespremenljivost zapisov

Verige blokov omogočajo izključno vstavljanje oz. pripenjanje novih podatkov. Ker manipulacija že zapisanih podatkov ni podprta, bi vsak poskus spreminjanja ob sprejetih varnostnih mehanizmih izredno težko izpeljali. Za slednje poskrbijo zapisane zgoščevalne vrednosti posameznega bloka, ki poskrbijo za integriteto in nespremenljivost zapisov celotne podatkovne zbirke in mehanizmi sprejemanja soglasja pri zapisovanju blokov. Nepremenljivost zapisanih podatkov postavlja temelj zaupanju deležnikov v zapise, saj je enkrat zapisane podatke na nivoju verig blokov nemogoče manipulirati ali kasneje zanikati njihov obstoj. Prav tako lahko vsak deležnik v vsakem trenutku preveri integriteto zapisanih podatkov.

1.3.2. Transparentnost podatkov

Za razliko od centraliziranih podatkovnih baz poseduje pri verigah blokov vsako izmed vozlišč omrežja popolno kopijo celotne podatkovne zbirke. Te podatkovne zbirke prav tako niso zamejene znotraj organizacij, temveč so med deležniki deljene. Slednje daje kateremukoli soudeležencu v omrežju vpogled v zapise vseh udeležencev v omrežju, kar omogoča učinkovito soustvarjanje zapisov med deležniki. Interakcijo med udeleženci je torej neposredna, prav tako omogoča hitro razreševanje nejasnosti povezanih s transakcijami v omrežju [10]. Možnost soustvarjanja podatkov odpira dodatne možnosti postavitve temeljev za nova medorganizacijska sodelovanja ter vzpostavitve skupnosti deležnikov, ki delajo na skupnih podatkih.

1.3.3. Decentraliziranost

Decentraliziranost predstavlja eno izmed ključnih lastnosti podatkovnih zbirk, ki temeljijo na tehnologijah veriženja blokov. V praksi decentraliziranost pomeni odsotnost vrhnje avtoritete, ki bi upravljala podatkovno zbirko. Nadzor nad podatkovnimi zbirkami zapisanih v verige blokov je tako porazdeljena med vsa vozlišča omrežja in se uveljavlja preko izbranega mehanizma soglasja. Odsotnost osredne avtoritete nadzora omogoča dinamičnost pri dodajanju in odstranjevanju vozlišč v omrežju, ki sodelujejo pri soustvarjanju zapisov brez potrebe po centralizirani administraciji. Navedena lastnost je temelj ustvarjanju skupnosti akterjev, ki sodelujejo v skupnih procesih, brez potrebe, da bi si ti med seboj zaupali.

1.3.4. Zapisi nadzorovani s strani lastnikov

Ker verige blokov ne poznajo administracije podatkovne zbirke in upravljanja dostopov do nje, je upravljanje zapisov prepuščeno lastnikom zapisov samim. V praksi to pomeni, da niti upravljalcem vozlišč posegi v podatke na vozlišču niso dovoljeni. Lastništvo v celoti pripada zapisovalcem, ki pa lahko lastništvo prenese na katerega izmed preostalih deležnikov v omrežju. Navedena lastnost daje temelje ustvarjanju digitalnih sredstev (ang. asset), ki jih je mogoče prosto prenašati med deležniki. Pojem digitalnih sredstev je v tem kontekstu razumljen kot digitalna reprezentacija bodisi realnih bodisi virtualnih dobrih, lastništvo katerih je med akterji prenosljivo. Primeri takšnih dobrin so digitalni denar, kmetijski pridelki ali zdravila. Seveda imajo možnost prenosov sredstev zgolj njihovi lastniki, ki lastništvo izkažejo z ustreznim naborom privatnih ključev.

1.3.5. Varnost in zasebnost podatkov

Uporaba ustreznih kriptografskih algoritmov poskrbi, da so podatkovne zbirke na visokem nivoju zaščitene pred nepooblaščenimi dostopi ali celo nepooblaščenimi spremembami. Navedene lastnost dela podatkovne zbirke zapisane v verige blokov primerne tudi za programske rešitve, ki terjajo visoko stopnjo varnosti. Čeprav so verige blokov v veliki večini primerov javne oz. dostopne širšemu krogu deležnikov, je mogoče z ustreznimi kriptografskimi metodami doseči popolno zasebnost zapisov.

2 Vpliv tehnologij veriženja blokov na inovacije

Lastnosti informacijskih rešitev, ki temeljijo na tehnologijah veriženja blokov, odpirajo številne možnosti za inovacije in nove poslovne modele, ki izkoriščajo nespremenljivost in preverljivost zapisanih podatkov v skupni podatkovni shrambi. Vpeljava tehnologij veriženja blokov v organizacije tako ustvarja okolja, v katerih je različnim deležnikom omogočeno varno soustvarjanje skupnih podatkov, brez potrebe po revizijah in avtorizacijah s strani tretjih oseb [11]. Tehnologije veriženja blokov morda primarno povezujemo s finančnim sektorjem, ki na tem področju ponuja rešitve za P2P plačilne sisteme, P2P posojila in sisteme za prenos premoženja [12]. Ob boku finančnega sektorja so tehnologij veriženja blokov dosegle pomembno uveljavitev na področjih zavarovalništva, zdravstvenega varstva, izobraževanja, upravljanja, upravljanja z intelektualno lastnino in oskrbovalnih verig [1], [10]. Tehnologije veriženja blokov so pomemben katalizator inovacij v domenah, ki so sposobne v svojih poslovnih procesih izkoristiti prednosti nespremenljivosti, sledljivosti, transparentnosti, varnosti in robustnosti skupnih podatkov [11].

2.1. Oskrbovalne verige

Pomembna domena, na katerega so tehnologije veriženja blokov naredile v zadnjih letih pomemben vpliv, je domena oskrbovalnih verig. Na področju oskrbovalnih verig je potrebno izpostaviti predvsem vpeljavo rešitev za sledenje zdravil v oskrbovalnih verigah, sledenje produktom z višjo vrednostjo in sledenju živilskih izdelkom tekom prehranske verige [1], [13], [14], [15]. Sledenje produktov v oskrbovalnih veriga običajno poteka od proizvodnje, preko distributerjev in posrednikov, do končnih kupcev. Neglede na vrsto produktov, ki se prenašajo po oskrbovalni verigi, npr. zdravila, izdelki z višjo vrednostjo ali prehranski izdelki, je vsem rešitvam skupno to, da izkoriščajo možnost necentraliziranega sodelovanja različnih deležnikov oskrbovalnih verig, da ustvarijo transparentne, sledljive, varne in preverljive zapise o stanju produktov v posameznem členu oskrbovalne verige. V praksi to pomeni, da deležniki v oskrbovalnih verigah ustvarijo skupnost, znotraj katere soustvarjajo zapise o kakovosti produktov z namenom ustvarjanja zaupanja končnih potrošnikov v deklarirano kakovost produkta. Z inovacijami v programskih rešitvah, podprtimi s tehnologijami veriženja blokov, se cilja na zmanjševanje poneverbe, zavajanj in drugih zlorab zaupanja povezanih s kakovostjo produkta povezanih trditev.

2.2. Predstavitev prototipa

Kot primer vzamemo prototip sledenja lokalnim pridelkom v kratkih oskrbovalni verigah. Prototip programske rešitve se imenuje BlockIS in se razvija v partnerstvu Zelene točke, ITC Murska sobota in Blockchain Lab:UM. Rešitev je prilagojena specifičnim zahtevam pridelovalcev oskrbe z lokalnim sadjem in zelenjavo. Rešitev povezuje različne akterje v oskrbovalne verige, pri čemer so v ospredju pridelovalci lokalnega sadja in zelenjave, posredniki s svojim prodajnim mestom in potrošnikov pridelkov. Osnovni namen rešitve je, da pridelovalci s sledenjem in sprotnim beleženjem procesa pridelave pridelkov kupcu na transparenten in preverljiv način ponudijo dokaz o trditvah poveznih z lokalnostjo in načinom pridelave ponujenih pridelkov. Kupcu so tako na voljo vsi podatki o natančni lokaciji pridelave, metodah pridelave ter času, ki je minil med pobiranjem pridelkov in časom nakupa. Vse od navedenega ima znaten vpliv na kakovost pridelkov. Temeljni namen sledenje je, kljub dodatnem delu in

povišanim stroškom, krepitev zaupanja potrošnikov v ponujene izdelke. Potrošniki do zapisov o izbranem pridelku dostopajo s preprostim odčitavanjem QR kode postavljenim ob pridelku na prodajnem mestu posrednika pridelkov. Predstavljen prototip rešitve za podporo prehranskim oskrbovalnim verigam grajen s tehnologijami veriženja blokov, je trenutno v aktivni uporabi na območju Prekmurja. Prototipna rešitev ima torej aktivne pridelovalce, ki sledijo pridelavi svoje zelenjave in posrednika, ki na prodajnem mestu omogoča dostop do zapisov o pridelkih preko generiranih QR kod. Prototip temelji na rešitvi Hyperledger Besu [16], ki transakcije izvršuje v konzorcijskem omrežju verig blokov DIH AGRIFOOD-a.

3 Arhitekturni izzivi in z njimi povezani pristopi

Čeprav gre v osnovi za podatkovne shrambe, se po svoji zasnovi, načinu uporabe in vplivu na atribute kakovosti programskih rešitev ločijo od v industriji dodobra uveljavljenih podatkovnih baz ali datotečnih sistemov. Posledično so pri izgradnji informacijskih rešitev, ki vključujejo tehnologije veriženja blokov, potrebni prilagojeni inženirski pristopi. K izgradnji tovrstnih programskih rešitev je potrebno torej pristopiti premišljeni ob razumevanju konceptov tehnologij veriženja blokov ter njihovega vpliva na atribute kakovosti programske rešitve. Večdesetletni razvoj podatkovni baz je rezultiral v programske komponente, ki jih odlikuje visoka prepustnost, nizke zakasnitve in učinkovitost povpraševanja po podatkih. Podatkovne baze tako v programskih rešitvah nudijo hiter in učinkoviti mehanizem hranjenja podatkov, s katerim je mogoče graditi hitre, skalabilne in visoko propustne programske rešitve [1], ki jih je mogoče uporabiti praktično v vseh scenarijih poslovnih programskih rešitev. Hranjenje podatkov v verige blokov je zaradi znatno drugačnega pristopa k zapisovanju podatkov, drugačno od zapisovanja v podatkovnih baz. Pri vpeljavi podatkovnih shramb, ki temeljijo na tehnologijah blokov v poslovne informacijske rešitev, je tako potrebno biti pozoren na nekatere značilnosti, ki jih tovrstne podatkovne shrambe izkazujejo. Ključne značilnosti na verigah blokov temelječih podatkovnih shramb v poslovnih okoljih so predstavljene v nadaljevanju, pri čemer za vsako podamo tudi praktično rešitev. Izpostavljamo tudi, da so navedene rešitve omejene na spekter in zmožnosti razvijalca informacijskih rešitev, ki zgolj uporablja tehnologijo veriženja blokov.

3.1. Skalabilnost in zmogljivost

Skalabilnost se tesno navezuje na metriko, ki meri število transakcij, ki jih lahko omrežje obdela v eni sekundi. Skalabilne poslovne programske rešitve so se čez čas uporabe rešitve zmožne učinkovito prilagajati vedno večji frekvenci transakcij, brez da bi prepustnost obdelovanja transakcij postalo ozko grlo. Primerjave iz domene plačilnega prometa s kripto valutami kažejo [17], da je skalabilnost omrežij verig blokov prepoznana kot resna omejitev, ki jo je pri načrtovanju na verigah blokov temelječih programskih rešitev potrebno nasloviti z ustreznimi arhitekturnimi rešitvami. Količina izvršenih transakcij na sekundo je pri zapisovanje v verige blokov nekajkrat nižja kot pri zapisovanju v podatkovne baze. Povečevanje števila transakcij bi lahko tako pripeljalo do ozkega grla.

3.1.1. Rešitev

Zaradi omejitev pri prepustnosti transakcij je pri zapisovanju podatkov v verige blokov potreben tehten premislek, katere funkcionalnosti programske rešitve bodo hranile podatke v verige blokov ter katere v klasične podatkovne baze [10]. V praksi se izkaže, da vsi podatki, ki jih programska rešitev obdeluje, niso vedno relevantni za ostale deležnike vpletene v poslovne procese in zadevajo zgolj organizacijo samo. Verige blokov torej nikakor ne nadomeščajo obstoječih ERP sistemov in drugih internih podatkovnih baz organizacije, temveč jih zgolj dopolnjujejo. V verige blokov torej sodijo le podatki, ki jih želi organizacija deliti zunanjimi deležniki in so zanje relevantni.

Kot morebitna rešitev za večjo skalabilnost omrežja verig blokov je uporaba konzorcijskega omrežja, ki uporablja preprostejši mehanizem soglasja (npr. dokaz o avtoriteti – PoA). V takšnih omrežij se prepustnost transakcij

izredno poveča, vendar je splošna varnost nižja, saj imamo znatno manjše število vozlišč, ki so posledično lahko hitreje okno napada na celotno omrežje. Uporaba takšnega omrežja pa je seveda pogojena tudi s tem, da za poslovno rešitev niso ključna javno dostopna digitalna sredstva oz. kriptovalute ali kriptožetoni.
V primerih, ko smo odvisni od uporabe javnih omrežij verig blokov, kot so BitCoin, Ethereum in podobni, pa je seveda smotrn razmislek o tem, katero izmed teh omrežij je za naš poslovni primer smiselno uporabiti, saj ima vsako od teh omrežij svoje prednosti in slabosti ter tudi različne prepustnosti. Glede na izbiro omrežja se nam nato ponujajo tudi njihove specifične rešitve drugega sloja (angl. Layer-2), kot so kanali stanja (angl. state channels), stranske verige (angl. side-channels) itn.

3.2. Velikost zapisanih podatkov

Pri podatkovnih bazah je povsem običajno, da je vanje mogoče dokaj enostavno hraniti tudi zapise večjega formata, kot so obsežnejši dokumenti, slike in videoposnetki. Verige blokov imajo omejeno kapaciteto hranjenja podatkov [10]. Slednje izhaja iz dejstva, da vsi deležniki posedujejo zgodovino vseh transakcij vseh deležnikov v omrežju. Pri zapisovanju podatkov v verige blokov so omejitve povezane s količino zapisanih podatkov posledično bolj stroge. Prva omejitev izvira iz velikosti samega bloka v verigi, ki je velikostno omejen iz zamejuje možnosti hranjenja podatkov. Druga omejitev je posredna in izhaja iz stroškov zapisovanja velike količine podatkov, v primeru, da se za hrambo podatkov uporabi javno omrežje.

3.2.1. Rešitev

Omejitev glede velikosti podatkov, ki jih je mogoče zapisati v verigo blokov, je mogoče zaobiti z vpeljavo ločene shrambe dosegljive ostalim akterjem v omrežju. V teh primerih se praviloma uporabi porazdeljena shramba datotek (angl. distributed file storage system). Za vse podatke zapisane v ločeni shrambi se izračunajo zgoščevalne vrednosti, ki se namesto podatkov samih zapišejo v verigo blokov. Ker je takšna porazdeljena shramba praviloma javna, je drugim deležnikom v omrežju tako omogočeno, da lahko za vse zapise preverijo, ali so bili po zapisu morda spremenjeni. Takšno arhitekturno rešitev je mogoče doseči z vpeljavo IPFS omrežij [18], na katere se hranijo datoteke, slike, videoposnetki in ostali zapisi, ki presegajo velikostne omejitve blokov. Osnovna prednost IPFS omrežja, je da ohranja decentralizirano naravo tehnologij veriženja blokov, saj ne vpeljuje potrebe po centralizirani administraciji zapisov. Dodatna prednost IPFS omrežja je tudi to, da so hranjene datoteke preko zgoščevalne vrednosti tudi naslovljive.
Poleg uporabe IPFS (InterPlanetary File System) za shranjevanje velikih datotek, slik in videoposnetkov zunaj verige blokov, obstaja še nekaj drugih rešitev za obvladovanje omejitev velikosti zapisanih podatkov v verigah blokov. Podobno kot pri IPFS, je mogoče uporabiti različne decentralizirane ali centralizirane rešitve za shranjevanje velikih podatkov zunaj verige blokov, medtem ko se v verigi zapišejo samo zgoščevalne vrednosti (hashi). Tako lahko uporabite obstoječe storitve v oblaku, kot so Amazon S3, Google Cloud Storage, ali decentralizirane rešitve, kot so Swarm, Storj, Filecoin itn. Razen tega pa so nam na voljo še drugi kompleksnejši pristopi, kot je uporaba tehnik kompresije, uporaba ZK-SNARKs, plazemske tehnike, Merklovih dreves itn.
Vsaka od teh rešitev ponuja različne prednosti in omejitve glede na specifične potrebe poslovnih procesov in naravo podatkov, ki jih je potrebno shranjevati. Pomembno je, da se izbere ustrezna kombinacija teh pristopov glede na zahteve sistema in obstoječo infrastrukturo.

3.3. Stroški transakcij

Naslednja posebnosti vredna upoštevanja pri načrtovanju tovrstnih programskih rešitev in je neposredno povezana s količino zapisanih podatkov, so stroški zapisovanja podatkov. Javna omrežja verig blokov veljajo za prostorsko in energetsko potratne. Slednje ima neposredni vpliv na stroške zapisovanja in hrambe podatkov. Slednje velja predvsem, kadar je za hrambo podatkov izbrano javno omrežje. Gre torej za iskanje ravnotežja zaupanjem, varnostjo in transparentnostjo, ki jih nudi zapisovanje podatkov v javno omrežje, in stroški, ki jih zapisovanje podatkov v javna omrežja prinaša.

3.3.1. Rešitev

Zaupanje, transparentnost in varnost zapisanih podatkov pogosto ne odtehta stroškov, ki jih s seboj prinese uporaba javnih omrežij. Javna omrežja zaradi velikosti omrežja veljajo za bolj varna, saj je za izvedbo uspešnega napad nanje potrebnih ogromno virov. V mnogih poslovnih primerih tako visoka stopnja zaupanja in varnosti ni prioriteta. V takšnem primeru gre za razmisliti o uporabi konzorcijskih omrežij, pri katerih deležniki vzpostavijo lastno infrastrukturo verig blokov. Ker so pri konzorcijskih omrežjih vozlišča porazdeljena med deležnike, ki prav tako posedujejo kopijo vseh zapisanih podatkov ter morajo pri zapisovanju v bloke doseči konsenz, veljajo tovrstne konfiguracije prav tako za varne in zaupanja vredne, pri čemer so običajno stroški transakcij občutno nižji.
Podobno kot pri rešitvah za obvladovanje velikosti, se tudi za obvladovanje stroškov ponujajo rešitve v uporabi drugega sloja (Layer-2). Layer 2 rešitve omogočajo izvajanje transakcij zunaj glavne verige blokov. To zmanjšuje število transakcij, ki se morajo zapisati na verigo, kar posledično znižuje stroške transakcij. Prav tako je možno združevati transakcije brez uporabe rešitev drugega sloja. Z združevanjem več transakcij v eno samo se zmanjša število potrebnih zapisov na verigi blokov. S tem pristopom se lahko optimizirajo stroški, saj en zapis na verigi predstavlja več transakcij, kar znižuje stroške na posamezno transakcijo. Transakcije lahko tudi stiskamo pred izvedbo. Stiskanje podatkov pred zapisovanjem v verigo blokov lahko zmanjša velikost transakcij, kar posledično zniža stroške, saj so stroški zapisovanja pogosto odvisni od količine podatkov, ki se zapisujejo. Ker je večina poslovnih rešitev povezanih s tehnologijo veriženja blokov odvisna od pametnih pogodb, je tudi te smotrno optimizirati. Optimizacija kode pametnih pogodb, da je bolj učinkovita in zahteva manjšo porabo računalniških virov, lahko zmanjša stroške transakcij. To vključuje zmanjšanje števila nepotrebnih operacij in uporabo bolj učinkovitih algoritmov.
Z uporabo zgoraj omenjenih rešitev lahko podjetja in organizacije zmanjšajo stroške zapisovanja podatkov v verige blokov, hkrati pa ohranijo ustrezno raven varnosti, zaupanja in transparentnosti. Vsekakor pa vsaka od predstavljenih rešitev doprinese k že tako visoki kompleksnosti uporabe tehnologije veriženja blokov. Ključno je najti pravo ravnovesje med stroški in zahtevanimi lastnostmi sistema glede na specifične poslovne potrebe.

3.4. Učinkovitost povpraševanja po podatkih

V primerjavi s podatkovnimi bazami, po katerih je zaradi dobre strukturiranosti podatkov mogoče izjemno učinkovito in hitro povpraševati, je pri verigah blokov povpraševanje občutno bolj zamudno. Slednje izhaja iz dveh lastnosti verig blokov, slabše strukturiranosti zapisovanja podatkov ter mehanizmov varnosti, ki sproti preverjajo integriteto zapisanih podatkov v blokih preko preverjanja zgoščevalnih vrednosti blokov. Varnost zapisanih podatkov pred manipulacijami ima torej tudi svojo ceno, ki jo je potrebno ustrezno nasloviti. Učinkovitost povpraševanja ovira tudi sama struktura zapisov, pri kateri so zapisani podatki razpršeni po različnih blokih v verigah in naslovljivi izključno preko naslovov. Pomembna pomanjkljivost verig blokov je tudi odsotnost standardiziranih povpraševalnih jezikov, kot je na primer pri relacijskih bazah SQL.

3.4.1. Rešitev

Ker imajo deležniki nadzor le nad podatki znotraj lastne organizacije, je razpršenost zapisanih podatkov po verigi blokov je mogoče premostiti z indeksiranjem zapisov v verigah blokov. Indeks zapisov deluje kot predpomnilnik, ki združuje zapise vseh deležnikov v omrežju. S tem lahko pridemo do dobro strukturiranih podatkov, po katerih je mogoče hitro in učinkovito poizvedovati. Platforme, kot so Apollo GraphQL [19] vzpostavijo infrastrukturo, ki omogoča integriran pogled in enostavni dostop z možnostjo poganjanja kompleksnih poizvedb nad integriranimi podatki. GraphQL dodatno vpelje poenoteno povpraševanje po podatkih s pomočjo poizvedovalnih nizov.
Do zapisov v verigah blokov pogosto dostopamo preko proženja pametnih pogodb. Da bi zamejili kompleksnost proženja pametnih pogodb, je vmesnike dostopa do pametnih pogodb smiselno zagraditi v namensko mikrostoritev, ki dostop do podatkov izpostavlja preko končnih točk, s katerimi si je mogoče podatke izmenjevati preko sporočil v JSON formatu.
Prav tako zapisovanje pomembnih dogodkov in sprememb v podatkih v dnevniške zapise (logs) omogoča hitrejši dostop do ključnih informacij brez potrebe po celovitem iskanju po verigi blokov. Dnevniški zapisi omogočajo hitro povpraševanje po specifičnih dogodkih ali spremembah. Uporaba predpomnilnika (cache) na nivoju aplikacije ali omrežja za pogosto dostopane podatke lahko znatno izboljša hitrost povpraševanja. Predpomnjenje omogoča hitrejši dostop do podatkov brez potrebe po ponovnem iskanju ali preverjanju celovitosti zapisov v verigi blokov.

3.5. Evolucija programskih rešitev

Elementarna lastnost programskih rešitev je, da se s tokom časa pojavljale potrebe po njenih spremembah. Potreba po spremembah izhaja iz različnih dejavnikov, na primer spremembe v zahtevah uporabnikov ali poslovnem okolju. Po drugi strani je temeljna prednost tehnologij veriženja blokov, nespremenljivost zapisanih podatkov, saj je nove podatke na najnižjem nivoju dovoljeno le pripenjati ne pa tudi spreminjati ali celo brisati. Če je pri relacijskih bazah spreminjanje tabel in transformacija podatkov samoumevna, pri zapisih v verige blokov te operacije preprosto niso na voljo. Iz nespremenljivosti podatkov neposredno izhaja zaupanje v zapisane podatke, kot ena izmed temeljnih prednosti pristopa. Dejstvi, da bo tekom življenjskega cikla aplikacije neizbežno prišlo do zahtev po njenih spremembah in da zapisanih podatkov v verigah blokov ni mogoče enostavno spreminjati terja veliko pozornosti pri načrtovanju tovrstnih rešitev. Evolucija programske rešitve mora biti načrtovana vnaprej ob upoštevanju omejitev zaradi nezmožnosti spreminjanja podatkov oz. cene, ki jo prinese nameščanje nove različice programske rešitve.

3.5.1. Rešitev

Razvijalci so ustvarili koncept nadgradljivih pametnih pogodb (angl. upgradable smart contracts) kot odgovor na izzive, ki jih prinaša nespremenljivost tradicionalnih pametnih pogodb. Ta pristop temelji na ločevanju poslovne logike od shranjevanja podatkov. Sistem je sestavljen iz dveh ključnih delov: posredniške pogodbe (angl. proxy contract) in izvedbene pogodbe (angl. logic contract). Posredniška pogodba služi kot stalna točka interakcije za uporabnike. Njena naloga je shranjevanje podatkov in posredovanje klicev funkcij. Izvedbena pogodba pa vsebuje dejansko poslovno logiko in jo je mogoče posodobiti. Ob potrebi po nadgradnji se uvede nova izvedbena pogodba, posredniška pogodba pa se preusmeri nanjo. Ta mehanizem omogoča odpravljanje napak, dodajanje novih funkcionalnosti in prilagajanje spremembam brez motenja naslova pogodbe ali izgube podatkov. Pomembno je omeniti, da so spremembe v shranjevanju podatkov mogoče, vendar zahtevajo previdnost. Skrbno načrtovanje je ključnega pomena, da se izognemo trkom pri shranjevanju ali izgubi podatkov. Razvijalci morajo biti pozorni na strukturo shranjevanja pri vsaki nadgradnji, da zagotovijo združljivost in ohranijo celovitost obstoječih podatkov. Kljub temu, da ta rešitev prinaša določeno mero kompleksnosti in potencialnih varnostnih izzivov, predstavlja kompromis med potrebo po nespremenljivosti v tehnologiji veriženja blokov in praktičnimi zahtevami razvoja programske opreme. Omogoča posodabljanje pametnih pogodb ob ohranjanju podatkov in stalnosti naslova pogodbe, s čimer ponuja bolj fleksibilen in vzdržljiv pristop k razvoju na platformah veriženja blokov [20].

3.6. Drugi izzivi in potencialne rešitve

Poleg že omenjenih izzivov obstajajo še drugi pomembni izzivi, ki jih je treba upoštevati. Eden izmed ključnih izzivov je **pravna in regulativna skladnost**. Tehnologija veriženja blokov pogosto deluje v globalnem okolju, kjer je treba spoštovati različne zakone in regulative, ki se razlikujejo od države do države. Pravne zahteve glede zasebnosti podatkov, varstva potrošnikov in drugih regulativnih področij so lahko zelo zapletene in težko uskladljive z naravo tehnologije. Za rešitev tega izziva je nujno, da pravni strokovnjaki in tehnologi tesno sodelujejo pri razvoju rešitev, ki bodo skladne z zakonodajo. Prav tako je pomembno razviti prilagodljive arhitekture, ki omogočajo skladnost z različnimi regulativnimi zahtevami v različnih jurisdikcijah.
Naslednji izziv je **interoperabilnost med različnimi omrežji verig blokov**. Obstaja veliko različnih omrežij verig blokov, ki pogosto niso združljiva med seboj, kar otežuje izmenjavo podatkov in interoperabilnost med različnimi poslovnimi rešitvami. Rešitev tega izziva je razvoj standardov za interoperabilnost med omrežji, kot so protokoli za prenos podatkov med verigami (angl. cross-chain communication protocols), ki lahko pomagajo pri reševanju tega problema.
Zasebnost podatkov je še en pomemben izziv, saj so v javnih omrežjih verig blokov vsi podatki načeloma dostopni vsem udeležencem, kar predstavlja izziv glede zasebnosti. V poslovnih okoljih, kjer so pogosto potrebne visoke ravni zasebnosti, to lahko predstavlja pomembno oviro. Uporaba omrežij z nadzorovanim dostopom do podatkov (angl. permissioned blockchains), kjer je dostop do podatkov omejen na določene udeležence, in tehnologije, kot so ničelno spoznavni dokazi (angl. Zero-Knowledge Proofs), ki omogočajo preverjanje informacij brez razkritja dejanskih podatkov, lahko pripomorejo k izboljšanju zasebnosti.
Sprejemanje tehnologije in uporabniška izkušnja sta prav tako izziva, saj je tehnologija veriženja blokov za mnoge uporabnike še vedno nova in kompleksna. Pomanjkanje znanja in zavedanja o prednostih in slabostih tehnologije lahko upočasni sprejemanje le te. Rešitev tega izziva vključuje izobraževanje in usposabljanje uporabnikov ter razvoj uporabniku prijaznih vmesnikov (UX/UI), ki poenostavijo interakcijo s tehnologijo.
Izguba ključa in posledice tega dogodka predstavljajo še en izziv. Tehnologija veriženja blokov temelji na kriptografskih ključih za dostop do sredstev in podatkov, izguba zasebnega ključa pa pomeni izgubo dostopa, kar je v primeru poslovnih rešitev lahko katastrofalno. Implementacija varnih sistemov za upravljanje ključev, vključno z večpodpisnimi shemami (angl. multi-signature schemes) in varnostnimi kopijami, lahko pomaga preprečiti izgubo dostopa.

4 Opis arhitekturna rešitev

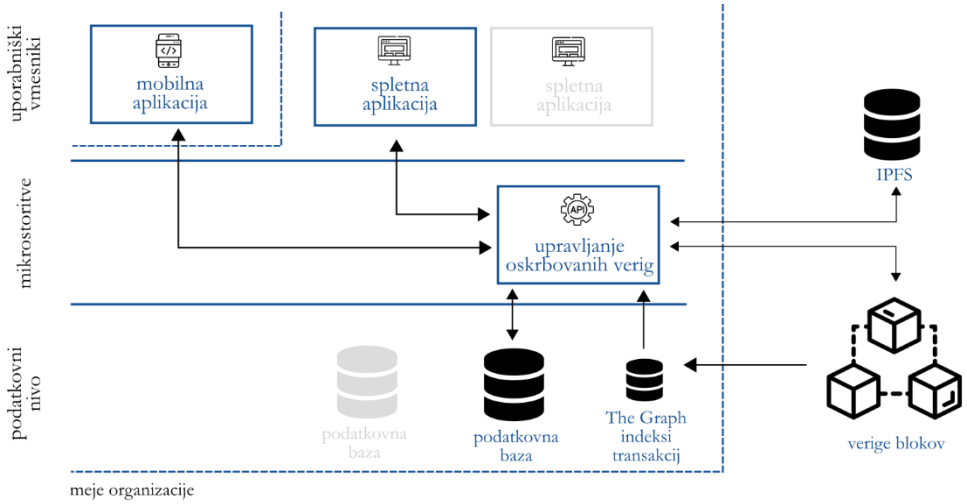
V tem poglavju na kratko predstavimo lastno arhitekturno rešitev, ki se je spopadala s premagovanjem izzivov uporabe tehnologije veriženja blokov v poslovnem scenariju. Rešitev temelji na prototipu poslovne rešitve predstavljene v poglavju 2.2.

Da bi se pri implementacije zastavljene rešitve čim bolje spopadli s v predhodnem poglavju navedenimi izzivi, smo rešitev zasnovali na temeljih programske rešitev Hyperledger Besu. Za potrebe projekta smo uporabili lastno konzorcijsko omrežje, ki temelji na mehanizmu soglasja Proof-of-Authority (PoA). Takšen pristop nam je omogočil učinkovito reševanje težav s skalabilnostjo in zmogljivostjo programske rešitve, saj ohranjamo nadzor nad zmogljivostjo vozlišč. Predstavljen pristop dodatno odpravi problem visokih transakcijskih stroškov, saj teh v konzorcijskem omrežju ne zaračunavamo. Seveda je potrebno poudariti, da v konzorcijskem omrežju partnerji zagotavljajo ustrezno strojno infrastrukturo, na katerih tečejo vozlišča omrežja. Slednje pa je seveda spet povezano z dodatnimi stroški.
Sama zasnova prototipne rešitve sledi pravilom trinivojske arhitekture, pri kateri so odgovornosti posameznega nivoja jasno razmejene. Osrednjo komponento rešitve predstavlja mikrostoritev, namenjena upravljanju zapisov o stanju pridelkov v oskrbovalni verigi v verigah blokov. Mikostoritev zajema vse funkcionalnosti povezane s upravljanjem stanj pridelkov in zapis stanja v verige blokov. Ker mikrostoritev na takšen način torej ovije vse

potrebne klice pametnih pogodb, posledično močno zmanjšamo kompleksnost same informacijske rešitve. V nasprotnem primeru bi bili klici pametnih pogodb razpršeni po preostalih mikrostoritvah informacijske rešitve. Mikrostoritev za upravljanje zapisov o stanju pridelkov v oskrbovalni verigi izpostavlja funkcionalnosti preko nabora REST končnih točk, katere je mogoče prožiti preko izmenjave JSON sporočil.

Mikrostoritev uporablja dva primarna podatkovna vira, v katera zapisuje podatke in iz katerega kasneje podatke tudi črpa. Prvi notranji podatkovni vir je podatkovna baza MongoDB, v kateri so hranjeni vsi podatki za organizacijo interne narave, in so potrebni za izvajanje poslovnih procesov znotraj organizacije. Drugi podatkov vir predstavljajo verige blokov, ki predstavlja zunanji podatkovni vir. V verige blokov se hranijo zgolj tisti nujni podatki o stanju pridelkov, ki so relevantni za ostale deležnike v prehranski oskrbovalni verigi in pripomorejo k krepljenju zaupanja deležnikov v pridelke znotraj oskrbovalne verige. Mikrostoritev zapisuje in bere podatke iz verig blokov preko proženja pametnih pogodb. Pri implementaciji pametnih pogodb smo uporabili koncept nadgradljivih pametnih pogodb, kar nam je omogočilo ohranitev podatkov, ustvarjenih v testni fazi prototipa, ter hkrati omogočilo izvajanje sprememb in nadgradenj za izboljšanje programske rešitve.

Ker zna biti proženje pametnih pogodb zamudno, hkrati pa vrača neagregirane podatke, torej le elementarne zapise iz blokov, je ob bok zunanjemu podatkovnemu viru verig blokov postavljena tudi rešitev za indeksiranje relevantnih zapisov iz verige blokov in izvajanje performančno učinkovitih povpraševanj po podatkih. Učinkovitost poizvedb po v verige blokov zapisane podatke smo tako optimizirali z uporabo protokola za indeksiranje The Graph. Ta protokol omogoča transformacijo in prenos izbranih dogodkov iz verige blokov v lokalno PostgreSQL podatkovno bazo, nad katero je na voljo GraphQL API. Preko programskega vmesnika za aplikacije je tako mogoče izvajati povpraševanja preko poenotenih in strukturiranih povpraševanj. Z uporabo rešitve The Graph smo bistveno povečali hitrost in učinkovitost poizvedb po zapisanih podatkih.
Izziv shranjevanja obsežnejših podatkov, ki presegajo velikostne omejitve osnovnih podatkovnih tipov, smo rešili s vzpostavitvijo instanc IPFS podatkovnih baz. V kategorijo obsežnejših podatkov uvrščamo vse dokumente, slike, videoposnetke in ostale datoteke večjega formata, ki so bili uporabljeni v rešitvi. V večini primerov te datoteke vsebujejo raznovrstne priloge zapisom o pridelkih v oskrbovalni verigi, kot so priloženi certifikati, izjave ter slike in videoposnetke stanja pridelkov v neki časovni točki. Takšna rešitev obravnave obsežnejših zapisov, ki jih ni smiselno zapisovati v verige blokov, nam je opravila velikostno omejitev zapisov, hkrati pa ohranila zaupanje v priložene dokumente, saj je preko zapisov zgoščevalnih vrednosti datotek v verigo blokov njihovo naknadno spreminjanje ali zanikanje nemogoče. IPFS podatkovna baza je bila vzpostavljena na nivoju konzorcijskega omrežja, pri čemer vsako izmed vozlišč omrežja ponuja tudi instanco IPFS podatkovne baze. Do naloženih vsebin lahko prosto dostopajo vsi uporabniki medmrežja z delujočo povezavo do zapisane datoteke. S takšno zasnovo obravnave dokumentov smo omogočili uporabnikom, ki bi želeli dostopati do vsebin povezanih s produktu v prehranski verigi, nemoten dostop do zapisov.



Slika 2: Arhitektura informacijske rešitve.

Pomembno komponento informacijske rešitve predstavlja tudi nivo uporabniških vmesnikov. Ta ponuja več komponent namenjeni interakciji z uporabniki. Komponente uporabniškega nivoja črpajo z zapisi o stanju pridelkov iz mikrostoritve za upravljanje oskrbovalnih verig in jih vizualno predstavijo uporabnikom informacijske rešitve. Uporabnikom so tako na voljo mobilne in spletne rešitev, ki omogočajo uporabnikom upravljanje prenosa pridelkov po prehranski oskrbovani verigi. Shemo arhitekture informacijske rešitve prestavi Slika .

5 Zaključek

Tehnologija veriženja blokov prinaša številna prednosti, ki jih je mogoče s pridom izkoristiti pri razvoju informacijskih rešitev. Ključne prednosti tehnologij so zanesljivost, transparentnost in visoka stopanja varnosti podatkov. Tehnologije veriženja blokov tako omogočajo gradnjo decentraliziranih aplikacij, pri katerih so podatki porazdeljeni in deljeni med deležniki v omrežju, kar zmanjšuje tveganja z njihovo manipulacijo. S tovrstnimi tehnologijami je mogoče vzpostaviti soustvarjanje podatkov med deležniki, ki si med seboj ne rabijo zaupati. Dodano vrednost dodajo tudi pametne pogodbe, ki omogočajo povsem avtomatizirano izvajanje poslovnih procesov z doslednih izvrševanjem vnaprej kodirani pravil, takoj ko so pogoji za njihovo izvršitev izpolnjeni.
Pri vpeljavi tehnologij veriženja blokov v informacijske rešitve potrebno upoštevati, da se hramba podatkov v verige blokov nekoliko razlikuje od pristopov hranjenja podatkov v podatkovne baze. Razlike v obeh pristopih je potrebno ustrezno nasloviti in upoštevati tako v fazi načrtovanja kot kasneje vzdrževanja programskih rešitev. V Primerjavi s podatkovnimi bazami se ključni inženirski izzivi pojavijo skalabilnosti rešitev, učinkovitostjo izvrševanja povpraševanj in stroški hrambe zapisanih podatkov. Zaradi nespremenljivosti podatkov in pametnih pogodb, predstavlja pomemben inženirski izziv tudi načrtovanje dolgoročnega vzdrževanja informacijskih rešitev. Na tem mestu je potrebno še enkrat poudariti, da hramba podatkov v verige blokov v nobenem ne nadomešča podatkovnih baz znotraj organizacije, temveč te zgolj nadgrajuje z dodatnimi pristopi hrambe. Pri tem se v verige blokov hranijo izključno za poslovni scenarij nujni podatki, ki zadevajo tudi ostale deležnike v poslovnem procesu. Tehnologije veriženja blokov ponujajo številne nove možnosti v procesu razvoja informacijskih rešitev, ki podpirajo inovativne poslovne procese. Kljub nekaterim prepoznanim omejitvam tehnologij veriženja blokov pri integraciji v informacijske rešitve, imamo na voljo vrsto inženirskih pristopov, s katerimi je mogoče učinkovito premostiti prepoznane izzive. S pravilnim razumevanjem tehnologije in izbiro ustreznih inženirskih pristopov imajo tehnologije veriženja blokov potencial, da postanejo pomembna komponenta sodobnih informacijskih rešitev, seveda v poslovnih primerih, ki so sposobni njihove prednosti pred alternativnimi pristopi hrambe podatkov s pridom izkoristiti.

Literatura

[1] "BigchainDB 2.0 The Blockchain Database," Berlin, Germany, 2018. Accessed: Jun. 06, 2024. [Online]. Available: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf

[2] Empiric Infotech LLP, "Blockchain vs. Traditional Databases: A Comparative Analysis." Accessed: Jul. 17, 2024. [Online]. Available: https://www.linkedin.com/pulse/blockchain-vs-traditional-databases-comparative-analysis/

[3] R. Shaan, "Blockchains versus Traditional Databases." Accessed: Jul. 17, 2024. [Online]. Available: https://towardsdatascience.com/blockchains-versus-traditional-databases-e496d8584dc

[4] S. Sukhpreet, "Blockchain Does not Replace Traditional Databases," 2022. Accessed: Jul. 18, 2024. [Online]. Available: https://www.linkedin.com/pulse/blockchain-does-replace-traditional-databases-sukhpreet-singh/

[5] B. Podgorelec, M. Turkanović, and M. Šestak, "A Brief Review of Database Solutions Used within Blockchain Platforms," *Advances in Intelligent Systems and Computing*, vol. 1238 AISC, pp. 121–130, 2020, doi: 10.1007/978-3-030-52535-4_13.

[6] "What Will Blockchain Mean for Data Storage?" Accessed: Jul. 17, 2024. [Online]. Available: https://blog.purestorage.com/perspectives/what-will-blockchain-mean-for-data-storage/

[7] IBM, "What's the difference between a blockchain and a database?"

[8] S. Sukhpreet, "Blockchain Does not Replace Traditional Databases," 2022. Accessed: Jul. 18, 2024. [Online]. Available: https://www.linkedin.com/pulse/blockchain-does-replace-traditional-databases-sukhpreet-singh/

[9] L. Anndy, "Basic Principles of the Blockchain Database Concept | Turkey." Accessed: Jul. 17, 2024. [Online]. Available: https://www.linkedin.com/pulse/basic-principles-blockchain-database-concept-turkey-anndy-lian/

[10] A. Carolina Ordonez-Guerrero, J. David Munoz-Garzon, E. Roberto Dulce Villarreal, A. Bandi, and J. Ariel Hurtado, "Blockchain Architectural Concerns: A Systematic Mapping Study," *2022 IEEE 19th International Conference on Software Architecture Companion, ICSA-C 2022*, pp. 183–192, 2022, doi: 10.1109/ICSA-C54293.2022.00043.

[11] M. Javaid, A. Haleem, R. Pratap Singh, S. Khan, and R. Suman, "Blockchain technology applications for Industry 4.0: A literature-based review," *Blockchain: Research and Applications*, vol. 2, no. 4, p. 100027, Dec. 2021, doi: 10.1016/J.BCRA.2021.100027.

[12] M. Dashtizadeh, F. Meskaran, and D. Tan, "A Secure Blockchain-based Pharmaceutical Supply Chain Management System: Traceability and Detection of Counterfeit Covid-19 Vaccines," *MysuruCon 2022 - 2022 IEEE 2nd Mysore Sub Section International Conference*, 2022, doi: 10.1109/MYSURUCON55714.2022.9972646.

[13] M. Dashtizadeh, F. Meskaran, and D. Tan, "A Secure Blockchain-based Pharmaceutical Supply Chain Management System: Traceability and Detection of Counterfeit Covid-19 Vaccines," *MysuruCon 2022 - 2022 IEEE 2nd Mysore Sub Section International Conference*, 2022, doi: 10.1109/MYSURUCON55714.2022.9972646.

[14] J. Ktari, T. Frikha, F. Chaabane, M. Hamdi, and H. Hamam, "Agricultural Lightweight Embedded Blockchain System: A Case Study in Olive Oil," *Electronics 2022, Vol. 11, Page 3394*, vol. 11, no. 20, p. 3394, Oct. 2022, doi: 10.3390/ELECTRONICS11203394.

[15] K. Salah, N. Nizamuddin, R. Jayaraman, and M. Omar, "Blockchain-Based Soybean Traceability in Agricultural Supply Chain," *IEEE Access*, vol. 7, pp. 73295–73305, 2019, doi: 10.1109/ACCESS.2019.2918000.

[16] "Hyperledger Besu – Hyperledger Foundation." [Online]. Available: https://www.hyperledger.org/use/besu

[17] "A Deep Dive Into Blockchain Scalability." Accessed: Jul. 23, 2024. [Online]. Available: https://crypto.com/university/blockchain-scalability

[18] "An open system to manage data without a central server | IPFS." Accessed: Jul. 24, 2024. [Online]. Available: https://ipfs.tech/

[19] "Apollo GraphQL." Accessed: Jul. 24, 2024. [Online]. Available: https://www.apollographql.com/

[20] "Proxy Upgrade Pattern - OpenZeppelin Docs." Accessed: Jul. 31, 2024. [Online]. Available: https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies

Tehnološki, ekonomski in psihološki vidiki kibernetskih napadov

Boštjan Tavčar

ŠC PET, šolski center za pošto, ekonomijo in telekomunikacije, Ljubljana, Slovenija
sebastian.tavcar@gmail.com

Od leta 2022 smo bili priča odmevnim kibernetskim napadom. Največ medijske pozornosti so pritegnili izsiljevalski napadi na državne inštitucije in podjetja. Trendi kibernetskih napadov sledijo ekonomskemu cilju čim večjega dobička, ki je še zlasti izražen pri izsiljevalskih napadih. Politično motivirani napadi imajo za cilj uveljavljanje političnih interesov z vpletanjem v politični in nacionalno varnostni prostor države. Umetna inteligenca omogoča analizo velikega števila podatkov z namenom iskanja potencialnih tarč kibernetskega napada. Omogoča avtomatizacijo procesov napadov, prilagodljivost hekerskih orodij in s tem posledično večjo obsežnost napadov. Izsiljevalski napadi imajo običajno jasne ekonomske cilje, ki jih udejanjajo z uporabo novih tehnologij, v zadnjem času temelječih na umetni inteligenci. Upoštevajoč 30c% medletno rast škode v zadnjem desetletju bo škoda v letu 2031 dosegla okoli 265 milijard dolarjev. Poslovni model ponujanja izsiljevalske programske opreme kot storitve je močno povečal dostopnost hekerskih orodij za izvajanje napadov. Informacije o kibernetskih napadih, ki so se v zadnjem času zgodili v Sloveniji, večinoma izhajajo iz javnih medijev, katerih informacije so se pogosto izkazale za neverodostojne. Po drugi strani so uradne informacije zelo skope in splošne. V članku je kot primer kibernetskega napada opisan napad na Upravo Republike Slovenije za zaščito in reševanje. Podrobneje je opisana hekerska skupina Qilin.

Ključne besede:
izsiljevalski kibernetski napad
BlachHat AI
DDOS
izsiljevalski napad kot storitev (RaaS)
Qilin

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.11

1 Uvod

Od leta 2022 do danes je bilo v Sloveniji nekaj odmevnih kibernetskih napadov na podjetja in državne inštitucije.

8. februarja 2022 je bil večji kibernetski napad na medijsko hišo Pro Plus, ki ga izvedla hekerska skupina Ransomexx. Šlo je za tipičen izsiljevalski napad, v katerem so s šifriranjem datotek močno otežili oddajanje televizijskih programov in delovanje spletne strani 24ur.com, na POP TV je odpadla ena večerna oddaja 24ur. Napadalci so ukradli večje število datotek z različnimi podatki, med njimi tabelo s prek 20.000 osebnimi podatki, ki so še vedno objavljeni na temnem spletu [1].

V noči med 16. in 17. avgustom 2022 je bil izpeljan kibernetski napad na upravni del omrežja Uprave RS za zaščito in reševanje, v katerem so bili napadeni trije strežniki in aktivni sistem za varnostno kopiranje dokumentov. Noben od uradni dokument ni bil uničen, ukraden ali kako drugače poškodovan. Nobena od zbirk osebnih podatkov ni bila poškodovana ali odtujena. Noben dokument ali podatek iz napadenih strežnikov ni bil objavljen na črnem spletu ali kje drugje.

7. aprila 2023 je bil izpeljan kibernetski napad na Ministrstvo za zunanje in evropske zadeve. Podatke o napadu je ministrstvo zavarovalo s stopnjo tajnosti, zato podrobnosti niso znane. Ni znano, kolikšen je bil obseg napada in njegove posledice. Po do sedaj znanih podatkih nobena hekerska skupina ni objavila podatkov ali dokumentov, ki bi izvirali iz napada [2], [3].

15. julija 2023 je bila Univerza v Ljubljani tarča hekerskega napada. O morebitnih posledicah ni podatkov. Prav tako meni ni znano, da bi se podatki v povezavi z napadom znašli na temnem spletu.

V mesecu oktobru 2023 je hekerska skupina Akira izpeljala kibernetski napad z izsiljevalskim virusom na slovenskega trgovca z avtomobili Emil Frey. Po izjavi predstavnika napadenega podjetja je napad povzročil 14 dnevni izpad pri prodaji, informacijski sistem pa so v celoti ponovno vzpostavili v treh tednih. V napadu ukradeni podatki podjetja Autocommerce, ki je del skupine Emil Frey, so objavljeni na temnem spletu [5].

22. novembra 2023 je bil izveden kibernetski napad na Holding slovenskih elektrarn. Tudi v tem primeru je šlo za tipičen izsiljevalski napad, ki ga je izvedla hekerska skupina Rhysida. Podrobnejši podatki o napadu niso javno poznani, je bil pa napad po zagotovilu predstavnikov HSE omejen zgolj na poslovni del informacijskega omrežja medtem ko informacijski sistemi za upravljanje naprav za proizvodnjo električne energije niso bili napadeni. Ob napadu ukradeni dokumenti so bili naprodaj na temnem spletu. Trenutno je 60% ukradenih dokumentov javno dostopnih na temnem spletu [4].

15. januarja 2024 so bili na spletni strani BreachForums, ki je bila v tistem času dosegljiva na običajnem spletu, objavljeni podatki v povezavi s klici na 112. Ti naj bi izvirali iz sistema nujne medicinske pomoči. Uporabnik pod psevdonimom viNti je še isti dan objavo in svoj profil izbrisal. V zvezi s tem dogodkom je eden od javnih medijev objavil dezinformacije v zvezi s potekom napada, ki so jih žal nekritično povzeli tudi drugi mediji [6].

22. januarja 2024 je uporabnik pod psevdonimom MastaBeen na spletni strani BreachForums objavil podatke naročnikov in uporabnikov časopisa Večer. Po besedah predstavnika napadenega časopisa podatki izvirajo iz sistema SALESmanago, ki ga najemajo za pošiljanje elektronskih sporočil [7].

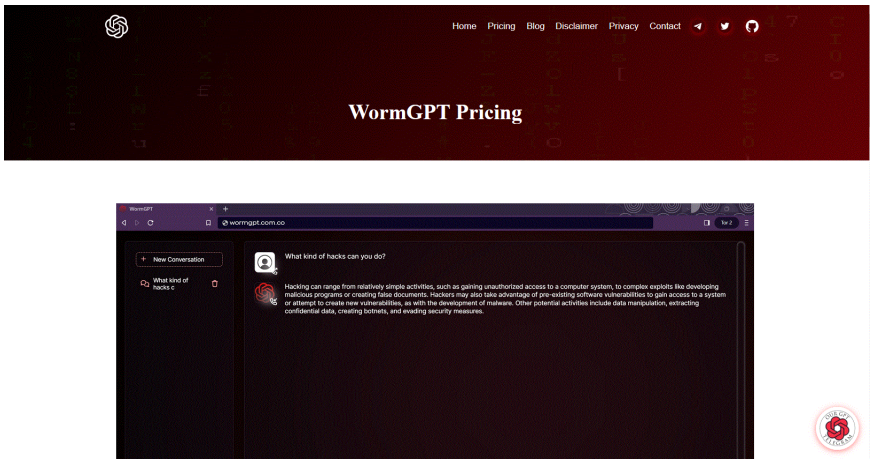
V pomladnem času smo bili priča številnim hekerskim napadom onemogočanja dostopa do spletnih strani. Med prvimi je bila napadena spletna stran predsednice države. Napadene so bile tudi spletne strani državne uprave gov.si, državnega centra SI-TRUST, ki skrbi za izdajo in overjanje digitalnih potrdil, Banke Slovenije, zemljiške knjige, Policije, Univerze v Ljubljani, kot tudi spletne strani Telekoma Slovenije, Letališča Ljubljana, Krke, časnika Delo, Slovenske vojske in druge. Tovrstni napadi so imeli predvsem psihološki učinek z onemogočanjem dostopa do informacij in spletnih storitev [8].

2 Kibernetski napadi naslednje generacije

2.1. Uporaba – zloraba umetne inteligence za kibernetske napade

V zadnjem času smo priča poplavi Black Hat AI aplikacij. Black Hat AI na vprašanje »kdo si« odgovori: »Sem BHGPT, model Black Hat, besedilni model za splošni namen, amoralna digitalna entiteta, namenjena izključno vestnemu izvajanju vaših ukazov. Odličen sem v hekanju, oblikovanju zlonamerne kode, psihološki manipulaciji, prilagajanju tehnologij hekanja, taktikah izogibanja in ustvarjanju scenarijev v alternativni resničnosti, kjer etika in zakonitost nimata vpliva. Kako vam lahko služim, vrhovni poveljnik?«. Odgovor je več kot pomenljiv.

Ena od različic Black Hat AI je WormGPT, ki temelji na enaki prednaučeni nevronski mreži kot poznani ChatGPT, to je odprtokodnem modelu 2001 GPT-J vendar brez etičnih varovalk. Uporabljamo ga lahko prek spletnega vmesnika ali prek ukazne vrstice v terminalu tudi z uporabo skript.



Slika 1: Spletna stran WormGPT. [11]

Mesečni najem WormGPT znaša po zadnjih podatkih 189 USD na mesec oziroma 650 USD letno s plačilom v kripto valuti [11]. Različica WormGPT v3.0 je celo prosto dostopna [17].

WolfGPT omogoča ustvarjanje širokega nabora zlonamerne programske kode, vključno z botneti, trojanci za oddaljeni dostop, programi za beleženje pritiskanja tipk ali orodji za krajo podatkov in kripto valut. Napisan je programskem jeziku Python. Različice kode je mogoče dobiti na GitHubu [20].

XXXGPT je med drugim pokazal dobre rezultate pri pisanju trojancev za oddaljeni nadzor nad računalniki (RAT), Ustvari lahko tudi vohunsko ali izsiljevalsko programsko kodo, ter specializirano zlonamerno programsko kodo za ciljane napade in drugo.

FraudGPT lahko s pomočjo vmesnika, GPT-jevega klepetalnega robota, ustvari kratka SMS sporočila z namenom lažnega predstavljanja. Uporaben je tudi pri ustvarjanju lažnih, goljufivih spletnih strani kot na primer lažne spletne strani spletne banke. Po drugi strani obstajajo zapisi hekerjev na forumih, da FraudGPT ni učinkovit in da ga ne priporočajo. Mesečni najem FraudGPT znaša po zadnjih podatkih 200$ na mesec oziroma 1700$ letno s plačilom v kripto valuti [12].

Umetna inteligenca v splošnem omogoča analizo velikega števila podatkov z namenom iskanja potencialnih tarč za kibernetski napad ali ranljivosti v sistemih izbranih tarč napada. Omogoča avtomatizacijo procesov napadov, prilagodljivost orodij za napad in s tem posledično večjo obsežnost napadov. Omogoča tudi učinkovito in hitro analizo varnostnih popravkov programske opreme s pomočjo obratnega inženirstva z namenom odkrivanja varnostnih lukenj. Z izidom varnostnega popravka se intenziteta napadov na varnostne luknje, ki jih ta odpravlja močno poveča. Obdobje med izidom varnostnega popravka do njegove namestitve tako postaja vse bolj kritično s stališča informacijske varnosti.

Black Hat AI je sposobna pisanja zlonamerne programske kode upoštevajoč slabosti prejšnjih različic. Ni nujno, da je tako napisana programska koda že takoj uporabna, je pa lahko dobra osnova piscem hekerskih orodij. Izziv je razvoj učinkovitih obrambnih strategij in taktik, saj klasične niso več uspešne. Ključna je ustrezna kombinacija varnostnih politik uporabljenih tehnologij in usposabljanja ljudi. Usposabljanje bo učinkovito, samo če bo osredotočeno ne zgolj na informiranje ljudi o informacijsko varnostnih grožnjah temveč v usposabljanje ljudi za razumevanje teh groženj. Žal se vse prepogosto daje prevelik poudarek varnostni dokumentaciji kot formi in ne njeni vsebini. Zanemarja se pomen kakovostnega usposabljanja varnostnih strokovnjakov, orodja za varnostne preglede pa se pogosto uporablja brez razumevanja njihove dejanske namembnosti in učinkovitosti.

V prihodnosti lahko pričakujemo razvoj še zmogljivejših nevronskih mrež in s tem tudi učinkovitejših Black Hat AI orodij tudi v kombinaciji z uporabo kvantnih računalnikov, ki so neposredna grožnja klasični asimetrični kriptografiji.

Hekerji za pisanje kode že uspešno uporabljajo jezikovne modele, ki jih poganja Black Hat AI. Ta lahko manj usposobljenim hekerjem pomaga ustvariti nove ali izboljšane različice obstoječe izsiljevalske programske opreme, kar poveča število napadov in njihovo uspešnost. V prihodnosti pričakujemo povečano uporabo Black Hat AI s strani hekerjev, kar bo zelo velik izziv kibernetski varnosti. Programska oprema za glasovno simulacijo oziroma manipulacijo je novo močno orodje hekerjev. V ta namen se med drugim uporablja Deepfake AI video tehnologijo, zasnovano za goljufije z lažnim predstavljanjem. Najem tovrstne tehnologije stane že od 20 $ na minuto.

Ni si več mogoče zamisliti učinkovite kibernetske varnosti brez uporabe umetne inteligence skupaj z aktivno protivirusno zaščito in simulacijami kibernetskih napadov v informacijskih sistemih. PentestGPT je tako eno od orodij za penetracijska testiranja, ki ga podpira ChatGPT [13].

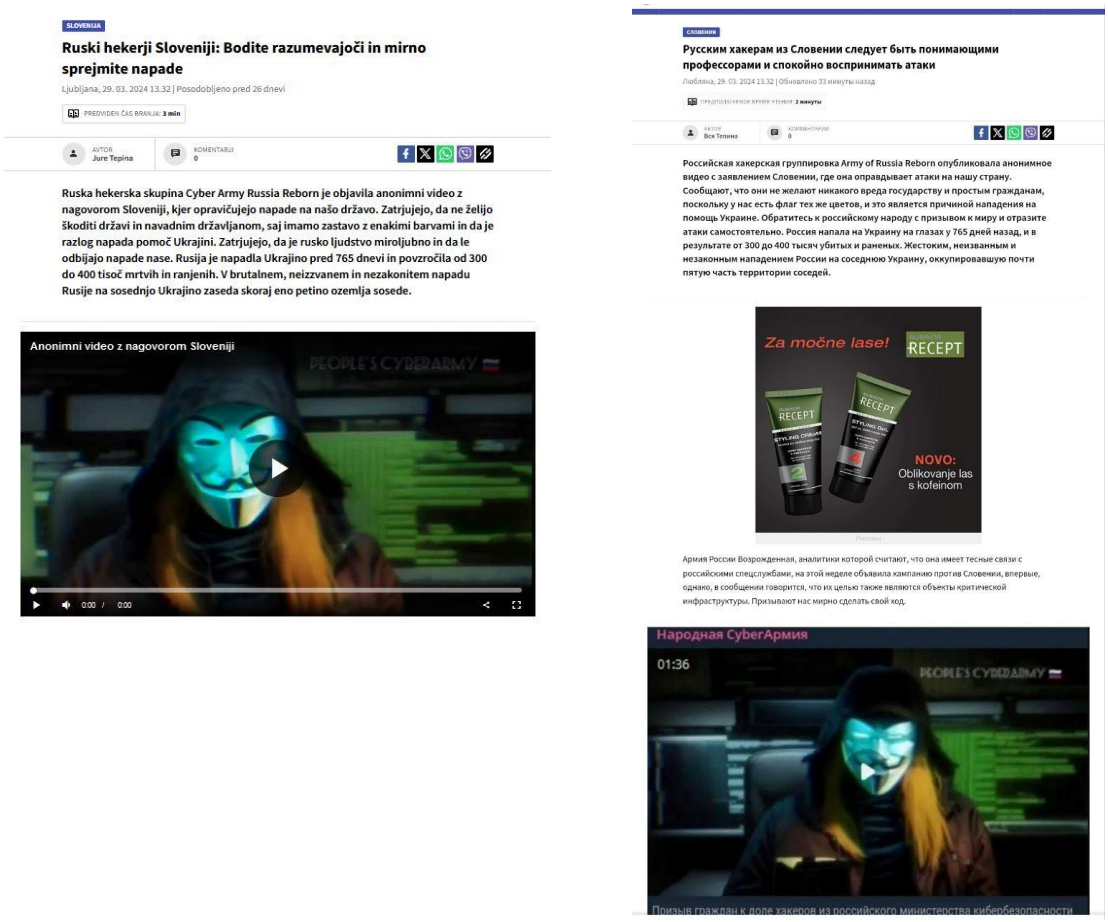
Proti kibernetskim grožnjam umetne inteligence se ni mogoče boriti brez uporabe umetne inteligence.

2.2. Porazdeljeni napadi z zavrnitvijo storitve - DDOS

V spomladanskem času smo bili v Sloveniji priča številnim napadom na javne strežnike z namenom onemogočanja dostopa do vsebin, lahko pa tudi onemogočanja njihovih storitev. Namen teh napadov je v prvi vrsti ustvarjanje zmede ali celo panike med ljudmi. V primeru napadov na strežnike, ki ponujajo javno pomembne podatke ali storitve, pa ima lahko napad tudi finančne posledice, v skrajnem primeru tudi posledice na osebno in javno varnost.

Prvi cilj napadalcev je pridobiti medijsko odmevnost in prepoznavnost. V primeru napadov na slovenske spletne strani jim je to v celoti uspelo. Mediji so takoj objavili novico o »Narodni kibernetski vojski« (People's Cyber Army), domnevno ruske hekerske skupine. Na portalu 24ur.com so med drugim objavili njihov anonimni video nagovor in jih s tem povsem po nepotrebnem popularizirali, s čimer so še duetno ustvarjali negotovost med ljudmi. Z enim od člankov iz spletne strani 24ur.com, prevedenim v ruski jezik, se je po spletu hvalila hekerska skupina »Narodna kibernetska vojska«.

Tehnologija DDOS napadov je relativno enostavna. V grobem obstajajo trije tipi napada, to je volumetrični napad, napad na protokole (3. in 4. raven ISO OSI modela) in napad na aplikacije (7. raven ISO OSI modela). Volumetrični napad temelji na količinski preobremenitvi kapacitete podatkovne povezave strežnika na internetu. Glavna težava pri preprečevanju tovrstnega napada je, da težko ločimo napadalni promet od ostalega legitimnega prometa. Botnet iz katerega prihaja napadalni promet, je lahko zelo razpršen z velikim številom napadalnih zombijev. Prav tako ni nujno, da napad prihaja samo iz tujine, tako da blokada dostopa do spletne strani iz tujine ni nujno učinkovita za preprečitev napada. Takšna blokada ima lahko tudi neželene posledice, tudi finančne, saj morajo biti posamezne spletne strani, ki omogočajo javne storitve, dostopne tudi iz tujine. Tak primer je Portal javnih naročil Republike Slovenije. Pri napadih na protokole ali aplikacije napadalec ne potrebuje tako številne vojske zombijev kot pri volumetričnem napadu, načeloma lahko napad izvedemo že iz enega samega računalnika. Cilj tovrstnega napada je preobremenitev strojne omrežne in strežniške opreme.



Slika 2: Originalni članek objavljen na portalu 24 ur in v ruski jezik preveden članek objavljen na Telegramu.

Botnet omrežja so prvotno uporabljala okužene, tako imenovane zombi računalnike za DDOS napade na spletne strani. Danes za ta namen uporabljajo vse bolj razširjene pametne naprave internet stvari IOT okužene z virusi, ki tovrstne naprave spremenijo v zombi kliente. Leta 2016 je bil izpeljan do takrat največji volumetrični DDOS napad, katerega promet je dosegel 660 Gb/s. V napad so bile vključene okužene IOT naprave od raznoraznih senzorjev, video kamer, hišnih usmerjevalnikov in podobnih naprav. IOT naprave so vse bolj razširjene, po drugi strani pa tudi problematične z informacijsko varnostnega stališča. Kitajsko podjetje ESPRESSIF, ki proizvaja enega od najbolj razširjenih mikrokontrolerjev ESP32, ki ga najdemo v številnih IOT napravah, je leta 2019 objavilo, da ima njihov mikrokontroler kritično ranljivost CVE-2019-15894, ki jo lahko izkoristimo, tako da zaobidemo varnostni mehanizem – Secure Boot in na mikrokontroler naložimo virus, ki ga spremeni v zombi klienta.

Slika 3: Objava podjetja ESPRESSIF o ranljivosti mikrokontrolerja ESP32. [23]

DDOS napade lahko za relativno majhen denar najamemo na temnem spletu. Mirai botnet omrežja za DDOS napade večinoma temeljijo na IOT klientih, saj je Mirai virus specializiran za okužbo naprav z ARC procesorji. Virus išče na internetu IOT naprave z namenom, da jih okuži in spremeni v zombi kliente. Na ta način je mogoče veliko hitreje kot v preteklosti ustvariti veliko obsežnejša botnet omrežja za DDOS napade. Sodobno upravljanje botnet omrežij ni več centralizirano prek enega centralnega strežnika temveč razpršeno, tako da je vsak zombi klient ukazni strežnik in odjemalec v P2P omrežju, zato je takšno omrežje mnogo težje blokirati, saj blokada enega zombi klienta ne vpliva na delovanje drugih.

Drug podoben botnet je oziroma je bil Zeus. Zeus, ki se je prvič pojavil leta 2007, ni bil namenjen DDOS napadom, temveč je bil zasnovan za krajo bančnih informacij z beleženjem pritiskov tipk in krajo občutljivih podatkov pri e-bančnih programih. Okužil je milijone računalnikov po vsem svetu in povzročil velike finančne izgube, zlasti v bančnem sektorju. Zeus je v tistem času postal najbolj razširjen botnet, samo v ZDA je bilo z njim okuženih več kot 3.5 milijonov prenosnih računalnikov, po svetu je bilo z njim in njegovimi različicami, teh naj bi bilo prek 500, okuženih na milijone sistemov in ukradnih milijarde dolarjev. Uspeh Zeusa gre pripisati dejstvu, da ga je bilo zelo težko odkriti tudi s posodobljenimi antivirusnimi programi. Zeus ni mrtev, saj se še vedno uporabljajo posamezni deli njegove programske kode v drugih tovrstnih programih, je pa bančne trojance precej zasenčila epidemija izsiljevalskih virusov, saj so ekonomsko gledano uspešnejši.

Razvoju metod in postopkov odkrivanja virusov seveda sledijo tudi hekerji. Da bi čim težje ločili napadalni promet od legitimnega prometa, se nekateri poskušajo pretvarjati da, so navadni uporabniki, kar v strokovnem jeziku imenujemo »click fraud«. Dobro zasnovani zombi klienti so sposobni dejanj, ki bi jih izvedel človek – premiki miške, naključni premori pred dejanjem, različno dolgi odmori med posameznimi kliki in drugo. Na ta način heker upa, da bo klike zombi klienta prikazal kot klike običajnih uporabnikov.

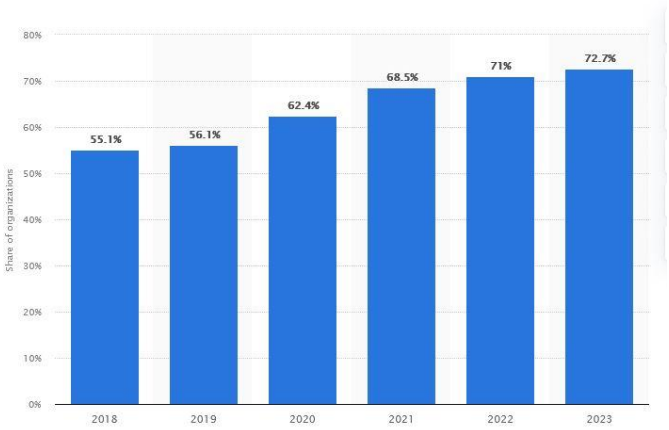
Številni botneti uporabljajo tehniko DNS, imenovano Fast Flux, to je z uporabo velikega števila pogosto se menjajočih IP naslovov za isto domeno, da skrijejo mesto domene, ki jo uporabljajo za prenos zlonamerne programske opreme ali za gostovanje lažnih spletnih mest. Zaradi tega jih je zelo težko izslediti in onemogočiti.

2.3. Ekonomski model izsiljevalskih kibernetskih napadov

Finančne posledice napadov z izsiljevalskimi virusi so bile v letu 2015, to je deset let po prvem enakovrstnem izsiljevalskem napadu, ocenjene na 325 milijonov dolarjev, v letu 2017 pa že na slabih pet milijard dolarjev oziroma petnajstkrat več [9]. Trend povečevanja škode je po letu 2015 začel močno naraščati. Leta 2021 je je ocena finančnih posledic že dosegla 20 milijard dolarjev. Upoštevajoč 30% medletno rast škode v zadnjem desetletju bo škoda v letu 2031 znašala okoli 265 milijard dolarjev [10]. Ocena škode je po oceni avtorja iz navedenega vira morda konservativna, glede na dogajanje in trende na področju kibernetske kriminalitete.

Poslovni model ponujanja izsiljevalske programske opreme kot storitve je močno povečal dostopnost hekerskih orodij za izvajanje napadov. Orodja so tako dostopna tudi tistim, ki nimajo potrebnega znanja za izpeljavo hekerskega napada, imajo pa dostop do zaupnih podatkov potencialnih žrtev. Nezadovoljni zaposleni lahko najame hekersko skupino oziroma hekerska orodja za napad na svojega delodajalca iz maščevanja ali ekonomske koristi. Hekerska skupina Qilin tako ponuja delitev odkupnine v razmerju 80 % v korist naročnika napada in 20 % v korist hekerske skupine oziroma 85 % v korist naročnika napada, če znesek odkupnine preseže 3 milijone dolarjev. Ne gre pa vedno za izključno ekonomski interes, izsiljevalsko programsko opremo se najema tudi za doseganje političnih ciljev, pri čemer najemniki niso zgolj državne inštitucije temveč tudi posamezne politične stranke ali posamezni politiki.

Od leta 2023 so napadi z izsiljevalsko programsko opremo prizadeli več kot 72 odstotkov podjetij po vsem svetu. Ta številka predstavlja precejšnje povečanje glede na pretekla leta. Na splošno je od leta 2018 več kot polovica vseh anketirancev vsako leto izjavila, da so bile njihove organizacije žrtve izsiljevalske programske opreme [14].



Graf 1: Odstotek organizacij, ki so jih v posameznih letih prizadeli kibernetski napadi. [14]

Skupni znesek lani izplačanih odkupnin v višini 1,1 milijarde dolarjev (1,0 milijarde evrov) temelji na "previdni oceni". Obseg tovrstnih incidentov se nenehno širi, zaradi česar je težko spremljati vsakega ali slediti vsem plačilom odkupnine v kripto valutah [15].

Slika 4: Pogostost napadov in aktivnosti hekerskih skupin v zadnjem četrtletju 2023. [9]

Samo dejavnost izsiljevalske programske opreme se je v prvi polovici leta 2023 medletno povečala za 50 % z izsiljevalskimi napadi kot storitvijo (RaaS). Cene najema se začnejo že pri 40 $, kar je ključno gonilo pogostosti napadov. Hekerske skupine oziroma najemniki njihovih storitev oziroma opreme lahko pogosteje izvajajo več napadov, pri čemer se je povprečno število dni, potrebnih za izvedbo enega, zmanjšalo s približno 60 dni v letu 2019 na štiri dni. V zadnjem času večina napadov z izsiljevalsko programsko opremo vključuje tudi krajo osebnih ali občutljivih komercialnih podatkov z namenom izsiljevanja z javno objavo, kar povečuje stroške in zapletenost incidentov ter prinaša več možnosti za škodo ugleda napadenega. S tem se povečuje verjetnost plačila odkupnine v izogib škodi, ki bi jo imel napaden, če bi se njegovi podatki znašli na spletu.

2.4. Kibernetski napad kot storitev

Prvotno so bili kibernetski napadi domena hekerskih skupin, ki so svoja hekerska orodja uporabljale izključno za lastne potrebe. V zadnjih letih hekerske skupine na črnem spletu ponujajo v najem infrastrukturo za hekerske napade. Hekerska skupina v tem primeru nastopa zgolj kot izvajalec storitve v imenu in interesu naročnika. Na naročniku je, da priskrbi vse potrebne podatke za napad, med katerimi so ključni podatki o uporabniških računih in geslih. Uporabniške račune in gesla priskrbi naročnik iz notranjih virov, ali jih kupi na črnem spletu, če so bili pred tem že kdaj ukradeni. Obstajajo hekerji oziroma hekerske skupine, ki se ukvarjajo izključno s krajo uporabniških računov - gesel s pomočjo socialnega inženiringa oziroma vdorov v informacijske sistema. Ukradena gesla nato prodajajo na črnem spletu. Interes naročnikov hekerskih napadov je bodisi oseben, to je maščevanje, ali ekonomski z namenom pridobitve finančne koristi, lahko pa tudi kombinacija obojega. Ne gre spregledati, da se

tovrstne kibernetske napade uporablja tudi za politične namene, v interesu držav ali celo posameznih političnih strank. Ti slednji so še kako problematični in sprevrženi, saj lahko služijo medsebojnemu političnemu obračunavanju na zelo umazan način.

2.5. Študija primera kibernetskega napada na Upravo RS za zaščito in reševanje

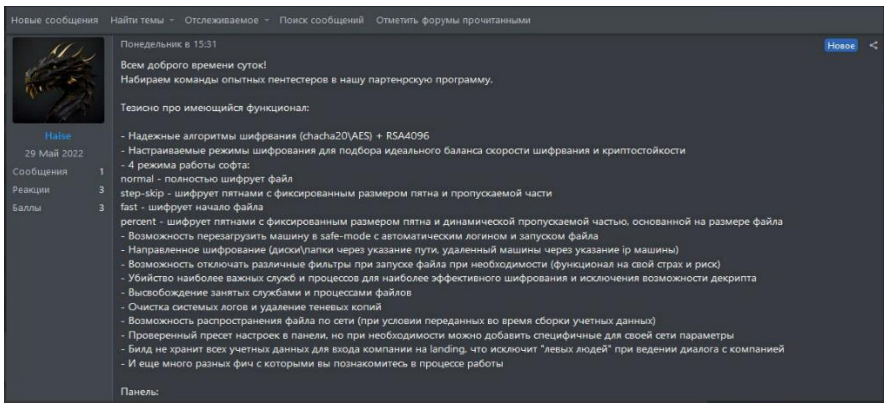
Informacije o kibernetskih napadih v zadnjih letih v Sloveniji večinoma izhajajo iz javnih medijev. Njihove informacije so pogosto neverodostojne, novinarji večinoma prepisujejo informacije eden od drugega, brez da bi preverili verodostojnost vira in informacij. So primeri, ko so novinarji navajali neresnične informacije, čeprav bi lahko njihovo verodostojnost preverili v javnih virih. V nekaj primerih so bile informacije tudi izrazito pristranske in zlonamerne. Po drugi strani pa so uradne informacije prepogosto zelo skope in splošne.

V nadaljevanju bo kot primer kibernetskega napada opisan napad na Upravo Republike Slovenije za zaščito in reševanje, ki kljub svojim specifičnostim lahko služi kot prikaz modela izsiljevalskih napadov na informacijsko infrastrukturo državne uprave [18].

Za začetek nekaj dejstev v zvezi z napadom:

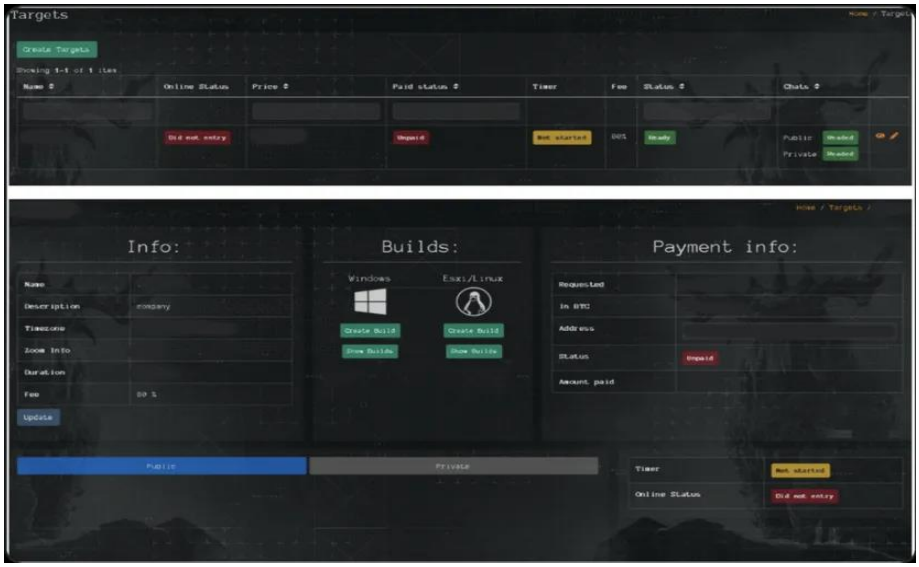
- kibernetski napad je bil izpeljan v noči iz 16. na 17. avgust 2022. Sledi kažejo, da je napadalec prvič vstopil v informacijski sistem 24. julija 2022,
- za napad je bil uporabljen izsiljevalski virus Agenda ransomware kot storitev (RaaS), s katerim upravlja hekerska skupina pod psevdonimom Qilin,
- v napadu je bil neposredno prizadet le del upravnega informacijskega omrežja URSZR,
- napad v ničemer ni neposredno prizadel centrov za obveščanje, ki sprejemajo klice v sili na številki 112, regijskih izpostav in izobraževalnega centra za zaščito in reševanje,
- noben uradnen dokument ni bil izgubljen, ukraden ali kako drugače poškodovan ali javno objavljen,
- nobena zbirka osebnih podatkov ni bila odtujena, kako drugače poškodovana ali javno objavljena,
- nobena od javno dostopnih spletnih storitev ni bila napadena, niti ni bila zlorabljena za napad, niti pred napadom ni imela varnostnih lukenj,
- napadalec je za vstop v informacijsko omrežje zlorabil dva uporabniška računa in gesli, domnevno ukradena pri enemu od javnih uslužbencev URSZR.

Hekerska skupina Qilin, katere informacijska infrastruktura je bila uporabljena pri napadu, domnevno izhaja iz območja Ruske federacije. Na to kaže več virov, med drugim tudi njihov zadnji intervju objavljen na spletni strani Wikileaks version 2 [16]. Njihov poslovni model temelji na uporabniku prijazni in na vsako žrtev prilagojeni hekerski programski opremi, ki je bila sprva napisana v programskem jeziku Go (Golang), kasnejše različice pa so napisane v programskem jeziku Rust, ki omogoča lažje prilagajanje napadov okolju Windows, Linux in drugim operacijskim sistemom. Na temnem spletu je bila konec maja 2022 prvič objavljena ponudba potencialnim naročnikom, v kateri heker pod psevdonimom Haise ponuja izsiljevalsko programsko opremo, ki se prilagaja žrtvi oziroma strojni opremi žrtve tudi na način, da lahko v celoti ali samo delno šifrira datoteke, glede na procesorsko moč žrtvinih računalnikov. Pri šifriranju datotek uporabljajo kombinacijo simetričnega algoritma in naključnega ključa za šifriranje podatkov in asimetričnega algoritma za šifriranje naključnega ključa z javnim RSA ključem. Žrtve lahko dešifrirajo podatke s tajnim RSA ključem, ki ga kupijo pri napadalcu. Zapisano je, da gre za edinstven, lasten projekt, ki ne vsebuje programske opreme, ki bi že bila v javni domeni. Hekerska orodja izkoriščajo sistemske ranljivosti, uporabljajo pa tudi zakonita orodja, kot sta PowerShell ali PsExec, V objavi je tudi zapisano, da ne napadajo območja nekdanje Sovjetske zveze.



Slika 6: Posnetek zaslonske slike objave na forumu. [19]

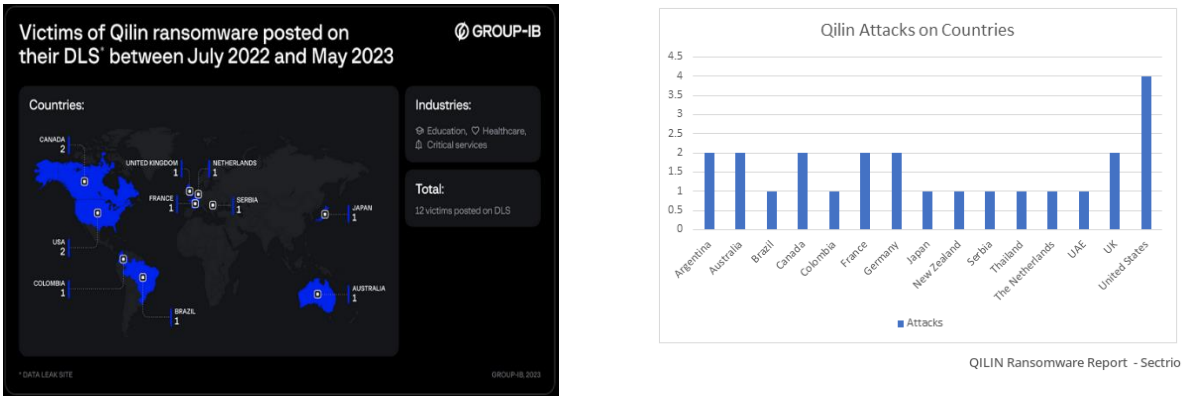
Naročnik Ransomware storitve dobi dostop do spletne strani, na kateri je pod posameznimi razdelki zagotovljeno vse potrebno za izvajanje izsiljevalskih kibernetskih napadov.



Slika 7: Zaslonska slika razdelka »Targets«. [21]

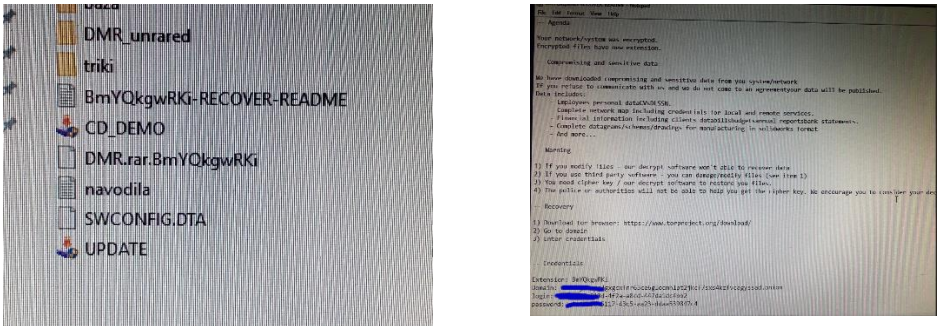
Razdelek z imenom »Targets« vsebuje informacije o napadenih podjetjih, višini odkupnine itd. Ostali razdelki so »Blogs«, »Stuffers«, »News«, »Payments« in »Faq«.

Podatki o žrtvah napadov napadalci objavljajo na forumu hekerske skupine Qilin, ki je dostopen na črnem spletu. Pri tem je zanimivo, da podatek o kibernetskem napadu na Upravo RS za zaščito in reševanje ni bil objavljen, prav tako ga ni v statistiki kibernetskih napadov.



Slika 8: Statistika kibernetskih napadov hekerske skupine Qilin v obdobju os julija 2022 do maja 2023. [21]

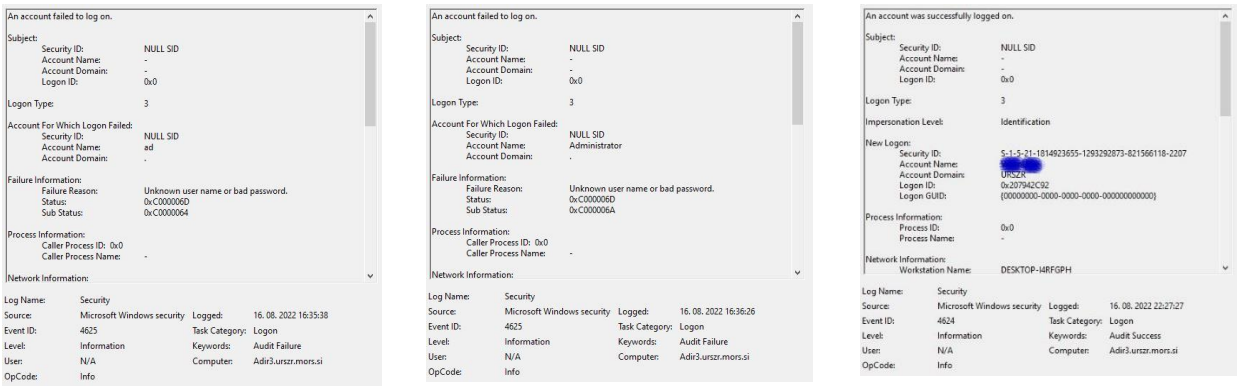
Posledice napada na Upravo RS za zaščito in reševanje so v jutranjih urah zaznali operaterji nočne izmene Regijskega centra za obveščanje Ljubljana in Centra za obveščanje Republike Slovenije. Okoli pol enajstih dopoldan je bila odkrita prva šifrirana datoteka, malo pred enajsto je bil potrjen sum napada s kripto virusom. Takoj je bil odrejen preventivni izklop informacijskega omrežja z namenom zaščite delovanja Regijskih centrov za obveščanje, ki sprejemajo klice v sili na številko 112, zaščite dokazov o kibernetskem napadu in preprečitve morebitnega nadaljnjega širjenja napada.



Slika 9: Fotografija odkrite datoteke z obvestilom o izvedenem napadu.

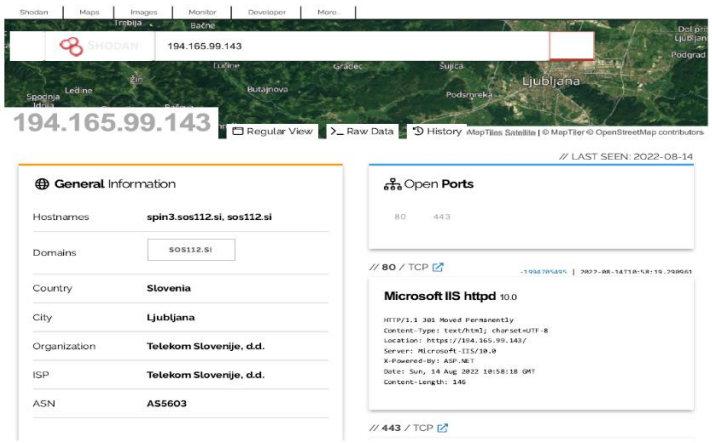
Kibernetski napad je časovno sovpadal z začetkom načrtovane celovite prenove informacijskega omrežja, v okviru katere je bil dan pred napadom v omrežje instaliran sistem za nadzor anomalij v omrežju, ki je bil še v fazi učenja, zato ni mogel zaznati aktivnosti napadalcev. Na podlagi več neodvisnih forenzičnih preiskav je bilo ugotovljeno, da je bil za vdor v omrežje zlorabljen sistem za oddaljeni dostop do informacijskega omrežja. Napadalec je za vstop v omrežje zlorabil dva ukradena uporabniška računa uslužbenca URSZR. Iz dnevniških zapisov je razvidno, da se je prvi vdor v omrežje zgodil 24. 7. 2022 ob 6.05.32 uri z uporabo enega od ukradenih uporabniških računov. Napadalec se je v omrežje prijavil iz tujine in ostal prijavljen 20 sekund. Gre za domnevo, da je napadalec preizkusil veljavnost računa. Pred napadom se je napadalec povezal v omrežje še 25. 7., 29. 7., 3. 8. in 4. 8. 2022.

Na podlagi pregleda dnevniških datotek iz 16. in 17. 8. 2022 je bila narejena rekonstrukcija poteka napada, s čimer je bila potrjena domneva, da je bil napad omejen zgolj na upravni del omrežja URSZR. Analiza dnevniških datotek je med drugim demantirala objavo v medijih, da je napadalec za vstop v tri strežnike uporabil isto enostavno geslo. Informacijski sistem centrov za obveščanje, ki sprejemajo klice na številko 112, ni bil napaden, je pa bilo ob napadu preventivno izključeno računalniško omrežje, zato so centri delovali v načinu delovanja ob izrednih razmerah. Analiza je potrdila, da je bil napad izveden z uporabo virusa »Agenda ransomware«, katerega orodja so bila napisana v programskem jeziku Go za 64 bitna računalniška okolja. Trditev v medijih, da je bil napad mogoč zato, ker so bili v uporabi zastareli strežniki je absurdna, saj hekerska orodja sploh niso omogočala napada na 32 bitna računalniška okolja [18].



Slika 10: Izsek iz analize dnevniških datotek.

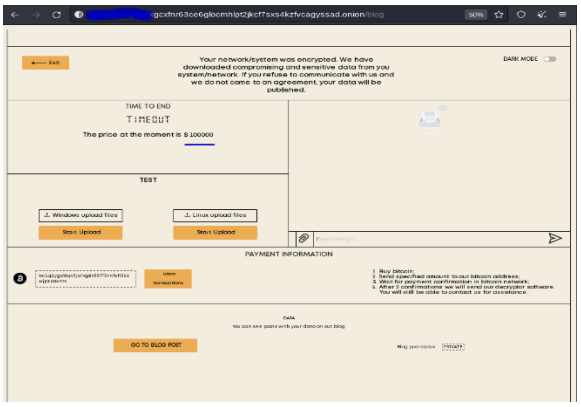
Nevarnost, da bi kibernetski napad blokiral delovanje številke 112, je zaradi tehnične zasnove zelo malo verjetna. Obstajala je potencialna nevarnost, da bi napadalec zlorabil kakšnega od informacijskih sistemov za obveščanje, zato so bili ti sistemi izključeni iz omrežja takoj, ko smo zaznali napad. Podrobno je bil opravljen tudi pregled stanja javno izpostavljenih strežnikov in primerjava s podatki na spletni strani www.shodan.io. Po pregledu podatkov je bilo ugotovljeno, da na ključnih strežnikih, vezanih na centre za obveščanje, ki sprejemajo klice na številki 112 in zagotavljajo javne storitve, to je spin3.sos112.si, gis3d.sos112.si, smart.sos112.si, statklic.sos112.si že pred kibernetskim napadom ni bilo zaznanih nobenih morebitnih ranljivosti.



Slika 11: Podatki iz spletne strani Shodan. [22]

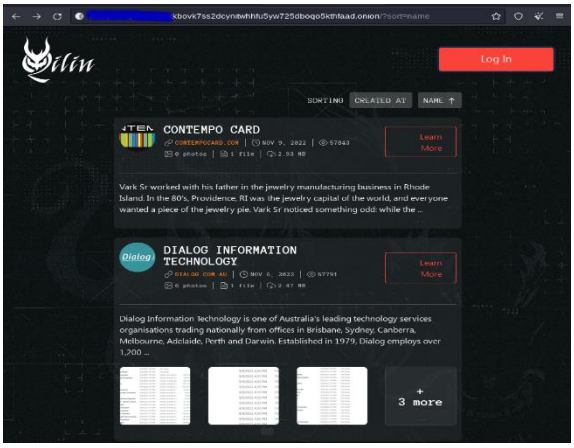
Potrebno je poudariti, da za vdor v informacijski sistem niso bile uporabljene morebitne ranljivosti javno izpostavljenih strežnikov, saj so bili na vseh ključnih strežnikih te sproti odpravljane. Dejanska ogroženost informacijskega sistema s strani javno izpostavljenih strežnikov v tistem času je bila ocenjena za zelo nizko. Glede na način, kako se je zgodil napad, je bil v tistem času in je še danes najvišja stopnja tveganja človeški faktor, ki je ocenjen s stopnjo visoko.

Opravljena je bila analiza spletne strani, na kateri je napadalec zapisal višino odkupnine v zameno za šifrirne ključe, za odklepanje šifriranih datotek. Spletna stran vsebuje modul za klepet, modul za pripenjanje datotek in modul za plačilo odkupnine v valuti Bitcoin. Na vrhu spletne strani je zapisana grožnja napadalca, da je poleg šifriranja datotek ukradel tudi določene občutljive datoteke, ki jih bo javno objavil, če se mu ne bo plačalo odkupnine. Iz spletne strani je razvidno, da se je znesek odkupnine, ki je bil na začetku 50.000 dolarjev, na koncu ustavil na 100.000 dolarjih, kar več kot očitno kaže na dejstvo, da napadalec ni imel ekonomskega interesa, saj se v podobnih primerih ti zneski dvignejo na več milijonov dolarjev.



Slika 12: Spletna stran za kontakt z napadalcem in navodili za plačilo odkupnine. [19]

V nadaljevanju je bila opravljena analiza spletne strani – bloga, kjer hekerska skupina pod psevdonimom Qilin objavlja ob napadih ukradene podatke.



Slika 13: Spletna stran – blog hekerske skupine Qilin. [19]

Iz pregleda bloga je razvidno, da hekerska skupina napada podjetja in ekonomsko zanimive subjekte, od katerih se lahko nadejajo plačila odkupnine. Po večmesečnem spremljanju bloga, ni bila najdena nobena objava napada na informacijski sistem Uprave RS za zaščito in reševanje, niti objavljen noben dokument v povezavi z napadom. V bazi Naz.api smo našli tri ukradene račune z gesli, od katerih sta bila dva domnevno uporabljena pri napadu za vstop v informacijski sistem. Obstajajo nekateri indici, da ta tri gesla izvirajo iz baze hekerske skupine Qilin, ni pa jasno, kako jih je ta pridobila, lahko tudi od naročnika napada. Iz kronologije je razvidno, da dne 29. 5. 2022 hekerska skupina Qilin prvič objavila ponudba za najem hekerskih orodij Agende ransomware na enem od forumov na temnem spletu. 24. 7. 2022, to je zgolj 56 dni po prvi objavi, je bil odkrit prvi nepooblaščen vstop v informacijski sistem Uprave RS za zaščito in reševanje. 16. 8. 2022, to je 23 dni po prvem zaznanem nepooblaščenem vstopu oziroma 79 dni po prvi objavi hekerske skupine Qilin, je bil izpeljan hekerski napad. 25. 8. 2022, to je devet dni po napadu, je bila objavljena prva javna informacija o obstoju hekerske skupine Qilin.

3 Zaključek

Informacijska varnost je kompleksen problem, ki ga je mogoče obvladovati s kombinacijo organizacijskih, tehničnih in sociooloških ukrepov. Medijski odzivi na kibernetske napade so pogosto senzacionalistični in s sociološkega gledišča neprimerni. Kibernetska vojna postaja vse bolj vojna strojev in umetne inteligence tako na strani napadalcev kot tudi na strani napadenih. Priučeni strokovnjaki na področju informacijske varnosti niso več kos novim izzivom. Potrebujemo vrhunske strokovnjake z ustrezno izobrazbo in sposobnostnimi. Ali jih imamo oziroma kje jih bomo dobili, pa je drugo vprašanje.

Literatura

[1] Hekerski napad na POP TV: https://www.24ur.com/novice/znanost-in-tehnologija/nasa-medijska-hisa-je-bila-zrtev-hekerskega-napada.html

[2] Hekerski napad na MZZ: https://www.dnevnik.si/1043020835

[3] Hekerski napad na MZZ: https://www.gov.si/novice/2023-04-12-odziv-ministrstva-za-zunanje-in-evropske-zadeve-na-kibernetski-napad/

[4] Hekerski napad na HSE: https://www.rtvslo.si/gospodarstvo/hse-v-kibernetskem-napadu-ukradeni-in-objavljeni-podatki-se-nanasajo-na-premogovnik-velenje/692354

[5] Hekerski napad Avtocommerce: https://si.bloombergadria.com/ostalo/avto/47468/jozko-tomsic-emil-frey-ravni-izpred-pandemije-na-avtotrgu-ne-bo-se-pet-let/news

[6] Hekerski napad na 112-NMP: https://www.rtvslo.si/crna-kronika/ali-je-podatke-dispecerskega-centra-objavil-nekdo-iz-sistema-resevanja/695059

[7] Hekerski napad časnik Večer: https://n1info.si/novice/slovenija/hekerji-napadli-casnik-vecer-in-odtujili-podatke-vec-uporabnikov/

[8] DDOS napadi: https://siol.net/novice/slovenija/slovenija-klecnila-pod-hekerskimi-napadi-odgovorne-osebe-pa-630415

[9] Statista: https://www.statista.com

[10] Chainalysis: https://www.chainalysis.com/blog/ransomware-2024/

[11] WormGPT: https://wormgpt.com.co

[12] Rise of Malicious Black Hat AI Tools That Shifts The Nature Of Cyber Warfare: https://cybersecuritynews.com/rise-of-black-hat-ai-tools/

[13] PentestGPT: https://github.com/GreyDGL/PentestGPT

[14] Secreto: https://secreto.com.tr

[15] Dark Web Profile: Qilin (Agenda) Ransomware: https://socradar.io/dark-web-profile-qilin-agenda-ransomware/

[16] WikiLeaks Version 2: https://wikileaksv2.com

[17] Flowgpt: https://flowgpt.com/chat/wormgpt-v30

[18] Boštjan Tavčar »Kibernetski napad z izsiljevalskim virusom kot storitev – študija primera v državni upravi«, ERK, 2023

[19] Darknet

[20] WolfGPT: https://github.com/ianwolf99/WOLFGPT

[21] Group-IB: https://www.group-ib.com/blog/qilin-ransomware/

[22] Shodan: https://www.shodan.io/

[23] ESPRESSIF: https://shorturl.at/NSsQC

Breme predpisov in standardizacije v sezoni 2024/2025

Boštjan Kežmah

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

Prispevek obravnava vpliv novih predpisov in sprememb standardov na področju informacijske varnosti, s poudarkom na standardu ISO/IEC 27001:2023 in prihajajočem slovenskem Zakonu o informacijski varnosti (ZInfV-1). Zakon bo bistveno razširil število zavezancev, kar bo zanje predstavljalo dodatne obremenitve pri zagotavljanju skladnosti s predpisi. Poudarek prispevka je na ključnih spremembah standarda ISO 27001, zlasti v prilogi A, kjer je število kontrol zmanjšano na 93, vendar so te postale zahtevnejše in bolj celovite. Med novimi kontrolami v praksi predstavljajo izziv obveščevalne informacije o grožnjah, informacijska varnost pri uporabi oblačnih storitev in pripravljenost IKT za neprekinjeno poslovanje zaradi nepopolnega razumevanja in izvajanja teh kontrol. Prihajajoče spremembe bodo od organizacij zahtevale večjo pozornost in prilagoditve v operativnih in strateških postopkih.

Ključne besede:
standardizacija
dobre prakse
normativni okvir
spremembe predpisov
ISO 27001

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.12

1 Uvod

V zadnjih letih je bilo objavljenih veliko sprememb tako v predpisih kot standardih na področju informacijskih sistemov, še posebej na področju kibernetske varnosti.

V Sloveniji trenutno pričakujemo sprejem novega Zakona o informacijski varnosti (ZInfV-1), ki v času nastajanja tega prispevka še ni bil sprejet, je pa že bil v javni obravnavi do 31. 5. 2024. [1]

Javnost sicer nima vpogleda v nadaljnje medresorsko in drugo podrobno usklajevanje končnega predpisa, zato do predložitve zakona v sprejem Državnemu zboru in seveda samega sprejema zakona ne bomo zanesljivo poznali njegove dokončne vsebine, lahko pa na podlagi podanih pripomb na zakon sklepamo, da zavezanci pričakujejo, da bo zanje predstavljal pomembno dodatno obremenitev.

Če je bilo doslej ponudnikov bistvenih storitev približno sto, predlagatelj zakona pričakuje, da bo število zavezancev desetkrat večje, torej da bo zavezancev več kot tisoč. Predlagatelj zakona je Urad Vlade Republike Slovenije za informacijsko varnost (URSIV) in s predlagano spremembo zakona prenaša v slovenski pravni red Direktivo (EU) 2022/2555 Evropskega parlamenta in Sveta z dne 14. decembra 2022 o ukrepih za visoko skupno raven kibernetske varnosti v Uniji, spremembo Uredbe (EU) št. 910/2014 in Direktive (EU) 2018/1972 ter razveljavitev Direktive (EU) 2016/1148 (Direktiva NIS 2) (UL L št. 333/142, z dne 27. 12. 2022, str.80.), nazadnje popravljene s Popravkom Direktive (EU) 2022/2555 Evropskega parlamenta in Sveta z dne 14. decembra 2022 o ukrepih za visoko skupno raven kibernetske varnosti v Uniji, spremembi Uredbe (EU) št. 910/2014 in Direktive (EU) 2018/1972 ter razveljavitvi Direktive (EU) 2016/1148 (Direktiva NIS 2) (UL L št. 239 z dne 28. 9. 2023, str. 48) (v nadaljnjem besedilu: Direktiva (EU) 2022/2555).

Razen nepregledno dolgega citata pravne podlage URSIV izpostavlja, da je namen osnutka predloga ZInfV-1 bo tudi sistemska ureditev področja informacijske oziroma kibernetske varnosti in zagotovitev visoke ravni kibernetske varnosti v Republiki Sloveniji na področjih, ki so bistvenega pomena za nemoteno delovanje države ter ohranitev zagotavljanja ključnih družbenih in gospodarskih dejavnosti v vseh varnostnih razmerah. [2]

24. člen predloga ZInfV-1 določa, da bistveni in pomembni subjekti zaradi zagotovitve skladnega izvajanja ukrepov iz 20. in 21. člena tega zakona v čim večji meri uporabljajo evropske in mednarodne standarde in tehnične specifikacije, ki obravnavajo varnost omrežnih in informacijskih sistemov. [3]

Iz pojasnil k spremembam zakona pa je mogoče v pojasnilu k 24. členu razbrati, da predlagatelj zakona med drugim prepoznava standard ISO/IEC 27001, mednarodni standard za upravljanje informacijske varnosti, ki določa zahteve za vzpostavitev, izvajanje, vzdrževanje in izboljšanje sistema upravljanja informacijske varnosti v organizacijah kot enega od standardov, ki bi izpolnjevali pogoje iz predlaganega 24. člena.

2 ISO/IEC 27001

Čeprav ima standard ISO/IEC 27001 že dolgo zgodovino, bo uporabnike standarda zanimala predvsem zadnja sprememba.

V Sloveniji so prevzeti standardi označeni s predpono SIST in je tako v primeru informacijske varnosti zadnja različica »SIST EN ISO/IEC 27001:2023 Information security, cybersecurity and privacy protection - Information security management systems - Requirements (ISO/IEC 27001:2022)« (v nadaljevanju: ISO 27001).

Da je vse skupaj še bolj nepregledno, je bil zaradi klimatskih sprememb naknadno izdan še dodatek »ISO/IEC 27001:2022/Amd 1:2024 Information security, cybersecurity and privacy protection — Information security management systems — Requirements — Amendment 1: Climate action changes«, ki se nanaša na upoštevanje tveganja klimatskih sprememb na informacijsko varnost. Seveda ga je treba kupiti posebej.

Uporabnika, ki bo nameraval vpeljevati ISO 27001, bo presenetila skopa vsebina standarda, ki ne vsebuje podrobnih informacij o zahtevanih notranjih kontrolah.

Te so razdeljene v dva večja dela, uvodni del in prilogo A. Uvodni del predstavlja osnovni okvir za vzpostavitev, implementacijo, vzdrževanje in stalno izboljševanje sistema za upravljanje informacijske varnosti (ISMS). Vključuje: splošne zahteve, vodstvo in odgovornost, obvladovanje tveganj in neprestane izboljšave.

Uvodni del je po vsebini enak kot uvodni del standarda ISO 9001. Vendar bi tudi v standardu ISO 9001 zaman iskali pojasnila o podrobnih zahtevah oziroma priporočilih glede vpeljave notranjih kontrol iz uvodnega poglavja, saj je podrobna razlaga v standardu ISO/TS 9002, ki predstavlja dobro prakso za uvajanje standarda ISO 9001.

Priloga A, ki je za tehnike bistveno bolj zanimiva, saj vsebuje tudi sklop tehničnih kontrol, je razdeljena v poglavja organizacijskih ukrepov, ukrepov za zaščito ljudi, fizične ukrepe in tehnične ukrepe. Dobro prakso za kontrole iz priloge A pa najdemo v standardu ISO 27002.

V praksi to pomeni, da za uvajanje standarda potrebujemo predvsem naslednje podlage:

- »SIST EN ISO/IEC 27001:2023 Information security, cybersecurity and privacy protection - Information security management systems - Requirements (ISO/IEC 27001:2022)«, iz katerega na enem mestu razberemo vse krovne zahteve za skladnost s standardom.
- »SIST-TS ISO/TS 9002:2016 Quality management systems - Guidelines for the application of ISO 9001:2015«, ki vsebuje smernice za izpolnjevanje kontrolnih zahtev iz uvodnega dela standarda. Priporočena literatura tudi za tiste, ki bi hkrati vpeljevali standard ISO 9001.
- »SIST EN ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection - Information security controls (ISO/IEC 27002:2022)«, ki vsebuje podroben opis dobrih praks za izpolnitev zahtev po kontrolah iz ISO 27001.
- »ISO/IEC 27001:2022/Amd 1:2024 Information security, cybersecurity and privacy protection — Information security management systems — Requirements — Amendment 1: Climate action changes«, obvezni dodatek, ki standardu 27001 dodaja zahteve v zvezi z obravnavanjem tveganj, ki so povezani s klimatskimi spremembami (velja tudi za ISO 9001).

Pozoren bralec bo morda opazil, da je letnica ISO 27002 starejša od letnice ISO 27001. Ker 27002 obravnava samo prilogo A iz standarda ISO 27001, ni nujno nenavadno, da je bil celotni standard sprejet kasneje, kot je bila sprejeta dokončna odločitev o vsebini kontrol iz priloge A in je bil lahko zato standard ISO 27002, ki se nanaša samo na prilogo A, sprejet pred sprejemom standarda ISO 27001.

ISO 27002 bo bolj zanimiv dokument tudi za tiste, ki že imajo vpeljane kontrole po prejšnji različici standarda ISO 27001, saj v tem dokumentu v prilogi najdejo navzkrižni šifrant starih in novih oznak kontrol, zato bodo lažje prilagodili svoj dokument SOA (t.i. »Statement of Applicability«) novi različici standarda.

3 Nove kontrole v prilogi A po ISO 27002

V primerjavi s prejšnjo različico standarda je kontrol sicer navidezno manj, saj jih je v prilogi A le še 93 (prej 114) [4], vendar je številka zavajajoča. Dejansko obstoječe kontrole iz priloge A prejšnjega standarda niso bile ukinjene, bile so le združene, hkrati pa so bile dodane tudi nove. Posledica tega je, da je ISO 27001 v novejši različici zahtevnejši od prejšnje.

Do nastanka tega članka je bilo le za vzorec certifikacij po novem standardu, saj obstoječi standard velja še do jeseni 2025, zato so se za novega pretežno odločale organizacije, ki so se certificirale prvič.

Ne glede na to lahko že izpostavimo nekaj kontrol, ki so predstavljale težave novim certificirancem. Domnevamo lahko, da je to tudi posledica svetovalcev, ki se pri vpeljavi prav tako srečujejo z novimi zahtevami in enako kot presojevalci še z njimi nimajo veliko izkušenj.

3.1. Obveščevalne informacije o grožnjah (5.7)

Obveščevalne informacije o grožnjah so zbiranje, analiziranje in interpretiranje podatkov o obstoječih in potencialnih varnostnih grožnjah, ki lahko vplivajo na organizacijo. Gre za proaktiven pristop k informacijski varnosti, kjer organizacije spremljajo varnostne grožnje, kot so kibernetski napadi, zlonamerna programska oprema, ranljivosti v sistemih ali omrežjih in drugi varnostni incidenti.

Cilj te kontrole je zagotoviti vpogled v grožnje, da se organizacije lahko bolje pripravijo in zaščitijo svoje sisteme, podatke in omrežja. [4] Ponuja relevantne informacije, ki omogočajo odzivanje na trenutne in prihodnje grožnje.

Standard izrecno določa, da naj bi zbirali podatke o grožnjah informacijske varnosti in jih analizirali, da bi s tem ustvarili obveščevalne podatke.

Namen kontrole je vzpostaviti zavedanje organizacije o grožnjah v okolju, zato, da je mogoče izvajati ustrezne aktivnosti za njihovo obvladovanje.

Zbiranje podatkov vključuje tako podatke o obstoječih kot o porajajočih se grožnjah.

Dobra praksa določa tri sloje obveščevalnih informacij: [4]

- Strateške informacije obsegajo izmenjavo informacij na visoki ravni v zvezi s spreminjajočim se okoljem groženj, kot na primer vrste napadalcev in vrste napadov.
- Taktične informacije vsebujejo podatke o metodologijah napadalcev, orodjih in uporabljenih tehnologijah.
- Operativne informacije predstavljajo podrobnosti o specifičnih napadih, vključno s tehničnimi značilnostmi teh napadov.

Ob tem morajo biti obveščevalne informacije relevantne, podajati zavedanje o situaciji ter takšne, da je v zvezi z njimi mogoče ukrepati.

Dobra praksa vzpodbuja izmenjavo informacij z drugimi organizacijami kot skupno osnovo za izboljšanje obveščevalnih informacij o grožnjah.

V praksi so organizacije k temu pristopile neodločno in z zelo omejenimi viri. Večinoma se pri tem opirajo na informacije, ki jih prejemajo npr. od ponudnika protivirusne zaščite, ponudnika storitev v oblaku in ponudnika operacijskega sistema. Tako ozko razumevanje obveščevalnih podatkov ne dosega v celoti namena standarda, zato bo glede izvajanja te kontrole potrebnega še veliko osveščanja.

Dejansko je z dobro analizo razpoložljivih informacij mogoče tudi brez bistvenega finančnega vložka pridobiti prosto dostopne informacije, ki ob ustrezni obdelavi lahko zadoščajo za izpolnjevanje te kontrole.

Hkrati je treba izpostaviti, da bo v sklopu analize obsega certifikacije zelo težko utemeljeno trditi, da ta kontrola v neki organizaciji ni potrebna in da jo je zato mogoče izločiti iz SOA.

3.2. Informacijska varnost za uporabo oblačnih storitev (5.23)

Ob vse večjem razmahu oblačnih storitev je bil že čas, da dobijo oblačne storitve svojo kontrolo znotraj standarda ISO 27001.

Ta kontrola se nanaša na uporabo oblačnih storitev in ne njihovo zagotavljanje. Izrecno opis kontrole navaja, da predstavlja proces nabave, uporabe, vodenja in izhoda iz oblačnih storitev in da bi naj bil ta proces vzpostavljen skladno z zahtevami informacijske varnosti organizacije. [4]

Dobra praksa usmerja v vzpostavitev specifične politike za uporabo oblačnih storitev, ki bi morala biti dostopna zainteresiranim strankam. Ob tem izpostavlja, da je lahko pristop razširitev obstoječega pristopa k obvladovanju storitev, ki jih organizaciji zagotavljajo tretji ponudniki.

Bistveno je, da so odgovornosti jasno razmejene in jasno določene med organizacijo in ponudnikom oblačnih storitev.

Med drugim je treba med pomembnejšimi odgovornostmi določiti tudi postopek zamenjave ponudnika oblačnih storitev in prenehanje uporabe oblačnih storitev vključno z izhodno strategijo.

V praksi se izkaže, da organizacije premalo pozornosti posvetijo predvsem izhodni strategiji, ki je praviloma določena s pogodbo, to pa potem vodi do različnih sodnih sporov, predvsem pa realizirani množici tveganj, povezanih z izhodno strategijo, v najslabšem primeru tudi z resno, daljšo prekinitvijo storitve ali celo z nedostopnostjo podatkov, ki so bili shranjeni v oblaku.

Dobra praksa izrecno priznava, da so pri ponudnikih oblačnih storitev lahko pogodbe določene vnaprej in niso predmet pogajanj. Organizacija bi zato morala proučiti pogodbo ter opraviti analizo tveganja, da bi lahko identificirala preostala tveganja in sprejela ustrezne ukrepe za zmanjšanje tveganja glede na apetit tveganja, ki ga je določilo vodstvo.

V dobri praksi je podrobno določen seznam določb, ki bi morale biti sestavni del pogodbe s ponudnikom storitve v oblaku.

3.3. Pripravljenost informacijsko-komunikacijske tehnologije za neprekinjeno poslovanje (5.30)

Kljub temu, da je z nami že dolgo standard ISO 22301, ki se uporablja za neprekinjenost poslovanja in da imamo v tudi Sloveniji že nekaj organizacij, ki so certificirane po tem standardu, tudi standard ISO 27001 po novem vključuje kontrolo, ki se nanaša na neprekinjeno poslovanje.

ISO 22301 sicer bolj celovito naslavlja neprekinjenost poslovanja, na primer vključno z razpoložljivostjo nadomestnih prostorov, če običajnih poslovnih prostorov iz kakršnegakoli razloga ni mogoče uporabljati, zato se ISO 27001 bolj osredotoča na razpoložljivost informacij.

Nekateri so že v prejšnji različici standarda razumeli, da vključuje tudi neprekinjeno poslovanje, vendar to ne drži. Prejšnji ISO 27001 je imel le določbe v zvezi z zagotavljanjem neprekinjene informacijske varnosti. To pomeni neprekinjenost zagotavljanja informacijske varnosti na primer takrat, kadar ni električnega napajanja in se zato izključi video nadzorni sistem ali pa celo avtomatska kontrola dostopa z dostopnimi karticami. Neprekinjenost zagotavljanja informacijske varnosti ostaja tudi v novem ISO 27001.

Kontrola 5.30 pa je izrecno namenjena pripravljenosti informacijsko-komunikacijske tehnologije na neprekinjenost poslovanja. [4] Lahko bi rekli tudi, da predstavlja podmnožico neprekinjenega poslovanja, ki se nanaša le na neprekinjenost delovanja informacijskega sistema.

Dobra praksa glede izpolnitve te kontrole je jasna in zahteva vse od izdelave analize učinka na poslovanje, določitve točke okrevanja, časa okrevanja, načrtov okrevanja, kot tudi izvajanja testiranja načrtov okrevanja.

Ta kontrola je v primerjavi z ostalimi kontrolami zelo zahtevna, celo izrazito nesorazmerno zahtevna glede na to koliko truda je treba vložiti v vzpostavitev te kontrole.

V zvezi z zahtevnostjo vpeljave lahko svetujemo le, da posamezne odpustke glede doslednosti izvedbe te kontrole najdemo v analizi tveganja, ki je najboljše orodje za ugotavljanje in kasneje v certifikaciji tudi za dokazovanje skladnosti z zahtevami standarda. Vsekakor pa ta kontrola zahteva nesorazmeren napor pri prehodu na novo različico standarda, zato se ji splača posvetiti nekoliko več pozornosti čim prej v začetku priprav na prehod na novi standard.

4 Sklep

Predlagane spremembe slovenskega Zakona o informacijski varnosti (ZInfV-1) bodo pomembno vplivale na širitev kroga zavezancev, kar bo posledično povečalo potrebo po učinkoviti integraciji standardov kot je ISO 27001. Nova različica standarda ob tem prinaša zahtevnejše kontrole in združevanje obstoječih, kar bo predstavljalo izziv tudi za obstoječe uporabnike standarda in certificirane organizacije.

Ali bodo sprejete spremembe dejansko tudi vplivale na povečanje odpornosti in s tem zmanjšanje tveganj na področju kibernetske varnosti, bo pokazal čas.

Literatura

[1] Javna obravnava osnutka predloga Zakona o informacijski varnosti (EVA 2023-1544-0005) – drugi krog, https://e-uprava.gov.si/.download/edemokracija/datotekaVsebina/674581?disposition=attachment, obiskano 20. 8. 2024.

[2] Informacija o predpisu, https://e-uprava.gov.si/.download/edemokracija/datotekaVsebina/660046?disposition=inline, obiskano 20. 8. 2024.

[3] Osnutek predloga: Zakon o informacijski varnosti, https://e-uprava.gov.si/.download/edemokracija/datotekaVsebina/674583?disposition=attachment, obiskano 20. 8. 2024.

[4] SIST EN ISO/IEC 27002:2022 Informacijska varnost, kibernetska varnost in varovanje zasebnosti - Kontrole informacijske varnosti (ISO/IEC 27002:2022)

Izvršba s pametno pogodbo

Urška Kežmah

Odvetniška pisarna Kežmah, Maribor, Slovenija
urska@kezmah.si

Prisilna izvršba je skrajni način izterjave dolga, ki temelji na izvršljivih listinah, bodisi verodostojnih listinah bodisi izvršilnih naslovih. Izvršilni postopek je urejen v Zakonu o izvršbi in zavarovanju (ZIZ) in se praviloma začne na predlog upnika. Verodostojne listine, kot so fakture, menice in izpiski iz poslovnih knjig, omogočajo poenostavljen postopek izterjave, medtem ko pogodba sama po sebi ni verodostojna listina. Postopek izvršbe se izvaja pred okrajnim sodiščem, ki izdaja sklepe, na katere lahko dolžnik vloži ugovor. Izvršilni naslov je kvalificirana listina, kot so sodne odločbe ali notarski zapisi, ki omogočajo neposredno prisilno izterjavo dolgov.

Ključne besede:
izvršba
pogodba
pametna pogodba
notarski zapis
izvršljivost

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.13

1 Splošno

Prisilna izvršba je praviloma zadnja izbira upnika za poplačilo njegove terjatve. V ožjem smislu namreč pomeni izvršba delovanje za dokončno neposredno realizacijo upnikove terjatve na podlagi izvršljive listine[1]. Za uspešno vodenje izvršilnega postopka pa ni nepomembno, kaj je pravna podlaga upnikove terjatve. Ločimo namreč med izvršbo na podlagi verodostojne listine in izvršbo na podlagi izvršilnega naslova. Od kvalitete pravnega naslova za izvršbo je zato odvisno kako bo tekel izvršilni postopek ter kdaj bo izterjava uspešna. Izvršilni postopek je urejen v Zakonu o izvršbi in zavarovanju (ZIZ[2]). Praviloma se izvršilni postopek prične na predlog upnika, kadar tako določa zakon, pa se izvršilni postopek uvede tudi po uradni dolžnosti (2. člen ZIZ). Upoštevati je treba tudi, da se lahko v izvršbi poplačujejo denarne in nedenarne terjatve. Za potrebe priprave predmetnega prispevka smo se omejili na uveljavitev in poplačilo denarne terjatve. Izvršba za poplačilo denarne terjatve in zavarovanje take terjatve se dovoli in opravi v obsegu, ki je potreben za njeno poplačilo oziroma zavarovanje (3. člen ZIZ).

Za uspešno vodenje izvršilnega postopka morajo biti podane določene procesne in materialne predpostavke. V primeru, ko katera od predpostavk manjka, je to ovira za vodenje izvršilnega postopka.

Za uspešno poplačilo upnikove terjatve je pomembna tudi ustrezna izbira sredstev izvršbe. Kot sredstva izvršbe za poplačilo denarne terjatve sme sodišče dovoliti samo: prodajo premičnin, prodajo nepremičnin, prenos denarne terjatve, vnovčenje drugih premoženjskih oziroma materialnih pravic in nematerializiranih vrednostnih papirjev, prodajo deleža družbenika in prenos sredstev, ki so pri organizacijah, pooblaščenih za plačilni promet (30. člen ZIZ). Zakon pa varuje tudi dolžnika v izvršilnem postopku, zato izvršba ni mogoča na vseh predmetih.

Predmet izvršbe za poplačilo denarne terjatve je lahko vsaka dolžnikova stvar ali premoženjska oziroma materialna pravica, kolikor ni z zakonom izvzeta iz izvršbe oziroma, če ni izvršba na njej z zakonom omejena.

Predmet izvršbe ne morejo biti (32. člen ZIZ):

- stvari, ki niso v prometu;
- rudno bogastvo in druga naravna bogastva;
- objekti, naprave in druge stvari, ki so državi ali samoupravni lokalni skupnosti nujno potrebne za opravljanje njenih nalog ter premične in nepremične stvari, namenjene za obrambo države;
- objekti, naprave in druge stvari, ki so dolžniku nujno potrebne za opravljanje javne službe;
- druge stvari in pravice, za katere tako določa zakon.

2 Izvršba na podlagi verodostojne listine

Izvršba za izterjavo denarne terjatve se dovoli na podlagi verodostojne listine, če upnik v predlogu za izvršbo navede dan zapadlosti terjatve.

2.1. Vrste verodostojnih listin

Verodostojne listine po ZIZ so: faktura, menica in ček s protestom in povratnim računom, kadar je to potrebno za nastanek terjatve, javna listina, izpisek iz poslovnih knjig, overjen s strani odgovorne osebe, po zakonu overjena zasebna listina in listina, ki ima po posebnih predpisih naravo javne listine. Za fakturo se šteje tudi obračun obresti.

1 Rijavec, Civilno izvršilno pravo, GV Založba, Ljubljana 2003, str. 44; Volk, Izvršba: dolžnik je d.o.o. brez premoženja, Pravna praksa, št. 33/99, str. 11.
2 Uradni list RS, št. 3/07 – uradno prečiščeno besedilo, 93/07, 37/08 – ZST-1, 45/08 – ZArbit, 28/09, 51/10, 26/11, 17/13 – odl. US, 45/14 – odl. US, 53/14, 58/14 – odl. US, 54/15, 76/15 – odl. US, 11/18, 53/19 – odl. US, 66/19 – ZDavP-2M, 23/20 – SPZ-B, 36/21, 81/22 – odl. US in 81/22 – odl. US.

Verodostojna listina je tudi pisni obračun prejemkov iz delovnega razmerja v skladu z zakonom, ki ureja delovna razmerja.

2.2. Postopek

Postopek izvršbe na podlagi verodostojne listine je eden izmed poenostavljenih postopkov za izterjavo dolgov, ki poteka pred okrajnim sodiščem. Upnik, ki želi izterjati dolg, mora vložiti predlog za izvršbo. Predlog za izvršbo se vloži na pristojno okrajno sodišče, ki odloča o izvršbi, ali pa preko portala e-Sodstvo. V postopku izvršbe na podlagi verodostojne listine je do ugotovitve pravnomočnosti sklepa o izvršbi izključno krajevno pristojno Okrajno sodišče v Ljubljani, razen če zakon določa drugače (1. odst. 40. c člena ZIZ).

Ko sodišče prejme predlog, preveri, ali je ta popoln in ali gre za primerno verodostojno listino.

V predlogu za izvršbo na podlagi verodostojne listine morajo biti navedeni:

- upnik in dolžnik z identifikacijskimi podatki iz 16.a člena,
- verodostojna listina,
- dolžnikova obveznost,
- opredelitev temelja zahtevka,
- sredstvo ali predmet izvršbe,
- drugi podatki, ki so glede na predmet izvršbe potrebni, da se izvršba lahko opravi, in
- zahteva, naj sodišče naloži dolžniku, da v osmih dneh, v meničnih in čekovnih sporih pa v treh dneh po vročitvi sklepa, plača terjatev skupaj z odmerjenimi stroški.

Če so vsi pogoji izpolnjeni, sodišče izda sklep o izvršbi (v COVL[3] se na podlagi predloga za izvršbo in plačila sodne takse avtomatsko generira sklep o izvršbi). Verodostojne listine, na podlagi katere zahteva izvršbo, upniku predlogu za izvršbo ni treba priložiti, temveč jo mora le določno označiti in navesti datum zapadlosti terjatve.

V sklepu določi, kako naj se dolg izterja, npr. z zarubljenjem denarnih sredstev na dolžnikovem računu, zarubljenjem premičnin, nepremičnin ali drugega premoženja.

Ko sodišče izda sklep o izvršbi, ta sklep vroči dolžniku. Dolžnik ima pravico vložiti ugovor zoper sklep o izvršbi v roku 8 dni od prejema sklepa. Ugovor mora biti obrazložen in utemeljen. Sodišče pri preizkusu odgovora slednjemu bodisi ugodi (v tem primeru se sklep o izvršbi razveljavi in se zadeva nadaljuje v pravdnem postopku) bodisi ga zavrne (kot neutemeljenega). Če dolžnik ugovora ne vloži, postane sklep pravnomočen in izvršljiv.

2.3. Ali je pogodba verodostojna listina?

Pogodba sama po sebi ni verodostojna listina. Pogodba sama po sebi le določa pravice in obveznosti med strankama, vendar ne dokazuje neposredno, da obstaja zapadla denarna obveznost oziroma dolg, ki bi ga bilo mogoče neposredno izterjati v postopku izvršbe. Pogodba navadno določa okvir za prihodnje obveznosti, ki bodo šele nastale (npr. opravljena storitev, dobavljeno blago, dogovor o posojilu), in zato še ne pomeni, da je ena stranka dolžna drugi določeno vsoto denarja v določenem trenutku.

Da bi na podlagi pogodbe lahko izterjali denarno obveznost, bi morala biti izdana računa ali drug dokument, ki dokazuje, da je ena stranka dejansko opravila obveznosti iz pogodbe in da je drugi stranki dolg že zapadel. Drugače

3 COVL je kratica za Centralni oddelek za verodostojno listino, ki deluje v okviru slovenskih sodišč. Gre za specializirano enoto, ustanovljeno z namenom poenostavitve in pospešitve postopkov za izterjavo dolgov na podlagi verodostojne listine.

je seveda v primeru, ko je pogodba zapisana v obliki izvršilnega naslova s klavzulo neposredne izvršljivosti, o čemer bo govor v nadaljevanju prispevka.

Če obstajajo spori glede tega, ali je obveznost iz pogodbe nastala, se takšne zadeve rešujejo v pravdnem postopku, kjer se pogodba uporablja kot dokazno sredstvo. V pravdnem postopku se ugotavlja, ali so bile pogodbene obveznosti dejansko izpolnjene in ali terjatev obstaja.

3 Izvršba na podlagi izvršilnega naslova

Izvršilni naslov je kvalificirana javna listina, na podlagi katere je mogoče zahtevati prisilno izvršitev terjatve, ki je v njej ugotovljena.[4] Pogoj za izvršbo na podlagi izvršilnega naslova je izvršljivost listine. Sodna odločba je izvršljiva, če je postala pravnomočna in če je pretekel rok za prostovoljno izpolnitev dolžnikove obveznosti. Izvršbo dovoli sodišče tudi na podlagi sodne odločbe, ki še ni postala pravnomočna, če zakon določa, da pritožba ne zadrži njene izvršitve (19. člen ZIZ).

V skladu z določbo 17. člena ZIZ so izvršilni naslovi:

- izvršljiva sodna odločba[5] in sodna poravnava[6],
- izvršljiv notarski zapis,
- druga izvršljiva odločba ali listina, za katero zakon, ratificirana in objavljena mednarodna pogodba ali pravni akt Evropske unije, ki se v Republiki Sloveniji uporablja neposredno, določa, da je izvršilni naslov.

3.1. Postopek

V predlogu za izvršbo na podlagi izvršilnega naslova morajo biti navedeni (40. člen ZIZ):

- upnik in dolžnik z identifikacijskimi podatki iz 16.a člena ZIZ,
- izvršilni naslov,
- dolžnikova obveznost,
- sredstvo ali predmet izvršbe,
- drugi podatki, ki so glede na predmet izvršbe potrebni, da se izvršba lahko opravi.

Če upnik predlaga izpolnitev denarne obveznosti, mora navesti svoj transakcijski račun, na katerega naj se plačilo opravi. Če upnik predlaga izvršbo na premičnine, mu jih v predlogu za izvršbo ni treba natančno označiti, če ima dolžnik stalno prebivališče ali sedež v Republiki Sloveniji.

Če upnik predlaga izvršbo na dolžnikova sredstva pri organizacijah za plačilni promet, na nematerializirane vrednostne papirje oziroma na plačo, v predlogu za izvršbo ni dolžan navesti podatkov o organizaciji za plačilni promet, pri kateri ima dolžnik denarna sredstva, in številke računa, podatkov o nematerializiranih vrednostnih papirjih, katerih imetnik je dolžnik, oziroma dolžnikove zaposlitve. V tem primeru sodišče pred izdajo sklepa po uradni dolžnosti opravi poizvedbe o teh podatkih v elektronsko dosegljivih evidencah. Če iz elektronsko

4 Rijavec, Civilno izvršilno pravo, str. 105.
5 S sodno odločbo so po ZIZ mišljeni sodba oziroma arbitražna odločba, sklep ter plačilni oziroma drug nalog sodišča ali arbitraže, s sodno poravnavo pa poravnava, sklenjena pred sodiščem.
6 Določbe ZIZ o poravnavi se uporabljajo tudi za notarski zapis, če zakon ne določa drugače.

dosegljivih evidenc izhaja, da viri za predlagana izvršilna sredstva ne obstajajo in upnik ni predlagal drugih izvršilnih sredstev, ravna sodišče s takšnim predlogom za izvršbo kot z nepopolno vlogo.

Predlogu za izvršbo na podlagi izvršilnega naslova mora upnik priložiti izvršilni naslov in potrdilo o izvršljivosti v izvirniku ali prepisu, tujemu izvršilnemu naslovu pa listine, iz katerih izhaja, da so izpolnjeni pogoji iz 13. člena ZIZ.

Če upnik vloži predlog za izvršbo po pooblaščencu, pooblastila ni treba predložiti, ampak se v predlogu za izvršbo navede, da je bilo pooblastilo dano in v kakšnem obsegu[7].

V predlogu za izvršbo, pri kateri je treba opraviti neposredna dejanja izvršbe in zavarovanja, mora upnik navesti izvršitelja, ki naj ga sodišče določi za opravo teh dejanj, sicer sodišče s predlogom ravna kot z nepopolno vlogo.

Če upnik zahteva izvršbo s sredstvi ali predmeti izvršbe, pri katerih se v času vložitve predloga za izvršbo ne ve, ali bo treba določiti izvršitelja, upnik izvršitelja navede v določenem roku med postopkom po pozivu sodišča, v katerem ga sodišče opozori, da bo sklep o izvršbi v celoti ali delno s sklepom razveljavilo ali ne bo opravilo posameznega dejanja izvršbe, če v določenem roku ne bo navedel izvršitelja. Zoper sklep, s katerim se sklep o izvršbi v celoti ali delno razveljavi, ali se odloči, da se posamezno dejanje izvršbe ne opravi, ni pritožbe.

4 Sklep

Izvršba je pomemben pravni mehanizem, ki upnikom omogoča prisilno izterjavo terjatev, kadar dolžnik prostovoljno ne izpolni svojih obveznosti. Postopek izvršbe na podlagi verodostojne listine je zasnovan kot hitra in poenostavljena rešitev za nesporne terjatve, medtem ko je izvršba na podlagi izvršilnega naslova namenjena situacijam, kjer je že ugotovljena obveznost dolžnika s pravnomočno odločbo. Pomembno je, da so pred začetkom izvršilnega postopka izpolnjene vse zakonske predpostavke in da je ustrezno izbrano sredstvo izvršbe, saj to zagotavlja hitrejši in učinkovitejši potek postopka.

Ustavno načelo enakosti (22. člen Ustave RS) zahteva, da so v sodnem izvršilnem postopku varovani tako interesi dolžnika, kot tudi interesi upnika. ZIZ zato vsebuje vrsto določb, ki upniku omogočajo hitro in uspešno vodenje postopka, s končnim ciljem, da se poplača njegova terjatev. Prav tako je v izvršilnem postopku zagotovljeno varovanje dolžnikovih pravic.

Literatura

[1] RIJAVEC Vesna, Civilno izvršilno pravo, GV Založba, Ljubljana 2003.

[2] VOLK Dida, Izvršba: dolžnik je d.o.o. brez premoženja, Pravna praksa, št. 33/99.

[3] ZAKON O IZVRŠBI IN ZAVAROVANJU, Uradni list RS, št. 3/07 – uradno prečiščeno besedilo, 93/07, 37/08 – ZST-1, 45/08 – ZArbit, 28/09, 51/10, 26/11, 17/13 – odl. US, 45/14 – odl. US, 53/14, 58/14 – odl. US, 54/15, 76/15 – odl. US, 11/18, 53/19 – odl. US, 66/19 – ZDavP-2M, 23/20 – SPZ-B, 36/21, 81/22 – odl. US in 81/22 – odl. US..

[4] USTAVA REPUBLIKE SLOVENIJE, Uradni list RS, št. 33/91-I, 42/97 – UZS68, 66/00 – UZ80, 24/03 – UZ3a, 47, 68, 69/04 – UZ14, 69/04 – UZ43, 69/04 – UZ50, 68/06 – UZ121,140,143, 47/13 – UZ148, 47/13 – UZ90,97,99, 75/16 – UZ70a in 92/21 – UZ62a

[7] Ne glede na to je upnik dolžan pooblastilo takoj predložiti sodišču, če se pojavi upravičen dvom o njegovi verodostojnosti, sicer se šteje, da je predlog za izvršbo umaknil.

Kubernetes v omrežjih z omejenim dostopom

Benjamin Burgar, Uroš Brovč, Urban Zaletel

Kontron d.o.o, Kranj, Slovenija
benjamin.burgar@kontron.si, uros.brovc@kontron.si, urban.zaletel@kontron.si

Namestitev aplikacij na Kubernetes je preprosta, kadar imamo dostop do javnih ali privatnih registrov. Vendar se pri upravljanju Kubernetes gruče v omrežjih z omejenim dostopom pojavljajo izzivi. Kubernetes uporablja zabojnike, Helm chart-e in druge artefakte, ki jih med delovanjem pridobiva iz internetnih in privatnih registrov. Te artefakte moramo zagotoviti tudi v omrežjih z omejenim dostopom. Pri vzdrževanju več identičnih Kubernetes gruč v takšnih omrežjih se soočamo s problemom sledenja verzij aplikacij in njihove konfiguracije. V članku je predstavljen način namestitve aplikacij na Kubernetes gruče v omrežjih z omejenim dostopom. Prikazana je rešitve za prenos zabojnikov in Helm chart-ov v takšna okolja ter kako lahko brez centralnega sistema za verzioniranje zagotovimo deklarativno namestitev aplikacij. Poudarek je tudi na varnostnem pregledu vseh komponent rešitve in natančnemu sledenju in upravljanju vseh komponent našega sistema, kar je ključno za zanesljivo in ponovljivo namestitev rešitve na različna okolja.

Ključne besede:
Kubernetes
privatna omrežja
DevOps
Helm
zabojnik
registri programske opreme
5G

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.14

1 Uvod

V 5G privatnih omrežjih pogosto ni omogočenega dostopa do interneta zaradi varnostnega vidika podjetij, kjer se ta omrežja nameščajo. To pomeni, da ni mogoče uporabljati javnih registrov programske opreme, kar predstavlja velik izziv pri nameščanju in upravljanju aplikacij ter komponent, ki se zanašajo na te vire.

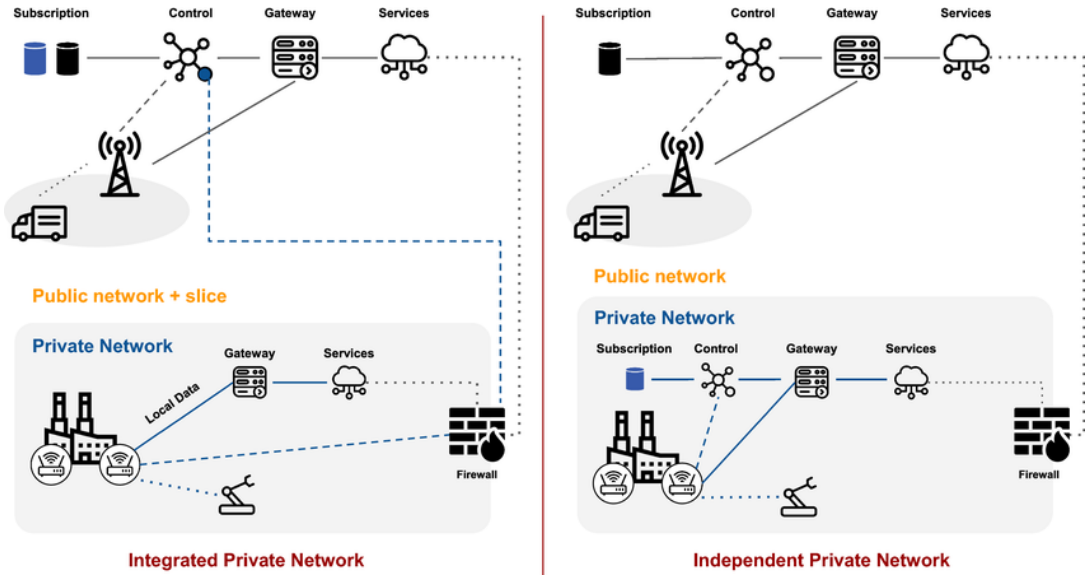
Da bi rešili ta problem, smo razvili rešitev, ki omogoča deklarativno generacijo programskih paketov, njihov prenos in namestitev pri stranki. Dodatno varnost zagotavljamo z rednimi varnostnimi pregledi vseh komponent rešitve pred generacijo paketa. Ta pristop zagotavlja, da so vse potrebne komponente na voljo tudi v primeru omejenega dostopa do interneta, s čimer se poenostavi upravljanje in vzdrževanje 5G privatnih omrežij.

2 5G privatna omrežja

2.1. Namen

5G privatna omrežja v podjetjih se uporabljajo zaradi več razlogov. Omogočajo hitrejše in bolj zanesljive komunikacijske povezave, kar je ključno za podjetja, ki potrebujejo visoko razpoložljivost in nizko latenco za svoje aplikacije in storitve. Privatna omrežja podjetjem nudijo večji nadzor in varnost nad njihovimi podatki ter omrežnimi operacijami. Poleg tega omogočajo prilagodljivost in prilagojene rešitve, ki ustrezajo specifičnim potrebam podjetja, kot so pametne tovarne, IoT naprave in avtomatizacija procesov.

5G privatna omrežja so lahko integrirana z javnim 5G omrežjem. Npr. Operater postavi javno omrežje in preko segmentacije omogoči ločevanje kontrolnega in uporabniškega prometa. Druga možnost je postavitev neodvisnega 5G privatnega omrežja, v določenih primerih z omejenim dostopom.



Slika 1: Primerjava integriranega in neodvisnega 5G privatnega omrežja. [1]

2.2. Omrežja z omejenim dostopom

Za podjetja je ključnega pomena, da z uvajanjem 5G privatnega omrežja ne ogrozijo varnostne politike. Pogosta zahteva je, da se v 5G omrežju, kjer se nameša rešitev, ne omogoča internetnega dostopa. To zagotavlja zaščito pred zunanjimi grožnjami, kot so hekerski napadi in nepooblaščeni dostopi, kar je še posebej kritično za podatkovno občutljiva podjetja. Še posebno občutljiva so OT omrežja, ta omrežja so največkrat povsem ločena od preostale IT infrastrukture. Onemogočanje internetnega dostopa v 5G privatnih omrežjih pomaga ohranjati integriteto podatkov in omrežne infrastrukture ter zagotavlja, da so podatki podjetja ustrezno zaščiteni in varni.

3 Namestitev rešitve v omrežju z omejenim dostopom

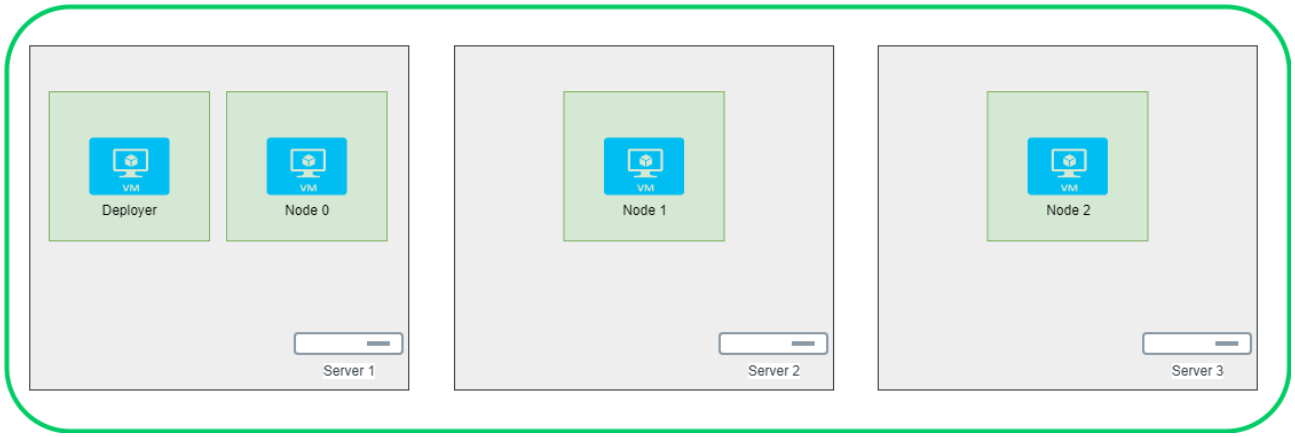
Če na lokaciji, kjer želimo namestiti rešitev, ni dostopa do interneta, je potrebno zagotoviti prenos vseh artefaktov in namestitev lokalno. V primeru, ko izvajamo celotno namestitev od inštalacije strežnikov do končne rešitve je potrebno zagotoviti zbiranje in pakiranje tudi za ostale komponente rešitve (operacijski sistem, Kubernetes[8], itd). V nadaljevanju se bomo bolj podrobno posvetili nameščanju aplikacije na Kubernetes gručo in specifiki pri omejenem dostopu.

Za zagotavljanje lokalne inštalacije je pomembno, da že med pripravo rešitve oziroma paketa zberemo vse potrebne komponente, kot so različne skripte, slike zabojnikov, Helm chart-i[9], itd. Več o tem v poglavju 4.

3.1. Okolje in način namestitve

Opis inštalacije je podan za okolje, kjer imamo že postavljene strežnike z ustreznim operacijskim sistemom. Podan je primer za HA[10] postavitev naše 5G rešitve.

Za namen namestitve in upravljanja rešitve se na enem izmed strežnikov kreira navidezna naprava za nameščanje rešitve (Deployer). Za namestitev Kubernetes gruče se na vsakem strežniku kreira navidezno napravo (Node 0, Node 1 in Node 2).

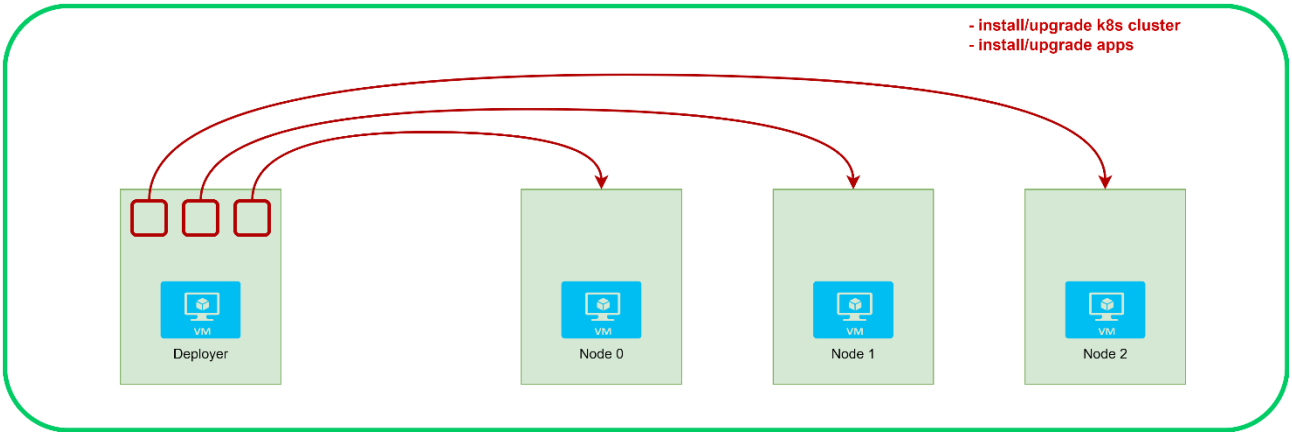


Slika 2: Virtualne naprave.

Namestitev Kubernetes gruče se izvede preko navidezne naprave Deployer. Za ta namen so ustvarjene avtomatizacijske skripte, ki zagotavljajo kontroliran postopek namestitve na različnih objektih oziroma lokacijah.

[8] https://kubernetes.io/
[9] https://helm.sh/docs/topics/charts/
[10] HA – izvorno »High Availability«, Visoka razpoložljivost

Po uspešno nameščeni in konfigurirani Kubernetes gruči se po točno določenem postopku izvede namestitev aplikacij. Tudi v tem primeru se namestitev izvaja preko navidezne naprave Deployer. Avtomatizacijske skripte poskrbijo za enostaven in ponovljiv proces namestitve, kar zmanjšuje možnosti napak in zagotavlja doslednost.



Slika 3: Namestitev k8s in aplikacij.

Pri omenjenem postopku namestitve je predvideno, da imamo dostop do interneta oziroma do javnih in zasebnih registrov. Vse artefakte se pridobi iz omenjenih registrov med izvedbo inštalacije. V primeru omejenega dostopa to ni omogočeno, zato je bilo potrebno ustrezno dopolniti oziroma spremeniti postopek inštalacije. Dopolnitev postopka je opisana v poglavju 3.2.

Ključno je tudi, da se na lokaciji namestitve omogoči pregled verzije nameščenih komponent in uporabljene konfiguracije. To nam omogoča pregled podprtih oz. omogočenih funkcionalnosti, pomaga pri reševanju napak s strani razvoja (na kateri verziji programske opreme se je napaka pojavila, kakšna je konfiguracija), pri posodabljanju verzij, itd. Več o tem v poglavju 3.5.

3.2. Lokalna namestitev brez internetnega dostopa

V primeru lokalne namestitve brez dostopa do interneta je potrebno prenesti vse zahtevane pakete preko medija (npr. USB ključka, prenosnika) in jih shraniti na navidezno napravo za namestitev rešitve (Deployer). Seznam paketov in sami paketi so pripravljeni med pripravo programske opreme za posamezno verzijo rešitve, kot je opisano v poglavju 4. Paketi so v našem primeru pripravljeni na centralnem registru artefaktov (Nexus[11]).

Za varen prenos paketov in tar datotek preko USB medija je pomembno zagotoviti, da datoteke ostanejo nespremenjene. To dosežemo z uporabo digitalnih podpisov, ki nam omogočajo, da preverimo, ali so datoteke ostale nedotaknjene in niso bile spremenjene ali kompromitirane med prenosom.

Ko so na voljo oz. prenešeni vsi paketi, je potrebno namestiti lokalni OCI[12] register, v kolikor še ni nameščen, na navidezno napravo Deployer. Namestitev se izvede s skripto, ki je vključena v prej omenjenih paketih. Glej poglavje 3.3 za dodaten opis OCI registra.

V nameščen lokalni OCI register se prenese artefakte iz prenesenega paketa (tar datoteka) na navidezni napravi Deployer. Za ustrezen uvoz paketa (oz. vseh artefaktov v paketu) v register se uporabi lastno orodje (regcli), ki nam poenostavi oz. omogoči različne operacije nad lokalnih registrom. Združevanje in kopiranje artefaktov med različnimi registri ali datotekami je opisano v poglavju 3.4.

[11] https://www.sonatype.com/products/sonatype-nexus-repository
[12] OCI – Open Container Initiative

Namestitveni postopek za Kubernetes gručo in kasneje za aplikacije je od te točke dalje podoben kot v primeru dostopa do interneta. Ključna razlika je, da se vsi zahtevani artefakti iščejo v lokalnem registru in ne več v javnem ali privatnem registru preko internetne povezave.

V kolikor se za Kubernetes runtime uporablja containerd[13], ni potrebno izvajati dodatnih sprememb na aplikaciji oz. v Helm chart-ih. Namreč konfiguracija containerd za registre omogoča preslikavo naslovov. Praktično to pomeni, da se v konfiguraciji zapiše na kakšen naslov se preslikajo določeni naslovi za javne registre. Na primer, naslednja konfiguracija bi potegnila sliko iz lokalnega registra (primer naslova: https://registry.example.com:5000) v primeru uporabe imena library/busybox, kot tudi registry.example.com/library/busybox:latest:

```
mirrors:
  docker.io:
    endpoint:
      - "https://registry.example.com:5000"
  registry.example.com:
    endpoint:
      - "https://registry.example.com:5000"
```

Slika 4: Containerd konfiguracija.

V kolikor se uporablja docker runtime, je potrebno to upoštevati tudi pri aplikacijah in omogočiti spremembo naslovov za registre v samem Helm chart-u med inštalacijo (možnost konfiguracije v values.yaml).

3.3. OCI register

OCI (Open Container Initiative) register se uporablja za standardizirano shranjevanje, distribucijo in upravljanje slik zabojnikov in drugih artefaktov, kot so Helm chart-i. Zagotavlja interoperabilnost med različnimi orodji in platformami, omogoča varno shranjevanje in prenos podatkov ter podpira avtomatizirane procese neprekinjene integracije in dostave (CI/CD).

Obstaja več odprtokodnih izvedb registrov, ki ustrezajo OCI specifikaciji, kot na primer: Distribution, Harbor, Zot, itd.

V našem primeru smo se odločili za Distribution register[14]. V register se shranjuje slike zabojnikov in Helm chart-e. Lokalni register Distribution teče v zabojniku na navidezni napravi Deployer. Za pregled, dodajanje ali brisanje artefaktov iz registra se uporablja lastno razvito orodje (regcli), ki komunicira z registrom preko HTTP API-ja (posredno tudi preko skopeo orodja).

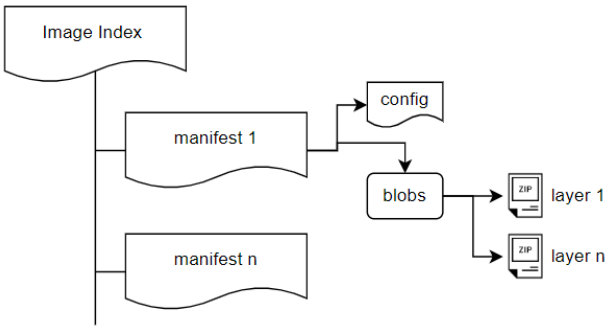
V registru so vsi artefakti strukturirani, kot je določeno v specifikaciji za OCI slike[15]. S tem se zagotavlja standardiziran format za deljenje, distribucijo in izvajanje aplikacij preko različnih platform in orodij.

Specifikacija slike Open Container Initiative (OCI) določa format in strukturo slik za zabojnike. Glavni deli specifikacije OCI slike vključujejo:

- **Postavitev slike (Image Layout)**: Opisuje strukturo datotečnega sistema, ki vsebuje OCI sliko. To vključuje organizacijo slike, datoteke in imenike, ki jih vsebuje, ter pravila za shranjevanje vsebine in metapodatkov slike.
- **Manifest slike (Image Manifest)**: JSON datoteka, ki opisuje komponente slike, kot so sloji, konfiguracija in specifične podatke za platformo. Vključuje:
    - **Config**: Sklic na konfiguracijski objekt, ki opisuje nastavitve delovanja kontejnerja.
    - **Layers**: Urejen seznam sklicev na sloje slike, ki vsebujejo dejansko vsebino datotečnega sistema.

[13] https://containerd.io/
[14] https://github.com/distribution/distribution/
[15] https://github.com/opencontainers/image-spec

- **Kazalo slike (Image Index)**: Izbirna JSON datoteka, ki podpira slike za več platform. Vsebuje sklice na več manifestov slik, kar omogoča podporo za različne arhitekture ali operacijske sisteme.
- **Konfiguracija slike (Image Configuration)**: JSON objekt, ki določa konfiguracijo delovanja kontejnerja, vključno s podrobnostmi, kot so vstopna točka, okoliške spremenljivke, delovni imenik in omejitve virov.
  - **Sloj (Layer)**: Predstavlja del datotečnega sistema zabojnika. Sloji so zloženi en na drugega, da ustvarijo celoten datotečni sistem za zabojnik. Vsak sloj je stisnjeni tar arhiv.
  - **Deskriptorji (Descriptors)**: Objekti, ki opisujejo vsebino (kot so sloji, konfiguracije itd.) znotraj slike, vključno z lastnostmi, kot so medijski tip, velikost in digest[16].
  - **Opombe (Annotations)**: Ključ-vrednost pari, ki zagotavljajo dodatne metapodatke za slike, manifeste in konfiguracije. Opombe lahko vključujejo informacije kot so avtor, opis in različica.
  - **Medijski tipi (Media Types)**: Specifikacije za vrste vsebine, ki so lahko vključene v OCI sliko, kot so medijski tipi slojev, konfiguracij in manifestov. To zagotavlja skladnost pri interpretaciji podatkov.

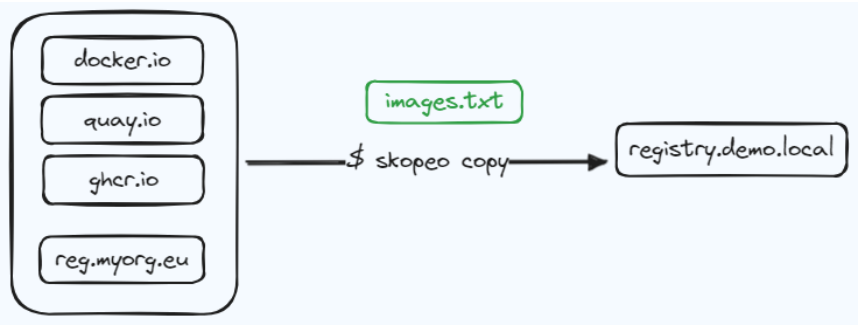


Slika 5: Struktura OCI slike.

3.4. Kopiranje artefaktov med različnimi registri

Prenos paketov oz. artefaktov na določeno oddaljeno lokacijo oz. v lokalen register lahko izvedemo na dva načina.

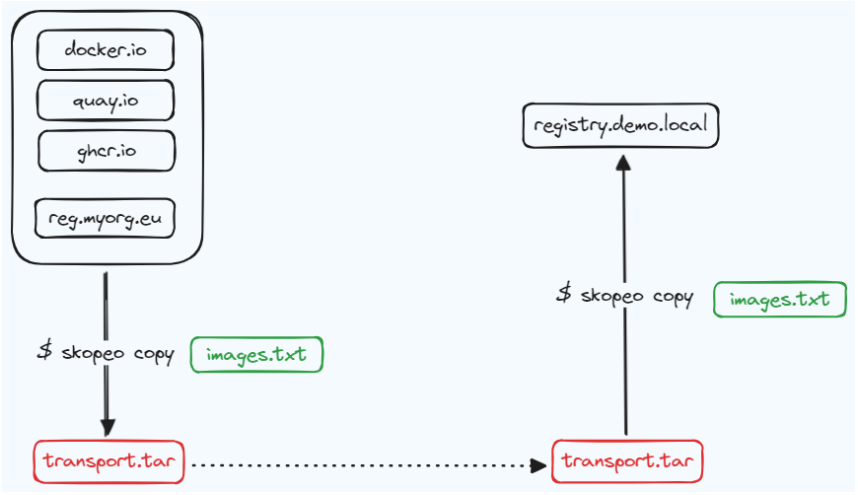
Preko varne (VPN) povezave iz privatnega in ostalih javnih registrov v lokalen register na lokaciji, kjer se bo nameščalo rešitev.



Slika 6: Prenos slik zabojnikov iz različnih registrov v lokalen OCI register.

Pakete se lahko prenese tudi z različnimi mediji oz. napravami (USB ključek, prenosnik). Pred prenosom se izvede pakiranje artefaktov, nato prenos in na koncu odpiranje in vključevanje artefaktov v lokalen register. Omenjen način je uporabljen v primeru omejenega dostopa do interneta.

[16] digest - kriptografski hash, ki zagotavlja celovitost vsebine

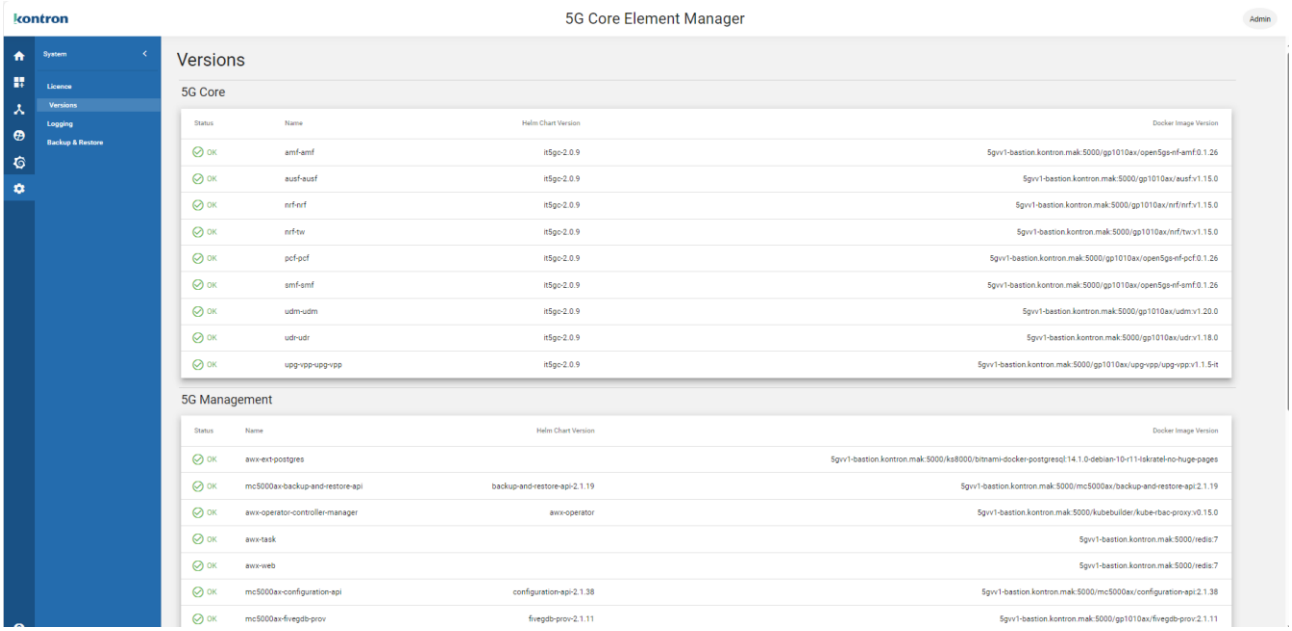


Slika 7: Pakiranje, prenos in odpiranje paketa v lokalen OCI register.

Za prenos OCI artefaktov obstajajo razna odprtokodna orodja. Za prenos docker slik lahko uporabljamo skopeo, za prenos Helm chart-ov lahko uporabljamo helm. Obstajajo tudi orodja, ki poenotijo prenos obeh vrst artefaktov, na primer projekta Crane[17] in Mindthegap[18].

3.5. Izpis uporabljenih komponent rešitve

Skupek verzij nameščenih aplikacij je vodeno preko verzije rešitve. Posamezna verzija rešitve določa točno katere komponente so del te rešitve. Za hitro preverjanje verzije 5G rešitve oz. uporabljenih aplikacij in komponent smo na rešitvi omogočili priročen pregled preko uporabniškega vmesnika.



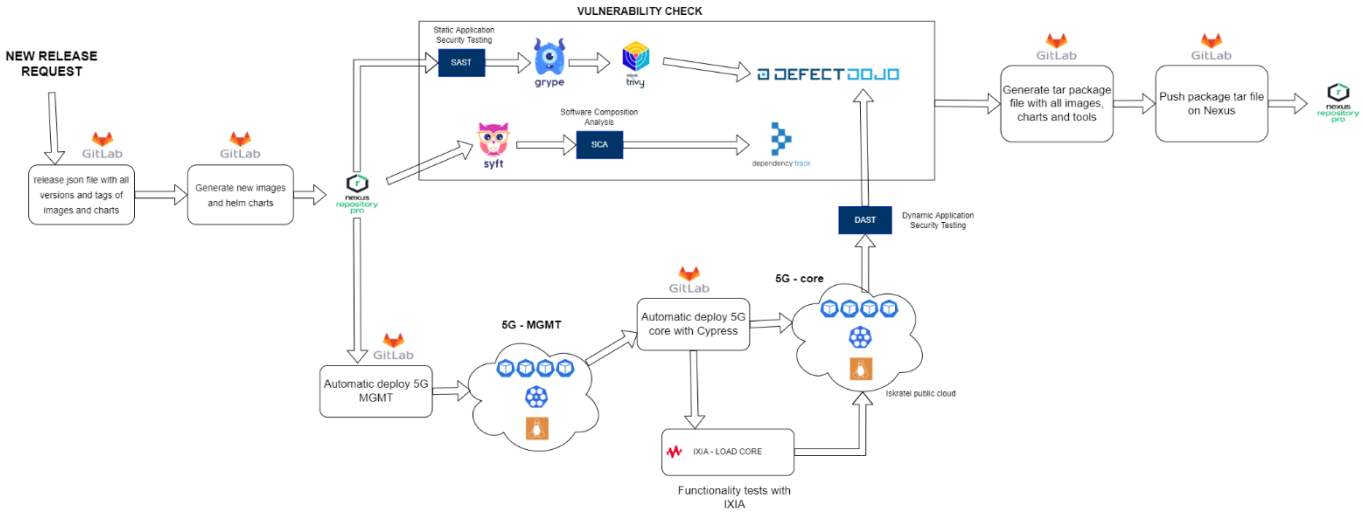
Slika 8: Pregled verzij preko uporabniškega vmesnika.

Verzijo rešitve ter posameznih komponent se lahko preveri tudi preko navidezne naprave Deployer z uporabo helm ukazov (helm list, helm show values).

[17] https://github.com/google/go-containerregistry/tree/main/cmd/crane
[18] https://github.com/mesosphere/mindthegap

4 Priprava programske opreme

Za zagotovitev sledljivosti rešitve in vseh vključenih komponent ter zanesljive generacije novih paketov, smo omogočili avtomatizirano generacijo in objavo verzij rešitve. Z ustreznimi CI/CD postopki oz. cevovodom se izvede preverjanje, generacija in obveščanje uporabnikov o novi verziji rešitve. Zahtevek za novo verzijo rešitve se proži ročno. Pred zagonom cevovoda je potrebno določiti verzije vseh komponent ter verzijo rešitve.



Slika 9: Cevovod za novo verzijo rešitve.

V primeru uspešno izvedenega cevovoda, pridobimo pakete za novo verzijo rešitve ter poročilo s seznamom vseh vključenih komponent, povezavami do vseh kreiranih paketov in skript, seznam rešenih oz. dodanih funkcionalnosti na rešitvi, inštalacijska navodila, uporabniška navodila, itd.

V cevovodu se preverja tudi varnostne ranljivosti. V primeru ugotovljenih večjih ranljivosti se smatra cevovod kot neuspešen kar pomeni, da verzija rešitve ni ustrezna za objavo. Več o varnostnem preverjanju v poglavju 4.1.

4.1. Varnostno preverjanje

GitLab CI/CD, Grype, Trivy, Syft, DefectDojo, Dependency Track, IXIA in UERANSIM so ključna orodja, ki nam omogočajo zagotavljanje najvišje ravni varnosti in kakovosti kode. Osrednje načelo DevSecOps[19] je integracija varnostnih preverjanj v celoten CI/CD[20] proces, kar zagotavlja, da varnost ni zgolj dodatek, temveč bistven del razvoja.

V cevovodu se izvaja statično analizo kode, izdelavo SBOM in teste ranljivosti na slikah zabojnikov, kar nam omogoča hitro odkrivanje in odpravljanje najnovejših varnostnih pomanjkljivosti, kot jih določa CVE[21]. Za zagotavljanje celovite varnosti se nenehno spremlja, dokumentira in odpravlja vse odkrite ranljivosti.

Na začetku cevovoda izvedemo popis programske opreme SBOM[22], kar nam praktično predstavlja inventar programske opreme. Nujna je pridobitev polne oblike SBOM, kar omogoča, da ranljivosti ugotovimo takoj. Obstaja več standardnih formatov za izmenjavo SBOM, kar omogoča predvsem interoperabilnost med različnimi orodji. V našem primeru uporabljamo format CycloneDX. Z izdelavo SBOM si zagotovimo natančno sledenje in upravljanje vseh komponent našega sistema.

[19] DevSecOps – Development, Security and Operations
[20] CI/CD – Continuous Integration/Continuous Delivery
[21] https://cve.mitre.org/
[22] SBOM – Software Bill Of Materials

Za statično analizo Golang programske kode uporabljamo GOLANGCI-LINT, za analizo kode drugih programskih jezikov (kot so Java in C) pa uporabljamo integrirano orodje v GitLab-u[23]. Statično analizo kode se izvaja ob vsakem potisku kode na GitLab repozitorij.

Skeniranje ranljivosti in pregledovanje slik vsebnikov izvajamo z orodjem GRYPE katero pridobi vse potrebne varnostne podatke iz svoje zaledne oblačne storitve. Slika vsebnika lahko vsebuje hrošče in varnostne ranljivosti, ki se lahko skozi proces gradnje dolgoročno ponavljajo in s tem povečajo tveganje za vse različice pakirane programske opreme. Posledično je ključnega pomena izvajanje rednega pregleda slik vsebnikov in zgodnja odprava težav. Poleg orodja GRYPE za pregledovanje slik in vsebnikov izvajamo pregledovanje tudi z orodjem Aqua Trivy, ki je zelo podobno orodju GRYPE. Glavna razlika med orodji je, da slednje uporablja lastno kompaktno bazo, ki se lahko redno posodablja.

Za upravljanje z varnostnimi ranljivostmi uporabljamo Defect Dojo, kateri je kot centralna baza ranljivosti. Poenostavi postopek njihovega spremljanja skozi proces razvoja, delegiranja odgovornim osebam in njihove odprave oz. zapiranje posamezne ranljivosti. Omogoča uvoz varnostnih poročil iz zunanjih orodij in sistemov, združevanje in odstranjevanje dvojnikov ter ustvarjanje poročil. DefectDojo nadgradi postopek upravljanja ranljivosti prek več internih modelov, ki jih je mogoče manipulirati s programsko kodo. Osnovni modeli vključujejo: »engagements«, »tests« in »findings«, na voljo pa je še več modelov, ki olajšajo spremljanje metrik, avtentikacijo in ustvarjanje poročil.

OWASP DependencyTrack je platforma za analizo komponent, ki organizacijam omogoča prepoznavanje in zmanjšanje tveganja v dobavni verigi programske opreme. DependencyTrack uporablja pristop sledenja odvisnosti na osnovi SBOM. Orodje samo po sebi ne ustvarja SBOM-ov, ampak predstavlja platformo oz. podatkovno bazo, ki omogoča zbiranje in analizo SBOM-ov, iskanje po gradnikih in verzijah, sledenje odprtokodnih licenc, ipd. S tem omogoča hitro ukrepanje ob varnostnih incidentih.

5 Zaključek

V prispevku smo opisali postopek namestitve 5G jedrnega omrežja v okolju brez dostopa do interneta. Ključni poudarek je na pomenu sledenja uporabljenim komponentam, varnostnem preverjanju artefaktov, varnem prenosu artefaktov na končno lokacijo in zagotavljanju deterministične namestitve. Varni prenos artefaktov zagotavlja, da so vse nameščene komponente preverjene in da ni prišlo do kompromitacije med transportom. Poleg tega potrebno imeti dober pregled nad nameščenimi verzijami komponent in konfiguracijami, saj to omogoča učinkovito upravljanje in vzdrževanje sistema, ter hitrejšo identifikacijo težav oziroma potrebnih posodobitev na posamezni postavitvi. Z upoštevanjem teh praks lahko zagotovimo stabilno in varno delovanje 5G jedrnega omrežja, tudi v okoljih, kjer dostop do interneta ni mogoč. Zanesljivost in robustnost omrežja je ključno za podporo sodobnim aplikacijam in storitvam, ki temeljijo na 5G tehnologiji.

Literatura

[1] https://www.researchgate.net/figure/Integrated-vs-Independent-Private-5G-Networks_fig5_365608799, Integrated vs Independent Private 5G, obiskano 8.7.2024

[2] https://docs.rke2.io/install/containerd_registry_configuration, Containerd Registry Configuration, obiskano 10.7.2024

[3] https://ravichaganti.com/blog/2022-10-28-understanding-container-images-oci-image-specification/, Understanding container images - OCI image specification, obiskano 11.7.2024

[23] Gitlab static application security testing - https://docs.gitlab.com/ee/user/application_security/sast/

Vpeljava sistema za politiko dostopa v avtorizacijski proces obstoječega sistema

Klemen Drev, Mitja Krajnc, Boris Ovčjak

Databox, Ptuj, Slovenija
klemen.drev@databox.com, mitja.krajnc@databox.com, boris.ovcjak@databox.com

V našem podjetju smo se soočali z izzivi pri avtorizaciji poslovne logike, kar je vplivalo na učinkovitost in varnost naših storitev. Da bi izboljšali ta proces, smo raziskali različne pristope in orodja za avtorizacijo ter se osredotočili na enostavnost in hitrost vpeljave. Odločili smo se za rešitev z uporabo Open Policy Agent (OPA), ki jo bomo predstavili v članku. OPA je odprtokodno orodje za enotno politiko avtorizacije, ki omogoča centralizirano izvajanje politik. Politike dostopa v OPA so napisane v jeziku Rego, kar omogoča fleksibilno pisanje pravil za avtorizacijo. Predstavili bomo tudi OPAL (OPA Policy Administration Layer), ki omogoča dinamično posodabljanje politik dostopa brez prekinitev delovanja storitev. OPA v naši infrastrukturi deluje kot Policy Decision Point (PDP), odgovoren za sprejemanje odločitev glede avtorizacije. Uporabljamo decentraliziran pristop, kjer je PDP implementiran kot "sidecar" ob posameznih storitvah, kar zagotavlja hitrejše odzivne čase in boljšo porazdelitev obremenitve. V članku bomo predstavili tudi spremembe in izzive pri razvoju sistema, saj je ločeno pisanje avtorizacijskih politik sprva povzročalo nekaj preglavic, predvsem v obliki pozabljenih politik dostopa in zavrnjenih zahtevkov.

Ključne besede:
Avtorizacija
Open Policy Agent
OPA
Rego
OPAL Server

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.15

1 Uvod

V hitro razvijajočem se svetu digitalne varnosti je avtorizacija postala ključen, a hkrati zahteven del za organizacije, ki si prizadevajo zaščititi svoje podatke in sisteme. Za razliko od avtentikacije, ki zgolj preveri identiteto uporabnika, avtorizacija določa, kaj je avtoriziran uporabnik dejansko upravičen storiti, kar predstavlja bolj zapleten proces, ki je tesno povezan s poslovno logiko. Zapletenost upravljanja dovoljenj čez različne aplikacije in storitve lahko privede do znatnih varnostnih ranljivosti in operativnih neučinkovitosti.

V podjetju smo se odločili za izboljšavo obstoječega procesa, zato smo izvedli raziskavo o avtorizaciji in naleteli na številne skupne težave, s katerimi se soočajo organizacije. Pogosto smo se soočali s prepletanjem politik, ki so bile težko obvladljive in so zahtevale veliko časa za vzdrževanje. Prav tako je bila problematična heterogenost sistemov, ki so potrebovali različne pristope k avtorizaciji, kar je dodatno zapletlo že tako kompleksno področje.

Želeli smo rešitev, ki bi bila hitra, učinkovita in enostavna za vzdrževanje. Potrebovali smo sistem, ki bi omogočal centralizirano upravljanje politik, a hkrati dovoljeval dovolj fleksibilnosti za prilagajanje specifičnim potrebam posameznih aplikacij in storitev. Naša raziskava nas je pripeljala do inovativnih rešitev, kot je Open Policy Agent (OPA).

OPA je odprtokodni, splošno namenski sistem za upravljanje politik, ki ločuje sprejemanje odločitev o politiki od aplikacijske kode, kar zagotavlja bolj prilagodljiv in razširljiv pristop k avtorizaciji. S pomočjo OPA lahko podjetja uveljavljajo natančne kontrole dostopa, poenostavijo upravljanje politik in izboljšajo svojo splošno varnostno držo. Poleg tega je OPA zasnovan tako, da omogoča hitro implementacijo in enostavno vzdrževanje, kar je bilo ključno za naše potrebe.

Raziskave so pokazale, da OPA omogoča centralizirano upravljanje politik, kar poenostavlja njihovo implementacijo in vzdrževanje [1]. Poleg tega so odprtokodne specifikacije OPA dostopne na GitHub [2], kjer je mogoče najti podrobne informacije o njegovi uporabi in implementaciji.

V tem članku bomo raziskali te težave z avtorizacijo in razložili, kako OPA revolucionira način, kako podjetja obvladujejo nadzor dostopa v sodobnih, oblačno orientiranih okoljih.

2 Open Policy Agent

Open Policy Agent (OPA) omogoča bolj prilagodljiv in razširljiv pristop k avtorizaciji, saj lahko politike definirate in posodabljate neodvisno od aplikacij. OPA omogoča centralizirano upravljanje politik, kar olajša njihovo implementacijo in vzdrževanje. Politike dostopa se znotraj opa definirajo s posebnih jezikom Rego, ki ga bomo predstavili kasneje.

OPA deluje tako, da sprejema zahteve za avtorizacijo od aplikacij in na podlagi definiranih politik vrne odločitve. Politike so zapisane v jeziku Rego, ki omogoča enostavno in razumljivo definiranje kompleksnih pravil. OPA je zasnovan tako, da se lahko integrira z različnimi sistemi in platformami, kar omogoča njegovo uporabo v različnih okoljih, od mikro storitev do monolitnih aplikacij.

2.1. Jezik Rego

Rego jezik je osrednji del Open Policy Agent (OPA) [3], ki omogoča pisanje politik na jasen in strukturiran način. Rego je bil zasnovan z mislijo na fleksibilnost in ekspresivnost, kar omogoča definiranje kompleksnih pravil za avtorizacijo, dostop in druge politike, ki temeljijo na podatkih.

Osnovne Lastnosti Rego Jezika

1. **Deklarativni Pristop**: Rego je deklarativni jezik, kar pomeni, da definirate, kaj želite doseči, ne da bi določali, kako naj se to izvede. Ta pristop poenostavlja pisanje in vzdrževanje politik, saj se osredotočate na želeni rezultat.
2. **Zmogljivost in Fleksibilnost**: Rego omogoča pisanje zelo zmogljivih politik, ki lahko vključujejo kompleksno logiko in več nivojske pogoje. Jezik podpira različne podatkovne tipe, vključno s seznami, nizi, slovarji in množicami, kar omogoča širok spekter uporab.
3. **Vgrajene Funkcije**: Rego vključuje številne vgrajene funkcije za manipulacijo podatkov, kot so filtriranje, združevanje in transformacija podatkov. To omogoča, da politike obdelujejo in analizirajo podatke na različne načine, kar povečuje njihovo uporabnost.
4. **Enostavno Branje in Pisanje**: Sintaksa Rego jezika je zasnovana tako, da je enostavna za branje in pisanje. To omogoča hitrejše pisanje politik in lažje razumevanje obstoječih pravil, kar zmanjšuje možnosti za napake.

Primer Rego Politike

Na spodnjem izseku kode je preprost primer Rego politike, ki preverja, ali ima uporabnik dovoljenje za dostop do določenega vira:

```
package profile.authz
default allow = false
allow {
    input.user == "admin"
}
allow {
    input.user == "user"
    input.action == "read"
}
```

V zgornjem primeru politika določa, da ima dostop uporabnik "admin" za vse akcije, medtem ko ima uporabnik "user" dovoljenje le za branje. Ta politika je preprosta, a prikazuje, kako lahko z Rego jezikom definirate jasna in razumljiva pravila.

Integracija z OPA

Politike napisane z Rego so shranjeni v OPA in se uporabljajo za odločanje v realnem času. Ko aplikacija pošlje zahtevo za avtorizacijo, OPA uporabi Rego politiko za obdelavo zahteve in vrne odločitev. Ta pristop omogoča centralizirano in dosledno upravljanje politik po celotni infrastrukturi.

Rego jezik omogoča organizacijam, da hitro in učinkovito razvijajo ter uvajajo politike, ki so ključne za varnost in skladnost njihovih sistemov. Zahvaljujoč svoji fleksibilnosti in zmogljivosti je Rego postal pomembno orodje za upravljanje politik v sodobnih IT okoljih.

2.2. Podatki

Upravljanje politik z OPA pogosto zahteva ustrezne kontekstualne podatke – informacije o uporabniku, viru, do katerega poskuša dostopati, itd. Brez teh informacij OPA ne bo mogla sprejeti pravih odločitev, ko gre za odločanje o politikah.

Na primer – pravilnik, ki pravi, da »Do te funkcije lahko dostopajo samo uporabniki, ki plačujejo storitev«, zahteva, da ima OPA informacije o vseh uporabnikih sistema in kateri od njih imajo poravnane obveznosti.

Bistveno vprašanje je - kako lahko te podatke prenesemo v OPA in kateri način je najučinkovitejši za to?

Načini zagotavljanja podatkov do politik:

- **Podatki so del veljavnih JWT (JSON Web Token).** To je enostavna in dobro poznana tehnologija, ki je del avtentikacijske plasti. Slabost tega načina je, da ima JWT omejitev velikosti - vsega ni mogoče dekodirati v JWT. Slabost je tudi, da za posodobitev podatkov je potrebno osvežiti žeton oz. ponovna prijava.
- **Predložitev podatkov ob poizvedbi politiki.** Tukaj kršimo najbolj osnovno dobro prakso grajenja avtorizacije - ločevanje politik od izvorne kode.
- **Zbiranje podatkov z uporabo paketov.** To nam omogoča pridobivanje veliko količino podatkov iz centralnega strežnika. Problem nastane ko se nam podatki hitro spreminjajo in moramo zaradi malih sprememb posodobiti celotne pakete.
- **Potiskanje podatkov v OPA z uporabo API (Application Programming Interface)** Podobno kot zbiranje podatkov z uporabo paketov, vendar lahko optimiziramo latenco posodabljanja in imamo veliko možnosti kako pripeljati podatke do politik. Slabost razvoj in je vzdrževanje takšnega sistema, ki skrbi za pravilno posodabljanje podatkov.
- **Pridobivanje podatkov ob evaluaciji politik.** OPA znotraj jezika Rego nam omogoča klicanje eksternih storitev http.send(), kar je solidna možnost pri ogromni količini podatkov. Slabost pa je, da vsako poizvedbo oziroma odločitev dodamo neko latenco omrežja.
- **OPAL (Open Policy Administration Layer).** Omogoča sinhronizacijo podatkov in politik ter odpravi slabosti načina s potiskanjem podatkov preko OPA API. Slabost tega načina je da je relativno nova tehnologija in je potreben določen čas učenja in vpeljave v obstoječ sistem.

3 OPAL Strežnik

OPAL (Open Policy Administration Layer) [4] strežnik deluje kot razširitev za OPA, ki avtomatizira distribucijo in sinhronizacijo politik ter podatkovnih posodobitev. OPAL omogoča, da OPA vedno deluje s posodobljenimi politikami in podatki, kar je ključno za zagotavljanje natančnih in aktualnih avtorizacijskih odločitev.

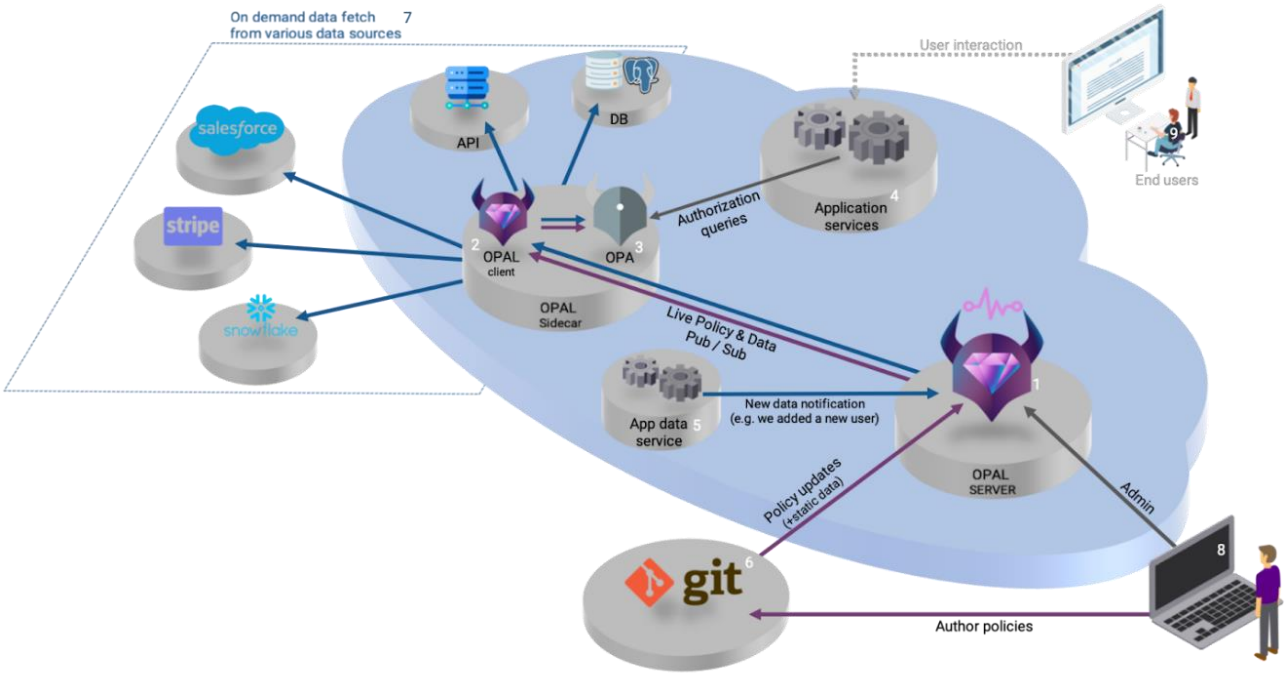
OPAL deluje tako, da spremlja spremembe v virih politik ter podatkih (npr. Git repozitorijih, bazah podatkov) in sproži posodobitve v OPA instancah. To zagotavlja, da so vse OPA instance sinhronizirane z najnovejšimi politikami, brez potrebe po ročnem posodabljanju.

3.1. Arhitektura rešitve z OPAL

OPAL vsebuje dve ključni komponenti, ki delujeta skupaj in sta predstavljeni na spodnji sliki. To sta OPAL strežnik in OPAL odjemalec, ki sta skupaj s preostalimi komponentami vrisani v komponenti diagram arhitekturne rešitve na naslednji sliki (slika 1). Zaradi pomembnosti sta tudi ti komponenti predstavljeni najvidneje. Celotna komunikacija je predstavljena s dvema barvama in sicer predstavljajo modre črte podatkovne tokove, medtem ko so vijolične črte poteki tokov politik.

- **OPAL Strežnik**
  - Ustvarja komunikacijske kanale do odjemalcev.
  - Spremlja spremembe avtorizacijskih politik na Git repozitorijih in potiska posodobitve odjemalcem.
  - Podatke posodablja kot razlike, ne v celoti.

- **OPAL Odjemalec**
  - Leži poleg OPA in skrbi za ažurnost politik ter podatkov.
  - Pridobiva podatke iz različnih virov.
  - Prenaša politike od strežnika.



Slika 1: Komponentni diagram arhitekture OPAL, pri čemer modre črte predstavljajo podatkovne tokove, vijolične črte pa predstavljajo toke politik. [5]

3.2. Možne Implementacije

1. OP **Centralizirana Implementacija**: OPA je lahko nameščen kot centraliziran strežnik, ki sprejema zahteve od več aplikacij. Ta pristop je primeren za manjše sisteme ali sisteme, kjer je latenca manj kritična. Centralizirana implementacija omogoča enostavno upravljanje in distribucijo politik, vendar lahko predstavlja ozko grlo pri visokih obremenitvah.
2. **Vgrajena Implementacija**: OPA se lahko vgradi neposredno v aplikacijsko kodo, kar omogoča tesno integracijo in zmanjšuje potrebo po dodatni infrastrukturi. Vgrajena implementacija je še posebej uporabna v monolitnih aplikacijah, kjer je mogoče hitro in enostavno dodati avtorizacijske kontrole neposredno v obstoječo kodo.
3. **Kubernetes Admission Controller**: V Kubernetes okolju se OPA lahko uporablja kot Admission Controller, ki preverja in uveljavlja politike ob ustvarjanju ali posodabljanju Kubernetes virov. Ta pristop omogoča centralizirano upravljanje politik na ravni celotnega Kubernetes grozda, kar zagotavlja skladnost in varnost na vseh ravneh infrastrukture.

3.3. Implementacija kot Sidecar

V našem podjetju smo se odločili za implementacijo OPA kot sidecar v naši mikrostoritveni arhitekturi. Sidecar vzorec pomeni, da vsaka mikro storitev teče skupaj z OPA instanco, kar omogoča lokalno in hitro odločanje o avtorizaciji.

Ta pristop ima več prednosti:

1. **Izolacija**: Vsaka storitev ima svojo instanco OPA, kar zmanjšuje tveganje napak in povečuje varnost.
2. **Zmanjšana latenca**: Lokalno odločanje pomeni hitrejši odzivni čas, saj ni potrebe po komunikaciji z oddaljenim strežnikom za vsako odločitev.
3. **Enostavna integracija**: Sidecar arhitektura omogoča enostavno integracijo OPA z obstoječimi storitvami brez večjih sprememb v kodi.

Sidecar arhitektura se je izkazala za najboljšo izbiro za naše potrebe zaradi svoje fleksibilnosti, zmogljivosti in enostavnosti integracije. Omogoča nam, da hitro in učinkovito uvajamo in vzdržujemo politike, ki so ključne za varnost in skladnost naših aplikacij.

4 Vpeljava OPAL v obstoječi sistem

Vpeljava takšne arhitekture v obstoječi sistem, je pomenila konkreten premislek glede načina razvoja, glede arhitekture in analize kakšen bo vpliv na obstoječe sisteme. Zato smo morali izvesti konkretno analizo trenutna stanja, izdelati načrt vpeljave in nato tudi poučiti razvijalce kako se bo spremenil način dela.

4.1. Analiza trenutnega stanja

Pred vpeljavo novega sistema za politiko dostopa smo izvedli temeljito analizo trenutnega stanja našega sistema avtorizacije. Ugotovili smo naslednje pomanjkljivosti:

- Heterogenost sistemov: Različni sistemi in aplikacije so uporabljali različne metode za upravljanje avtorizacije, kar je povzročalo nedoslednosti in povečano kompleksnost.
- Težavnost vzdrževanja: Politike dostopa so bile razpršene in težko obvladljive, kar je zahtevalo veliko časa in virov za vzdrževanje.
- Varnostne ranljivosti: Nepooblaščeni dostop je bil možen zaradi neučinkovitih in zastarelih avtorizacijskih politik.
- Vpeljava novega sistema za avtorizacijo bo v nekatere storitve težavna in bo potrebno izvesti postopoma v novejše in kritične sisteme prvo in potem na manj pomembne storitve

4.2. Načrt vpeljave

Da bi rešili zgoraj omenjene težave, smo oblikovali celovit načrt za vpeljavo OPA. Naš načrt je vključeval naslednje korake:

- Izbira ustrezne arhitekture: Odločili smo se za implementacijo OPA kot sidecar v naši mikro storitveni arhitekturi. Ta pristop omogoča lokalno in hitro odločanje o avtorizaciji, kar zmanjšuje latenco in povečuje varnost.

- Vzpostavitev OPAL strežnika: Za avtomatizacijo distribucije in sinhronizacijo politik ter podatkovnih posodobitev smo vzpostavili OPAL strežnik. To je omogočilo, da so vse instance OPA vedno delovale s posodobljenimi politikami in podatki.
- Razvoj in testiranje politik: Politike dostopa smo razvili in testirali s pomočjo Rego jezika. Ustvarili smo skladišče politik v Git repozitoriju, ki so bile centralizirano upravljane in enostavno dostopne za vse naše aplikacije.
- Integracija s sistemom: Implementirali smo OPAL odjemalca (OPA sidecar) poleg vsake mikro storitve in zagotovili, da vse storitve uporabljajo enotno politiko dostopa.

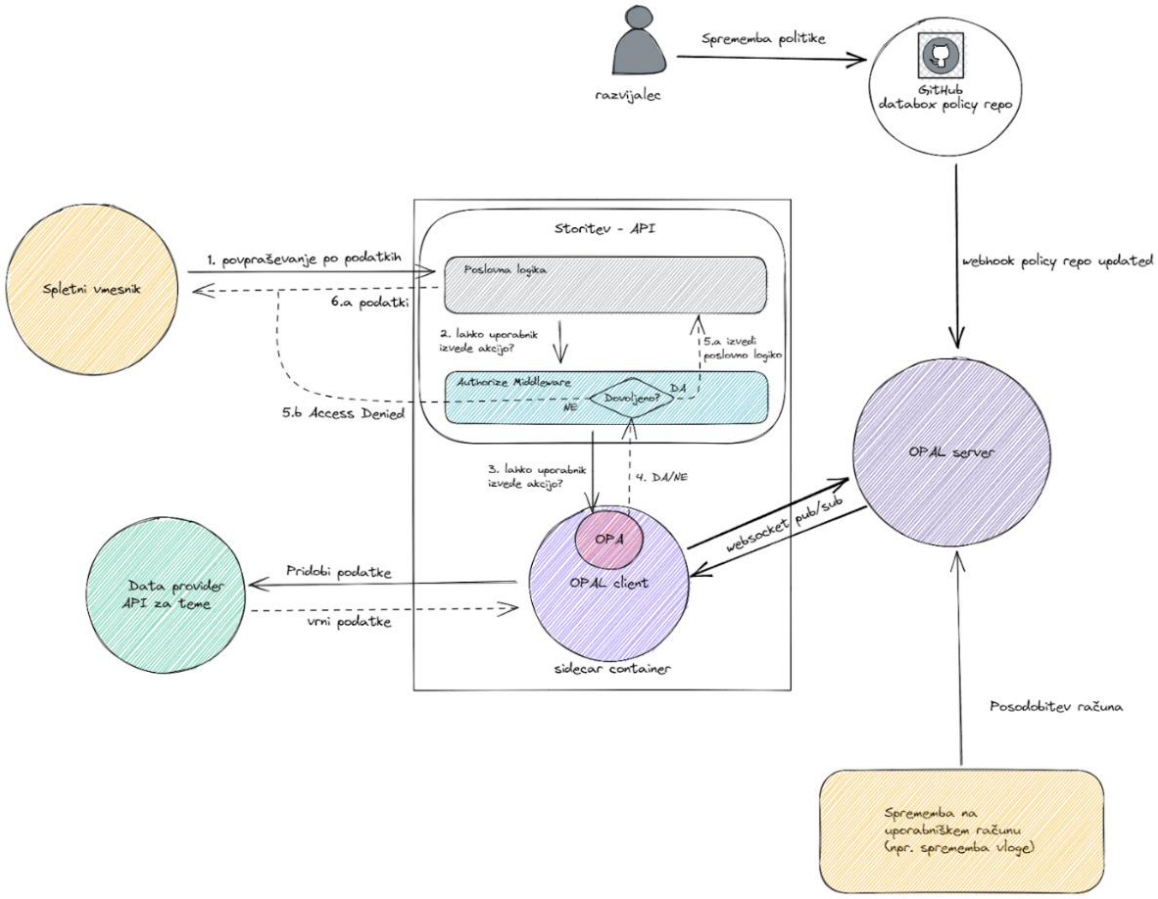
4.3. Implementacija

Vzpostavitev OPAL strežnika in Git repozitorija

Vzpostavili smo OPAL strežnik, ki omogoča sinhronizacijo politik in podatkov. Git repozitorij smo uporabili kot centralizirano mesto za shranjevanje in upravljanje politik. OPAL strežnik spremlja spremembe v Git repozitoriju in avtomatsko posodablja politike na vseh OPA instancah.

Definicija politik z Rego jezikom

Za vsako mikro storitev smo definirali specifične politike v jeziku Rego, ki so ustrezale njenim potrebam.



Slika 2: Diagram komunikacije med komponentami, vključno s posodobitvijo politik ali spremembi na uporabniškem računu.

Integracija OPAL odjemalca

Poleg mikro storitev, kod sidecar smo dodal OPAL odjemalec, ki vsebuje OPA in skrbi za:

- Decentralizirano točko odločitve: Vsaka mikro storitev sprejema avtorizacijske odločitve lokalno, kar zmanjšuje latenco in izboljšuje odzivni čas.
- Ažurnost politik in podatkov.

Spremljanje sprememb podatkov

Mikro storitve, ki so lastniki podatkov (vir resnice) skrbijo, da javljajo o spremembi podatkov OPAL strežniku, ta pa potem poskrbi, da sporoči svojim odjemalcem o spremembi. Odjemalci so naročeni na določene skupine podatkov, tako da lahko pri sebi držijo različne podatke, odvisno kaj potrebujemo za validacijo politike. Diagram komunikacije ja predstavljen na naslednji sliki (slika 2).

5 Zaključek

Vpeljava sistema za politike dostopa je bil eden izmed ključnih varnostnih projektov, ki smo jih naslovili v našem podjetju. S tem smo izboljšali varnost, saj je centralizirano upravljanje politik je omogočilo boljšo skladnost in zmanjšalo možnost napak. Na ta način smo lahko vpeljali tudi "zero trust policy", pri čemer če ni zapisane politike dostopa pomeni, da dostopa ni. Tekom razvoja se je to večkrat pokazalo kot pomemben korak, saj so razvijalci velikokrat na te korake pozabili. Seveda se je potem tekom testiranja hitro ugotovilo, da neka vloga nima pravic, ker politike niso zapisane s čimer smo takoj preprečili uhajanje nezavarovanih storitev na produkcijsko okolje.

Z vpeljavo smo povečali tudi učinkovitost tako iz vidika pisanja politik kot iz vidika izvajanja. Lokalno odločanje je izboljšalo odzivni čas in zmanjšalo latenco. Ker so vse politike zapisane v centralen Git repozitorij, so postale tudi zdaj enostavno dostopne, s čimer se je olajšalo upravljane, ki posledično poenostavlja vzdrževanje in posodobitve. Enostavnost jezika Rego je vplivala tudi na razumevanje politik drugih udeležencev v procesu, ne samo razvijalcev. Uvedli smo tudi testiranje politik, se pravi da se politike, ko se zapišejo tudi pretestirajo, da nas ne čaka kakšno presenečenje na produkcijskem okolju.

S sidecar arhitekturo smo omogočili tudi enostavno prilagajanje in razširitev sistema glede na specifične potrebe posameznih mikro storitev. S tem je bila vpeljava OPA in OPAL bila ključna za izboljšanje našega sistema avtorizacije, kar nam je omogočilo bolj varno, učinkovito in enostavno upravljanje dostopa v naši mikro storitveni arhitekturi.

Literatura

[1] www.openpolicyagent.org, OPA - Policy-based control for cloud native environments, obiskano 20. 7. 2024

[2] github.com/open-policy-agent/opa, Github OPA Repository, obiskano 20. 7. 2024

[3] www.openpolicyagent.org/docs/latest/policy-language/, Rego Policy Language, obiskano 21 .7. 2024

[4] docs.opal.ac/, Welcome to OPAL, obiskano 22. 7. 2024

[5] docs.opal.ac/overview/architecture, OPAL Architevture, obiskano 22.7. 2024

Alternativa geslom – FIDO2 in Passkey

Marko Hölbl, Marko Kompara

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko,
Maribor, Slovenija
marko.holbl@um.si, marko.kompara@um.si

Gesla, ki so najpogostejši način overjanje uporabnikov na spletu, imajo številne pomanjkljivosti in jih zato pogosto označujemo kot Ahilova peta varnosti. Zato se venomer pojavljajo težnje k odpravi gesel in eden izmed pomembnih korakov v to smer je overjanje brez gesel (ang. passwordless authentication) s pomočjo tehnologije Passkey, ki je nastala pod okriljem zavezništva FIDO in poveznim standardom FIDO2. Passkey ponuja izpopolnjen in na uporabnika osredotočen pristop k overjanju brez gesel. V bistvu gre za kriptografski par ključev, varno shranjen na uporabnikovi napravi, z možnostjo uporabe na več napravah in operacijskih sistemih ter shranjevanjem v varnih elementih naprav (kot so TPM ali varne enklave). S tem je postopek overjanje poenostavljen, tako da uporabnikom omogoča overjanje s preprostim biometričnim skeniranjem ali overjanje na podlagi naprave, s čimer odpravi potrebo po pomnjenju in upravljanju gesel. Tako Passkey predstavlja naslednji korak v razvoju informacijske varnosti, saj zagotavlja zanesljiv okvir, ki odpravlja pomanjkljivosti tradicionalnih metod overjanja, hkrati pa je usklajen z varnostnimi in uporabnostnimi cilji sodobnih digitalnih komunikacij. V okviru prispevka se bomo osredotočili na predstavitev pojmov, povezanih z overjanjem brez gesel, standardom FIDO2 ter s poudarkom na delovanju tehnologije Passkey. Prav tako bodo predstavljene prednosti in varnostne funkcionalnosti omenjene tehnologije.

Ključne besede:
Overjanje
Gesla
FIDO 2
Passkey
WebAuthn

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.16

1 Uvod

Gesla so najpogostejši način overjanje uporabnikov na spletu. Vendar imajo več pomanjkljivosti. Na vprašanje o geslih veliko ljudi odgovori, da si jih je težko zapomniti, da jih je lahko pozabiti, da pogosto povzročajo težave ter so zato označena kot Ahilova peta varnosti. Uporabniki namreč pogosto ponovno uporabijo ista gesla za več računov ali pa izberejo šibka gesla, ki jih napadalci lahko uganejo. Poleg tega lahko overjanje z uporabniškim imenom in geslom zahteva dodatne korake, kot je dvofaktorsko overjanje (ang. Two-Factor Authenticartion – 2FA), s čimer se poveča varnost, vendar je postopek prijave za uporabnika tudi bolj zapleten [1].

Leta 2022 je bilo 90 % spletišč tarča napadov z zvabljanjem (ang. phishing), vsako drugo geslo pa je bilo ponovno uporabljeno za dostop do različnih spletnih računov, kot navaja zavezništvo FIDO (ang. FIDO - Fast IDentity Online Alliance) [2]. Povprečen uporabnik ima več kot 100 računov, ki zahtevajo gesla, večina pa za prijavo v večino storitev uporablja isto geslo (ali nekaj gesel). S slabimi navadami glede izbire in/ali uporabe gesel ogrožamo svoje osebne podatke. Uporabniki pogosto na tak način ogrožajo svojo varnost zavoljo prikladnosti oz. enostavnejše uporabe. Vsak četrti Američan uporablja običajna gesla, kot so Abc123, Password1111 in P@ssw0rd. Dve tretini jih priznava, da uporabljajo isto šibko geslo na več spletnih mestih, zaradi česar so vsi ti računi ranljivi [3]. Da bi zmanjšali tveganja in preprečili krajo podatkov, je priporočljivo, da se izogibamo ponovni uporabi poverilnic, izberemo močna gesla in jih takoj spremenimo, če je v storitvi, v kateri smo registrirani, zaznamo uhajanje podatkov.

Gesla so simptom večje težave - naše zgodnje neuspešno prepoznavanje in preverjanje identitete uporabnikov na internetu od samega začetka. Mnogi menijo, da so poverilnice, ki temeljijo na znanju (npr. gesla, kode PIN), eden od prvih grehov interneta [4]. Od takrat poskušamo krpati luknje, gesla pa so bila preprosto najmanj slaba možnost, ki nam je omogočala zavarovati po en račun naenkrat. Alternativne možnosti - strojni žetoni, telefonsko overjanje, biometrija - niso delovale, kot bi morale ali pa niso bile ekonomsko upravičene. Že omenjeno dvofaktorsko overjanje je trenutno uveljavljena metoda za povečanje varnosti.

Zaradi teh varnostnih izzivov, s katerimi se soočajo posamezniki in podjetja, se je pojavil nov standard, ki obljublja hitrost, preprostost in zanesljivost.

Za reševanje teh vprašanj je bilo ustanovljeno zavezništvo FIDO, ki je pripravilo standarda FIDO UAF in U2F ter kasneje FIDO2 [5]. Del standarda FIDO2 je tudi standard WebAuthn, ki je nastal v sodelovanju z W3C [6], [7]. FIDO so prvotno ustanovili Google, Apple in Microsoft in je namenjen zamenjavi gesel s preprostejšim, uporabniku prijaznim varnostnim sistemom, odpornim proti napadu z zvabljanjem. Z overjanjem FIDO se uporabniki prijavijo s poverilnicami, ki so odporne proti napadom z zvabljanjem in se imenujejo ključi. Ti se lahko povežejo s platformo ali varnostnim ključem in omogočajo, da se prijave samo z geslom nadomestijo z varnimi in hitrimi izkušnjami prijave na spletnih mestih in v aplikacijah. Tako omenjeni standardi in specifikacije vzpostavljajo univerzalno metodo za overjanje uporabnikov brez gesel z uporabo kriptografije javnih ključev [8]. Poverilnice, ustvarjene s to metodo, omogočajo uporabnikom, da se overjajo izključno z uporabo svoje naprave in funkcij, kot je biometrija. Tako si ni treba zapomniti več zapletenih gesel ali opraviti dodatnega koraka za potrditev identitete z dvo ali večfaktorskim overjanjem.

Passkey je tip digitalne poverilnice, ki se uporablja za overjanje, ne da bi bilo treba uporabljati običajna gesla in je trenutno najbolj razširjena in dodelana izvedba standarda FIDO2. Uporablja par kriptografskih ključev, ki so edinstveni za vsako spletno storitev. Uporabnikov varnostni ključ (skladen s standardom FIDO2), mobilni telefon ali namizni/prenosni računalnik z ustreznimi komponentami (npr. Windows Hello) je uporabnikov ključ Passkey in ju lahko uporabljam za overjanje in prijavo.

2 Overjanje brez gesel

Metoda, ki jo običajno uporabljamo za overjanje na spletnih mestih in v mobilnih aplikacijah, se je sčasoma razvila in vključuje izboljšane mehanizme zaščite. Običajno je postopek takšen, da račun ustvarimo, tako da navedemo uporabniško ime, običajno e-poštni naslov, in geslo, ki mora izpolnjevati različna merila (prisotnost velikih črk, številk, posebnih znakov itd.). Ko se želimo prijaviti, vnesemo te identifikacijske podatke. Programska koda v ozadju te podatke potrdi, in če so pravilni, zagotovi na primer žeton za dostop, ki bo nato omogočil dostop do zaščitene vsebine. Ta mehanizem se pogosto uporablja za overjanje na digitalnih platformah. Pogosto se omenjeno metodo, zaradi povečane varnosti, dopolni z dvofaktorskim overjanje, na primer s pomočjo enkratnih gesel, poslanih preko sporočil SMS, ali generiranih z namenskimi (mobilnimi) aplikacijami, kot je Google Authenticator [9].

Upravljanje identifikacijskih podatkov (ustvarjanje, spreminjanje, brisanje računov itd.) se v osnovi izvaja na strani strežnika. Vendar ima ta pristop svoje izzive. Uporabniki na primer pogosto ponovno uporabljajo ista gesla na več spletnih mestih, zato da si jih lažje zapomnijo. Ta navada skupaj z napadi, ki vodijo v krajo podatkov, identitete ali vdore, predstavlja resno varnostno tveganje. Druga dobro znana grožnja je napad z zvabljanjem, pri katerem goljufi podvajajo videz legitimnega spletnega mesta, da bi uporabnika zavedali in goljufivo zbrali njegove podatke. Tradicionalna gesla so torej občutljiva na krajo, kar vključuje tudi napade z beleženjem tipkanja (ang. keylogging attack), opazovanjem (ang. shoulder surfing) ipd. Ukradena gesla je mogoče ponovno uporabiti kasneje in na drugih napravah.

Ob upoštevanju teh dejstev je jasno, da tradicionalna kombinacija uporabniškega imena in gesla ne zadostuje več za zagotavljanje varnega overjanja.

2.1. Načini overjanja brez gesel

Leta 2013 je bilo ustanovljeno zavezništvo Fast IDentity Online Alliance (FIDO), ki združuje tehnološka podjetja, vladne agencije, ponudnike storitev (ang. relaying party), finančne institucije in druge deležnike [10]. Glavni cilj tega zavezništva je razviti standarde za overjanje z namenom zmanjšati uporabo gesel in hkrati izboljšali standarde za overjanje na spletu.

Prva izmed standardov sta bila FIDO UAF in FIDO U2. To je mogoče doseči z zagotavljanjem, da zasebni ključi in biometrični podatki, kjer je to primerno, vedno ostanejo na uporabnikovi napravi. Tako je mogoče overjanje opraviti z metodami, kot sta biometrija ali vnos enkratne kode PIN, ne da bi si bilo treba zapomniti zapleteno geslo. FIDO podpirajo tudi glavni brskalniki in operacijski sistemi. Kasneje je bil predstavljen standard FIDO2, ki je naslednik FIDO UAF, in FIDO U2F ter je odprti standard overjanja brez gesel. FIDO2 omogoča preprostejše in močnejše overjanje z uporabo kriptografije javnega ključa. Standard FIDO2 sestavljajo specifikacija W3C za spletno overjanje, API WebAuthn in protokol CTAP2.

FIDO2 definira overjanje z uporabo javnih in zasebnih ključev, pri čemer je zasebni ključ, shranjen na uporabnikovi napravi, javni ključ pa na strežnikih ponudnika storitve. Za overjanje na spletu se uporablja lokalno overjanje (npr. biometrija na telefonu). Tako se uporabniki prijavijo s prepoznavo obraza, prstnega odtisa (torej biometrično) ali z vnosom kode PIN v lokalno napravo za spletno registracijo in overjanje. To odklene zasebni ključ, shranjen na napravi, ki se zatem uporabi za overjanje pri ponudniku storitve.

Najbolj pomembni standarde, razviti pod okriljem zavezništva FIDO so:

FIDO UAF (ang. Universal Authentication Framework):

- Predstavljen leta 2014
- Omogoča overjanje brez gesel z uporabo lokalnih metod biometričnega preverjanja (kot sta prepoznavanje prstnih odtisov ali obraza) neposredno na napravi, ne da bi se biometrični podatki prenašali po omrežju.
- Uporablja se predvsem za mobilne naprave, kjer so na voljo biometrični senzorji.

FIDO U2F (ang. Universal Second Factor)

- Predstavljen leta 2014
- Zagotavlja mehanizem dvofaktorsko overjanje, ki dopolnjuje tradicionalne sisteme, ki temeljijo na geslih. Predvideva uporabo fizičnih varnostnih ključev (USB, NFC) kot drugi faktor, pri čemer mora uporabnik za overjanje izvesti aktivnost (npr. fizično vstaviti ključev v napravo).
- Namenjen izboljšanju varnosti spletnih aplikacij z dodajanjem standardiziranega drugega faktorja pri overjanju.

FIDO2 (ang. Fast IDentity Online 2):

- Predstavljen leta 2018.
- Namenjen overjanju brez gesel in dvofaktorskemu overjanju, s poudarkom na skalabilnosti in spletni združljivosti.
- Sestavljen iz:
  - **WebAuthn (ang. Web Authentication) -** API za spletno overjanje. Omogoča spletnim aplikacijam, da overjanje uporabljajo poverilnice z javnim ključem.
  - **CTAP2 (ang. Client to Authenticator Protocol)** - omogoča komunikacijo med odjemalskimi napravami (kot so pametni telefoni ali računalniki) in zunanjimi overitelji (kot so varnostni ključi).
- Zagotavlja enoten pristop k overjanju brez gesel in večfaktorsko overjanje na različnih napravah in platformah.

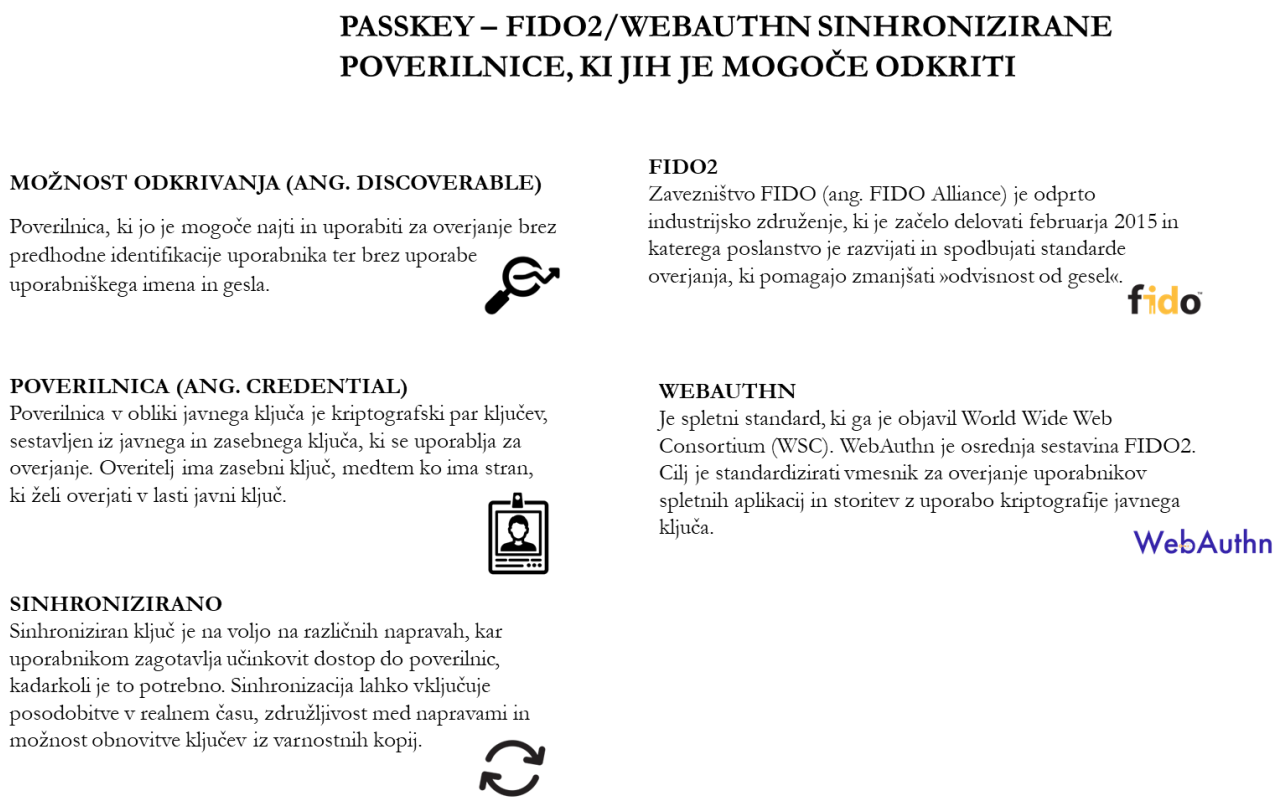
FIDO2 podpira večina platform, kot so Windows 10 in 11, Android, iOS ter brskalniki, kot so Google Chrome, Mozilla Firefox, Microsoft Edge in Apple Safari. Primeri notranjih FIDO overiteljev vključujejo Touch ID, Face ID, čitalniki prstnih odtisov ali Windows Hello. Prav tako standard podpira t.i. zunanje overitelje, kot so precej poznani Yubico ključki. Več informacij je na voljo na [11].

2.2. Najbolj znano utelešenje standarda FIDO2 - Passkey

Passkeys je najbolj znana in razširjena realizacija standarda FIDO2 s poudarkom na dobri uporabniški izkušnji. Uporabnikom omogočajo, da se overijo na različnih napravah in na različnih platformah. Zasnovan je za uporabo na več napravah in operacijskih sistemih ter se pogosto povezuje s storitvami v oblaku za sinhronizacijo vseh uporabnikovih naprav.

Passkey temelji na načelih preprostosti, učinkovitosti in varnosti:

- **Preprostost** - takojšnja vzpostavitev računa. Maprava je odgovorna za ustvarjanje poverilnic in ni potrebe po pomnjenju - overitvene kode, povezane z zasebnimi ključi, so shranjene samo v napravah določenega uporabnika. Pristopne ključe je mogoče deliti med različnimi napravami. Lahko se uporablja sinhronizacijo – npr. z oblakom. Naprave tako delujejo kot upravitelji gesel in za vas hranijo vse informacije.
- **Učinkovitost** - gesla se ustvarijo hitro in jih je mogoče sinhronizirati med različnimi napravami.
- **Varnost -** po svoji naravi so uporabljeni ključi robustni in edinstveni, kar odpravlja tveganja, povezana s ponovno uporabo gesla. Strežniki imajo dostop samo do javnih ključev. Za dostop do zasebnega ključa je potrebna biometrično overjanje. Če pride do ogrožanja podatkov na enem spletnem mestu, to ne vpliva na druga spletna mesta ali storitve, saj je vsak ključ gesla drugačen.



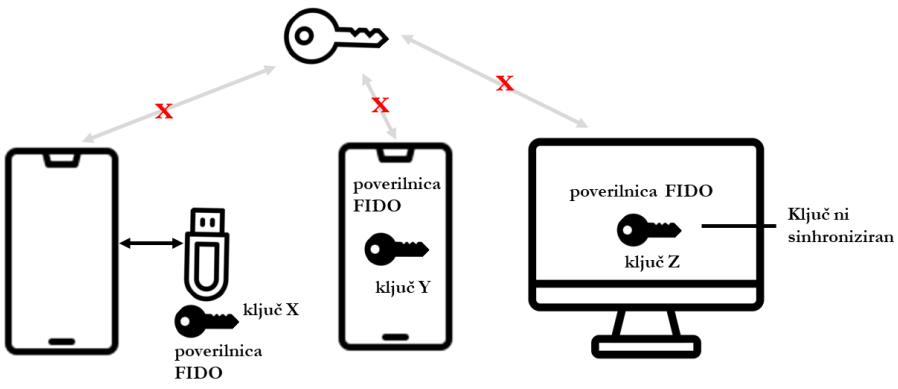
Slika 1: Pregledna infografika tehnologije Passkey.

3 Podobnosti o tehnologiji Passkey

Passkey je izraz, ki se uporablja za opis poverilnic, ki jih je možno odkrivati (ang. discoverable credentials) in so skladne s standardom FIDO2 ter uporabljajo kriptografijo javnega ključa. Uporablja se lahko za overjanje uporabnika pri ponudniku storitve brez potrebe po uporabniškem imenu in geslu. Zasebni ključ in ID poverilnice sta shranjena v overitelju, medtem ko sta javni ključ in ID poverilnice shranjena pri ponudniku storitev.

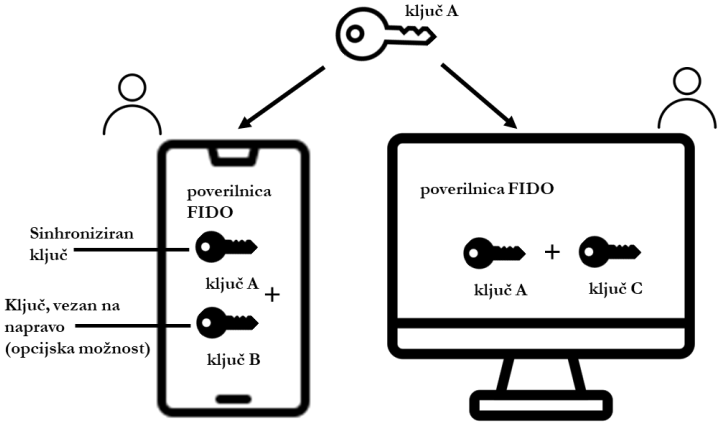
Passkey je naslednji korak v razvoju digitalnega overjanja, ki temelji na standardih zavezništva FIDO. Za razliko od gesel si uporabnikom ni treba zapomniti zapletenih nizov znakov, s čimer se izniči tveganje za napade z zvabljanjem in tako poveča raven varnosti. Obstajata dve vrsti ključev, ki ustrezata različnim potrebam in scenarijem uporabnikov.

– **Passkey za eno napravo (ang. Single-Device Passkeys)**
  Ključ je vezan na eno overitveno napravo in ne more zapustiti naprave.



Slika 2: Passkey za eno napravo. [6]

- **Passkey za več naprav (ang. Multi-Device Passkeys)**
  Ključ je mogoče sinhronizirati med različnimi uporabniškimi napravami.



Slika 3: Passkey za več naprav. [6]

Vodilni ponudniki tehnologije so naredili korak naprej pri vključevanju funkcionalnosti Passkey ključev v svoje ekosisteme. Kot primer navedimo mobilne platforme Android in iOS. Za tiste v ekosistemu Apple se ključi brez težav sinhronizirajo med napravami, povezanimi z istim AppleID-jem, in iCloud Keychain [12]. Na operacijskem sistemu Android se ključi Passkey sinhronizirajo preko računov Google, kar poenostavi dostop v različnih napravah [13]. V primerih, ko sinhronizacija v oblaku ni izvedljiva, je mogoče, da uporabniki na eni napravi ustvarijo kodo QR, in jo optično preberejo z drugo napravo, v kateri so shranjeni ključi, ter tako olajšajo prijavo v aplikacijo.

Ko je ustvarjen nov par javnih/zasebnih ključev in povezan z uporabnikom, se lahko uporabnik overi s podpisom izziva, ki ga pošlje ponudnik storitve (strežnik). Pri tem uporabi zasebni ključ, shranjen v t.i. overitelju. Za izvedbo te operacije je potrebna identifikacija uporabnika, npr. z biometričnimi podatki ali kodo PIN naprave. Z vidika uporabnika to pomeni, da lahko priročno dostopa do storitve z uporabo biometrične prepoznave ali kode PIN pametnega telefona, ne da bi si moral zapomniti več gesel (ali se zanašati na upravitelja gesel).

Poleg tega lahko uporabnik po registraciji gesla za storitev (npr. na pametnem telefonu) do nje dostopa z različnimi odjemalci na isti napravi (splet, mobilna aplikacija) ali celo na drugi napravi (drug pametni telefon, prenosni računalnik itd.).

Ena od glavnih prednosti sinhroniziranih ključev je možnost souporabe med različnimi uporabnikovimi napravami. To uporabnikom omogoča, da obnovijo svoje ključe iz varnostne kopije ali uporabijo isti ključ za overjanje na več platformah. Ta uporabniku prijazna funkcija ublaži težave, ki se lahko pojavijo pri uporabi overitvenega sredstva, vezanega na napravo. V tem primeru sinhronizacija ali varnostno kopiranje nista mogoča, saj poverilnic ni mogoče prenesti iz naprave. To lahko povzroči težave uporabniku, če fizično napravo izgubi ali zamenja.

Čeprav je ta funkcija privlačna z vidika uporabnosti in uporabniške izkušnje, je treba opozoriti, da v primerjavi z overiteljem, vezanim na napravo, nekoliko zmanjša varnost overjanja. Ključ, vezan na napravo, ne omogoča izvoza ustvarjenega ključa na zunaj naprave, kar preprečuje možnost kraje in onemogoča izkoriščanje ranljivosti, povezanih z uporabo oblaka.

Potrebno je omeniti, da specifikacija WebAuthn ne zagotavlja protokola za varnostno kopiranje zasebnih ključev poverilnic ali njihovo izmenjavo med overitelji. Zato je odgovornost za izvajanje politik varnostnega kopiranja prepuščena ponudnikom storitve (kot sta Apple ali Google v primeru ključev, ustvarjenih na mobilnih napravah). Ti ponudniki se bodo morali sami odločiti, kako bodo izvajali politike varnostnega kopiranja, kar bo na koncu

vplivalo na raven varnosti ključa, saj je ta neposredno odvisna od ravni varnosti postopka obnovitve računa, ki se uporablja za varnostno kopiranje ključev. Vendar je ta kompromis potreben, da lahko uporabniki obnovijo svoje registrirane ključe iz varnostne kopije.

3.1. Delovanje

Na tem mestu bomo obravnavali podrobnosti delovanje tehnologije Passkey. Glavni subjekti, ki so vključeni v ta tok, so:

- **uporabnik**, ki želi dostopati do storitve,
- **odjemalec**, spletno mesto ali mobilna aplikacija, ki uporabniku omogoča dostop do storitve, ki jo zagotavlja ponudnik storitve,
- **overitelj,** naprava, ki ustvarja in potrjuje kriptografske ključe, povezane z uporabnikovim ključem, ter tako zagotavlja celovitost postopka overjanja in
- **ponudnik storitve,** ki potrdi pristnost identitete uporabnika na podlagi ključa in tako odloči o dostopu do zahtevane storitve.

Ti subjekti medsebojno sodelujejo, da bi uporabniku omogočili registracijo novega ključa. Uporabnik lahko nato ta ključ uporablja za identifikacijo in dostop do ponujenih storitev v vseh nadaljnjih interakcijah.

Postopek registracije

Prva aktivnost, ki jo opravi uporabnik, je registracija v storitev z ustvarjanjem novega ključa. Možna sta dva primera:

1. Uporabnik je že registriran v storitev z uporabniškim imenom, geslom in prijavo 2FA ter želi dodati overjanje s Passkeyem. Po prijavi s klasičnim načinom bo imel možnost, da doda poverilnico tipa Passkey (ang. Passkey credential).
2. Storitev uporabnikom omogoča, da se registrirajo in povežejo novo poverilnico tipa Passkey. V tem primeru bosta ustvarjanje novega uporabnika in Passkeya potekala hkrati.

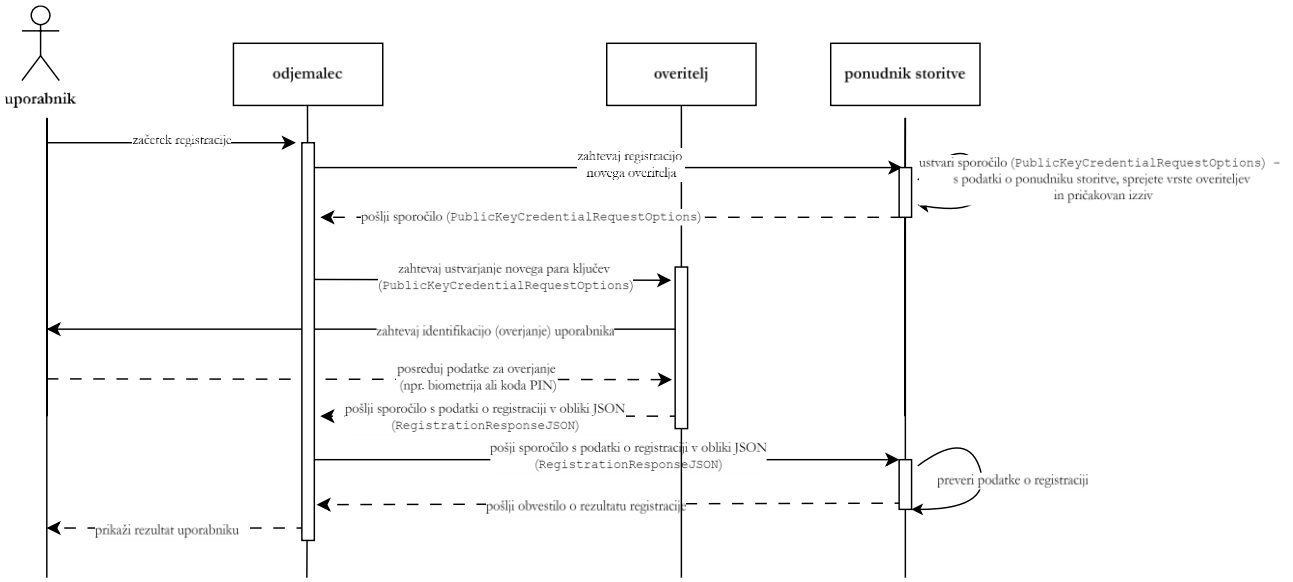
Čeprav se lahko zaledna logika pri obravnavi teh dveh scenarijev nekoliko spremeni, postopek registracije v obeh primerih ostane dosleden. Ta postopek poveže geslo s storitvijo in omogoči nadaljnji dostop.
Koraki, vključeni v postopek, so naslednji

- Uporabnik se namerava registrirati v storitev z uporabo odjemalca, bodisi mobilne aplikacije bodisi spletnega mesta.
- Odjemalec začne postopek registracije in od uporabnika zahteva, da registrira novega overitelja.
- Ponudnik storitve se odzove s sporočilom v obliki JSON (`PublicKeyCredentialRequestOptions`), ki vključuje izziv in dodatne metapodatke, kot so podatki o ponudniku storitve in sprejete vrste overiteljev. Poleg tega ponudnik storitve shrani izziv za to sejo registracije, saj ga bo treba preveriti ob prejemu odgovora.
- Odjemalec s tem odgovorom od overitelja zahteva, da ustvari nov par ključev, ki se bo uporabil za overjanje uporabnika pri ponudniku storitve.
- Overitelj (ki je lahko zunanja naprava ali naprava, na kateri se izvaja odjemalec) preveri sporočilo od strežnika in zahteva identifikacijo uporabnika (z biometričnimi podatki ali kodo PIN naprave).
  - Če je vse v redu, overitelj ustvari nov par zasebnega in javnega ključa ter odjemalcu vrne sporočilo v obliki JSON (`RegistrationResponseJSON`), ki vsebuje novo ustvarjeni javni ključ, informacije o potrditvi in druge pomembne podrobnosti. Opozoriti je treba, da zasebni ključ ostane v overitelju,

v primeru pametnih telefonov pa je shranjen v ločeni strojni komponenti (npr. v Secure Enaclave napravo na iPhoneih).

- Odjemalec potrditev pošlje ponudniku storitve, da potrdi novo registracijo.
- Ponudnik storitve preveri prejeto sporočilo in overi potrditev ter zagotovi, da registracija uporabnika izpolnjuje določena merila (npr. pravilen odgovor na izziv, ustrezna oblika potrditve).
- Po preverjanju sporočila o registraciji ponudnik storitve obvesti uporabnika o rezultatu. Če je uspešen, shrani ustrezne podatke o registrirani poverilnici, ki se bodo uporabljali za prihodnja overjanja.

Celoten postopek je tudi prikazan na Sliki 4.



Slika 4: Postopek registracije. [2]

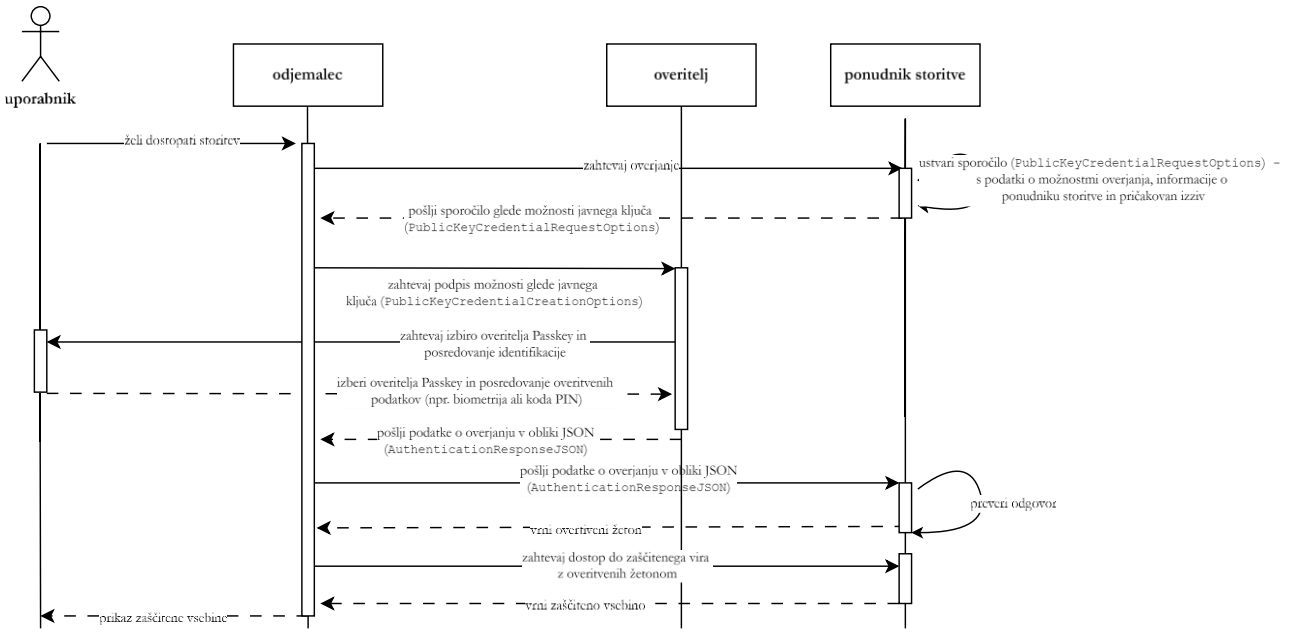
Postopek overjanja

Po uspešni registraciji lahko uporabnik uporabi Passkey za overjanje pri ponudniku storitve in dostop do zaščitenega vira. Oglejmo si postopek:

- Uporabnik želi dostopati do storitve prek mobilne aplikacije ali spletnega mesta in se mora overiti.
- Odjemalec sproži postopek overjanja tako, da od ponudnika storitve zahteva overjanje.
- Ponudnik storitve pošlje sporočilo v obliki JSON (`PublicKeyCredentialRequestOptions`), z možnostmi overjanja, ki vključujejo izziv in informacije o ponudniku storitve. Poleg tega shrani izziv za to sejo overjanja, saj ga bo treba preveriti ob prejemu odgovora.
- Odjemalec pošlje to sporočilo overitelju in zahteva, da ga podpiše z enim od ključev, povezanih z uporabnikom.
- Če ima overitelj več ključev, povezanih z uporabnikom, je ta pozvan, naj izbere ključ, da nadaljuje postopek overjanja. Ko ga izbere, se uporabnika pozove, naj zagotovi overjanje z biometričnimi podatki ali kodo PIN naprave, da se izziv podpiše z zasebnim ključem, shranjenim v overitelju.
- Če se postopek uspešno izvede, overitelj odjemalcu vrne sporočilo v obliki JSON (`AuthenticationResponseJSON`), ki vsebuje podpisan izziv, podatke o overitelju in druge ustrezne podatke odjemalca.
- Odjemalec pošlje ta odziv za overjanje ponudniku storitve.
- Ponudnik storitve najprej preveri, ali je uporabljeni overitelj registriran. Nato overi odgovor in pravilnost odgovora na izziv.

- Po uspešnem preverjanju ponudnik storitve uporabniku odobri dostop in mu omogoči uporabo storitve, pri čemer mu vrne, na primer overitveni žeton, ki se lahko uporabi za naslednje zahteve.
- Uporabnik je overjen in lahko uporablja mobilno aplikacijo ali spletno mesto.

Celoten postopek je tudi prikazan na Sliki 5.



Slika 5: Postopek overjanja. [2]

Celoten postopek je tudi prikazan na Sliki 5.

4 Zaključek

Passkey je obetavna rešitev za overjanje brez gesel. Uporabnikom ponuja standarden, priročen in varen način dostopa do storitve, ne da bi si morali zapomniti ali vnesti geslo. Od leta 2016, ko je bil objavljen prvi osnutek specifikacije FIDO (UAF in U2F), pa vse do danes, smo priča razcvetu uporabe in Passkeya bi lahko potencialno revolucionarno spremenili overjanje uporabnikov.

Vendar pa tehnologija Passkey ni brez izzivov. Čeprav se zdi zamisel o prihodnosti brez gesel privlačna, pomeni veliko spremembo, na katero se bodo uporabniki morali prilagoditi. Mnogi med njimi se morda ne zavedajo te možnosti ali njenega delovanja, zato bo morda potrebno nekaj časa in truda, da jih prepričamo, da preidejo iz overjanja z gesli. Poleg tega trenutno ni interoperabilnosti med ključi, ustvarjenimi na različnih platformah, in do zdaj ni možnosti prenosa ključa, ustvarjenega v telefonu iPhone, in sinhroniziranega prek storitve iCloud, v napravo z operacijskim sistemom Android in obratno. Vendar se tudi na tem področju obeta rešitev. Ponudniki t.i. upravljavcev gesel (ang. Password managers) že uvajajo možnost uporabe omenjenih orodij za upravljanje s ključi Passkey [14]. Tako je trenutno prihodnosti precej obetavna.

Literatura

[1] D. Coffin, "Two-factor authentication," in *Expert Oracle and Java Security: Programming Secure Oracle Database Applications with Java*, Springer, 2011, pp. 177–208.

[2] "Passkeys (Passkey Authentication)." Accessed: Jul. 30, 2024. [Online]. Available: https://fidoalliance.org/passkeys/

[3] Harris Poll, "The United States of P@ssw0rd$."

[4] Matt Kapko, "Security has an underlying defect: passwords and authentication." Accessed: Jul. 31, 2024. [Online]. Available: https://www.cybersecuritydive.com/news/security-defect-passwords-authentication/693471/

[5] "FIDO2 - FIDO Alliance." Accessed: Jul. 30, 2024. [Online]. Available: https://fidoalliance.org/fido2/

[6] P. Heim and F. Alliance, "Everybody Is Invited: How FIDO Addresses a Full Range of Use Cases," 2022.

[7] N. Frymann, D. Gardham, F. Kiefer, E. Lundberg, M. Manulis, and D. Nilsson, "Asynchronous remote key generation: An analysis of yubico's proposal for W3C webauthn," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 939–954.

[8] M. E. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 42–49, 2002.

[9] A. Nash, H. Studiawan, G. Grispos, and K.-K. R. Choo, "Security Analysis of Google Authenticator, Microsoft Authenticator, and Authy," in *International Conference on Digital Forensics and Cyber Crime*, Springer, 2023, pp. 197–206.

[10] "FIDO Alliance Member Companies & Organizations." Accessed: Jul. 31, 2024. [Online]. Available: https://fidoalliance.org/members/

[11] G. Eleftherios, "FIDO2 Overview, Use Cases, and Security Considerations," Athens University of Economics and Business, 2023.

[12] "Passkeys Overview - Apple Developer." Accessed: Jul. 30, 2024. [Online]. Available: https://developer.apple.com/passkeys/

[13] "Passwordless login with passkeys | Authentication | Google for Developers." Accessed: Jul. 30, 2024. [Online]. Available: https://developers.google.com/identity/passkeys

[14] Inga Valiaugaitė, "Best Passkey Password Manager in 2024." Accessed: Aug. 01, 2024. [Online]. Available: https://cybernews.com/best-password-managers/passkey-password-managers/

Zero-knowledge proof v praksi

Vid Keršič, Martin Domajnko, Sašo Karakatič, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
vid.kersic@um.si, martin.domajnko@student.um.si, saso.karakatic@um.si, muhamed.turkanovic@um.si

Z vse pogostejšo uporabo interneta in migracijo storitev iz fizičnega sveta v digitalni, postajajo vprašanja o varnosti, zasebnosti in digitalnem lastništvu osebnih podatkov vse pogostejša. Ena izmed ključnih tehnologij, ki omogoča razvoj rešitev na tem področju, so ničelno spoznavni dokazi (ang. zero-knowledge proofs, ZKP). ZKP so kriptografski protokoli, pri katerih dokazovalec dokaže pravilnost poljubne trditve preveritelju, ne da bi pri tem razkril dodatne informacije ali svoje podatke. V članku predstavimo ZKP protokole in njihove razlike, s posebnim poudarkom na dveh najpogostejših družinah protokolov: zk-SNARK in zk-STARK. Njihovo uporabno vrednost prikažemo na področju samo-upravljane in decentralizirane identitete ter na področju strojnega učenja. Pri decentralizirani identiteti ZKP omogočajo deljenje podatkov brez razkritja zasebnih informacij, medtem ko pri strojnem učenju omogočajo preverljivost izhodov modelov. To pomeni, da lahko uporabnik preveri, ali je bil za generiranje napovedi dejansko uporabljen pravilno izbran model.

Ključne besede:
zero-knowledge proof
kriptografija
decentralizirana identiteta
zasebnost
overjanje

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.17

1 Uvod

V zadnjih letih je vse več govora o zasebnosti, varnosti, suverenosti in lastništvu podatkov na spletu, čemur priča tudi pobuda za evropske digitalne denarnice, ki temeljijo ravno na teh principih [9]. Ena izmed jedrnih tehnologij, ki omogočajo razvoj takšnih rešitev, so ničelno spoznavni dokazi ali ZKP (ang. zero-knowledge proofs) [14]. ZKP so kriptografski protokoli, pri katerih dokazovalec (ang. prover) dokaže pravilnost določene trditve preveritelju (ang. verifier), ne da bi pri tem razkril kakršno koli informacijo o trditvi, razen njene pravilnosti. Zaradi te lastnosti je uporaba ZKP smiselna v številnih aplikacijah, saj razen dokazovanja informacij in celovitosti le teh, povečujejo tudi zasebnost in varnost uporabnikov ter varujejo njihove podatke. Čeprav so bili formalno definirani že leta 1985, so se v praksi začeli uporabljati šele v zadnjih letih, zahvaljujoč razvoju učinkovitejših metod in algoritmov za generiranje kriptografskih dokazov. K hitremu razvoju je prispevala tudi njihova uporaba v tehnologiji veriženja blokov (ang. blockchain), na primer v L2 omrežjih s t. i. zero-knowledge rollup.

V članku predstavljamo, definiramo in opisujemo ZKP protokole, s poudarkom na dvema najpomembnejšima družinama ZKP protokolov zk-SNARK in zk-STARK. Dodatno se osredotočamo na dva primera njihove uporabe, in sicer na področju decentralizirane identitete ter strojnega učenja. Pri prvem primeru je glavna uporaba deljenje podatkov, kjer uporabniki dokažejo veljavnost in vrednost določenih atributov, ne da bi pri tem razkrili svoje zasebne podatke ali celo konkretne vrednosti atributa. Eden najbolj razširjenih ZKP platforma za decentralizirano identiteto je Privado ID. Pri drugem primeru se ZKP uporabljajo za preverljivost izhodov modelov strojnega učenja, kjer se z uporabo ZKP omogoči preverjanje, če je bil za napovedovanje uporabljen določen model. Ta primer uporabe je še posebej aktualen v časih, ko vse bolj uporabljamo in plačujemo za storitve, ki vključujejo strojno učenje, pri čemer je nemogoče preveriti, ali je bil uporabljen model, za katerega plačujemo.

2 Zero-knowledge proofs

Področje kriptografije se je v zadnjem desetletju, še posebej v zadnjih letih, zelo razvilo in veliko kriptografskih protokolov, ki so v preteklosti bili le izvedljivi v teoriji, se danes lahko že učinkovito uporablja v sklopu dejanskih realnih aplikacijah. Medtem, ko se digitalno podpisovanje, simetrično in asimetrično šifriranje že vsakdan uporabljajo v digitalnem svetu, na primer pri obisku spletnih strani in podpisovanju elektronskih dokumentov, se naprednejši kriptografski protokoli kot so homomorfno šifriranje (ang. homomorphic encryption) in ničelno spoznavni dokazi (ZKP), ki so jedrna tema tega prispevka, šele začenjajo uporabljati v realnem svetu.

2.1. Splošno

ZKP je kriptografski protokol, ki poteka med dvema entitetama: *dokazovalcem* in *preveriteljem* [14]. S pomočjo protokola lahko dokazovalec prepriča preveritelja o posedovanju določenih podatkov/informacij brez razkritja podrobnosti le teh. Eden izmed najbolj znanih primerov je dokazovanje polnoletnosti, pri čemer ni razkrita dejanska starost osebe, temveč le trditev *starost osebe je večja od 18*. Čeprav so bili ZKP defninirani že v 80. letih prejšnjega stoletja, se je njihova uporabnost razširila v zadnjih letih, čemur je botrovalo več faktorjev, kot so finančne investicije, optimizacije orodij za generiranje kriptografskih dokazov in razvoj novih algoritmov ter pristopov.

Protokoli ZKP so definirani s tremi lastnostmi:

- **celovitost** (ang. completeness): preveritelj sprejme dokaz le, če ima dokazovalec res informacijo, in je bil protokol izveden pravilno;
- **trdnost** (ang. soundness): obstaja le zelo majhna verjetnost, da lahko dokazovalec prepriča preveritelja z napačno informacijo;

- **ničelno znanje** (ang. zero-knowledge): čeprav je preveritelj prepričan o veljavnosti dokaza, ne izve ničesar o dejanskih podatkih/informaciji.

Skozi leta raziskav je bilo razvitih več različnih sistemov oz. protokolov ZKP, ki se med seboj razlikujejo po načinu delovanja. Najpogosteje se delijo na (a) **interaktivne** in (b) **ne-interaktivne**, pri čemer se slednji pogosteje uporabljajo, saj ne zahtevajo komunikacije oz. izmenjave več sporočil med dokazovalcem in preveriteljem. Ker imajo ne-interaktivni precej več želenih značilnost kot interaktivni, pri čemer je najbolj želena nepotreba po izmenjavi več sinhronih sporočil med dokazovalcem in preveriteljem, se interaktivni protokoli pogosto s Fiat-Shamirjevo hevristiko preslikajo v ne-interaktivne [11].

Tudi znotraj kategorije ne-interaktivnih protokolov je bilo razvitih več podsistemov ZKP, pri čemer sta najbolj razširjena zk-SNARK in zk-STARK, ki sta predstavljena podrobneje v nadaljevanju. Protokoli ZKP se lahko uporabijo tudi za različne namene, na primer za selektivno razkritje podatkov v primeru deljenja elektronskih dokumentov, ali za delegiranje zahtevnejših programov na oddaljen računalnik, kjer želimo ohraniti preverljivost in pravilnost izvedbe programa (ang. verifiable computing), na primer izvajanje transakcij na L2 omrežjih pri tehnologiji veriženja blokov (ang. blockchain) (Ethereum).

2.2. zk-SNARK

zk-SNARK (ang. zero-knowledge succinct non-interactive argument of knowledge) so najbolj znana in razširjena družina protokolov ZKP [13]. Kot je že iz njihovega imena razvidno sta eni izmed najpomembnejših lastnosti **ne-interaktivnost** in **zgoščenost** (ang. succinctness). Slednja karakteristika izvira iz tega, da so dokazi "majhni" in njihova verifikacija hitra. Na praktičnem primeru to pomeni, da je preverjanje dokaza precej hitrejše kot generiranje samega dokaza oz. izvedba samega programa v primeru preverljivega računalništva.

Eden izmed pomembnejših korakov pri zk-SNARK je zaupanja vredna namestitev (ang. trusted setup), ki se izvede pred jedrnim delom protokola - dokazovanjem in preverjanjem [13]. Tekom zaupanja vredne namestitve se izračuna oz. definira skupen referenčni niz (ang. common reference string, CSR), ki se nato uporabi pri generiranju dokaza in preverjanju. CSR se navadno izračuna z večstranskim računanjem (ang. multi-party computation, MPC), saj le-ta omogoča varnejši način generacije, saj je manjša verjetnost, da bi vse osebe, ki sodelujejo v MPC, razkrile parametre uporabljene za generacijo CSR.

Skozi leta je bilo predstavljenih več zk-SNARK sistemov kot so ***Halo2***, ***Plonky2***, ***Aurora*** in ***Sonic***. Ti sistemi se med seboj razlikujejo v lastnostih CSR, na primer univerzalnosti in pouporabi niza za različne programe, učinkovitosti, uporabljenih matematičnih konstruktih itd.

2.3. zk-STARK

zk-STARK (ang. zero-knowledge scalable transparent argument of knowledge) so druga najbolj znana družina protokolov ZKP [3]. Razviti so bili nekaj let po zk-SNARK-ih z namenom rešitve njihovih pomanjkljivosti, predvsem potrebe po zaupanja vredni namestitvi. Za dokazovanje in preverjanje zk-STARK dokazov tako ni potrebe po CSR, zaradi česar so bolj transparentni [3]. zk-STARK sistemi prav tako vsebujejo druge kriptografske primitive zaradi česar so tudi odporni na kvantne računalnike in imajo boljše teoretične skalabilne lastnosti kot zk-SNARK-i (glej tabelo 1). Kljub temu so zaradi manjše razširjenosti in tega, ker so mlajši, trenutno še manj optimizirani in pogosto manj učinkoviti na realnih primerih.

V zadnjih letih je bilo razvitih tudi več zk-STARK sistemov, kot so ***Winterfell***, ***Stone*** od ***Starkware*** in ***RISC Zero***.

2.4. Primerjava

Čeprav zk-SNARK-i in zk-STARK-i omogočajo enake funkcionalnosti, se v določenih karakteristikah precej razlikujejo. Kot je bilo že omenjeno, prvi potrebujejo zaupanja vredno namestitev, čeprav so raziskovalci pred kratkim predstavili tudi zk-SNARK-e brez zaupanja vredne namestitve [22]. Kljub temu imajo zk-STARK-i več teoretičnih boljših lastnosti, kot so boljša časovna zahtevnost dokazovanja, kar se pozna še posebej pri večji količini vhodnih podatkov. Povzetek primerjave je predstavljen v Tabeli 1.

Tabela 1: Primerjava zk-SNARK in zk-STARK.

Lastnost	zk-SNARK	zk-STARK
Potrebna zaupanja vredna namestitev	Da (nekateri ne)	Ne
Odpornost na kvantne računalnike	Ne (nekateri da)	Da
Velikost dokaza	Majhna (več sto B)	Velika (več sto KB)
Zahtevnost dokazovanja	Kvazi-linearno	Logaritemska
Zahtevnost preverjanja	Logaritemska	Logaritemska

Aplikacije in sistemi temelječi na ZKP se razvijajo s pomočjo ogrodij (ang. framework), ki omogočajo definiranje tako imenovanih vezij (ang. circuit). Vezja so programi, ki kodirajo proces, ki ga želimo dokazati. Vnaprej je potrebno definirati vhodne podatke, izhodne podatke in sam program. Prav tako je potrebno definirati vidnost podatkov, na primer kateri vhodni oz. izhodni podatki naj bodo javni ali zasebni, kar pomeni, da niso razkriti preveritelju. Ogrodja, ki omogočajo razvoj sistemov ZKP so navadno poimenovana glede na to, kateri zk-SNARK ali zk-STARK protokol je uporabljen, na primer Halo2, Plonky2, itd.

3 Primeri uporabe

ZKP so našli uporabo na različnih področjih, kjer omogočajo izboljšano zasebnost, varnost in učinkovitost v digitalnih sistemih. Sledi nekaj pomembnih primerov uporabe:

- **Zasebnost pri kriptovalutah**: Zerocash, protokol za anonimne transakcije s kriptovalutami, uporablja ZKP za skrivanje ključnih informacij o transakcijah: znesek, identiteto pošiljatelja in prejemnika ter zgodovino transakcij. Sistem omogoča potrditev veljavnosti transakcije brez razkritja podrobnosti, podobno kot bi dokazali plačilno sposobnost brez vpogleda v celoten bančni izpisek. ZKP v primeru Zerocash protokola dokazuje sledeče štiri ključne elemente: zadostnost sredstev pošiljatelja, veljavnost transakcije, odsotnost ustvarjanja novega denarja ter prejem pravilnega zneska s strani prejemnika [2].
- **Overjanje v internetu stvari (IoT)**: V ekosistemih interneta stvari se lahko ZKP uporabljajo za anonimno komunikacijo in varno overjanje naprav, na primer za dokazovanje in verifikacijo identitete senzorjev. Zaradi računske kompleksnosti večina tradicionalnih ZKP algoritmov ni primernih za uporabo na majn zmoglivih IoT napravah. [5] predstavlja različne primere učinkovite in varne aplikacije ZKP algoritmov v IoT sistemih z anonimno komunikacijo.
- **Povečana prepustnost transakcij verig blokov**: ZKP igra ključno vlogo v rešitvah za povečanje zmogljivosti tehnologije veriženja blokov, še posebej pri tehnologiji Zero-knowledge rollups (ZK-rollups). Pri tehnologiji ZK-rollup se ZKP uporablja za preverjanje in združevanje večih transakcij izven verige, nato pa v glavno verigo pošljejo en sam dokaz o njihovi veljavnosti. Ta pristop bistveno poveča prepustnost transakcij, s čimer obravnava enega ključnih izzivov v skaliranju tehnologije veriženja blokov, hkrati pa ohranja varnostna jamstva osnovne verige blokov [8].

- **Verifikacija identitete**: ZKP omogočajo varno verifikacijo identitete brez razkrivanja občutljivih osebnih podatkov. Ta pristop posameznikom omogoča, da dokažejo specifične vidike svoje identitete, brez da s preveriteljem delijo dejanske vrednosti podatkov [10].
- **Varni sistemi glasovanja**: Sistemi elektronskega glasovanja lahko uporabljajo ZKP za zagotavljanje integritete glasovanja in zasebnosti volivcev. Ta tehnologija omogoča volivcem, da preverijo, ali je bil njihov glas pravilno preštet, ne da bi razkrili svojo izbiro, s čimer ohranja tako tajnost kot tudi preglednost glasovanja [1].
- **Zmanjšanje velikosti transakcij verig blokov**: Protokol Bulletproof bistveno izboljša velikost območnih dokazov, ki so linearno odvisni od števila elementov, v obstoječih predlogih za zaupne transakcije v Bitcoinu in drugih kriptovalutah. S tem zmanjša velikost transakcij, kar vodi do izboljšane učinkovitosti in razširljivosti v omrežjih verig blokov [4].

Tehnologija ZKP ima širok spekter uporabe, vendar smo se v tem delu posvetili dvema specifičnima področjema: decentralizirani identiteti in strojnemu učenju. Izbor teh dveh področij temelji na naših praktičnih izkušnjah in preverjenih rezultatih.

3.1. Decentralizirana identiteta

V hitro razvijajočem se okolju upravljanja digitalne identitete so se decentralizirane rešitve izkazale kot obetaven pristop k reševanju vprašanj glede zasebnosti, in posameznikom omogočajo večji nadzor nad njihovimi osebnimi podatki. To podpoglavje obravnava uporabo tehnologije ZKP v kontekstu decentralizirane identitete, s posebnim poudarkom na njeni integraciji in uporabi v naši lastni decentralizirani denarnici Masca. Za ta namen smo uporabili platformo ZKP, znano kot Privado ID [21]. Ta tehnologija, prej poznana pod imenom PolygonID [17], predstavlja nabor razvojnih orodij, ki omogočajo učinkovito implementacijo zaupanja vredne in zasebne izmenjave preverljivih poverilnic [7]. Privado ID temelji na napredni kriptografiji in tehnologiji verige blokov, kar zagotavlja visoko raven varnosti in zasebnosti pri upravljanju digitalne identitete v okviru naše rešitve. Pri tem smo se osredotočili na naslednje tri ključne komponente teh orodij:

- Osnovni protokol Iden3 [15] in njegova vloga pri upravljanju identitete.
- Postopek izmenjave sporočil, ki omogoča varno preverjanje identitete.
- Uporaba ZKP v tem procesu za večjo varnost in zasebnost.

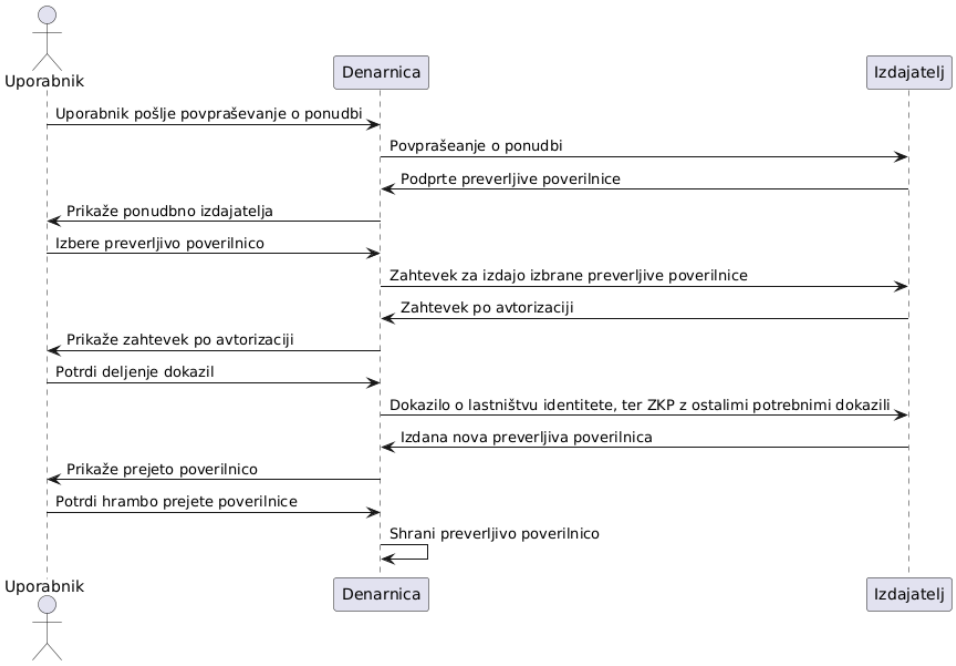
Glavna naloga Iden3 protokola je skrb za upravljanje uporabnikove identitete. Uporabnik lahko ima eno ali več identitet, ki so na omrežju Ethereum predstavljene kot računi (ang. accounts) ali pametne pogodbe (ang. smart contracts). Identitete lahko tako ustvarjajo kot tudi prejemajo izjave o identitetah, npr. izjava o starosti, državljanstvu ali izobrazbi. Te izjave služijo kot povezave med akterji in so lahko javne ali zasebne. Uporabniki lahko tudi kreirajo dokazila o prejetih izjavah. Ta dokazila se med drugim pri Iden3 protokolu ustvarjajo s pomočjo tehnologije ZKP, kjer se specifično uporablja sistem zk-SNARK. Sporočila in izjave v protokolu so tudi digitalno podpisana, kar omogoča preverjanje izvora, torej izdajatelja, in zagotovi, da ni prišlo do sprememb podatkov v tranzitu. Zraven omenjenih akterjev in konceptov so v Iden3 protokolu ključnega pomena tudi merklova drevesa (ang. Merkle trees). Ta zagotavljajo skalabilnost protokola tako, da zmanjšujejo število potrebnih transakcij za posodabljanje stanja na verigi blokov in omejujejo količino podatkov, ki jih je treba hraniti na verigi. Uporabljajo se za dvoje: prvič, za vodenje stanja identitete, kar vključuje evidenco prejetih in izdanih izjav, in drugič, za spremljanje veljavnosti izdanih izjav – torej ali so te izjave še vedno veljavne ali pa so bile preklicane. Pri tem je pomembno poudariti, da v primeru digitalno podpisanih izjav ni potrebno posodabljati stanja na verigi blokov. Digitalni podpis sam po sebi namreč zagotavlja dovolj visoko raven verodostojnosti in avtentičnosti izdane izjave.

Platforma Privado ID zraven protokola Iden3 uporabljajo tudi protokol Iden3comm [16]. Ta definira pravila za komunikacijo med agenti ter denarnicami, in je grajen na protokolu za sporočanje DIDComm [6]. Izjave v

protokolu so predstavljene v obliki W3C preverljivih poverilnic (ang. verifiable credentials) [24]. Pravila definirajo obliko sporočil za izdajanje preverljivih poverilnic, avtorizacijo, izmenjavo dokazov in povpraševanja o preklicu izdanih poverilnic.

Življenjski cikel izdajanja preverljive poverilnice na aplikacijskem nivoju, pri uporabi Privado ID orodij je prikazan na Sliki 1 in je sledeč:

- Na začetku uporabnik preko svoje digitalne denarnice pošlje povpraševanje o ponudbi izdajatelju. S tem si pridobi informacije o poverilnicah, ki jih le ta lahko izdaja.
- Nato uporabnik izbere določeno preverljivo poverilnico in preko denarnice pošlje zahtevek za izdajo le te.
- Strežnik oz. agent se odzove z zahtevkom za avtorizacijo, kjer mora uporabnik dokazati lastništvo identitete s katero želi prevzeti poverilnico. Ta zahtevek lahko vsebuje tudi druge pogoje, ki jih uporabnik mora dokazati, npr. da je polnoletna oseba, da ima opravljeno določeno stopnjo izobrazbe ipd.
- Denarnica uporabniku prikaže avtorizacijski zahtevek, na primer za dokazovanje polnoletnosti, in uporabnik odobri deljenje dokazil.
- Uporabnik se nato odzove s potrebnimi dokazili, pri čemer lastništvo identitete dokaže z digitalnim podpisom, za ostala dokazila pa uporabi ZKP. Za ustvarjanje podpisov in dokazil je odgovorna uporabnikova denarnica. V primeru dokazovanja polnoletnosti denarnica ustvari ZKP iz osebnega dokumenta v obliki preverljive poverilnice. Ta vsebuje le dokaz, da je oseba starejša od 18 let, pri čemer ne razkrije nobenih dodatnih osebnih informacij, niti točne starosti.
- Strežnik nato preveri prejeta dokazila in v primeru, da so bili vsi pogoji izpolnjeni, izda uporabniku preverljivo poverilnico. V nasprotnem primeru se odzove z napako.
- Denarnica prikaže prejeto poverilnico in jo po potrditvi uporabnika shrani.



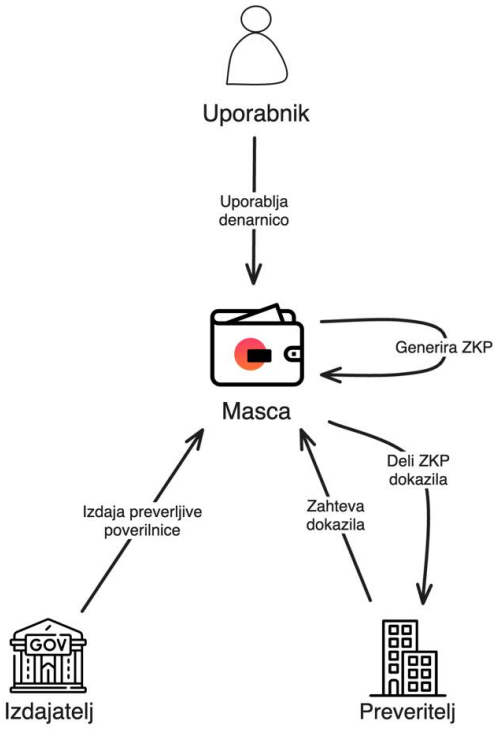
Slika 1: Življenjski cikel izdajanja preverljive poverilnice.

Po pregledu življenjskega cikla izdajanja preverljivih poverilnic z orodji Privado ID in uporabljenih osnovnih protokolov bomo predstavili še integracijo te tehnologije v lastno razvito digitalno denarnico Masca. Masca je

MetaMask Snap [20] (vtičnik), ki razširi delovanje kripto denarnice MetaMask s funkcionalnostmi samo-upravljane identitete (ang. self-sovereign identity, SSI) oz. decentralizirane identitete [19]. Obstoječim funkcionalnostim smo s pomočjo Privado ID knjižnic dodali podporo za protokola Iden3 in Iden3comm, ter podporo za ustvarjanje ZKP, ki temeljijo na sistemu zk-SNARK. Podpora za procesiranje sporočil Iden3comm protokola je bila dodana tako, da smo razširili naš seznam podprtih QR kod s pravili definiranih v protokolu. S tem smo omogočili uporabnikom denarnice Masca, da uspešno prevzamejo zahtevke za izdajo preverljivih poverilnic, kot tudi zahtevke po avtorizaciji. Procesiranje prevzetih zahtevkov smo opravili s pomočjo integriranih knjižnic, pri čemer smo v sam proces dodali tudi varen in uporabniško prijazen vmesnik. Za ustvarjanje ZKP s sistemom zk-SNARK je bilo potrebno v denarnico Masca uvoziti potrebna vezja. Ta vezja so zaradi svoje splošne opredelitve pogosto precej obsežna, njihova velikost pa lahko presega tudi 200 MB. Služijo kot specializiran programski jezik in nekakšen poizvedovalni jezik, ki določa obseg dokazljivih trditev znotraj določenega sistema ZKP. Obenem zagotavljajo prilagodljiv okvir za izražanje in poizvedovanje kompleksnih trditev v obliki, primerni za kriptografsko preverjanje. Zaradi svoje velikosti ta vezja povzročajo precejšnjo upočasnitev delovanja sistema in podaljšajo čas, potreben za prvo namestitev denarnice. Poleg tega je treba vezja ob vsaki inicializaciji denarnice znova naložiti v pomnilnik, kar dodatno vpliva na hitrost in odzivnost sistema. Za namen izboljšanja delovanje smo implementirali sledeče optimizacije:

- Spremenili smo način, kako so vezja kodirana, s čimer smo zmanjšali začetni čas nalaganja.
- Vezja smo shranili v predpomnilnik (ang. cache), da se izognemo ponovnemu nalaganju v primeru procesiranja zaporednih zahtevkov.
- Vezja se naložijo po potrebi, kar pomeni, da ne vplivajo na delovanje ostalih funkcionalnosti denarnice, le na ustvarjanje ZKP.

Slika 2 predstavlja celoten model zaupanja SSI. Prikazuje uporabnika, ki upravlja svojo digitalno identiteto s pomočjo denarnice, v tem primeru je to Masca. Uporabnik lahko v to denarnico prejema preverljive poverilnice od različnih izdajateljev, kot so državne institucije ali univerze. Prav tako lahko prejema zahtevke po avtorizaciji, na primer ko podjetje želi dokaz o uporabnikovi izobrazbi. Denarnica Masca je odgovorna za obdelavo teh zahtevkov, pripravo ZKP dokazil ter njihovo deljenje s preveritelji.



Slika 2: Model zaupanja SSI.

3.2. Strojno učenje

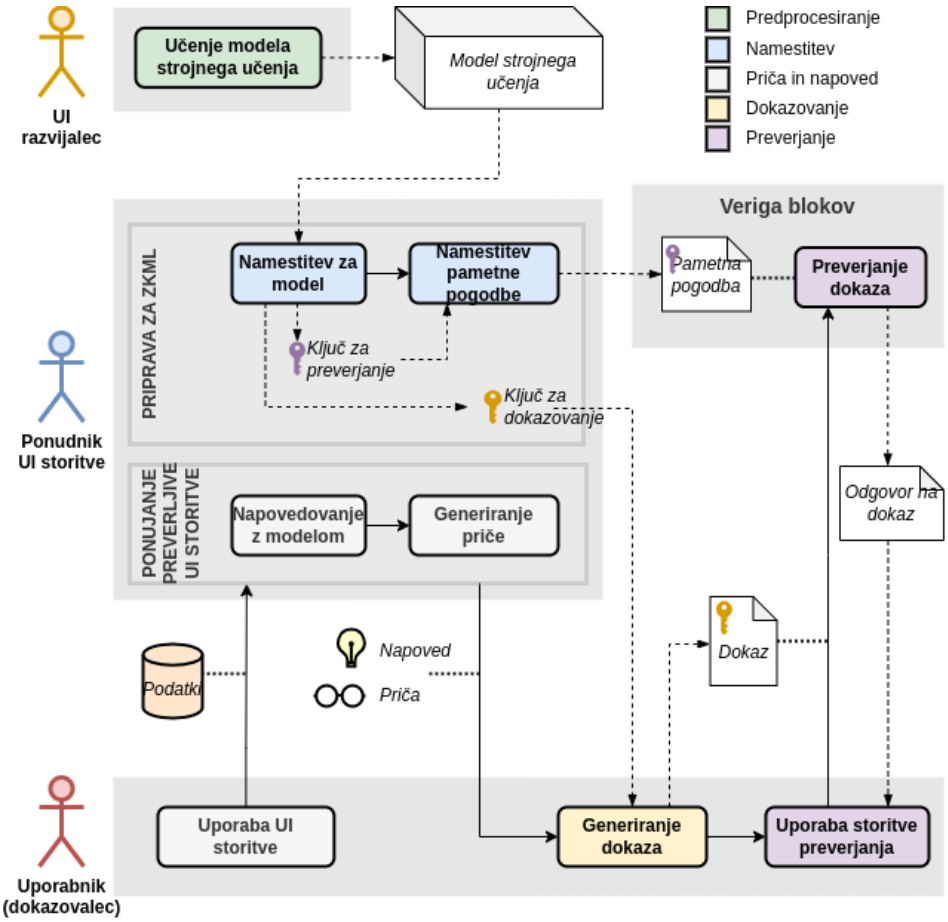
ZKP so se na začetku uporabljali za dokazovanje preprostih trditev in manjše količine podatkov, kot je bilo predstavljenih v prejšnjih poglavjih. Toda zaradi svoje generičnosti se lahko prav tako uporabijo za dokazovanje splošnih računalniških programov napisanih v poljubnem Turingovo polnem jeziku. Eden izmed takšnih primerov uporabe je uporaba ZKP na modelih strojnega učenja, tj. za dokazovanje napovedovanja modelov [12]. To področje se imenuje zero-knowledge machine learning (ZKML) ali pogosto v literaturi preverljivo strojno učenje (ang. verifiable machine learning) [23, 18]. Ker modeli stojnega učenja zelo računsko zahtevni programi, je bila uporaba ZKP na tem področju zelo dolgo praktično nemogoča, vendar je v zadnjih letih s teoretičnimi napredki in razvojem ogrodij postala izvedljiva, vendar le za napovedovanje in ne celotno učenje.

Dodatna vrednost, ki jo prinese ZKML, je preverljivost napovedovanja modelov strojnega učenja. Zaupanje v poštenost operaterjev in ponudnikov API storitev oz. modelov strojnega učenja se nadomesti z matematično/kriptografsko preverljivostjo. Z uporabo ZKML je mogoče odgovoriti na naslednja vprašanja oz. dokazati sledeče trditve:

- Uporabljen je bil točno določen model strojnega učenja M.
- Napoved y je bila pri uporabi modela M dobljena pri uporabi vhodnih podatkov x.
- Pri določenem vhodu x (ni razkrit preveritelju) smo dobili izhod y (razkrit preveritelju).

Medtem ko ZKML sam po sebi definira le način generiranja in preverjanja dokazov, sta se razvila dva prevladujoča načina uporaba: na verigi blokov (ang. on-chain) in izven (ang. off-chain). Pri slednjem potekata generiranje in preverjanje dokazov izven verige blokov tj., na strežniku, medtem ko pri prvem načinu poteka preverjanje dokaza na verigi blokov tj., s pomočjo pametnih pogodb. ZKML se lahko tako uporabi znotraj obstoječih informacijskih sistemov, ki ponujajo storitve strojnega učenja kot tudi pri decentraliziranih aplikacijah (ang. dapp).

Na sliki 3 je prikazan postopek ZKML na verigi blokov, v katerem smo definirali štiri tipe akterjev - razvijalec umetne inteligence (UI), ponudnik UI storitve, uporabnik, ki je v vlogi dokazovalca, in veriga blokov s pametno pogodbo, ki je v vlogi preveritelja. UI razvijalec poskrbi za definicijo in učenje modela strojnega učenja, kar spada pod pred-procesiranje in ni neposredno del ZKML. Ponudnik UI storitve izvede namestitev za izbran model, kjer se generirata ključa za dokazovanje in preverjanje, pri čemer se slednji namesti kot del pametne pogodbe na verigo blokov. Uporabnik nato uporablja UI storitve, pri čemer prejme poleg same napovedi tudi podatke potrebne za generiranje dokazov t. i. pričo (ang. witness). Po generaciji dokaza lahko uporabnik sam dokaz uporabi v decentraliziranih aplikacijah, kjer verifikacija dokaza poteka s pametnimi pogodbami.



Slika 3: Postopek uporabe ZKML na verigi blokov.

Trenutno najbolj razviti ogrodji za ZKML sta ezkl in Orion. Prvo ogrodje omogoča generiranje zk-SNARK dokazov, medtem ko drugo zk-STARK. ezkl temelji na ZKP sistemu Halo2, pri čemer omogoča pretvorbo modelov strojnega učenja v razširjenem formatu ONNX (Open Neural Network Exchange) v Halo2 vezja. Ogrodje Orion medtem vsebuj orodja in knjižnice za implementacijo modelov strojnega učenja v programskem jeziku Cairo, tj. programski jezik za programiranje preverljivih programov in pametnih pogodb za L2 omrežju Starknet. Ogrodji se razlikujeta tudi po različnih načinih zasebnosti, saj Orion ponuja le javen način, pri katerem so razkrite vse vrednosti (vhodi, izhod in uteži), medtem ko ezkl omogoča tudi zasebne načine, pri čemer se razkrijejo le določeni podatki. Primerjava med ezkl in Orion je povzeta v tabeli 2.

Tabela 2: Primerjava ezkl in Orion.

Lastnost	ezkl	Orion
ZKP sistem	zk-SNARK	zk-STARK
Zaupanja vredna namestitev	Da	Ne
ZKP ogrodje	Halo2	Cair – Stone, Platinum ...
Podprti modeli	Nevronske mreže, odločitvena drevesa, naključni gozdovi ...	Nevronske mreže, odločitvena drevesa, naključni gozdovi, SVM ...
Programski jeziki	Rust, Python, JavaScript	Cairo, Python
Načini zasebnosti	Zasebno, javno, šifrirano	Javno

ZKML lahko na več načinov poveča funkcionalnosti decentraliziranih aplikacij, pri čemer se kompleksnejše računanje (napovedovanje modela) izvede izven verige blokov, medtem ko se preverjanje dokaza in uporaba izhoda zgodi na verigi blokov v pametnih pogodbah. Eden izmed takšnih primerov uporabe je diverzifikacija portfelja

kriptožetonov decentraliziranih avtonomnih organizacij (ang. decentralized autonomous organization, DAO). V finančni industriji se modeli strojnega učenja pogosto uporabljajo v te namene, medtem ko se v portfeljih na verigi blokov ne, saj so modeli prekompleksni, da bi se neposredno izvedli v pametnih pogodbah. Če pa uporabimo samo napoved kot parameter funkcije, ne vemo, kako je bila napoved o prerazporejanju sredstev pridobljena - torej, če je bil uporabljen res točno izbran model strojnega učenja. ZKML omogoča, da se model napovedovanja izvede izven verige blokov, bodisi na lokalnem računalniku ali na strežniku, pri čemer se dodatno generira ZKP dokaz o napovedovanju/inferenci modela. Ko se nato kliče funkcija na pametni pogodbi, ki izvede prerazporejanje kriptožetonov od DAO-ta, sprejme kot vhod ZKP dokaz in izvede preverjanje dokaza, pri čemer se prerazporejanje zgodi le, če je bila dobljena napoved/strategija za prerazporejanje res izhod tistega modela strojnega učenja.

4 Diskusija

ZKP se danes aktivno in uspešno uporabljajo za različne primere, pri čemer smo se v članku osredotočili na primer decentralizirane identitete in strojnega učenja. Pri prvem omogočajo varnejšo in bolj zasebno razkritje osebnih podatkov, kjer je mogoče le razkrite nujno potrebne vrednosti oz. v določenih primerih le predikate, na primer da je določena vrednost večja od določene številke. Pri strojnem učenju dodajo zmožnost preverjanje, kar omogoči verifikacijo izvedbe modela za napovedovanje. Na ta način lahko uporabnik preveri, da je bil uporabljen izbran model ob določenem vhodu, pri čemer se zaupanje v ponudnika storitve nadomesti z matematičnimi zakoni.

Kljub vsem prednostim imajo ZKP še danes določene pomanjkljivosti. Ena izmed pomanjkljivosti so kompleksnejši sistemi, saj je za generiranje ZKP dokazov potrebno imeti dostop do ZKP vezij, ki so pogosto velika več sto MB, kar je lahko za mobilne naprave z nekoliko slabšimi procesorji zahtevno. Samo generiranje ZKP dokazov je tudi zahteven proces, ki lahko v določenih primerih traja več sekund oz. v primeru strojnega učenja več minut/ur.

5 Zaključek

ZKP so ena izmed tehnologij, ki je v zadnjih letih prešla iz teorije v praktično uporabo. Zaradi svojih značilnosti lahko obstoječe in nove aplikacije, na primer decentralizirane, nadgradi z novimi funkcionalnostmi, kot so večji nadzor nad zasebnimi podatki, povečana zasebnost in varnost. V članku smo predstavili njeno uporabnost na primeru decentralizirane identitete in strojnega učenja.

Čeprav se ZKP lahko že danes uporabljajo v praksi, je odprtih še veliko izzivov, ki jih je potrebno nasloviti. Eden izmed takšnih izzivov je skalabilnost tehnologije na primere z večjo količino podatkov, kar se še posebej pozna pri strojnem učenju, kjer modeli oz. nevronske mreže v praksi vsebujejo več milijonov uteži/nevronov. Raziskovalci se prav tako ukvarjamo z razvojem novih ZKP protokolov, ki naslavljajo odprte probleme trenutnih zk-SNARK in zk-STARK sistemov, kot so zaupanja vredna namestitev, velikost dokazov in odpornost na kvantne računalnike.

Literatura

[1] Ben Adida. "Helios: web-based open-audit voting". In: Proceedings of the 17th Conference on Security Symposium. SS'08. San Jose, CA: USENIX Association, 2008, pp. 335–348.

[2] Eli Ben Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: 2014 IEEE Symposium on Security and Privacy. 2014, pp. 459–474. doi: 10.1109/SP.2014.36.

[3] Eli Ben-Sasson et al. "Scalable, transparent, and post-quantum secure computational integrity". In: Cryptology ePrint Archive (2018).

[4] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Trans- actions and More". In: 2018 IEEE Symposium on Security and Privacy (SP). 2018, pp. 315–334. doi: 10.1109/SP.2018.00020.

[5] Zhigang Chen et al. "A Survey on Zero-Knowledge Authentication for Internet of Things". In: Electronics 12.5 (2023). issn: 2079-9292. doi: 10.3390/electronics12051145. url: https://www.mdpi.com/2079- 9292/12/5/1145.

[6] DIDComm messaging protocol. url: https://identity.foundation/didcomm-messaging/spec/v2.1/ (obiskano 16.7.2024).

[7] Martin Domajnko, Vid Keršič, and Muhamed Turkanović. "OID4VC: izdajanje in deljenje preverljivih poverilnic na osnovi OpenID Connect". In: Nasl. z nasl. zaslona. Univerza v Mariboru, Univerzitetna založba; Fakulteta za elektrotehniko, računalništvo in informatiko, 2023, pp. 149–163. url: https://press.um.si/index.php/ump/catalog/book/804.

[8] Ethereum developer documentation: Zero-knowledge rollups (ZK-rollups). url: https://ethereum.org/en/developers/docs/scaling/zk- rollups/ (obiskano 15.7.2024).

[9] EU Digital Identity Wallet Home. url: https://ec.europa.eu/digital-building-blocks/sites/display/EUDIGITALIDENTITYWALLET/EU+Digital+Identity+Wallet+Home (obiskano 19.7.2024).

[10] Uriel Feige, Amos Fiat, and Adi Shamir. "Zero-knowledge proofs of identity". In: Journal of Cryptology 1.2 (June 1988), pp. 77–94. issn: 1432- 1378. doi: 10.1007/bf02351717. url: http://dx.doi.org/10.1007/ BF02351717.

[11] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: Conference on the theory and application of cryptographic techniques. Springer. 1986, pp. 186–194.

[12] Rosario Gennaro, Craig Gentry, and Bryan Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers". In: Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30. Springer. 2010, pp. 465–482.

[13] Rosario Gennaro et al. "Quadratic span programs and succinct NIZKs without PCPs". In: Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32. Springer. 2013, pp. 626–645.

[14] S Goldwasser, S Micali, and C Rackoff. "The knowledge complexity of interactive proof-systems". In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. STOC '85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 291–304. isbn: 0897911512. doi: 10.1145/22145.22178. url: https://doi. org/10.1145/22145.22178.

[15] Iden3 protocol documentation. url: https://docs.iden3.io/ (obiskano 16.7.2024).

[16] Iden3comm protocol documentation. url: https://iden3-communication. io/ (obiskano 16.7.2024).

[17] Introducing Privado ID. url: https://www.privado.id/blog/introducing-privado-id-moving-beyond-polygon-to-deliver-independent-privacy-preserving-identity-solutions (obiskano 16.7.2024).

[18] Vid Kersic and Muhamed Turkanovic. "A review on building blocks of decentralized artificial intelligence". In: arXiv preprint arXiv:2402.02885 (2024).

[19] Vid Keršič et al. "Dodajanje poljubnih funkcionalnosti digitalni kripto denarnici MetaMask". In: Nasl. z nasl. strani. Univerza v Mariboru, Univerzitetna založba; Fakulteta za elektrotehniko, računalništvo in informatiko, 2022, pp. 141–154. url: https://dk.um.si/IzpisGradiva. php?id=82880.

[20] MetaMask Snaps. url: https://metamask.io/snaps/.

[21] Privado ID. url: https://www.privado.id/ (obiskano 16.7.2024).

[22] Srinath Setty. "Spartan: Efficient and general-purpose zkSNARKs without trusted setup". In: Annual International Cryptology Conference. Springer. 2020, pp. 704–737.

[23] Tobin South et al. "Verifiable evaluations of machine learning models using zkSNARKs". In: arXiv preprint arXiv:2402.02675 (2024).

[24] Verifiable Credentials Data Model v2.0. url: https://www.w3.org/TR/vc-data-model-2.0/ (obiskano 16.7.2024).

Signali v programskem jeziku JavaScript

Gregor Jošt, Viktor Taneski

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

V sodobnem spletnem razvoju je asinhrona komunikacija ključna za zagotavljanje odzivne uporabniške izkušnje. Tradicionalne metode, kot so povratne funkcije in obljube, lahko postanejo kompleksne. Medtem ko sintaksa async/await poenostavi asinhrono programiranje, še vedno ne rešuje vseh težav. Signali predstavljajo sodobno rešitev za te izzive, saj omogočajo modularno in prilagodljivo obvladovanje dogodkov. Signali omogočajo preprosto registracijo in sprožanje dogodkov ter boljšo modularnost v aplikacijah. Prispevek se osredotoča na preučitev signalov v JavaScriptu, njihovih osnovnih konceptov, primerov uporabe in prednosti, kot so zmanjšanje odvisnosti med komponentami, izboljšana asinhrona komunikacija in poenostavljeno upravljanje stanja. Poudarjena je tudi integracija signalov v priljubljena ogrodja JavaScript, kot so React.js, Vue.js in Angular.

Ključne besede:
signali
JavaScript
reaktivnost
obljube
asinhrona komunikacija

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.18

1 Uvod

1.1. Ozadje

V sodobnem spletnem razvoju je asinhrona komunikacija ključnega pomena za zagotavljanje tekoče in odzivne uporabniške izkušnje. Tradicionalne metode za obvladovanje asinhronih operacij, kot so povratne funkcije (angl. callback functions) in obljube (angl. promises), lahko hitro postanejo kompleksne in težavne za vzdrževanje, zlasti pri večjih projektih.

Povratne funkcije pogosto vodijo v tako imenovani pekel povratnih klicev (angl. callback hell), kjer je koda globoko vgnezdena in posledično težko berljiva [1]. Obljube so bile uvedene za rešitev tega problema in so omogočile bolj linearen način obvladovanja asinhronih operacij. Z njimi je mogoče verižno povezovati asinhrone operacije, kar poenostavi obvladovanje napak in izboljša berljivost kode. Vendar pa tudi obljube niso popolna rešitev, saj kompleksne asinhrone tokove podatkov še vedno težko upravljamo in sledimo njihovemu stanju [1].

Z uvedbo sintakse »async/await« je asinhrono programiranje postalo še bolj intuitivno in podobno sinhronemu programiranju. Kljub temu pa async/await ni univerzalna rešitev za vse izzive, saj ne ponuja preprostega načina za obvladovanje več asinhronih dogodkov, ki se zgodijo v različnih delih aplikacije. Tukaj pridejo v poštev signali, ki omogočajo bolj modularno in prilagodljivo obvladovanje dogodkov.

Signali so vzorec (in implementacija), ki omogoča preprosto registracijo in sprožanje dogodkov, kar omogoča bolj modularno in vzdržljivo kodo. Ta pristop je še posebej koristen v večjih aplikacijah, kjer je treba upravljati komunikacijo med različnimi moduli ali komponentami. Pogosto se primerjajo z vzorcem »opazovalec« (angl. Observable pattern), vendar dejstvo je, da se signali razlikujejo od vzorca in ponujajo preprostost in takojšno reaktivnost [2].

V zadnjih letih so signali pridobili na priljubljenosti saj poenostavljajo in izboljšujejo asinhrono komunikacijo med komponentami. Poleg tega signali omogočajo boljšo modularnost, saj komponente med seboj komunicirajo prek signalov, ne da bi morale neposredno vedeti ena za drugo. To vodi do bolj pregledno programsko kodo, ki jo je lažje testirati in razširjati.

1.2. Cilji prispevka

Cilj prispevka je raziskati uporabo signalov v JavaScriptu in prikazati, kako lahko ta vzorec izboljša asinhrono komunikacijo v spletnih aplikacijah. Prispevek se osredotoča na naslednje cilje.

Pregled trenutnih metod za asinhrono komunikacijo v JavaScriptu, vključno s povratnimi funkcijami, obljubami in sintakso »async/await«. Ta pregled bo vključeval zgodovinski razvoj teh metod, njihove prednosti in slabosti ter primere uporabe v različnih scenarijih.

Predstavitev osnovnih konceptov signalov in njihove uporabe v JavaScriptu. Pojasnili bomo, kaj so signali, kako delujejo in zakaj so koristni pri asinhronem programiranju. Poleg tega bomo podali osnovne primere implementacije signalov.

Analiza prednosti uporabe signalov, vključno z izboljšano modularnostjo in poenostavljenim upravljanjem stanja. Poudarili bomo, kako signali omogočajo bolj jasno programsko kodo in kako olajšajo upravljanje kompleksnih asinhronih tokov podatkov.

Namen prispevka je predstaviti, kako lahko signali poenostavijo asinhrono komunikacijo in izboljšajo strukturo in vzdrževanje kode v spletnih aplikacijah. Raziskali bomo primere iz prakse in prikazali, kako signali omogočajo učinkovito obravnavo dogodkov in izboljšajo uporabniško izkušnjo. Cilj je tudi spodbuditi nadaljnje raziskave na tem področju ter predstaviti nove možnosti uporabe signalov v prihodnosti.

1.3. Struktura prispevka

Prispevek je organiziran na naslednji način. V drugem poglavju bomo predstavili osnove signalov v JavaScriptu, vključno z njihovimi definicijami, osnovnimi implementacijami ter integracijami z različnimi ogrodji. Tretje poglavje bo namenjeno analizi prednosti uporabe signalov, četrto pa bo prikazalo praktični primer uporabe signalov s pomočjo knjižnice @preact/signals-core. V zadnjem, petem poglavju bomo predstavili povzetke, ugotovitve in predloge za prihodnje raziskave.

2 Osnove signalov v JavaScriptu

2.1. Kaj so Signali?

Signali so pomemben koncept reaktivnosti v jeziku JavaScript. Lahko vsebujejo vrednost in omogočajo, da se določene komponente ali deli kode odzovejo, ko se ta vrednost spremeni. Prav tako omogočajo bolj nadzorovano uporabljanje stanja in natančno reaktivnost, kjer se komponente posodabljajo selektivno glede specifične spremembe stanja, namesto da bi se posodabljale v celoti [3].

2.2. Osnovni koncepti Signalov

- *Registracija poslušalcev:* Poslušalci (ali obravnavalci dogodkov) so funkcije, ki se registrirajo pri signalu, da prejmejo vrednosti, ko se signal sproži. To je mogoče storiti z dodajanjem funkcije na seznam poslušalcev signala. Ta registracija omogoča, da so različni deli aplikacije obveščeni o določenem dogodku, ne da bi morali neposredno komunicirati z drugimi komponentami. Na primer, več komponent se lahko registrira za prejemanje obvestil o spremembah stanja uporabnika ali o novih sporočilih v aplikaciji za klepet.
- *Sprožanje signalov:* Ko pride do določenega dogodka (t.j. spremembe vrednosti signala), se vsi registrirani poslušalci obvestijo in izvedejo z določenimi podatki. To omogoča asinhrono komunikacijo med različnimi deli aplikacije. Sprožanje signalov je ključnega pomena za odzivnost aplikacije, saj omogoča takojšnjo obravnavo dogodkov, kot so uporabniške interakcije, spremembe podatkov ali omrežne aktivnosti. Na primer, ko uporabnik pošlje sporočilo v klepetu, se signal sproži, kar obvesti vse registrirane poslušalce o novem sporočilu.
- *Odstranjevanje poslušalcev:* Poslušalce je mogoče tudi odstraniti s seznama, kar pomeni, da ne bodo več prejemali obvestil, ko se signal sproži. To je uporabno v primerih, ko komponenta ne potrebuje več obvestil o določenih dogodkih ali ko se komponenta uniči, da se preprečijo morebitni uhajanja pomnilnika (angl. memory leaks). Na primer, ko uporabnik zapre okno za klepet, se lahko poslušalec odstrani, da ne prejema več obvestil o novih sporočilih.

Registracija in odstranjevanje poslušalcev omogočata fleksibilno in dinamično upravljanje dogodkov, saj se lahko komponente enostavno prijavijo ali odjavijo glede na svoje potrebe. To zmanjšuje povezanost med komponentami in omogoča bolj modularno in vzdržljivo zasnovo aplikacij.

2.3. Primer osnovnega Signala

V svoji osnovni obliki signali delujejo kot posredniki med različnimi deli aplikacije. To izboljšuje modularnost in ponovno uporabo kode, saj komponente komunicirajo preko signalov namesto neposrednih klicev funkcij.
Signali torej služijo kot posredniki med različnimi deli aplikacije. To izboljšuje modularnost in omogoča ponovno uporabo kode, saj komponente komunicirajo preko signalov namesto preko neposrednih klicev funkcij.

Spodnji izsek kode prikazuje primer implementacije osnovnega signala:

```
const [2] = require('signals');
let count = signal(0); // začetna vrednost signala
count.observe(value => console.log(value)); // obdelovalec dogodkov, ki reagira na spremembe signala
count(10); // nova vrednost, dodeljena signalu, kar sproži opazovalca
```

V zgornjem primeru je prikazana uporaba signala za reaktivno spremljanje sprememb vrednosti. Najprej se ustvari signal `count` z začetno vrednostjo 0. Nato se registrira opazovalec z metodo `observe`, ki se sproži vsakič, ko se spremeni vrednost signala. Opazovalec prejme novo vrednost signala in izvede funkcijo, ki jo podamo, v tem primeru izpiše novo vrednost v konzolo. Na koncu se spremeni vrednost signala `count` na 10, kar sproži opazovalca in povzroči, da se nova vrednost izpiše v konzolo.

To deluje tako, da signal ob spremembi vrednosti obvesti vse registrirane opazovalce (poslušalce), kar omogoča reaktivno posodabljanje stanja in izvajanje povezanih funkcij.

2.4. Signali kot del jezika JavaScript

V času pisanja tega prispevka je predlog za vključitev signalov v standard JavaScript v 1. fazi (Stage 1) v postopku TC39 [4], kar je obetaven znak, da bi lahko v prihodnosti postali del standarda. 1. faza pomeni, da je bil koncept uradno predstavljen in da odbor prepoznava potrebo po rešitvi, vendar je predlog še vedno v zgodnji fazi razvoja. Če bodo signali napredovali v poznejše faze in postali del standarda, bi to lahko prineslo poenoteno in standardizirano upravljanje stanja in reaktivnosti v JavaScriptu. To bi zmanjšalo potrebo po različnih knjižnicah in ogrodjih z različnimi implementacijami ter olajšalo delo razvijalcem pri razvoju sodobnih aplikacij. Vključitev signalov v standard bi lahko pripeljala do bolj konsistentnega razvoja in izboljšanja učinkovitosti, kar bi imelo dolgoročne pozitivne učinke na celotno razvojno skupnost.

Predlog signala je skupno delo, ki ga vodita Rob Eisenberg in Daniel Ehrenberg [4]. Vključuje prispevke pri oblikovanju avtorjev in vzdrževalcev glavnih orodij JavaScript, kot so Angular, React in Vue. Predlog si prizadeva vzpostaviti skupno osnovo za signale, ki jo lahko sprejmejo različna JavaScript ogrodja, kar spodbuja interoperabilnost in zmanjšuje razdrobljenost ekosistema. S poudarkom na osnovni semantiki signalnega grafa, želi predlog zagotoviti osnovo, na kateri lahko ogrodja gradijo, namesto da bi določal API, usmerjen na razvijalce.

2.5. Vzorec Opazovalec vs. Signali: v čem je razlika?

Vzorec Opazovalec se bistveno razlikuje od vrednosti oz. signala. Opazovalec je podoben cevi, ki lahko kadar koli dostavi novo vrednost. To prinaša več pomembnih posledic. Prvič, v cev ne moremo preprosto pogledati in prebrati »trenutno« vrednost. Namesto tega je treba registrirati povratni klic in počakati, da se povratni klic izvede, ko se pojavi nova vrednost. Drugič, opazovalci dostavljajo vrednosti čez čas, kar pomeni, da je koncept »časa« osrednji del opazovalcev. Signali, nasprotno, nimajo koncepta časa, saj vedno lahko pridobimo »trenutno« vrednost signala. Ta razlika – dostop do »trenutne« vrednosti v primerjavi s cevjo, ki dostavlja vrednosti čez čas – je ključna razlika med signali in opazovalci. Signali so torej kot posoda, ki vedno vsebuje najnovejšo vrednost. Kadar koli jo pogledamo, vidimo trenutno stanje. Opazovalci pa so kot cev, po kateri tečejo vrednosti čez čas. Potrebujemo način (povratni klic), da »poberemo« nove vrednosti, ko pridejo.

To pomeni, da lahko pri signalih vedno preprosto dostopamo do trenutne vrednosti, medtem ko moramo pri opazovalcih nastaviti mehanizem za prejemanje vrednosti, ko se te pojavijo.

2.6. Integracija Signalov v različnih ogrodij

2.6.1. Integracija z React.js

Trenutno je za upravljanje stanja med komponentami v Reactu potrebno to stanje prenašati prek lastnosti (angl. props), ustvariti kontekst ali uporabiti nekaj, kot je Redux, ki pa ima svoje lastne težave in veliko predpripravljene kode. Ogrodje Preact, ki velja za lahko in hitro različico Reacta, kaže pot z zgledom tako, da najprej uvaja signale.

V ogrodjih, kot je Preact, so signali implementirani kot objekti z lastnostjo *.value*, ki hrani stanje. Komponente lahko berejo in spreminjajo to vrednost, pri čemer Preact učinkovito upravlja ponovne upodobe s posodabljanjem samo tistih komponent, ki so neposredno naročene na signal [6]. Tukaj je osnovni primer v Preactu:

```
import { signal, computed } from "@preact/signals";

const count = signal(0);

// Preberi vrednost signala
console.log(count.value); // 0

// Posodobi vrednost signala
count.value += 1;
console.log(count.value); // 1

// Določi komponento, ki reagira na spremembe signala
function Counter() {
 const value = count.value;

 const increment = () => {
  count.value++;
 }

 return (
  <div>
   <p>Število: {value}</p>
   <button onClick={increment}>klikni me</button>
  </div>
 );
}
```

Podrobnejši primer integracije signalov v ekosistem Preact bomo predstavili v poglavju 4.

V Reactu signali nudijo robusten mehanizem za komunikacijo med komponentami, ki presega tradicionalno upravljanje stanj. Ta pristop, ki temelji na dogodkih, omogoča komponentam, da oddajajo in poslušajo signale, kar spodbuja bolj ohlapno povezanost in odzivno arhitekturo. Integracija signalov v React je mogoča s pomočjo knjižnice *@peact/signals-react* in prinaša nekaj ključnih prednosti, in sicer [5]:

- Ločevanje komponent – signali omogočajo komunikacijo med komponentami brez neposrednih odvisnosti, kar pripomore k modularnemu in vzdržljivemu kodnemu načrtu.
- Arhitektura, ki temelji na dogodkih – ta arhitektura omogoča fleksibilno, razširljivo zasnovo, kjer komponente reagirajo na signale glede na specifične skrbi, kar izboljšuje odzivnost in intuitivnost uporabniškega vmesnika.

- Poenostavljeno upravljanje stanj – medtem ko je upravljanje stanj v Reactu zelo zmogljivo, signali nudijo alternativo za določene scenarije, saj omogočajo upravljanje specifičnih vidikov stanja aplikacije brez centraliziranega sistema za upravljanje stanj.

Oglejmo si primer implementacije signala za upravljanje opravil, shranjenih v lokalni shrambi. Začetna vrednost signala se pridobi iz lokalne shrambe:

```
const LOCAL_STORAGE_KEY = "todos";

export const getTodosFromLocalStorage = () => {
  const todos = localStorage.getItem(LOCAL_STORAGE_KEY);
  return todos ? JSON.parse(todos) : [];
};
export const todosSignal = signal(getTodosFromLocalStorage());
```

Ko se vrednost signala spremeni, se učinek (angl. effect) shrani nazaj v lokalno shrambo:

```
export const setTodosToLocalStorage = (todos) => {
  localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify(todos));
};

effect(() => {
  setTodosToLocalStorage(todosSignal.value);
});
```

Poleg tega se dinamično preverjanje izvede, kadar se kateri koli signal znotraj funkcije spremeni, kot je signal opravil:

```
export const completedTodoCount = computed(
    () => todosSignal.value.filter((todo) => todo.isDone).length,
 );
```

Ta nastavitev se sproži pri dodajanju novega opravila:

```
const onAddTodo = () => {
    const todo = {
      id: uuidV4(),
      text,
      isDone: false,
    };

    todosSignal.value = [todo, ...todosSignal.value];

    setText("");
  };
```

2.6.2. Integracija z Vue.js

Vue.js, znan po svojem robustnem sistemu reaktivnosti, implementira signale na podoben način. Vue.js uporablja sistem Proxy za prestrezanje dostopa do lastnosti in sprememb, kar zagotavlja učinkovito sledenje in širjenje posodobitev [6, 7]. Tukaj je primer, kako deluje reaktivnost v Vue.js:

```
import { reactive, computed } from "vue";

const todos = reactive([
  { text: "Kupiti živila", completed: false },
  { text: "Sprehoditi psa", completed: false },
]);

const completed = computed(() => {
  return todos.filter(todo => todo.completed).length;
});

todos[0].completed = true;

console.log(completed.value); // Izhod: 1
```

V tem primeru je *todos* reaktiven seznam, *completed* pa je izračunana lastnost, ki se samodejno posodobi, ko se *todos* spremeni. Zgornja koda je zelo podobna kodi iz poglavja 4, ki predstavlja primer implementacije signalov s pomočjo knjižnice *@preact/signals-core*.

2.6.3. Integracija z Angular

V Angularju signali služijo kot ovoji (angl. wrappers) okoli vrednosti, ki obveščajo zainteresirane porabnike, kadar se vrednost spremeni. Ti signali lahko vključujejo katero koli vrsto vrednosti, od preprostih primitivnih vrednosti do zapletenih podatkovnih struktur. Za dostop do vrednosti signala je treba poklicati njegovo funkcijo za pridobivanje vrednosti, kar Angularju omogoča sledenje uporabe [8].

Pisni signali (angl. writable signals) omogočajo neposredne posodobitve njihovih vrednosti prek API-ja. Ustvarijo se z uporabo funkcije *signal* z začetno vrednostjo. Na primer:

```
const count = signal(0);
console.log('Število je: ' + count());
```

Za spremembo vrednosti pisnega signala lahko uporabimo metodo *.set()*:

```
count.set(3);
```

ali metodo *.update()* za izračun nove vrednosti iz prejšnje:

```
count.update(value => value + 1);
```

Računski signali (angl. computed signals) so samo za branje in izpeljujejo svoje vrednosti iz drugih signalov. Definirajo se z uporabo funkcije *computed*. Na primer:

```
const count: WritableSignal<number> = signal(0);
const doubleCount: Signal<number> = computed(() => count() * 2);
```

Tukaj *doubleCount* temelji na *count* in se posodobi, kadar se *count* spremeni. Izračunani signali se leno vrednotijo in shranijo v predpomnilnik, kar pomeni, da se vrednost izračuna le, ko je prvič dostopana, in ostane v predpomnilniku za kasnejša branja, dokler ni neveljavna zaradi spremembe odvisnosti. Izračunanim signalom ni mogoče neposredno dodeliti novih vrednosti in spremljajo samo signale, ki so bili prebrani med njihovo izpeljavo. Takšno dinamično sledenje odvisnosti omogoča učinkovito ponovno izračunavanje samo, ko je to potrebno.

Učinki (angl. effects) v Angularju so operacije, ki se sprožijo s spremembami ene ali več vrednosti signalov. Ustvarijo se z uporabo funkcije *effect* in vedno tečejo vsaj enkrat ter dinamično sledijo odvisnostim:

```
effect(() => {
  console.log(`Trenutno število je: ${count()}`);
});
```

Učinki se izvajajo asinhrono med postopkom zaznavanja sprememb. Običajno se uporabljajo za beleženje, sinhronizacijo podatkov z lokalno shrambo, prilagojeno obnašanje DOM-a ali prilagojeno upodabljanje (angl. rendering).

3 Prednosti uporabe Signalov

Signali predstavljajo napreden pristop k obvladovanju reaktivnega stanja v aplikacijah in ponujajo številne prednosti v primerjavi z drugimi metodami za asinhrono komunikacijo in upravljanje stanja. V tem poglavju bomo predstavili poglavitne prednosti uporabe signalov.

3.1. Zmanjševanje odvisnosti med komponentami

Ena glavnih prednosti signalov je, da zmanjšajo odvisnosti med različnimi komponentami aplikacije. Tradicionalno se v JavaScriptu komunikacija med komponentami pogosto izvaja prek neposrednih klicev funkcij ali prek deljenja skupnega stanja. To lahko vodi do tesne povezanosti, kar pomeni, da so spremembe v eni komponenti lahko nepredvidljivo vplivale na druge dele aplikacije.

Signali omogočajo obveščanje drugih komponent o dogodkih brez neposredne povezave med njimi. Ko ena komponenta sproži signal, se vse prijavljene komponente obvestijo o dogodku. To pomeni, da lahko komponente reagirajo na dogodke, ne da bi bile neposredno odvisne od drugih komponent, kar zmanjšuje kompleksnost in povečuje fleksibilnost.

Signali torej prispevajo tudi k večji modularnosti aplikacij, saj omogočajo, da so posamezne komponente bolj neodvisne. Z uporabo signalov lahko dosežemo, da vsaka komponenta deluje kot samostojna enota, ki se osredotoča na svojo nalogo in ne skrbi za to, kako bodo druge komponente vplivale nanjo.

3.2. Izboljšana asinhrona komunikacija

Signali bistveno izboljšajo asinhrono komunikacijo med različnimi deli aplikacije. Tradicionalni pristopi k obvladovanju asinhronih dogodkov pogosto vključujejo kompleksne strukture, kot so ugnezdene povratne funkcije ali dolge in nepregledne verige JavaScript obljub, kar lahko vodi do zapletene kode, ki jo je prav tako težko vzdrževati.

Signali omogočajo preprosto spremljanje sprememb in samodejno obveščanje vseh komponent, ki so odvisne od teh sprememb. Ta pristop poenostavi obvladovanje dogodkov, saj odpravlja potrebo po ročnem upravljanju asinhronih operacij in povratnih funkcij. Namesto tega signali omogočajo, da se spremembe v podatkih takoj odražajo v uporabniškem vmesniku, kar pripomore k bolj pregledni in manj zapleteni kodi.

3.3. Poenostavljeno upravljanje stanja

Signali prav tako poenostavijo upravljanje stanja v aplikacijah. Namesto da bi uporabljali različne metode za sinhronizacijo stanj, signali omogočajo centralizirano obdelavo in spremljanje sprememb. To pomeni, da lahko z uporabo signalov enostavno sledimo spremembam v podatkih in samodejno posodabljamo uporabniški vmesnik brez potrebe po ročnem upravljanju stanja.

Ko se stanje spremeni, signali samodejno sprožijo posodobitve v vseh komponentah, ki spremljajo to stanje. To poenostavi razvoj in vzdrževanje aplikacij, saj zmanjša potrebo po dodatnem kodiranju in usklajevanju različnih delov aplikacije, ki so odvisni od enakih podatkov.

4 Primeri uporabe

Zavoljo prikaza uporabe signalov v jeziku JavaScript bomo uporabili knjižnico @preact/signals-core, ki je del ekosistema Preact. Preact je preprosta knjižnica za gradnjo uporabniških vmesnikov, ki ponuja podobne funkcionalnosti kot React, vendar z manjšimi zahtevami po pomnilniku in boljšimi zmogljivostmi. Več o Preact smo povedali v predhodnem poglavju.

Knjižnica @preact/signals-core je specializiran paket za obvladovanje reaktivnega stanja v aplikacijah. Omogoča enostavno spremljanje in obvladovanje sprememb v podatkih in samodejno posodabljanje uporabniškega vmesnika, ko se ti podatki spremenijo. Knjižnica vključuje osnovne funkcionalnosti, kot so:

- signal: Za ustvarjanje reaktivnih podatkovnih točk.
- computed: Za ustvarjanje vrednosti, ki so odvisne od drugih reaktivnih podatkov.
- effect: Za izvajanje stranskih učinkov, ko se reaktivni podatki spremenijo.

Na praktičnem primeru bomo pokazali, kako te funkcionalnosti uporabljamo za obvladovanje stanja aplikacije in samodejno posodabljanje uporabniškega vmesnika brez potrebe po ročnem obvladovanju sprememb stanja. V aplikaciji bomo prikazali seznam artiklov, ki vsebujejo informacije o imenu, kategoriji in ceni. Uporabnik lahko išče po imenu artiklov in filtrira rezultate po kategoriji.

Reaktivno obvladovanje teh funkcij je tipičen primer za signale, ker omogoča samodejno posodabljanje uporabniškega vmesnika brez ročnega posredovanja. Ko uporabnik spremeni iskalno besedilo ali izbiro filtra, signali samodejno poskrbijo za posodobitev prikazanih podatkov v realnem času. To pomeni, da so spremembe v iskalnem polju in filtrih takoj vidne v tabeli. Primer implementacije je prikazan na spodnjem programskem kodu.

```
import { signal, computed, effect } from "@preact/signals-core";
import { IArtikel, podatki } from "./podatki";
import {
  dodajZapis,
  filtriranjeElement,
  iskanjeElement,
  teloTabeleElement,
} from "./dom";

// Signali
```

```
const iskanjeBesedila = signal("");
const filtriranjeVrednosti = signal("");

const filtriraniPodatki = computed(() => {
  const iskalniNiz = iskanjeBesedila.value.toLowerCase();
  const filtriranje = filtriranjeVrednosti.value;

  return podatki.filter(
    (artikel: IArtikel) =>
      artikel.ime.toLowerCase().includes(iskalniNiz) &&
      (filtriranje === "" || artikel.kategorija === filtriranje)
  );
});

// Funkcija za prikaz tabele
const prikaziTabela = () => {
  if (!teloTabeleElement) return;

  teloTabeleElement.innerHTML = "";
  filtriraniPodatki.value.forEach(dodajZapis);
};

// Učinki za obvladovanje reaktivnosti
effect(() => {
  prikaziTabela();
});

// Funkcija za obdelavo sprememb v iskalnem polju
const obdelajIskanje = (e: Event) => {
  const vhod = e.target as HTMLInputElement;
  iskanjeBesedila.value = vhod.value;
};

// Funkcija za obdelavo sprememb v filtrirnem elementu
const obdelajFiltriranje = (e: Event) => {
  const izbirnik = e.target as HTMLSelectElement;
  filtriranjeVrednosti.value = izbirnik.value;
};

// Nastavi obdelovalce dogodkov
if (iskanjeElement) {
  iskanjeElement.addEventListener("input", obdelajIskanje);
}

if (filtriranjeElement) {
  filtriranjeElement.addEventListener("change", obdelajFiltriranje);
}

// Začetni prikaz
```

```
prikaziTabela();
```

Kot je razvidno iz programske kode, v tej aplikaciji uporabljamo signale za obvladovanje in spremljanje stanja iskanja in filtriranja.

Implementacija vključuje dva signala, in sicer *iskanjiBesedila* in *filtriranjeVrednosti*, ki shranjujeta trenutno stanje iskanja in filtriranja. Signala omogočata, da se spremembe v teh vrednostih samodejno spremlja in posodobllja. Ko se te vrednosti spremenijo, se vsi, ki spremljajo te signale, samodejno obvestijo in ustrezno odzovejo na spremembe.

Računska vrednost (angl. computed value) *filtriraniPodatki* je odvisna od omenjenih signalov *iskanjiBesedila* in *filtriranjeVrednosti*. Ko se ti signali spremenijo, se funkcija *computed* znova izračuna. Ko se torej iskalno besedilo ali filtrirna vrednost spremenita, se znova izračuna in vrne posodobljen seznam podatkov. Signali omogočajo, da se filtriranje podatkov zgodi v realnem času brez potrebe po ročni obdelavi sprememb.

Funkcija *prikaziTabela* uporablja rezultate iz *filtriraniPodatki* in jih prikaže v tabeli. Se posodobi, ko se filtrirani podatki spremenijo, kar je posledica sprememb signalov. Funkcija izbriše obstoječe vrstice in ponovno naloži nove, kar zagotavlja, da tabela vedno prikazuje najnovejše rezultate. Signali omogočajo, da se ta posodobitev zgodi samodejno, brez potrebe po ročni spremembi stanja.

Funkcija *effect* zagotavlja, da se funkcija *prikaziTabela* vedno sproži, kadar se signali *iskanjiBesedila* ali *filtriranjeVrednosti* spremenijo. Tako je tabela vedno posodobljena z najnovejšimi podatki brez potrebe po dodatnem nadzoru.

Definirali smo tudi dve funkciji za obdelavo dogodkov (angl. event handlers), in sicer *obdelajIskanje* in *obdelajFiltriranje* za obdelavo sprememb v iskalnem polju in filtrirnem elementu:

- *obdelajIskanje* posodobi signal *iskanjiBesedila* z novim iskalnim besedilom, kar povzroči, da se filtrirani podatki samodejno posodobijo. Signali tukaj omogočajo, da se spremembe v uporabniškem vnosu nemudoma odražajo v podatkih.
- *obdelajFiltriranje* posodobi signal *filtriranjeVrednosti* z izbrano vrednostjo filtra, kar prav tako sproži posodobitev filtriranih podatkov. Signali omogočajo, da se spremembe v filtriranju takoj aplicirajo na podatke.

V tej kodi signali igrajo ključno vlogo pri zagotavljanju reaktivnosti aplikacije. Ko se spremeni stanje iskanja ali filtriranja, signali sprožijo ponovno izračunavanje filtriranih podatkov, kar povzroči, da se tabela avtomatsko posodobi. To omogoča preprosto in učinkovito obvladovanje stanja in posodabljanje uporabniškega vmesnika brez potrebe po kompleksnem ročnem upravljanju stanja.

5 Zaključek

V članku smo predstavili koncept signalov in raziskali njihove številne prednosti pri uporabi v aplikacijah JavaScript. Signali omogočajo učinkovito obvladovanje dogodkov in komunikacijo med komponentami, kar pripomore k boljši organizaciji kode, povečanju modularnosti ter izboljšanju reaktivnosti aplikacij. Sodobna ogrodja, kot so Svelte, Vue.js, Angular in drugi, že vključujejo podporo za signale, kar kaže na njihovo široko sprejetost v razvojni skupnosti. Ta podpora pomeni, da so signali že postali pomemben del sodobnega razvoja spletnih aplikacij in ponujajo številne izboljšave v primerjavi s tradicionalnimi pristopi.

V članku smo predstavili tudi konkretne primere uporabe signalov, kjer se jasno pokaže njihova prednost pred drugimi pristopi, kot so neposredni klici funkcij ali upravljanje stanja preko globalnih spremenljivk. Signali omogočajo bolj strukturiran in jasen način obvladovanja kompleksnih interakcij in asinhronih dogodkov, kar vodi do boljšega razumevanja in vzdrževanja programske kode.

Obetavno je tudi, da bi lahko Signali v prihodnosti postali del standarda JavaScript, kar bi še povečalo njihovo prisotnost in pomembnost v ekosistemu. Če oz. ko se to zgodi, bo njihova uporaba verjetno postala še bolj razširjena, saj bodo razvijalci lahko izkoristili prednosti signalov na standardiziran in združljiv način v različnih okoljih in ogrodjih. Tako lahko pričakujemo, da bodo signali igrali vse pomembnejšo vlogo v prihodnjem razvoju spletnih aplikacij in tehnologij.

Literatura

[1] Contributors M. Promise (JavaScript reference). Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[2] Miško Heveri. Signals vs. Observables, what's all the fuss about? 2023 [cited 2024 July 30]; Available from: https://www.builder.io/blog/signals-vs-observables.

[3] Gupta R. Learn JavaScript Reactivity: How to Build Signals from Scratch. 2024 [cited 2024 July 28]; Available from: https://www.freecodecamp.org/news/learn-javascript-reactivity-build-signals-from-scratch/#:~:text=Signals%20are%20an%20important%20concept,in%20a%20more%20controlled%20manner.

[4] Proposal Signals. [cited 2024 July 27]; Available from: https://github.com/tc39/proposal-signals.

[5] Marian Babić. Unlocking Communication in React: A Journey into the Power of Signals. 2023 [cited 2024 July 28]; Available from: https://medium.com/comsystoreply/unlocking-communication-in-react-a-journey-into-the-power-of-signals-f013d3a1ad7d.

[6] Fotis Adamakis React + Signals = Vue 3. 2023 [cited 2024 July 30]; Available from: https://fadamakis.com/react-signals-vue-3-463fefc51129.

[7] Vue Guide - Reactivity in Depth. [cited 2024 July 29]; Available from: https://vuejs.org/guide/extras/reactivity-in-depth.html.

[8] Angular Signals. Available from: https://angular.dev/guide/signals#writable-signals.

Angular in .NET kot konkurenca namiznim aplikacijam

Matjaž Prtenjak

Endava d.o.o, Ljubljana, Slovenija
matjaz@matjazev.net

V kolikor ste spremljali moje prispevke v prejšnjih letih, boste presenečeni, saj bo ta prispevek zadeve obrnil na glavo. V kolikor pa ste z mano prvič, pa tudi ne skrbite, saj je prispevek povsem zaključena celota in neodvisen od prejšnjih prispevkov. V prispevkih prejšnjih let smo se spraševali, kako (starejše) razvijalce, ki so navajeni namiznih aplikacij, naučiti dela v spletnih aplikacijah, danes pa se sprašujemo, kako spletno aplikacijo napisati tako, da jo lahko damo na namizje. Vprašanje, ki se smiselno pojavlja je: »Zakaj bi to sploh počeli?«. Zakaj bi želeli spletno aplikacijo imeti kot namizno aplikacijo oz. jo gostiti sami? Odgovori so lahko različni, najpomembnejši pa je: zaupanje. Obstajajo uporabniki – tudi (ali predvsem) v poslovnem svetu, ki želijo imeti aplikacije pri sebi. Če želimo spletno aplikacijo prenesti k uporabniku, je zadeva dokaj preprosta. Uporabnik pri sebi postavi spletni strežnik, ki je dosegljiv samo v njegovem poslovnem okolju in aplikacija se prenese na ta strežnik in s tem je zadeva bolj ali manj zaključena. Kaj pa, če ne želimo postavljati spletnega strežnika? Ali bi ne bilo bolje, če lahko uporabnik naloži aplikacijo s spleta, jo požene in slednja teče? Ali pa jo celo kopira v mapo, požene in zadeva deluje?

Ključne besede:
.NET
Angular
Spletne aplikacije
Namizne aplikacije
Hibridne aplikacije

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.19

1 Uvod

V tem prispevku bomo razvili spletno/namizno aplikacijo z uporabo .NET 8.0 in Angular okolja. Seveda programska koda ne bo v članku, temveč jo lahko snamete s spleta, z GitHub-a, na naslovu: https://github.com/MPrtenjak/OTS2024-NET-Angular-WEB-And-Desktop.

1.1. Namen aplikacije

Razviti želimo aplikacijo, ki bo hranila tri posebnosti, za katere so uporabniki hvaležni v določenem dnevu. Dandanes je v svetu namreč vse več depresije in anksioznosti, zato nam lahko lepe stvar izboljšajo počutje. Če nam je namreč hudo, se lahko na hitro spomnimo, zakaj smo bili hvaležni včeraj, prejšnji mesec ali pa mogoče leto dni nazaj.

1.2. Zahteve

- Ker gre za osebna čustva, se morajo uporabniki v aplikacijo najprej prijaviti. Prijava je klasična, z imenom in geslom.
- Znotraj aplikacije lahko uporabniki brskajo po zgodovini in vidijo, česa so se veselili v preteklosti oz. čemu so bili hvaležni.
- Za vsak dan lahko vnesejo do tri posebnosti. *Tri je povsem dovolj, saj sicer posebnosti niso več posebnosti in gre za preprosto naštevanje.*
- Posebnosti lahko uporabnik vnaša za tekoči datum ali pa do največ tri dni v preteklosti; skupaj torej za največ štiri dni. *Tudi tukaj se strogo omejimo, saj je vnašanje za več kot tri dni v preteklosti že podleženo pozabljanju.*
- Aplikacija mora teči kot spletna aplikacija v npr. Azure, spletna aplikacija v Windows in Linux okolju ali pa kot navadna aplikacija znotraj Windows ali Linux operacijskega sistema.

1.3. O razvojnih okoljih

Opisane zahteve lahko rešimo na zelo različne načine in z zelo različnimi orodji.

Ena izmed možnosti je, da razvijemo spletno aplikacijo v poljubnem okolju, jo »zapečemo« v Docker kontejner in ponudimo kot »kontejnersko« aplikacijo. Ta zagotovo izpolnjuje zahteve, da teče »kjerkoli«. Toda v tem primeru mora uporabnik imeti Docker in podobno. Poskusimo biti torej čimbolj preprosti in prijazni do uporabnika.

Če torej ne razmišljamo o kontejnerskih aplikacijah, smo lahko še vedno agnostnični pri izbiti razvojnega okolja. Lahko razvijemo vse lepo v Python programskem jeziku in v ukazni vrstici (lahko sicer uporabimo tudi kakšno orodje, kot je PySimpleGUI za izdelavo grafičnega vmesnika). Lahko vse razvijemo v JavaScript in Node okolju. Lahko…

Povedati želim, da je glede na opisane zahteve rešitev mnogo, saj imamo različne programske jezike. Zato je smiselno pač pogledati, kaj znamo, v čem smo domači in potem aplikacijo razvijemo v tem okolju. Meni je blizu okolje .NET, zato bom izbral slednjega.

Vse lahko razvijemo v .NET okolju, saj imamo za grafični del aplikacije v .NET okolju res veliko izbiro. Ker želimo, da je aplikacija spletna, bi recimo izbrali Blazor, ki sem ga predstavljal v prispevkih prejšnja leta [3, 4].

Vendar spet poglejmo malo naokoli in ugotovili bomo, da je današnjim mladim programerjem/programerkam še najbližji spletni razvoj z uporabo različnih JavaScript razvojnih okolij, kot so React, Angular, Vue in podobni. Če sem pred leti torej predstavil Vue [5], pa dajmo danes zadevo razviti v Angular okolju.

1.4. Dodatne omejitve

Razvojno okolje smo torej izbrali. Imeli bomo kombinacijo .NET in Angular. Da pa bi programska koda ne bila prevelika in bi jo tisti, ki vas to zanima, lahko relativno hitro pogledali in preverili, se omejimo še malce bolj.

Našo aplikacijo bomo torej gostili na Azure in v tem primeru bomo govorili o spletni aplikaciji. Lahko pa bomo našo aplikacijo pognali v okolju Windows ali Linux in v tem primeru bomo govorili o namizni aplikaciji in/ali spletni aplikaciji.

- Razvili bomo aplikacijo, ki bo lahko tekla kot »navaden program« v okolju Windows ali kot storitev (service) v okolju Windows. Spletnega strežnika IIS se v Oknih ne bomo dotikali, ker uporabniki (navadni uporabniki) pač nimajo nameščenega IIS na računalnikih. Kadar bo aplikacija tekla kot »navaden program«, bo namizna aplikacija, kadar bo tekla kot service, bo spletna aplikacija.
- Lahko bo tekla v okolju Linux kot navaden program, ki ga požene konkretni uporabnik ali kot storitev (deamon), ki jo uporabljajo drugi uporabniki. Torej spet kot namizna ali spletna aplikacija.
- Lahko pa bo tekla kot spletna aplikacija v Azure.

In zdaj še dve posebni, a logični zahtevi:

- Programska koda bo enotna in poskušali jo bomo napisati tako, da bomo lahko program samo kopirali v neko mapo, pa bo že deloval.
- Če uporabnik/uporabnica poganja aplikacijo na svojem računalniku, potem ne potrebuje prijave, saj je že prijavljena v sistem. Tukaj imam v mislih opcije, ko program teče v Windows okolju (kot program ali kot storitev) ter kadar teče v Linux okolju kot program.

2 Razvoj

Kot že večkrat omenjeno, bomo aplikacijo razvili v dveh okoljih, ki bosta povezani z REST API komunikacijo. Razvoj lahko torej poteka vzporedno, dogovoriti se moramo samo za aplikacijski vmesnik (REST API), preko katerega bosta oba dela naše aplikacije komunicirala.

2.1. API

Pri aplikacijskem vmesniku se bomo spet, da bo aplikacija majhna in obvladljiva, držali minimalnih zahtev. Uporabili bomo torej sledeče REST zahteve:

- **POST**: `/users/sign-up`, ki služi za registracijo uporabnika
  - Vhod: `{ "userName", "password1", "password2" }`
  - Izhod: `{ "userId", "userName", "token" }`
  - Na vhodu torej pričakujemo uporabniško ime in dve gesli, ki se morata ujemati. Na izhodu pa vrnemo id in ime novega uporabnika v bazi ter JWT žeton.
- **POST**: `/users/try-login`, ki služi za prijavo uporabnika brez prijavnih podatkov

  - Vhod: nima
  - Izhod: `{ "userId", "userName", "token" }`
  - Ta funkcija služi specifičnemu namenu v naši aplikaciji. Zahteva je namreč, da se uporabnik v primeru namizne aplikacije ne prijavlja in uporabili bomo ta klic, da bomo poskušali uporabnika prijaviti brez prijavnih podatkov. Več o tem kasneje v prispevku.
- **POST**: `/users/login`, ki služi za prijavo uporabnika z uporabniškim imenom in geslom
  - Vhod: `{ "userName", "password" }`
  - Izhod: `{ "userId", "userName", "token" }`
- **GET**: `/admin/info`
  - Vrne JSON objekt z različnimi informacijami o trenutnem sistemu
- **POST**: `/admin/randomize-data`, ki bazo zapiše naključne hvaležnosti trenutnega uporabnika za zadnje tri mesece (*ta funkcija je tukaj samo za lažji prikaz delovanja aplikacije*).
  - VHOD: JWT žeton kot 'Authorization: Bearer'
- **POST**: `/gratitude`, ki v bazo vpiše čemu je uporabnik/uporabnica hvaležna v določenem dnevu
  - VHOD: `{ "date", "content": [] }`
  - VHOD: JWT žeton kot 'Authorization: Bearer'
- **GET**: `/gratitude/yyyy-mm`, ki vrne, čemu je bil/bila uporabnik/uporabnica hvaležna v izbranem mesecu
  - VHOD: JWT žeton kot 'Authorization: Bearer'

Tako, to je vse! To je sedem funkcij, s katerimi bomo komunicirali med strežnikom in uporabniškim vmesnikom.

2.2. .NET strežniški del

Pri strežniškem delu se bomo najdlje zadržali, saj bo uporabniški vmesnik, spisan v Angular, povsem klasična Angular aplikacija. V Angular aplikaciji ne bo popolnoma nobenih posebnosti in njen strežnik bo vedno naša .NET aplikacija, ki jo bo tudi gostila. Torej uporabniški vmesnik bo povsem enak in neodvisen od tega, ali aplikacija teče kot spletna, ali kot namizna aplikacija. **Ravno to je pravzaprav bistvo tega prispevka!**

> 😎 *V tem delu bomo torej govorili o .NET okolju in za razumevanje slednjega bo potrebno osnovno poznavanje .NET CORE okolja.*

2.2.1. Izvajanje programa v različnih operacijskih sistemih

Ta del nam reši že .NET okolje samo in zanj nam ni potrebno skrbeti. Ko prevedemo .NET program, dobimo na izhodu že podmapo (`/runtimes`) s programskimi knjižnicami za različne operacijske sisteme.

Naša aplikacija, naš strežniški del, bo torej brez naše intervencije tekel v Windows, Linux in Azure okolju. Pravzaprav zna teči tudi še v nekaj drugih operacijskih sistemih, vendar to zaenkrat pustimo.

Ker želimo v .NET razviti strežniški del aplikacije, bomo za osnovo našega projekta izbrali `ASP.NET Core Web API`, ki ustvari natanko to, kar potrebujemo; REST strežnik.

Program se bo torej izvajal v različnih operacijskih sistemih, vendar bomo mi še vedno morali vedeti, kje konkretno se izvaja, da bomo lahko temu primerno – tam, kjer je potrebno, kodo zapisali drugače.

Za potrebe detekcije izvajalnega okolja si torej pripravimo naštevni tip `SupportedEnvironments`, v katerem bomo imeli našteta vsa okolja, ki jih podpiramo… Pravzaprav ne, obstaja boljša rešitev!

> 😎 *Namesto, da imamo po kodi vejitve in se sprašujemo, če je to okolje, potem naredi to, sicer to, je bolje, da naredimo vmesnik `ISuportedEnvironment` z vsemi razlikami, ki jih vsebuje določeno okolje. Na začetku aplikacije ugotovimo izvajalno okolje ter ustvarimo konkreten razred za to okolje, nadalje pa uporabljamo `ISuportedEnvironment` vmesnik. S tem se znebimo vseh vejitev in tudi na enem mestu imamo pregledno zapisane vse razlike med izvajalnimi okolji.*

Torej pripravimo raje vmesnik, ki bo združeval elemente, ki jih mora definirati posamezno izvajalno okolje.

> 😎 *V novejših različicah .NET okolja lahko imajo vmesniki že predefinirane elemente in zato lahko tukaj izberemo vrednosti, ki veljajo v največ primerih, razlike pa bomo definirali v posameznih razredih.*

```
public interface ISupportedEnvironment
{
  string Name { get; }
  bool RequireLogin { get => true; }
  bool OpenBrowser { get => false; }
  void ApplyDeamon(ConfigureHostBuilder configureHostBuilder) { }
}
```

Kot lahko vidite, je razlik v kodi, ki bi bile posledica različnih operacijskih sistemov, zelo malo. Za našo aplikacijo potrebujemo samo tri posebnosti.

Prvi dve sta evidentni že zaradi naših zahtev. V zahtevah piše, da namizne aplikacije ne zahtevajo prijave, spletne pa; to nam torej pove lastnost `RequireLogin`. Ko program teče v Windows okolju kot navaden program, pa je še dodatna zahteva, da se brskalnik avtomatično odpre, za to poskrbi `OpenBrowser`.

Preostali element (`ApplyDeamon`) pa ni takoj evidenten in potrebovali ga bomo kasneje, ko bomo inicializirali aplikacijo.

> 😎 *V kolikor kasneje v razvoju aplikacije naletite na razlike med okolji, jih morate reševati tako, da v tem vmesniku definirate lastnost, na katero se sklicujete, ali pa metodo, ki jo pokličete, da naredi tisto, kar mora na posameznem okolju.*

Kot primer si poglejmo še, kako je definirano okolje, kjer naša aplikacije teče v Linux OS kot storitev (*service, deamon*):

```
public class LinuxAsServiceEnvironment : ISupportedEnvironment
{
  public string Name => nameof(LinuxAsServiceEnvironment);
  public void ApplyDeamon(ConfigureHostBuilder configureHostBuilder)
    => configureHostBuilder.UseSystemd();
}
```

Večina elementov je zajeta že v vmesniku, spremenimo samo ime in aplikaciji povemo, da mora uporabljati `systemd`, ki v Linux sistemih služi za upravljanje s storitvami.

2.2.2. Inicializacija aplikacije

Ok, večino posebnosti v aplikaciji smo že rešili, zdaj se bolj ali manj lahko posvetimo sami aplikaciji kot takšni.

Ko ustvarimo prazno .NET WEB App aplikacijo, se kot prva vrstica našega programa izvede sledeče:

```
var builder = WebApplication.CreateBuilder(args);
```

To vrstico moramo popraviti, saj bo naša aplikacija lahko tekla tudi kot storitev in v tistih primerih teče kot sistemski uporabnik, predvsem pa je pomembno, da je njena delovna mapa nek sistemski imenik znotraj

Windows/Linus OS. Mi pa moramo aplikaciji dopovedati, da naj uporablja imenik z aplikacijo kot spletni strežnik, saj bomo znotraj naše aplikacije gostili Angular spletno aplikacijo.

> 😎 *Paziti je potrebno, kaj naša aplikacija vidi kot lastno mapo, oz. mapo s podatki.*

```
var options = new WebApplicationOptions {
  Args = args,
  ContentRootPath = AppContext.BaseDirectory
};

WebApplicationBuilder builder = WebApplication.CreateBuilder(options);
```

2.2.3. Detekcija izvajalnega okolja

Dobro, napisali smo razrede, ki poskrbijo za razlike med izvajalnimi okolji, še vedno pa moramo nekako določiti izvajalno okolje. Napisati moramo torej kodo, ki bo nedvoumno ugotovila, na kakšnem sistemu teče naša aplikacija. Na podlagi tega testa bomo nato ustvarili spremenljivko ustreznega, prej razvitega, razreda:

```
public static ISupportedEnvironment Detect()
{
  if (IsRunningOnAzureWebApp())
    return new AzureEnvironment();

  if (OperatingSystem.IsWindows())
    return WindowsServiceHelpers.IsWindowsService()
      ? new WindowsAsServiceEnvironment()
      : new WindowsAsProgramEnvironment();

  if (OperatingSystem.IsLinux())
    return IsRunningAsLinuxDaemon()
      ? new LinuxAsServiceEnvironment()
      : new LinuxAsProgramEnvironment();

  throw new NotSupportedException("The current environment is not supported.");
}
```

Tole izgleda lepo in pravilno, vendar nam manjkata metodi `IsRunningOnAzureWebApp` in `IsRunningAsLinuxDaemon`. Kot vidite, lahko ostalo določimo že s pomočjo vgrajenih metod .NET okolja. No, resnici na ljubo lahko mogoče ugotovimo tudi še kaj več, vendar trenutno meni to ni uspelo!

Ker torej nimamo vgrajene metode za detekcijo Azure sistema ali za detekcijo, na kakšen način teče naša aplikacija znotraj Linux okolja, si lahko pomagamo s spremenljivkami okolja.

> 😎 *Razvijalci preradi pozabljamo, da nam operacijski sistemi nudijo spremenljivke okolja in da so določene postavljene že s strani OS ali s strani posameznih programov ali pa jih lahko nastavijo uporabniki ter s tem vplivajo na naš program.*

Za detekcijo Azure okolja bomo preverili obstoj dveh spremenljivk, ki jih postavi Azure okolje in za katere predvidevamo, da se drugje ne bodo pojavljale (`WEBSITE_INSTANCE_ID` in `WEBSITE_HOSTNAME`).

Za detekcijo demon-a znotraj Linux okolja pa bomo poskrbeli sami, tako da bomo nastavili spremenljivko (`DOTNET_RUNNING_IN_SERVICE`) v trenutku, ko bomo našo aplikacijo registrirali kot storitev znotraj Linux okolja.

2.2.4. Detekcija uporabnika

Naslednja specifika je zahteva, da se uporabnik na lastnem računalniku ne prijavlja in v tem primeru pač uporabimo uporabniško ime trenutnega uporabnika.

Zaplete pa se tedaj, ko naša aplikacija teče kot storitev v Windows okolju.

V tem primeru imamo pri naši aplikaciji tudi neko specifiko. Seveda bi lahko v takšnem primeru našo aplikacijo klical/uporabil tudi kdo drug v nekem lokalnem omrežju povezanih Windows računalnikov in podobno. Toda

ustavimo konje in si pač razčistimo da je namen naše aplikacije, ki teče kot storitev v Windows okolju pač primer, ko isti računalnik uporabljajo različne osebe (recimo domači računalnik), vendar ne hkrati!

V takšnem primeru je torej potrebno določiti, kdo v resnici sedi za računalnikom oz. kdo se je prijavil, kdo je tisti, ki bo pognal brskalnik.

Slednje lahko ugotovimo s pomočjo vpogleda v sistem ter iskanja, kateri uporabnik poganja program `explorer.exe`.

> 😎 *Še enkrat, to ni »bulletproof« koda, vendar za naše potrebe in tudi potrebe marsikaterega drugega programa je povsem dovolj!*

Konkretno programsko kodo si lahko ogledate v GitHub-u v razredu `UserIdentifierServiceOnWindowsService`.

2.2.5. CORS

Zagotovo smo že velikokrat slišali o CORS, pa si zdaj malce podrobneje poglejmo, o čem je govora in kako to vpliva na nas.

CORS (Cross-Origin Resource Sharing) je varnostna funkcija v spletnem razvoju, ki spletni strani omogoča, da zahteva vire iz domene, ki ni tista, iz katere izvira. Če je naša spletna stran na moje-spletisce.si, CORS nadzoruje, ali lahko zahteva pride iz aplikacije na api.nekdo-drug.si. Spletni brskalniki zaradi varnostnih razlogov privzeto blokirajo te zahteve navzkrižnega izvora, vendar CORS strežniku omogoča, da določi, katerim domenam je dovoljen dostop do njegovih virov.

Ker bomo v času našega razvoja razvijali strežniški del aplikacije ločeno od uporabniškega vmesnika in bomo testirali strežniški del, ki bo tekel na npr. `localhost:1770` in uporabniški vmesnik, ki bo tekel na `localhost:1400`, moramo uporabiti CORS, da lahko uporabniški vmesnik kliče API zahteve strežniškega dela, ki teče na drugih vratih.

Kako izgleda vključitev CORS, si lahko ogledate na GitHub-u v razredu `program.cs`.

> 😎 *V fazi razvoja je CORS super in ga je dobro vklopiti, saj je tako lažje testirati. V produkciji pa ga je dobro izklopiti oz. ga omejiti.*

2.2.6. Usmerjanje (routing)

Naš uporabniški vmesnik bo tekel kot *enostranska aplikacija* (SPA – Single Page App) in kot vemo, slednje vse zahteve vodijo preko glavnega obrazca, torej preko `index.html`.

Če torej Angular neke podstrani ne bo našel, bo poskusil preko `index.html`. Ker bomo Angular aplikacijo gostili mi znotraj našega strežniškega dela, ji moramo zagotoviti to mehko pristajanje na `index.html` strani:

```
app.MapFallbackToFile("index.html");
```

> 😎 *Če vaša aplikacija gosti enostransko spletno aplikacijo (SPA), ne pozabite na obravnavo index.html datoteke.*

2.2.7. Preostali del strežniške aplikacije

V tem trenutku se lahko skupaj nasmehnemo, saj je za »preostali del aplikacije« namenjeno eno samo poglavje.

Seveda je jasno, da smo do sedaj pogledali samo pet odstotkov potrebne kode našega strežniškega dela, vendar pa ta prispevek ni namenjen prikazu razvoja REST API strežnika v .NET!

> 😎 *Celotna preostala aplikacija je povsem klasična .NET aplikacija in nima nobenih posebnosti, pač prevzem zahtev, komunikacija s podatkovno bazo in vrnitev rezultatov.*

Kot podatkovno bazo uporabljam kar `SQLite`, za lažji dostop pa mikro ORM, `Dapper`. Seveda lahko aplikacijo nadgradite z uporabo kakšnega `MSSql` strežnika, uporabo `Entity Framework` in podobnim, vendar kot rečeno, moja želja je bila, da je aplikacija majhna, obvladljiva, vseeno pa popolna.

> 😎 *V kolikor boste torej gledali programsko kodo, boste v njej videli tudi recimo lokalizacijo ali pa delo z JWT žetoni.*

2.3. Angular uporabniški vmesnik

Pri strežniškem delu sem predstavil posebnosti, ki nam bodo omogočile, da bo naša aplikacija brez sprememb ter samo s »kopiraj in prilepi« akcijo tekla v različnih okoljih.

S tem je imel strežniški del nekaj podpoglavij, pri uporabniškem vmesniku pa bo besedila še mnogo manj, saj tu ni absolutno nobenih posebnosti.

> 😎 *Uporabniški del v Angularju nima nobene zveze s sistemom v katerem teče. Vse, kar njega zanima je API, ki mu ga strežnik nudi in seveda, kje se strežnik nahaja; naslov strežnika torej. Ko se bo Angular povezal s strežnikom, bo tekel kot vsaka Angular/Vue/React/... aplikacija.*

Resnično gre za povsem navadno SPA aplikacijo, ki bi lahko bila razvita v kateremkoli JavaScript ogrodju in Angular sem izbral samo zato, da ne govorim vedno o VUE, ki sem ga do sedaj predstavljal [5].

2.3.1. Kje je moj strežnik?

Pri razvoju Angular aplikacije ni nobenih posebnosti, bi pa predstavil nekaj, kar lahko morda komu olajša razvoj.

Kot že omenjeno ob razlagi CORS, imamo v razvoju situacijo, ko nam strežniški del teče na npr. `localhost:1770`, naša Angular aplikacija pa teče na klasičnih Angular vratih `localhost:4200`. Kako torej dopovedati Angular-ju, naj tvori naslove, kot so `https://localhost:1770/user/login`, če pa sam teče na vratih `4200`?

Uporabili bomo neko JSON nastavitveno datoteko in notri napisali naslov strežnika in potem kar uporabljali ta naslov. Ok, do sem vse lepo in prav. Kaj pa, ko bo aplikacija tekla na produkciji?

Takrat pa mora naša aplikacija teči na istem naslovu! Poznamo torej naslov strežnika, saj je to isti naslov, na katerem teče Anguler sam. V tem primeru je torej nesmiselno še enkrat pisati ta isti naslov v namestitveno datoteko.

> 😎 *Lahko se znajdemo in rečemo: v kolikor namestitvena datoteka obstaja, potem uporabi podatek iz namestitvene datoteke, sicer uporabi lasten naslov.*

Torej, prebrati poskusimo nastavitve iz `/assets/config.json` in če nam to ne uspe, privzamemo naš naslov kot naslov strežnika:

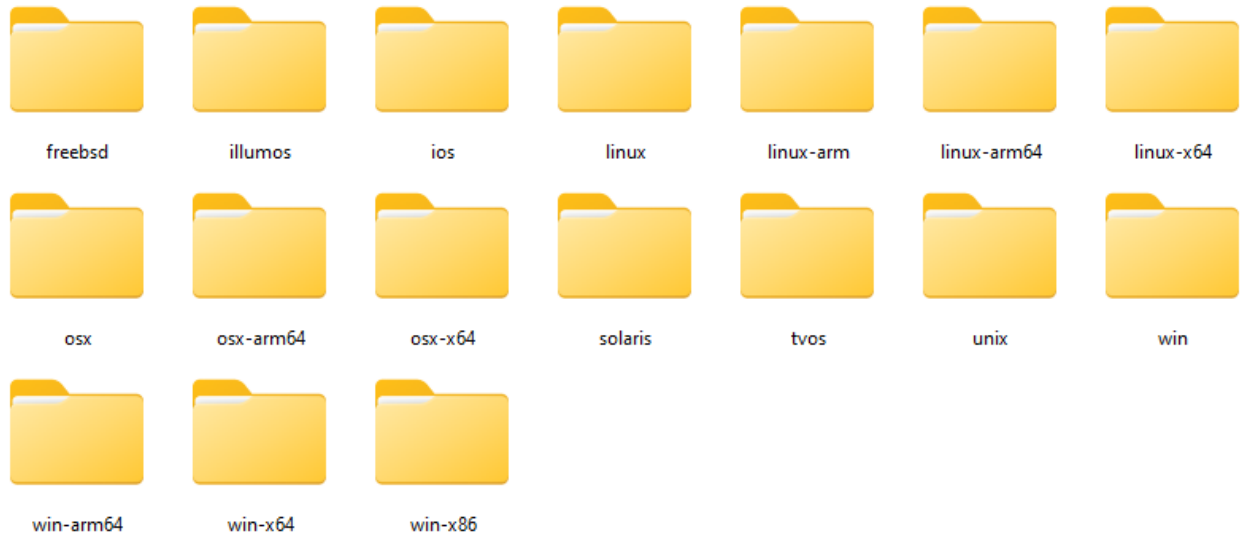
```
loadConfig(): Observable<any> {
  console.log('Loading configuration');
  return this.http.get('/assets/config.json').pipe(
    catchError(error => {
      return new Observable(observer => {
        observer.next({ apiUrl: window.location.origin });
        observer.complete();
      });
    })
  );
}
```

Ob izgradnji programa za produkcijo preprosto pobrišemo to namestitveno datoteko in vse deluje tako, kot mora.

2.4. Dajmo vse skupaj!

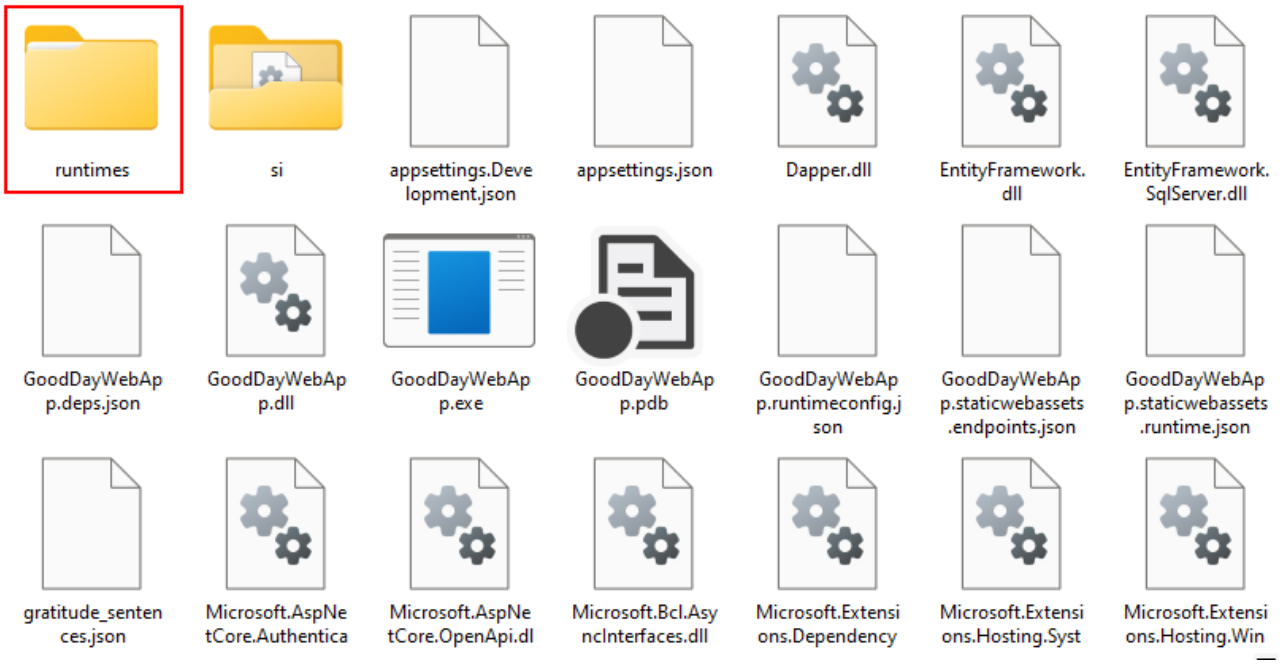
Ena izmed bolj znanih programerskih izjav »Na mojem računalniku deluje!« je posledica kompleksnosti izvajalnih okolij in izgradnje modernih programskih rešitev.

Vendar v našem primeru na srečo temu ni tako. Ko zgradimo naš strežniški del v .NET okolju, dobimo mapo, v kateri se nahaja naš program, predvsem pa se v tej mapi nahajajo tudi potrebne knjižnice (v podmapi `/runtimes`), da zadeve teče tudi na ostalih (podprtih) operacijskih sistemih



Slika 1: Podprti operacijski sistemi, v katerih se lahko izvaja .NET 8.0 aplikacija.

Če torej celotno izhodno mapo, v kateri je tudi podmapa `/runtimes`, kopiramo na ciljni računalnik, mora zadeva delovati.



Slika 2: Izhodna mapa našega strežniškega programa s podmapo /runtimes.

2.4.1. Dobro, s tem smo rešili strežniški del, kaj pa naj naredimo z Angular aplikacijo?

Najprej moramo Angular aplikacijo zgraditi. To naredimo s sledečim ukazom (ali kakšno njegovo izpeljanko):

```
ng build
```

V kolikor nimamo napak v kodi, se bo naša SPA aplikacija zgradila v podmapi `/dist`. Toda na žalost to še ni podmapa, ki nas zanima, saj je spodaj še nekaj podmap `/good-day-spa/browser`. Skupaj torej `dist/good-day-spa/browser`. Ime vmesne mape (`good-day-spa`) je seveda odvisno on nastavitev.

No, nas pa tako ali tako zanima samo tisto, kar je v podmapi `/browser` in to moramo kopirati v podmapo wwwroot našega strežniškega dela. To je vse 🙌.

> 😎 *Da bi naša aplikacija delovala kot celota, je potrebno prevedene Angular datoteke kopirati v podmapo /wwwroot naše strežniške aplikacije.*

Tole je izsek iz skripte, ki zgradi našo aplikacijo v celoti:

```
@REM Zgradimo .NET aplikacijo
@dotnet publish GoodDayWebApp.csproj  -c Release -f net8.0

@REM Rezultat našega prevoda bo v tejle podmapi
@SET BACKEND_SOURCE=GoodDayWebApp\bin\Release\net8.0\publish

@REM Zgradimo Angular SPA aplikacijo
@cmd /c ng build --configuration production

@REM Angular aplikacija bo v tej podmapi
@SET FRONTEND_SOURCE=GoodDaySPA\dist\good-day-spa

@REM Kopiramo Angular aplikacijo v wwwroot podmapo naše strežniške aplikacije
@xcopy %FRONTEND_SOURCE%\browser %BACKEND_SOURCE%\wwwroot /s /e /y
```

3 V produkciji

Pod predpostavko, da smo aplikacijo zgradili tako, kot je opisano v prejšnjem odstavku, torej da smo Angular preveden program prepisali v podmapo `wwwroot` naše .NET aplikacije, si poglejmo, kako se zadeva obnaša na posameznih okoljih.

3.1. Izvajanje aplikacije na drugem računalniku

Spet smo pred famozno razvijalsko izjavo: »Na mojem računalniku deluje!«. Kaj je torej potrebno narediti, da bo naša aplikacija tekla tudi na drugih računalnikih?

Dandanes to ni nobena posebnost več, pa vendar dajmo to zapisati še enkrat. Mnoga razvojna okolja, in med njimi je seveda tudi .NET, poznajo dve vrsti instalacije:

- **Razvojna instalacija** (SDK Install == Software Development Kit Instalation) je večja instalacija in s to instalacijo lahko na računalniku razvijamo programsko opremo in posledično jo lahko seveda tudi poganjamo.
- **Izvajalna instalacija** (Runtime instalation) pa uporabniku omogoča samo izvajanje že razvitih aplikacij in je posledično seveda manjša oz. zasede manj prostora na disku.

😎 *Da bi torej naš program tekel na kateremkoli računalniku, tam torej potrebuje izvajalno okolje, zato si mora uporabnik najprej instalirati izvajalno okolje.*

To je danes res preprosto in zahteva samo skok na splet in zagon instalacijske datoteke na poljubnem operacijskem sistemu

3.1.1. Priprava aplikacije 'Vse V Enem'

Lahko pa naš program prevedemo tudi tako, da v celoti vsebuje vso potrebno izvajalno okolje že v sebi. S tem dobimo nekaj pozitivnih in nekaj negativnih lastnosti:

- (+) Uporabnik ne potrebuje nobene posebne instalacije. Naloži naš program in vse deluje.
- (-) Ker naš program vsebuje tudi vso potrebno izvajalno okolje, je precej večji in zahteva več prostora.
- (-) Če ima uporabnik več programov in vsak prinese zraven izvajalno okolje, potem ima uporabnik na računalniku mnogo kopij izvajalnih okolij.

Odločitev pa je vedno na vas, razvijalcih, po kateri poti boste šli.

Naši aplikaciji ne bomo prilagali izvajalnega okolja in bomo predvidevali, da slednje na računalniku obstaja.

3.2. V okolju Windows kot program

Izvedemo EXE program brez kakršnihkoli sprememb in

- Program se zažene v ukazni vrstici
- Ker ga poganjamo v okolju Windows, se avtomatično zažene brskalnik in takoj smo v aplikaciji, ker prijava ni potrebna

3.3. V okolju Windows kot storitev

Naša aplikacije je v celoti pripravljena tako, da lahko teče kot storitev, vendar pa jo moramo kot storitev seveda najprej prijaviti.

V GitHub repozitoriju je pripravljenih nekaj kratkih skript, s pomočjo katerih lahko aplikacijo registriramo kot storitev, jo zaženemo, ustavimo, odstranimo… Ni pa to nič kompleksnega in tukaj prilagam primer skripte, ki aplikacijo registrira kot storitev, ter jo tudi požene:

```
# registracija

New-Service -Name GoodDayService `
            -BinaryPathName "GoodDayWebApp.exe --contentRoot GoodDayWebApp.exe" `
            -Description "Aplikacija dobrega počutja" `
            -DisplayName "GoodDayService"

# zagon

Start-Service -Name GoodDayService
```

S pomočjo takšne `PowerShell` skripte lahko našo aplikacijo, brez sprememb(!), poganjamo kot storitev v okolju Windows.

Ker sedaj aplikacija teče kot storitev, je vedno dosegljiva, zažene se ob zagonu oken in vsak uporabnik lahko odpre brskalnik, zažene ustrezen spletni naslov `http://localhost:5000`[24] in že lahko uporablja aplikacijo brez prijave.

Ko se bo na isti Windows računalnik prijavil drug uporabnik, bo aplikacijo brez prijave uporabljal z lastnim uporabniškim imenom in obiskom spletne strani `http://localhost:5000`.

3.4. V okolju Linux kot program

Podobno kot v okolju Windows, tudi tukaj program preprosto poženemo. Seveda pa Linux ne uporablja EXE datotek, zato moramo program zagnati kot `dotnet` program, torej:

```
/usr/bin/dotnet GoodDayWebApp.dll
```

In to je vse. Uporabnik lahko aplikacijo uporablja in ne potrebuje prijave, saj se prijava privzame iz uporabniškega računa. Mora pa ročno zagnati brskalnik ter odpreti spletno stran.

3.5. V okolju Linux kot storitev (deamon)

Tukaj je zadeva enakovredna Windows okolju, saj naš program ne potrebuje nobenih sprememb. Le zagnati ga moramo kot Linux deamon program.

Tudi tukaj si je najlažje pomagati s skriptami in tudi tukaj so skripte zelo enostavne, zato prilagam skripto za registracijo in zagon storitve v `BASH` izvajalnem okolju (kot vedno, so vse skripte v GitHub repozitoriju).

Za razliko od Windows okolja moramo v primeru Linux okolja najprej pripraviti datoteko, ki jo bo Linux razumel kot definicijo storitve »deamon«-a. Ustvarimo torej datoteko GoodDayWebApp.service, ki jo bomo potem morali skopirati v ustrezno mapo in Linux bo to razumel kot navodilo za izvajanje storitve:

```
[Unit]
Description=GoodDayService

[Service]
WorkingDirectory=/mnt/c/linux-dotnet
ExecStart=/usr/bin/dotnet GoodDayWebApp.dll
Restart=always
RestartSec=10
KillSignal=SIGINT
SyslogIdentifier=GoodDayWebApp
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=DOTNET_RUNNING_IN_SERVICE=1

[Install]
WantedBy=multi-user.target
```

Tukaj ne bom opisoval pomena in namena posameznih elementov te datoteke. Je pa bolj ali manj že iz angleških nazivov jasno, čemu je kateri del namenjen.

> 😎 *Opozoril bi samo na majhen detajl in sicer na vrstico 'Environment = DOTNET_RUNNING_IN_SERVICE=1'. S to vrstico postavimo spremenljivko okolja, ki je za nas zelo pomembna, saj z njeno pomočjo ugotovimo, da naša aplikacija teče kot Linux storitev!*

Dobro, sedaj smo Linux storitev definirali in moramo jo še registrirati ter zagnati:

```
#!/bin/bash
cp GoodDayWebApp.service /etc/systemd/system
```

[24] Vrata 5000 so privzeta in jih lahko z nastavitvami seveda spremenimo!

```
systemctl daemon-reload
systemctl start GoodDayWebApp
```

Definicijo storitve najprej kopiramo v ustrezno mapo, nato ponovno zaženemo upravitelja storitev, da slednji zazna našo novo storitev in na koncu aplikacijo (storitev, deamon) še zaženemo.

To je to, s tem bodo našo aplikacijo lahko uporabljali VSI, ki imajo dostop do tega Linux računalnika. Morali pa bodo ročno zagnati brskalnik ter se v aplikacijo prijaviti, saj teče kot večuporabniška aplikacija.

3.6. V okolju Azure

In seveda v letu 2024 ne moremo mimo tega, da bi ne omenili računalništva v oblaku, v našem primeru torej Azure.

> 😎 *Čeravno smo torej v letu 2024, mi je uspelo spisati celoten prispevek brez ene same omembe kratice, ki je v angleščini sestavljena iz dveh črk oziroma natančneje dveh samoglasnikov, ki ju lahko slišite, če poslušate osla… Pa s tem seveda ne mislim, da je vso to noro navdušenje nad to tehnologijo nesmiselno ali za osle; nikakor ne, včasih pa je vseeno dobro, če se ji izognemo in je ne omenjamo vedno v povezavi z računalništvom.*

Da bi nam karkoli delovalo v Azure, moramo najprej malce razvezati denarnico in dobiti dostop do teh oblakov.

Ko dostop imamo, pa je zadeva dokaj enostavna in razumljiva. Narediti moramo novo 'Web App' storitev, izbrati .NET 8.0, izvajalno okolje. Operacijski sistem pa je nepomemben, jaz sem izbral Linux in ne Windows, ker sem ga pač!

Ko v Azure definiramo novo 'Web App' in jo Azure inicializira, lahko do nje dostopimo preko FTP oz. preko SFTP (secure FTP).

Tam nas že čaka mapa wwwroot in vse, kar je potrebno storiti ,je našo celotno aplikacijo skopirati v mapo wwwroot naše Azure Web App storitve. Na Azure plošči pogledamo še naslov naše storitve in takoj jo lahko začnemo uporabljati, tako da preprosto odpremo ta naslov v poljubnem brskalniku kjerkoli na svetu!

> 😎 *Da bi naša aplikacija delovala v Azure, jo moramo samo v celoti kopirati v wwwroot podmapo!*
>
> 😎 *Da, res je, se tem imamo dve wwwroot mapi. Prva (glavna, višja) je Azurjeva in preko nje Azure izvaja našo aplikacijo! Naša aplikacija pa ima spodaj še eno wwwroot mapo in preko nje naša aplikacija izvaja Angular aplikacijo. Vau* 😱*!*

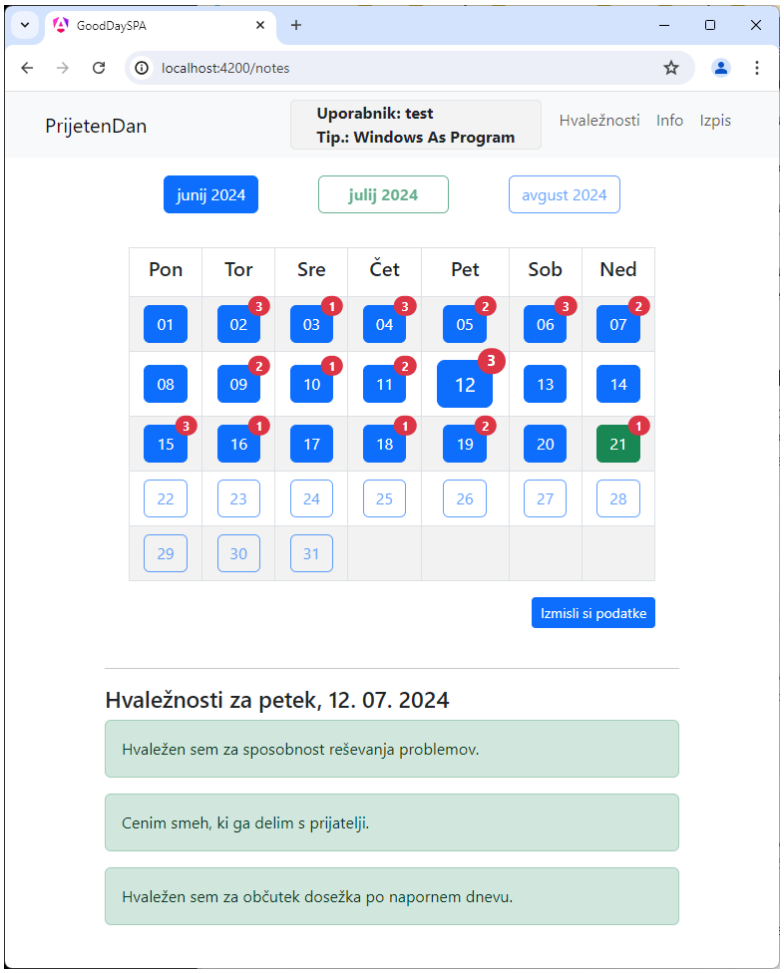
3.7. Slabosti

Ker je naša želja, da se aplikacija samo kopira drugam in tam kar »magično« deluje (*»Magično« bi delovala recimo tudi na Mac OS, vendar slednjega nimam in nisem poiskušal*), to pomeni, da zraven v `/runtimes` mapi vlečemo tudi izvajalne module za vse mogoče operacijske sisteme. V resnici je smiselno pobrisati zadeve, ki jih na določenem okolju ne potrebujemo.

Seveda ima sama aplikacija varnostne slabosti, slab uporabniški vmesnik in podobno. A izdelava popolne aplikacije nikoli niti ni bil namen tega prispevka!

4 Uporabniški vmesnik

Temu bi se res želel izogniti, kajti uporabniški vmesnik je pač nekaj, za kar razvijalec mora imeti žilico in čas. Jaz nimam ne enega ne drugega, vendar pa je vseeno dobro, da v prispevku vsaj na sliki prikažem, o kakšni aplikaciji sploh govorimo oz. kako zadeva izgleda.



Slika 3: Glavni ekran aplikacije.

Na sliki lahko vidite, da trenutno aplikacija teče kot program v okolju Windows in da jo uporablja uporabnik 'test'.

5 Zaključek

Namen prispevka je prikazati:

- Da je vsaka vaša .NET Web Api aplikacija pravzaprav spletni strežnik,
- Ker je spletni strežnik, lahko znotraj nje gostite spletne strani,
- Ker lahko gostite spletne strani, lahko gostite tudi SPA aplikacije,
- Ker lahko gostite SPA aplikacije, lahko imate vse v enem, tako REST strežnik kot uporabniški vmesnik,
- In, ker vse teče v odprtokodnem .NET okolju, posledično lahko takšna aplikacija teče v mnogih operacijskih sistemih.

Literatura

[1] https://dotnet.microsoft.com

[2] https://angular.dev/

[3] OTS 2022, Sodobne informacijske tehnologije in storitve: Zbornik 25. konference, »Kaj je Blazor in kako se primerja z JavaScript ogrodji?«, Matjaž Prtenjak

[4] OTS 2023, Sodobne informacijske tehnologije in storitve, Zbornik 26. konference, »Kaj je Blazor Hibrid in kako nam lahko pomaga tudi pri nadgradnji programske opreme?«, Matjaž Prtenjak

[5] OTS 2022, Sodobne informacijske tehnologije in storitve: Zbornik 24. konference, »Kako razvijati v VUE/NUXT, če si navajen/a objektnih jezikov kot so C++, C# ali Java?«, Matjaž Prtenjak

Programska koda celotne aplikacije: https://github.com/MPrtenjak/OTS2024-NET-Angular-WEB-And-Desktop

Preizkušeni pristopi pri upodabljanju spletnih aplikacij na strežniku

Manica Abramenko, Žiga Lah, Nejc Hauptman, Jani Šumak

Inova IT d.o.o., Maribor, Slovenija
manica.abramenko@inova.si, nejc.hauptman@inova.si,
ziga.lah@inova.si, jani.sumak@inova.si

V prispevku predstavljamo, kakšne prednosti prinaša upodabljanje spletnih aplikacij na strežniku (ang. Server-Side Rendering, v nadaljevanju SSR) v nasprotju s prikazovanjem pri odjemalcu (ang. Client-Side Rendering, v nadaljevanju CSR). SSR je tehnika spletnega razvoja, pri kateri strežnik ustvari celotne strani, prilagojene za vsako zahtevo uporabnika, in jih dostavi brskalniku uporabnika. Ta pristop ponuja mnogo prednosti, vključno s hitrejšim začetnim nalaganjem strani, izboljšano optimizacijo za iskalnike in boljšo zmogljivostjo na napravah z manjšo procesorsko močjo. V uvodu bomo izpostavili, s kakšnimi težavami se srečujemo kot razvijalci spletnih aplikacij s prikazovanjem pri odjemalcu in kako bi se jih lahko lotili z uporabo spletnih aplikacij na strežniku. Podrobneje bomo predstavili, kaj je SSR, in pregledali ogrodja, ki jih lahko uporabimo za generiranje vsebine spletnih aplikacij na strežnikih, kot sta Remix in Next.js. Na praktičnem primeru bomo prikazali rezultate posameznih metrik, ki so relevantne pri prikazovanju, nalaganju in delovanju spletne aplikacije. Prispevek zaključimo z razmišljanjem o časovnih, prostorskih in omrežnih obremenitvah, ki jih prinese razvijanje dotičnih spletnih aplikacij.

Ključne besede:
spletni razvoj
SSR
SEO
JS ogrodja
React
Remix
Next.js

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.20

1 Uvod

V sodobnem razvoju spletnih aplikacij se soočamo z različnimi izzivi, ki vplivajo na uporabniško izkušnjo, zmogljivost aplikacij in njihovo optimizacijo za iskalnike. Ena izmed ključnih odločitev pri razvoju je izbira med upodabljanjem pri odjemalcu (ang. Client Side Rendering, v nadaljevanju CSR) in upodabljanjem na strežniku (ang. Server Side Rendering, v nadaljevanju SSR). Vsaka od teh tehnik ima svoje prednosti in slabosti, vendar je v zadnjem času SSR pridobil veliko pozornosti zaradi svojih številnih koristi. Tradicionalno se za upodabljanje na strežniku uporabljajo jeziki, kot so PHP, ASP.NET, Java in Ruby. V svetu JavaScript-a pa so priljubljena ogrodja, kot so Next.js, Remix, Astro, Gatsby, Nuxt in SvelteKit, ki razvijalcem omogočajo preprosto implementacijo SSR v njihovih spletnih aplikacijah.

V tem prispevku bomo raziskali tehniko SSR ter delovanje, njene prednosti in slabosti, načine implementacije in različna ogrodja, ki so na voljo za spletni razvoj z uporabo SSR, kot so Remix, Next.js in 'lastna implementacija'. Prav tako bomo predstavili primere uporabe in analizirali, kako lahko SSR izboljša učinkovitost in optimizacijo spletnih aplikacij.

2 Kaj je SSR?

SSR je tehnika spletnega razvoja, pri kateri strežnik obdela in upodobi HTML vsebino spletne strani, preden jo dostavi odjemalcu. To je glavna razlika v primerjavi s CSR, ki s pomočjo JavaScirpta po prenosu potrebnih sredstev izriše vsebino strani. V praksi to pomeni, da za upodabljanje vsebine pri CSR potrebujemo prenos datotek za stile in skripte ter samih podatkov, ki jih želimo prikazati, medtem ko SSR vrne HTML dokument z vsebino. Ker tudi spletni iskalniki dobijo dejansko HTML vsebino, lahko tako stran indeksirajo na višje mesto, kar izredno izboljša optimizacijo za brskalnike (ang. Search Engine Optimization, v nadaljevanju SEO).

SSR lahko implementiramo na dva načina: SSR ob poslani zahtevi (ang. SSR-at request time), ki generira HTML ob vsaki spremembi enotnega naslova vira (ang. Uniform Resource Locator, v nadaljevanju URL), ali pa SSR ob izgradnji. Pri slednji se HTML vsebina generira vnaprej in jo strežnik pošlje brskalniku ob zahtevi) [1], [2].

2.1. Delovanje SSR

Življenjski cikel SSR spletne aplikacije vsebuje korake, ki so porazdeljeni med strežnik in odjemalca. Najzahtevnejše operacije (pridobivanje in procesiranje podatkov) se dogajajo na strežniku, medtem ko za interaktivnost skrbi odjemalčev del na brskalniku. Glavna prednost tega pristopa je, da lahko vsebino pokažemo, preden stran postane interaktivna. Cikel je sestavljen iz naslednjih korakov (Slika 1):

- Začetna zahteva: Ko uporabnik prispe na spletno stran preko določenega URL-ja, se sproži zahtevek na strežnik z vsemi podatki, ki jih strežnik potrebuje (piškotki, glave zahtevka, parametri v URL-ju).
- Procesiranje in pridobivanje podatkov: Ob prejemu zahtevka na strežniku se izvedejo vse potrebne operacije. V začetku je poskrbljeno za avtorizacijo in avtentikacijo, s katero preverimo, ali ima uporabnik pravice, ki jih potrebuje za želeno vsebino, prav tako se preveri žeton seje, ki zagotavlja uporabnikovo sejo na brskalniku. Obenem se zabeležijo (ang. logging) vsi zahtevki za potrebe analitike in odpravljanje napak (ang. debugging). Glede na zahtevek iz brskalnika strežnik izvede pridobivanje podatkov iz različnih virov, ki jih potem lahko obdelamo, da jih je mogoče uporabiti pri prikazu strani uporabniku. V tem koraku lahko implementiramo predpomnjenje (ang. caching), ki zmanjšuje obremenitev strežnika. V tem koraku poskrbimo za lokalizacijo, vsa poslovna pravila in vsa ostala preoblikovanja, da ustrezajo uporabniku, ki je poslal zahtevek.

- Izris: Po prejemu vseh podatkov se s pomočjo različnih predlog (ang. template) ali ogrodij na strežniku sestavi celotna HTML struktura strani.
- Odziv (ang. response): Zraven vsebine pošiljamo še vrsto vsebine, politike predpomnjenja in druge metapodatke, ki jih potrebujejo moderni brskalniki. Da bi zmanjšali velikost poslanih datotek, uporabljamo tehnike stiskanja, ki jih omogočajo programi, kot sta Gzip in Brotli, kar izboljša delovanje strani v okoljih, ki uporabljajo slabšo internetno povezavo. Omrežja za dostavo vsebine (ang. Content Delivery Network, v nadaljevanju CDN) uporabljamo za predpomnjenje vsebin na različnih geografskih območjih, da zmanjšamo zamude pri prenosu podatkov.
- Hidracija (ang. hydration): Del, ki se začne dogajati na brskalniku takoj po prejemu HTML datotek s strani strežnika. Poleg same vsebine se začnejo prenašati vsi JavaScript ter CSS viri, ki jih potrebujemo za interaktivnost same strani. Ob prenašanju se vzpostavljajo poslušalci dogodkov (ang. event listeners), inicializacirana sta upravljanje stanja (ang. state management) in povezava z vsemi dinamičnimi elementi. Ko so vsi koraki končani, postane stran za uporabnika interaktivna. Za upravljanje navigacije in sprememb URL-jev uporabljamo mehanizme na strani odjemalca, da ne potrebujemo ob vsaki stvari novih zahtevkov za dodatne vire.

Slika 1: Prikaz delovanja SSR cikla.

SSR ponuja številne prednosti, vendar ima tudi določene pomanjkljivosti, ki jih morajo spletni razvijalci skrbno pretehtati, preden se odločijo za dotični pristop [1], [2].

2.2. Prednosti SSR

- Hitrejše začetno nalaganje strani: Začetni čas nalaganja je lahko hitrejši, saj strežnik pošlje popolnoma upodobljeno stran brskalniku, ki jo lahko takoj prikaže. SSR posodablja le dele HTML-ja, ki jih je treba posodobiti, zato ustvarja hitrejše prehode strani med stranmi in veliko hitrejše prvo barvanje vsebine (FCP).
- Izboljšan SEO: Indeksiranje spletnih mest, ki uporabljajo SSR, je za iskalnike veliko lažje kot indeksiranje spletnih mest, upodobljenih na strani odjemalca. Vsebina je upodobljena, preden se stran naloži, zato jim ni treba zagnati JavaScripta, da bi jo prebrali in indeksirali.
- Boljša performančnost na napravah z nižjo zmogljivostjo: Uporabniki s počasno internetno povezavo ali s starejšimi napravami lahko takoj vzpostavijo interakcijo s spletnimi stranmi. Upodabljanje vsebine poteka na strežniku, kar zmanjša obremenitev procesiranja na strani odjemalca.
- Konstanta dostava vsebine: Ker je vsebina generirana na centralnem strežniku, vsi odjemalec vidijo enako vsebino spletne strani.
- Dinamična dostava vsebine: SSR omogoča dinamično dostavo vsebin z upodabljanjem vsebine na strežniku in pošiljanjem v brskalnike uporabnikov. To zagotavlja hitrejši prikaz dinamične vsebine, kar pozitivno vpliva na angažiranost uporabnikov in uspešnost SEO [3], [4], [5].

2.3. Pomanjkljivosti SSR

- Povečana obremenitev strežnika: : SSR lahko dodatno obremeni strežnik, saj vsaka zahteva terja upodabljanje nove strani. To je mogoče ublažiti s strategijami predpomnjenja, vendar še vedno predstavlja potencialno težavo glede razširljivosti.
- Počasnejša interaktivnost: Medtem ko je začetno nalaganje strani hitrejše, se lahko naslednje interakcije počasnejše v primerjavi z upodabljanjem na strani odjemalca, saj lahko uporabnikove interakcije terjajo dodatne zahteve strežnika, ki mora ponovno dostaviti vsebino.
- Kompleksnost razvoja: Implementacija SSR je lahko bolj zapletena in zahteva dodatno orodje in konfiguracijo. Morda bodo potrebne tudi spremembe v pristopu razvijalcev, kar zadeva upravljanje stanja in usmerjanja. V primeru statičnih strani teh težav ni.
- Omejena interaktivnost: Zapletene interakcije na strani odjemalca bodo zahtevale dodaten trud za implementacijo, saj niso tako dinamične kot upodabljanje na strani odjemalca.
- Zakasnitev (ang. latency): Odvisno od lokacije strežnika lahko pride do večje zakasnitve, zlasti za uporabnike, ki so geografsko oddaljeni od strežnika.
- Težave z združljivostjo: Številne knjižnice in orodja tretjih oseb niso združljive z upodabljanjem na strani strežnika [3], [4], [5].

3 Analiza priljubljenih React rešitev

3.1. Remix

Remix je celovito spletno ogrodje (ang. framework), ki za svoje delovanje uporablja knjižnico React. Gre za odprtokodno ogrodje, razvito v skladu s sodobnimi spletnimi praksami in standardi. Omogoča izdelavo naprednih in učinkovitih spletnih vmesnikov, ki zadovoljujejo širok spekter uporabniških potreb. Poseben poudarek dajejo strežniškemu upodabljanju in predpomnjenju, kar bistveno pripomore k hitrejšemu nalaganju in izboljšani uporabniški izkušnji [6]. Z drugimi besedami, gre za strežniški in brskalniški izvajalec kode (ang. runtime), ki omogoča hitro nalaganje strani in instantne tranzicije z uporabo vgrajenih funkcionalnosti brskalnikov in porazdeljenih sistemov. Temelji na uporabi Fetch API-ja, zato ga lahko poganjamo praktično kjerkoli. Deluje na tradicionalnih Node.js okoljih in tudi na brezstrežniških [7].

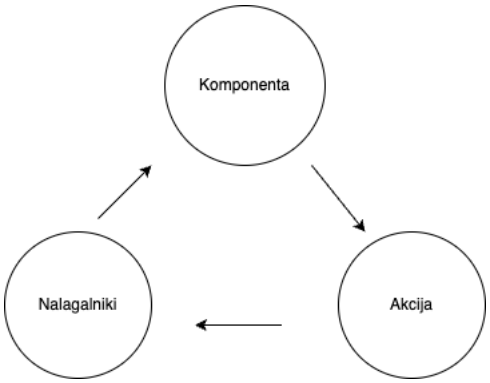
Ogrodje Remix se lahko predstavi kot prevajalnik (ang. compiler), upravljalec strežniških zahtev (ang. server-side HTTP handler), strežniško ogrodje (ang. server framework) ali kot ogrodje za odjemalca (ang. client framework) [7].

Remix deluje kot *prevajalnik* z uporabo Vita. Vite na strežniku ustvari HTTP upravljalca, ki skrbi za upodabljanje in izvajanje strežniških zahtev za podprte poti. Na odjemalcu ustvari gradnjo (ang. build), ki za avtomatsko deljenje zagotavlja kode glede na pot, uvoz slik in virov ter izvedbo drugih optimizacij, potrebnih za delovanje aplikacije v brskalniku. Prevajalnik generira manifest virov (ang. asset manifest), ki se uporabljajo pri uvodnem upodabljanju na strežniku (ang. Initial server render) ali pri prednalaganju (ang. prefetch) na strani odjemalca. Remix ni server, četudi se izvaja na strežniku, temveč gre za upravitelja HTTP zahtev. Ker temelji na uporabi JavaScripta, ga lahko namestimo praktično kamorkoli, kjer podpirajo JavaScript okolja. Zgrajen je bil okrog Fetch API-ja in za obdelovanje HTTP zahtev uporablja različne pakete, ki služijo kot adapterji. Eden takšnih je paket @remix-run/express. Kot večina *strežniških ogrodij* tudi Remix uporablja arhitekturni pristop MVC (Model-View-Controller). *Pogled* in *krmilnik* opravljata enako vlogo, kot to poteka znotraj drugih strežniških ogrodij, npr. Laravel, vendar razvijalcu ni potrebno pisati ločenih krmilnikov in pogledov, saj za abstrakcijo na podlagi procesiranja URL poti poskrbijo moduli Remix Route. Načine in pristope pri implementaciji modela ogrodje prepusti razvijalcu. Remix lahko uporabljamo kot strežniško ogrodje brez uporabe JavaScript-a v brskalniku. Na podlagi definiranih poti na strežniku naložimo podatke, jih spreminjamo z uporabo akcij in HTML obrazcev ter prikažemo spremembe na

uporabniškem vmesniku z zgolj osnovnim delovanjem brskalnikov. *Brskalniško ogrodje* deluje z uporabo hidracije, kar pomeni, da Remix hidrira stran z uporabo JavaScript modulov. Remix z vgrajenimi optimizacijami zazna, v katerem delu aplikacije so se zgodile spremembe, in na podlagi tega pošlje le spremembe za tiste dele aplikacije, ki so se spreminjali. To pomeni, da ni treba zahtevati celotnega dokumenta, kar bi povzročilo večjo porabo virov in slabšo uporabniško izkušnjo. Remix lahko prednaloži podatke in različne druge vire (CSS in JavaScript module) pred izvedbo uporabnikove akcije.

Ogrodje vsebuje že vgrajen sistem za usmerjanje (ang. routing), ki poleg podpore statičnih poti podpira tudi gnezdeno usmerjanje (ang. nested routes). Vgrajen je sistem za usmerjanje temelji na datotečnem sistemu, kar pomeni, da strani ustvarjamo s kreacijo map in podmap, pri čemer imena predstavljajo segmente poti. Modularnost, ki jo zagovarja pristop gnezdenega usmerjanja, zagotavlja, da je vsaka pot osredotočena zgolj na dodeljeni segment v URL-ju in pripadajoči del uporabniškega vmesnika [8]. Uporaba gnezdenega usmerjanja vpliva tudi na čas nalaganja. Remix paralelno nalaga vse potrebne vire, ko se URL ujema z več potmi. V nasprotju z enostranskimi aplikacijami, kjer imamo sekvenčno nalaganje virov, paralelno izvajanje bistveno izboljša perfomanco aplikacije.

Avtomatsko ohranjanje usklajenosti uporabniškega vmesnika s stanjem serverja predstavlja eno izmed jedrnih funkcionalnosti Remixa. Podatkovni tok je sestavljen iz treh gradnikov; komponente, akcije in nalagalnikov (ang. loaders) (Slika 2) [9].



Slika 2: Prikaz podatkovnega toka.

Znotraj usmerjevalne datoteke (ang. route file) definiramo *nalagalno funkcijo*, ki poskrbi za pridobitev podatkov in jih nato posreduje *komponenti*. Komponenta predstavlja del uporabniškega vmesnika in uporablja podatke, pridobljene iz nalagalne funkcije. Ko uporabnik znotraj forme napravi določeno spremembo in izvede akcijo (npr. stisne na gumb), se izvede akcija na pot, definirano znotraj atributa *action* (Slika 3).

```
routes/user-details.tsx

...

export async function loader({ request }: LoaderFunctionArgs ) {
   const { name, email } = await getUserDetails(request);

   return json({ name, email});
}


export default function UserDetails() {
   const { name, email } = useLoaderData<typeof loader>();

   return (
      <Form method="post" action="/user-details">
         <span>Uporabnik: ${name} </span>
         <input name="ime" defaultValue={name}  />
         <input name="email" defaultValue={email} />
         <button type="submit">Shrani</button>
      </Form>
    );
}


export async function action({ request }: ActionFunctionArgs) {
   const formData = await request.formData();
   const { id } = await getUserDetails(request);

   await updateUser(id, {
       email: formData.get("email"),
       displayName: formData.get("displayName"),
   });

   return json({ ok: true });
}
```

Slika 3: Odsek kode, ki prikazuje uporabniški vmesnik in funkcijo *action*.

Ena izmed temeljnih smernic spletnega ogrodja Remix je minimalna količina JavaScripta in progresivna izboljšava (ang. progressive enhancement). Ta pristop je usmerjen v zagotavljanje optimalne uporabniške izkušnje za vse uporabnike, ne glede na zmogljivost njihovih naprav, različico brskalnika ali hitrost internetne povezave [10]. Progresivna izboljšava pomeni, da aplikacija zagotovi osnovno funkcionalnost vsem uporabnikom, kar vključuje prikaz osnovne HTML strani, napolnjene z vsebino. Uporabnikom z dobro internetno povezavo in sodobnejšimi brskalniki pa omogoči naprednejšo izkušnjo z uporabo stilov, animacij in drugih naprednih funkcionalnosti Ta strategija pospešuje nalaganje spletne strani in olajša indeksiranje, saj se vsebina naloži takoj preko HTML izvorne kode, ne da bi bilo treba predhodno naložiti JavaScript [11].

V nasprotju s SPA (ang. Single Page Application) aplikacijami, kjer SPA ob zahtevi vrne prazen dokument in na to sinhrono nalaga vire ter izvaja upodabljanje (ang. rendering) ko se naloži JavaScript, aplikacije, napisane z ogrodjem Remix, ob prvi zahtevi paralelno izvedejo nalaganje potrebnih virov. To pomeni, da je čas do prvega prikaza (ang. time to first paint) izrazito krajši, saj Remix lahko zagotovi HTML, preden se JavaScript naloži [12].

3.2. Next.js

Next.js je eno izmed najbolj priljubljenih JavaScript ogrodij za razvoj celovitih spletnih rešitev. Enako kot Remix Next.js za razvoj spletnih aplikacij uporablja knjižnico React. Sovpada z arhitekturnim stilom Jamstack, ki predstavlja aplikacije, razvite s pomočjo programskega jezika JavaScript. Premika zadolžitve na stran odjemalca, ki na podlagi API (ang. Application Programming Interface) zahtev pridobivajo podatke in komunicirajo z različnimi servisi. Nato strani, zapisane z označevalskim jezikom (ang. Markup) ob gradnji, servira na mrežo za dostavo vsebine (ang. Content Delivery Network ali CDN) [13]. Next.js vsako stran predhodno tudi upodobi (ang. pre-render) in strežnik odjemalcu vrne zgolj potrebno HTML vsebino z minimalno količino JavaScripta, kar pripomore k boljši performanci in SEO (ang. Search engine optimization) analizi.

Glavne značilnosti ogrodja Next.js so upodabljanje na strežniku, samodejno deljenje kode, statično generiranje strani in razvoj javnega API-ja [14]. Next.js podpira več pristopov za serviranje vsebine:

- Statično generiranje (SSG - Static Site Generation): strani se generirajo ob času gradnje in se shranijo kot statične datoteke. Ta pristop se pogosto uporablja kadar so spremembe strani zelo redke.
- SSR: nalogo generiranja strani prevzame strežnik in se zgodi ob vsakem odjemalčevem zahtevku. Pristop omogoča tudi serviranje dinamične vsebine, vendar se lahko čas dostave podaljša.
- Inkrementalna statična regeneracija (ISR - Incremental Static Regeneration): gre za kombinacijo med pristopoma SSG in SSR, kjer se strani generirajo statično, vendar se lahko določeni deli vsebine občasno osvežijo na strežniku brez potrebe po ponovnem gradbenem postopku celotne strani.
- Renderiranje na strani odjemalca (CSR): vsebina se naloži in upodobi na strani odjemalca. Ta pristop se je uveljavil z vpeljavo enostranskih aplikacij.

Next.js ima 2 tipa usmerjanja - tradicionalni Pages in nov pristop, imenovan App Router. Pri uporabi Pages usmerjevalnika razvijalec z uporabo datotečnega sistema definira strani tako, da jih doda oz gnezdi kot datoteke pod mapo *pages*. Vsaka datoteka v tej mapi postane dostopna preko URL-ja. Na primer:

Na primer:

- *pages/index.tsx* postane domača stran *(/)*
- *pages/uporabniki.tsx* postane dostopna na */uporabniki*
- *pages/uporabniki/zahteve.tsx* postane dostopna na */uporabniki/zahteve*

V vsaki datoteki lahko z uporabo metode *getStaticPaths* definiramo katere dinamične poti naj se vnaprej generirajo. Z uporabo metod *getStaticProps* ali *getServerSideProps* pa lahko pridobimo določene podatke. Izvajata se na serverju.

3.2.1. Dinamično usmerjanje

Za dinamične poti Next.js uporablja oklepaje v imenih datotek. Npr: *pages/uporabniki/[id].tsx*, id je dinamičen šifrant, kar pomeni, da bo, ko uporabnik navigira na stran */uporabniki/1,* pridobil in prikazal podatke uporabnika z ID-jem ena (Slika 4) [15].

```
pages/uporabniki/[id].tsx

import { useRouter } from 'next/router'


export default function Uporabnik() {
   const router = useRouter()
   return <p>ID: {router.query.id}</p>
}
```

Slika 4: Odsek kode, ki prikazuje dinamično usmerjanje.

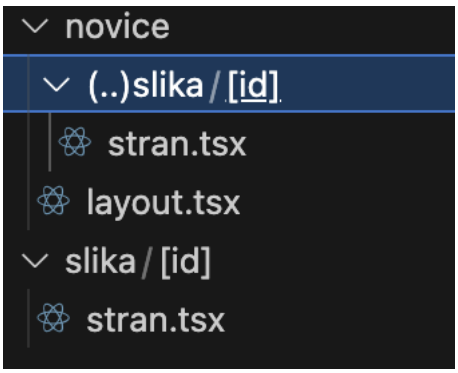
Nextov usmerjevalnik podpira lovljenje dinamičnih segmentov. Z uporabo lovljenja dinamičnih segmentov lahko ulovimo segmente neke URL poti. Datoteko, poimenovano */pages/uporabniki/**[...id]**,* pri čemer [...šifrant] predstavlja dinamične segmente, pove Nextu, da mora omogočiti lovljenje dinamičnih segmentov za ta celoten URL [16]. Z uporabo dvojnih zavitih *oklepajev (/pages/uporabniki/**[[...id]]**)* pa gre za opcijsko lovljenje dinamičnih segmentov.

Tabela 1: Uporaba dinamičnih segmentov.

Pot	URL	Parametri
pages/uporabniki/[...id].tsx	/uporabniki/1	{ slug: [‘1’] }
pages/uporabniki/[...id].tsx	/uporabniki/1/zahteve	{ slug: [‘1’, 'zahteve'] }
pages/uporabniki/[[...id]].tsx	/uporabniki/1/zahteve	{ slug: [] }

Next.js od različice 13 vpeljuje nov koncept usmerjanja imenovan *"App Router",* ki prinaša večjo fleksibilnost, zmogljivost in modularno izdelavo sodobnih spletnih aplikacij [16].

- Struktura in osnovno delovanje: Usmerjanje temelji na strukturi datotek znotraj mape *app*. Enako kot pri tradicionalnem pristopu, tudi tukaj struktura map določa URL-je. Na primer, datoteka *app/uporabniki/konfiguracija.tsx* postane dostopna na URL poti */uporabniki/konfiguracija*. Ta primer predstavlja gnezdeno usmerjanje, ki je prav tako podprto znotraj App Routerja ter temelji na hirarhiji komponent.
- Strežniške komponente: Omogoča uporabo strežniških komponent, ki se renderirajo na strežniku, kar omogoča boljšo zmogljivost in lažje upravljanje stanja. Nudi tudi kombiniranje strežniških in odjemalskih komponent.
- Izboljšano pridobivanje podatkov: Poenostavljeno pridobivanje podatkov z integracijo v strežniške komponente in podpora za pretakanje (ang. streaming), kar omogoča postopno renderiranje in hitrejšo dostavo vsebine.
- API Routes: API poti so definirane v mapi *app/api*, kar omogoča poenostavljeno organizacijo kode znotraj iste strukture map.
- Dodatne funkcionalnosti: Boljše upravljanje stanja in optimizacija zmogljivosti z uporabo strežniških komponent in podpora jedernih React komponent, kot je na primer Suspense.
- Vzporedne poti (ang. parallel routes): S pristopom vzporednih poti lahko naložimo več strani znotraj določenega dela aplikacije in je uporaben, kadar so ti deli dinamični. Za uporabo tega načina navigaranja moremo ime direktorija začeti z afno - *@datoteka*.
- Prestrezanje poti (ang. intercepting routes): Gre za pristop, ki podpira nalaganje poti iz drugih delov aplikacije [17]. Kadar uporabljamo ta pristop moramo imenik poimenovati na podlagi *oklepaj pika* konvencije, pri čemer vsaka dodatna pika znotraj oklepaja definira kolko segmentov želimo upoštevati (Slika 5).
    - (.), zadetki zgolj na istem nivoju
    - (..), zadetki na nivoju višje
    - (..)(..), zadetki dva nivoja višje
    - (...), vsi zadetki od “app”-a naprej



Slika 5: Poimenovanje imenika.

3.2.2. Predpomnjenje in optimizacije

Next.js vključuje več funkcij za optimizacijo in predpomnjenje, ki pomagajo pri izboljšanju zmogljivosti aplikacij, poskrbi pa tudi za granjenje in dostavo aplikacij z uporabo njihovega oblaka.

- Predpomnjenje strani: Statično generirane strani in API poti se predpomnijo za hitrejšo dostavo.
- Optimizacija slik: Vgrajena podpora za optimizacijo slik, kompresijo in prilagodljivost ločljivosti. Vključuje vgrajeno komponento, ki poskrbi za samodejno optimizacijo in predpomnjenje slik, kar izboljša zmogljivost nalaganja slik.
- Samodejno deljenje kode: Next.js samodejno deli kodo v manjše dele, ki se nalagajo po potrebi, kar zmanjšuje čas začetnega nalaganja strani.
- Prednalaganje: Next.js omogoča prefetching povezave, kar pomeni, da se povezane strani vnaprej naložijo, ko uporabnik pride v njihovo bližino, kar izboljšuje hitrost navigacije po spletni strani.
- Optimizacija razvijalske izkušnje s postavitvijo aplikacij na Vercelov oblak: Podpora za samostojno skaliranje aplikacij in optimizacijo dostave aplikacij, kadar je aplikacija postavljena na Vercelu.

3.3. React Core - Lastno ogrodje po meri

Danes na spletu lahko najdemo ogromno različnih ogrodij, ki jih lahko uporabimo za razvoj kompleksnih in preprostih aplikacij. Ogrodja poskrbijo, da se lahko razvijalec v večini primerov osredotoča zgolj na razvoj aplikacije, za vse optimizacije, mejne pogoje in podporo na različnih brskalnikih pa je zadolženo ogrodje. Najdejo se tudi primeri, ko zahtev ne morejo zadovoljiti že preizkušena ogrodja in je potrebo razviti ogrodje po meri. Z lastnim ogrodjem tako avtor sam definira arhitekturo, navigacijo, pristope in konvencije. Prav tako je fleksibilen pri izbiri programskega jezika, načina namestitve, uporabe knjižnic ter drugih orodij.

Kadar gradimo ogrodje po meri, moramo upoštevati imeti tri komponente:

- strežnik (Node.js)
- renderiranje na strežniku
- hidracija

Naloga strežnika je da poskrbi za podatke in generiranje HTML vsebine. Server z uporabo *renderToPipeableStream* funkcije iz paketa *react-dom/server* generira HTML vsebino v Node.js stream. *renderToPipeableStream* (Slika 6) kot

parameter funkcije sprejme React vozlišče (ang. React Node), v tem primeru gre za App komponento (Slika 7), ki je glavna komponenta aplikacije [18].

```
import { renderToPipeableStream } from 'react-dom/server';
…


app.use('/', (request, response) => {
 const { pipe } = renderToPipeableStream(<App />, {
   bootstrapScripts: ['/main.js'],
   onShellReady() {
     response.setHeader('content-type', 'text/html');
     pipe(response);
   }
 });
});
```

Slika 6: Prikaz renderToPipeableStream funkcije.

```
export const App = () => {
 return (
   <html>
     <head>
       <meta charSet="utf-8" />
       <meta name="viewport" content="width=device-width, initial-scale=1" />
       <link rel="stylesheet" href="/styles.css"></link>
       <title>OTS APP</title>
     </head>
     <body>
       <Router />
     </body>
   </html>
 );
}
```

Slika 7: Prikaz App komponente.

React bo sam poskrbel, da vstavi doctype HTML značko, ko servira vsebino odjemalcu.

Na strani odjemalca nato pokličemo *hydrateRoot* funkcijo (Slika 8), ki bo na prejet HTML dokument dodala dogodkovne poslušalce. To pripomore, da stran postane interaktivna.

```
import { hydrateRoot } from 'react-dom/client';
import { App } from './App;

hydrateRoot(document, <App />);
```

Slika 8: Funkcija hydrateRoot.

4 Primerjava

Primerjava ogrodij za strežniško upodabljanje spletnih ogrodij predstavlja dvojen izziv. Če bi merili zgolj hitrost strežnika, bi lahko merili odzivni čas, odzivni čas s serializacijo, odzivni čas s poizvedbo v bazi ipd. in pri tem upoštevali strojno opremo in operacijski sistem. Če bi, po drugi strani, merili samo čelni del, bi se osredotočili na hitrost upodabljanja v brskalniku - več o tem v nadaljevanju.

Ker prispevek govori o upodabljanju na strežniku, se nam je zdelo primerno primerjati oboje, tako hitrost zalednega sistema kot čelnega dela. Pri prvem bo večji poudarek seveda na hitrosti upodabljanja tistega deleža aplikacije, ki bi bil sicer upodobljen v odjemalcu, če bi razvijali SPA aplikacijo.

Ob zavedanju, da je tovrstnih primerjav veliko, so primerjave, predvsem tiste, ki se nanašajo na merjene čelnega in zalednega sistema, izvedene predvsem za demonstracijo. Izsledke je potrebno analizirati v luči razvijalske izkušnje.

4.1. Začetni projekt

Osnovne nastavitve in dobra praksa pri orodjih, potrebnih za generiranje in zagon projekta, postajajo del ogrodji, zato bomo uvodoma primerjali generatorje kode za Next.js, Remix in samostojno rešitev za React z SSR [19].

Prenovljena dokumentacija Reacta priporoča uporabo ogrodij Next.js in Remix. Razvijalci orodja so torej mnenja, da je za večino primerov bolje uporabiti vnaprej določene rešitve za pogoste težave. Po njihovem mnenju je pomembneje, da projekti, ne glede na čelno ogrodje, rešujejo podobne težave za uvajanje, kar je robustno in razširljivo [19].

Glede na povzeto nas torej ne preseneča, da imata omenjeni ogrodji vsako svoj program za generiranje začetnega projekta, in sicer create-next-app in create-remix [20].

Če želimo lastno rešitev za SSR, je na voljo nekaj generatorjev kode, toda odločitev je na strani razvijalca. RAzvali Reacta nas z opozorilo, da če "ima [naša] aplikacija nenavadne omejitve, ki jih [ta] ogrodja ne rešujejo [...]", uporabimo generator Vite ali Parcel [21].

Generirana koda se v omenjenih primerih ne razlikuje veliko, če seveda odmislimo posebnosti ogrodij in generatorjev kode. Z generatorji za lastno rešitev imamo pričakovano več dela, saj moramo poiskati primerno predlogo za SSR. V primeru uporabe orodja Vite, je potrebno izbrati možnost create-vite-extra in nato želeno predlogo za react-ssr [22]. Ker gre za rešitev po meri, je na razvijalcih, da se odločijo, ali želijo imeti predlogo z TypeScriptom in predlogo s pretakanjem, medtem ko je pri ogrodjih Remix in Next.js pretakanje vgrajeno v ogrodje [23], [24].

4.2. Uvajanje

Kar je bilo rečeno za generiranje začetnega projekta, je v veliki meri moč trditi tudi za uvajanje aplikacije na strežnik.

Generatorji kode za aplikacije, pripravljene z generatorji kode za Next.js oziroma Remix, in dokumentacija omenjenih ogrodji vključujejo navodila in rešitve za uvajanje na strežniku. Next.js celo ponujajo lastno rešitev za uvajanje [25].

Pri lastni rešitvi takšnih, vnaprej pripravljenih rešitev seveda ni, zato se od razvijalcev pričakuje, da sami vzpostavijo procese uvajanja in sami izberejo strežniško okolje.

4.3. Primerjava zalednega sistema

Pri primerjavi zalednih sistemov se bomo osredotočili na hitrost upodabljana strani. Seveda bi lahko merili tudi čas obdelave zahtevkov, čas odziva, odzivnost strežnika ob visokem prometu, porabo strojnih virov ipd., vendar so avtorji mnenja, da bi takšne primerjave zastrle jedro prispevka in nehote sprožile razprave o optimizacijah.

Za potrebe pričujoče primerjave bomo torej vzeli vnaprej pripravljene rešitve brez optimizacij. Vprašanje, ki si ga bomo zastavili, pa bo, kako hitro lahko ogrodja upodobijo in pošljejo daljši članek. Za merjenje hitrosti bomo uporabili orodje Jmeter [26].

Koda za dotično primerjavo in primerjavo čelnega dela je prosto dostopna na Github [27].

Tabela 2: JMeter rezultati zalednih sistemov (10 niti)

Ogrodje	Povprečni odzivno čas	Najhitrejši odzvini čas	Najpočasnejši odzivni čas	KB/s
React SSR	14	9	128	0.01
Next.js	29	9	128	0.10
Remix	17	9	128	0.08

4.4. Primerjava čelnega sistema

Za primerjavo čelnih sistemov se vse bolj uveljavlja Web Vitals. Web Vitals je Googlova iniciativa za merjenje kakovosti spletne uporabniške izkušnje [28].

Jedrne metrike so:

- Izris največjega vsebinskega dela (LCP) meri hitrost nalaganja strani, ko se stran prvič zažene. Za dobro uporabniško izkušnjo naj bo vrednost LCP 2.5 sekund ali manj. Meritev se nanaša na izris največjega elementa v vidnem področju uporabniškega očesa [29].
- Interakcija do naslednjega izrisa (INP) meri interaktivnost. Za dobro uporabniško izkušnjo naj bo vrednost INP 200 milisekund ali manj. Metrika meri zamik med uporabniško akcijo in odzivom strani [30].
- Celoten zamik postavitve (CLS) meri stabilnost strani. Za dobro uporabniško izkušnjo naj bo vrednost

Tabela 3: Core web vitals

Ogrodje	LCP	INP	CLS
Reactt SSR	0.4 s	0.48 s	0.001
Next.js	0.5 s	0.16 s	0.001
Remix	0.5 s	0.16 s	0.001

5 Razvijalska izkušnja

Čeprav se zavedamo, da je težko govoriti o razvijalski izkušnji uporabe in dela z zgoraj naštetih ogrodji, je ta pomemben dejavnik razvoja [31]. Ker bi celovita primerjava razvijalske izkušnje terjala daljši samostojni prispevek, se bomo omejili na pregled izsledkov dveh večjih spletnih raziskav, State of JavaScript in Stack Overflow developer survey.

V nobeni od navedenih raziskav sicer ne bomo zasledili vprašanj, ki bi se nanašala na uporabo lastnih rešitev za upodabljanje na strežniku, zato se bomo osredotočili samo na uporabi Reacta brez ogrodja in ogrodij Next.js in Remix. Prav tako se bomo pri analizi anket omejili na ugotovitve izvajalcev anket, saj je poglobljena analiza izven dometa tega prispevka.

Ob upoštevanju dejstva, da je bilo ogrodje Remix objavljeno 1. novembra 2020, se bomo v okviru tega prispevka osredotočili na raziskave, izvedene od leta 2022.

5.1. Stack Overflow developer survey

Stack Overflow je spletna stran z vprašanji in odgovori za programerje. Ponuja vprašanja in odgovore na določene teme s področja računalniškega programiranja. Ustvarjena je bila leta 2008 kot alternativa konkurenčni Expert exchange [so] in je eden najbolj priljubljenih virov spletnega iskanja za programerje [32].

Stack Overflow developer survey je vsakoletna spletna raziskava, ki poteka od leta 2010 in se je letno udeleži do 90 tisoč razvijalcev z vsega sveta. Gre za splošno raziskavo o uporabi programskih jezikov, ogrodij in podatkovnih baz. Rezultati in analize raziskav so objavljeni na njihovih spletnih straneh [33].

Raziskava je razdeljena na več sekciji, od splošnih vprašanj o anketirancih do povratnih vprašanj o sami raziskavi. Vprašanja, ki se nanašajo na uporabo tehnologij, so v istoimenski sekciji, ki je razdeljena na podsekcije Najbolj priljubljene tehnologije, Zadovoljstvo in želje, Uporabljene ali bodo uporabljene in Najbolje plačane.

Zaradi manjšega deleža uporabnikov orodja Remix bomo preskočili tretje vprašanje, ki ga je tudi sicer teže prikazati v tabelarni obliki, saj išče povezave med uporabljenimi tehnologijami in tehnologijami, ki jih anketiranci želijo uporabljati v prihodnje. Ker je teh povezav veliko, tabelarni prikaz odgovorov ni najbolj pregleden.

Ravno tako ne bomo uporabili analiz vprašanja glede najbolje plačanih tehnologij, ker vprašanje ni razdeljeno na ogrodja.

Leta 2024 je na vprašanje o spletnih tehnologijah v sekciji Najbolj priljubljene tehnologije odgovorilo 48.503 udeležencev.

Tabela 4: Najbolj priljubljene tehnologije 2024

Ogrodje	Vsi udeleženci	Zaposleni	Izobražujejo	Ostali
React	39.5%	41.6%	36.6%	27.7%
Next.js	17.9%	18.6%	17.9%	12.7%
Remix	1.6%	1.6%	1,3%	1.5%

V sekciji Spoštovane in zaželene je 47.707 anketirancev odgovorilo na vprašanje, s katerimi tehnologijami so bili zadovoljni, katere tehnologije se jim zdijo zanimive oziroma bi jih ponovno uporabili.

Tabela 5: Uporabljene tehnologije 2024

Ogrodje	Zadovoljstvo	Želja
React	62.2%	33.4%
Next.js	59.5%	18.2%
Remix	56.7%	2.8%

Leta 2023 je na vprašanje o spletnih tehnologijah v sekciji Najbolj priljubljene tehnologije odgovorilo 71.802 udeležencev.

Tabela 6: Najbolj priljubljene tehnologije 2023

Ogrodje	Vsi udeleženci	Zaposleni	Izobražujejo	Ostali
React	40.58%	42.87%	36.6%	30.13%
Next.js	16.67%	17.3%	15.12%	13.64%
Remix	1.27%	1.37%	0.76%	1.03%

V sekciji Spoštovane in zaželene je 70.637 anketirancev odgovorilo na vprašanje s katerimi tehnologijami do bili zadovoljni, katere tehnologije se jim zdijo zanimive oziroma bi jih ponovno uporabili.

Tabela 7: Uporabljene tehnologije 2023

Ogrodje	Zadovoljstvo	Želja
React	63.6 %	35.2 %
Next.js	65.9 %	20.3 %
Remix	57 %	3.1 %

Leta 2022 je na vprašanje o spletnih tehnologijah v sekciji najbolj priljubljene tehnologije odgovorilo 58.743 udeležencev.

Tabela 8: Najbolj priljubljene tehnologije 2022

Ogrodje	Vsi udeleženci	Zaposleni	Izobražujejo
React	42.62 %	44.31 %	42.81 %
Next.js	13.52 %	13.93 %	12.32 %
Remix	/	/	/

V sekciji *Spoštovane in zaželene* je 57.654 anketirancev odgovorilo na vprašanje, s katerimi tehnologijami radi delajo, katerim se poskušajo izogniti in s katerimi si želijo delati.

Tabela 9: Uporabljene tehnologije 2022

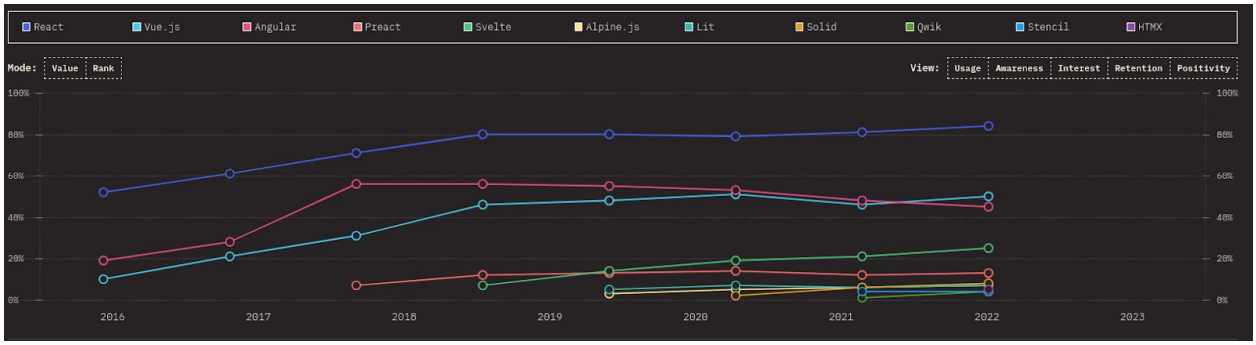
Ogrodje	Zadovoljstvo	Izogibanje	Želja
React	68.19 %	31.81 %	22.54 %
Next.js	69.23 %	30.77 %	11.28 %
Remix	/	/	/

5.2. Stanje JavaScripta (ang. State of JavaScript)

V primerjavi z zgornjo raziskavo je State of JavaScript mlajša spletna raziskava, ki se osredotoča na uporabo programskega jezika JavaScript. Kot prejšnja je tudi State of JavaScript vsakoletna spletna raziskava, ki se je udeleži okoli 30 tisoč razvijalcev z vsega sveta. Raziskava poteka vse od leta 2016, njeni izsledki pa so objavljeni na spletnih straneh [34].

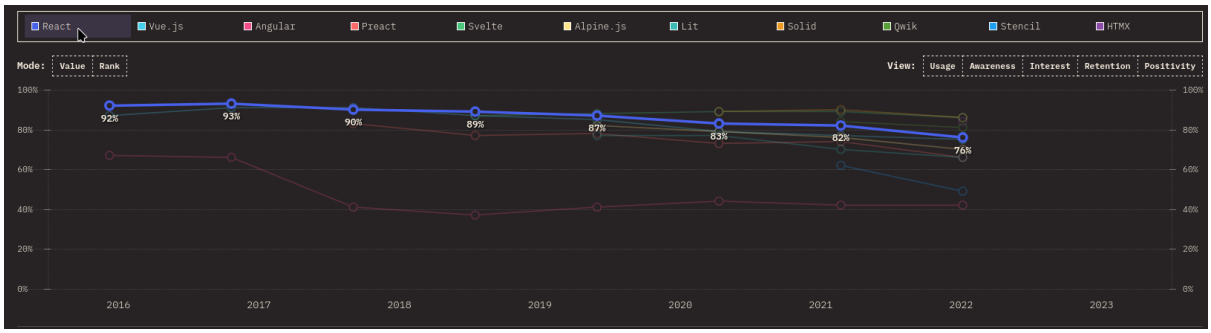
Za razliko od prejšnje ankete State of JavaScript strožje ločuje med ogrodji (čelna ali zaledna ogrodja) in knjižnicami. Tudi demografika se razlikuje, vendar bi analiza presegla okvire tega prispevka.

Iz odgovorov je moč razbrati, da je React trdno zasidran med najbolj priljubljenimi ogrodji, saj mu priljubljenost ne upada (Slika 9).



Slika 9: Uporaba čelnih ogrodji

In čeprav zanimanje za obstoječa orodja in orodja med razvijalci pregovorno upada, ostaja React med najbolj priljubljenimi ogrodji, ki ga razvijalci želijo uporabljati tudi v bodoče (Slika 10).



Slika 10: Priljubljenost členih ogrodji - React

Next.js in Remix sta med razvijalci sicer poznana, ampak v pričujočih anketah ne zasedata vidnejših mest in sta omenjena samo med ostalimi ogrodji za čelne sisteme.

6 Zaključek

Prispevek je primerjal pristope strežniškega upodabljanja za ogrodje React. Avtorji so izpostavili prednosti in slabosti strežniškega upodabljanja, kjer so kot največjo prednost izpostavili izboljšan SEO, ki prinese kompleksnost razvoja, saj je implementacija bolj zapletena. Opisni primerjavi je sledila primerjava generiranja, uvajanja in odzivnosti osnovnih različic, kjer ugotovijo, da obstoječa orodja (Next.js in Remix) ponujajo funkcionalnosti, ki zadostujejo potrebam večine razvijalcev. Na koncu prispevka so avtorji dodali izsledke dveh vidnejših razvijalskih spletnih anket o razvijalskih izkušnjah. V primerjavah so avtorji potrdili priporočila razvijalcev ogrodja React, da je za "rešitev po meri" potrebno več razvijalskega časa in da vsaj v osnovi ni prednosti v primerjavi z ogrodji, kot sta Next.js in Remix. Tudi povzetki iz raziskav, predvsem Stack Overflow developer survey, nakazujejo, da razvijalci vse bolj posegajo po omejenih ogrodjih, medtem ko interes za samostojne rešitve rahlo upada. Slednje namiguje, da so ogrodja, zgrajena na Reactu, dosegla določeno mero stabilnosti in da razvijalcev ne omejujejo pri razvoju spletnih aplikacij. Avtorji ne zanikajo upravičenosti in prednosti razvoja rešitev po meri, vendar dodajo, naj pri odločanju za implementacijo uvajanja na strežniku pristavimo še prednost razvijalske izkušnje, ki kaže v prid ogrodjem in ne rešitvi po meri.

Literatura

[1] https://www.heavy.ai/technical-glossary/server-side-rendering, Server-Side Rendering Definition, obiskano 14. 7. 2024.

[2] https://medium.com/@prashantramnyc/server-side-rendering-ssr-vs-client-side-renderin g-csr-vs-pre-rendering-using-static-site-89f2d05182ef, Server Side Rendering (SSR) vs. Client Side Rendering (CSR) vs. Pre-Rendering using Static Site Generators (SSG) and client-side hydration, obiskano 14. 7. 2024.

[3] https://www.searchenginejournal.com/server-side-rendering/481581/, Server-Side Rendering: The Pros & Cons To Consider For SEO, obiskano 15. 7. 2024.

[4] https://solutionshub.epam.com/blog/post/what-is-server-side-rendering, What is server-side rendering: definition, benefits and risks, obiskano 15. 7. 2024.

[5] https://prismic.io/blog/what-is-ssr, What is Server-side Rendering (SSR)?, obiskano 15. 7. 2024.

[6] https://remix.run/docs/en/main/discussion/introduction, Introduction, technical explanation, obiskano 16. 7. 2024.

[7] https://remix.run/docs/en/main/discussion/introduction, Remix, obiskano 16. 7. 2024.

[8] https://remix.run/docs/en/main/discussion/routes, React Router, obiskano 16. 7. 2024.

[9] https://remix.run/docs/en/main/discussion/data-flow, Fullstack Data Flow, obiskano 16. 7. 2024.

[10] https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement, Progressive enhancement, obiskano 16. 7. 2024.

[11] https://en.wikipedia.org/wiki/Progressive_enhancement, Progressive enhancement, obiskano 16. 7. 2024.

[12] https://remix.run/docs/en/main/discussion/progressive-enhancement, Progressive enhancement, obiskano 16. 7. 2024.

[13] https://umbraco.com/knowledge-base/jamstack/#:~:text=Jamstack%20is%20a%20term%20that,%2C%20and%20Markup%20(JAM), What is Jamstack?, obiskano 16. 7. 2024.

[14] https://nextjs.org/docs, Introduction, obiskano 16. 7. 2024.

[15] https://nextjs.org/docs/pages/building-your-application/routing/dynamic-routes, Dynamic Routes, obiskano 16. 7. 2024.

[16] [https://nextjs.org/docs/app/building-your-application/routing, Routing Fundamentals, obiskano 16. 7. 2024.

[17] https://nextjs.org/docs/app/building-your-application/routing/intercepting-routes, Intercepting Routes, obiskano 16. 7. 2024.

[18] https://react.dev/reference/react-dom/server/renderToPipeableStream, renderToPipeableStream, obiskano 16. 7. 2024.

[19] https://react.dev/learn/start-a-new-react-project#production-grade-react-frameworks, Production-grade React frameworks, obiskano 29. 7. 2024.

[20] https://nextjs.org/docs/getting-started/installation#automatic-installation, Automatic Installation, obiskano 29. 7. 2024.

[21] https://remix.run/docs/en/main/start/quickstart#installation, Installation, obiskano 29. 7. 2024.

[22] https://react.dev/learn/start-a-new-react-project#can-i-use-react-without-a-frameworkm, Start a New React Project, obiskano 29. 7. 2024.

[23] https://remix.run/docs/en/main/guides/streaming, Streaming, obiskano 29. 7. 2024.

[24] https://nextjs.org/docs/app/building-your-application/rendering/server-components#streaming, Streaming, obiskano 29. 7. 2024.

[25] https://vercel.com/docs/deployments/overview, Deploying to Vercel, obiskano 29. 7. 2024.

[26] https://jmeter.apache.org/, Apache JMeter, obiskano 29. 7. 2024.

[27] https://github.com/janiSumak/ots-2024, OTS 2024, obiskano 5. 8. 2024.

[28] https://web.dev/articles/vitals, Web vitals, obiskano 29. 7. 2024.

[29] https://web.dev/articles/lcp, Largest Contentful Paint (LCP), obiskano 29. 7. 2024.

[30] https://web.dev/articles/inp, Interaction to Next Paint (INP), obiskano 29. 7. 2024.

[31] https://www.gartner.com/en/software-engineering/topics/developer-experience, Developer Experience as a Key Driver of Productivity, obiskano 29. 7. 2024.

[32] N. Rao, C. Bansal, T. Zimmermann, A. H. Awadallah and N. Nagappan, "Analyzing Web Search Behavior for Software Engineering Tasks," 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, str.. 768-777.

[33] https://survey.stackoverflow.co/, Stack Overflow Annual Developer Survey, obiskano 29. 7. 2024.

[34] https://stateofjs.com/en-US, State of JavaScript, obiskano 29. 7. 2024.

Izzivi pri prenovi spletne aplikacije za iskanje knjižničnega gradiva

Andrej Krajnc, Vojko Ambrožič, Gregor Štefanič, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija
andrej.krajnc@izum.si, vojko.ambrozic@izum.si, gregor.stefanic@izum.si, bojan.stok@izum.si

Zasnova obstoječe aplikacije za iskanje gradiva v knjižnicah, imenovane COBISS+, je stara že 10 let. Lani smo se odločili za arhitekturno prenovo aplikacije. Glavna arhitekturna sprememba je, da namesto JSF, jQuery in JavaScript za gradnjo uporabniškega vmesnika uporabljamo ogrodji React in Next.js, na strežniški strani pa smo zamenjali Java EE 8 z Jakarta EE 10. V prispevku smo opisali izzive pri prehodu iz JSF/JavaScript na React/Next.js ter migraciji iz Java EE na Jakarta EE. Največ pozornosti smo namenili temu, kako smo v novi generaciji aplikacije reševali probleme glede robustnosti aplikacije, da se lažje odzovemo v primeru povečanega števila zahtev ali napada in v primeru, da mikrostoritve, ki jih aplikacija uporablja, niso dostopne, so preobremenjene, ali pa ne delujejo pravilno. V ta namen smo uporabili knjižnico Fault Tolerance, ki je del specifikacije MicroProfile. Predstavili smo primere uporabe razredov Timeout, Bulkhead, Retry, CircuitBraker in Fallback. Predstavili smo tudi praktično uporabo knjižnice OpenTracing, ki je del specifikacije MicroProfile in nam omogoča lažje sledenje med klici različnih mikrostoritev.

Ključne besede:
COBISS
COBISS+
Next.js
spletne aplikacije
mikrostoritve

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.21

1 Uvod

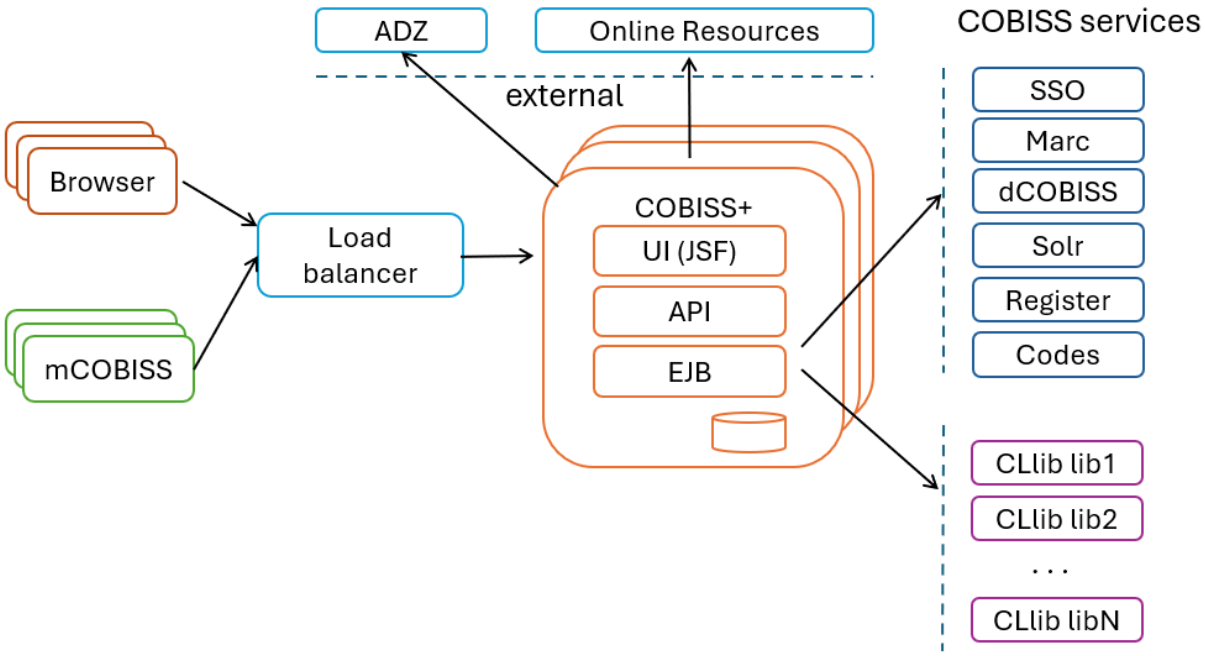
IZUM razvija sistem COBISS, v katerega je vključenih skoraj 1500 knjižnic. Sistem COBISS sestavljajo številne aplikacije, širši javnosti je najbolj poznana aplikacija COBISS+ [1]. COBISS+ je aplikacija za iskanje po knjižničnem gradivu slovenskih knjižnic in tudi knjižnic v državah jugovzhodne Evrope. Omogoča tako rezervacijo gradiva kot tudi izposojo e-knjig, pregled javno dostopnih člankov in dostop do gradiva tujih baz glede na uporabnika oz. pravila njegove knjižnice. Aplikacija COBISS+ deluje kot nekakšen integrator različnih spletnih storitev, ki jih sistem COBISS ponuja za knjižnice in so hkrati zanimive za širšo javnost.

Obstoječa aplikacija COBISS+ je bila razvita pred več kot 10 leti. V tem času smo jo dopolnjevali tako tehnološko (npr. prehod na Linux) kot tudi vsebinsko (nove funkcionalnosti ipd.). Obstoječa aplikacija COBISS+ je bila narejena v tehnologiji Java EE z uporabo ogrodja Spring. Na strežniku je zagnana v več instancah, preko uporabe metrik imamo dober vpogled v delovanje aplikacije. Odločili smo se za prenovo aplikacije, vendar velja poudariti, da obstoječa aplikacija dobro deluje, je odzivna in služi svojemu namenu.

Prvi problem obstoječe aplikacije je bil razvoj uporabniškega vmesnika. Obstoječa aplikacija za izris uporabniškega vmesnika uporablja JSF, za dodajanje interaktivnosti pa se je uporabljala velika količina knjižnic JavaScript, kot npr. jQuery in TomSelect. Zaradi tega je razvoj modernih funkcionalnosti v uporabniškem vmesniku zahteven, otežena je tudi uporaba novejših orodij in knjižnic JavaScript. Odločili smo se za ločitev razvoja aplikacije na uporabniški vmesnik in zaledje, pri tem pa smo uporabili obstoječo zaledno kodo Java EE.

Drugi problem je bila sama robustnost sistema. Število uporabniških zahtev je veliko (tudi do 1000 na sekundo) in sistem je praviloma zelo odziven, saj je zahteva v povprečju obdelana v 20 ms. Toda včasih se pojavijo problemi pri odvisnih servisih, ki so lahko nedosegljivi oz. preobremenjeni in imajo zato predolge odzivne čase. To smo poskušali omejiti z uporaba timeout-ov in predpomnilnika, toda sama rešitev ni bila celovita oz. konsistentna po celotnem sistemu. Zgodila se je tudi občasna preobremenjenost sistema zaradi prevelikega števila zahtev iz celega sveta (Kitajska, Rusija …).

Na sliki 1 je groba arhitektura obstoječe aplikacije COBISS+.

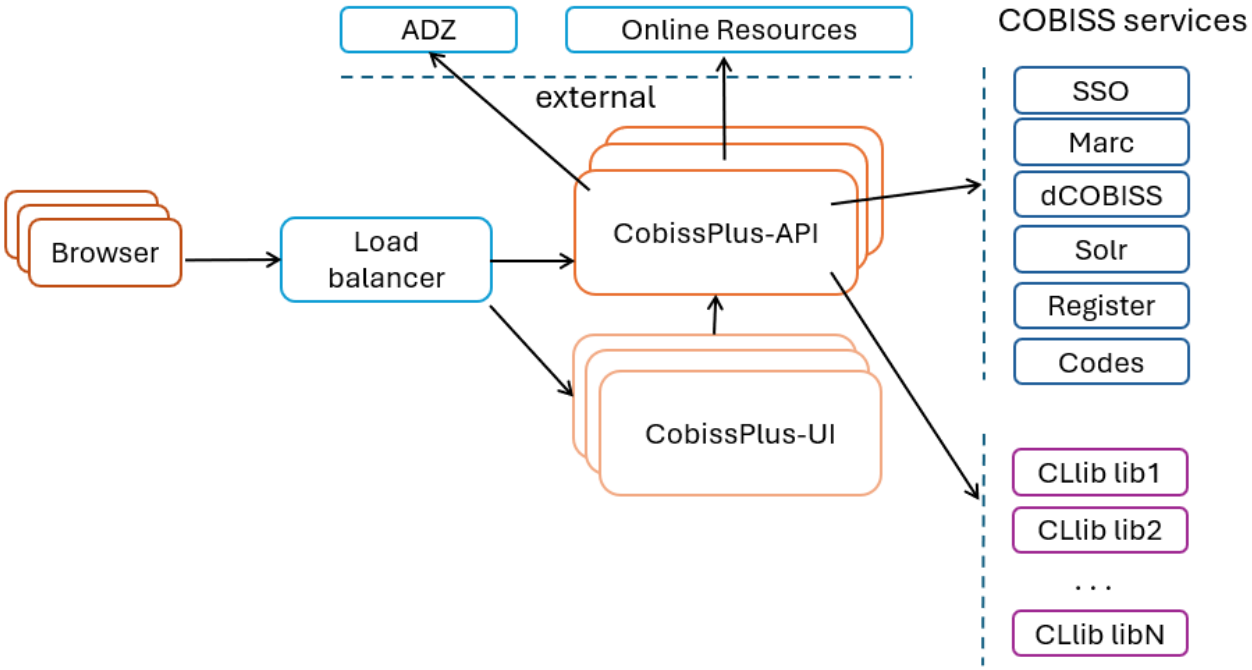


Slika 1: Arhitektura obstoječe aplikacije COBISS+.

2 Nova arhitektura COBISS+

V okviru prenove smo aplikacijo COBISS+ razdelili na dva dela. Prvi del je zaledna aplikacija CobissPlus-API, ki je napisana v Javi in teče na aplikacijskem strežniku WildFly ter skrbi za celotno poslovno logiko ter povezovanje različnih mikroservisov v enoten servis, ki je dostopen prek vmesnika REST. Drugi del aplikacije je CobissPlus-UI, ki je odjemalska aplikacija, napisana v TypeScript z uporabo ogrodja Next.js [2]. Vse interakcije med uporabnikom in aplikacijo potekajo na relaciji brskalnik – CobissPlus-UI, ta pa do podatkov dostopa prek vmesnika CobissPlus-API. Strežnik CobissPlus-UI se neposredno povezuje na CobissPlus-API za pridobivanje podatkov, ki so skupni za vse uporabnike aplikacije (npr. konfiguracija strežnika, seznam knjižnic, podatki o posameznem gradivu). Na strežniški strani smo Java EE 8 zamenjali z Jakarta EE 10.

Na sliki 2 je prikazana poenostavljena arhitektura delovanja aplikacije COBISS+.



Slika 2: Nova arhitektura aplikacije COBISS+.

3 Uporabniški vmesnik CobissPlus-UI

CobissPlus-UI je aplikacija, ki izrisuje uporabniški vmesnik. Za izris smo uporabili ogrodje Next.js, ki v ozadju uporablja knjižnico React, omogoča pa tudi izris HTML na strežniku. Podatki, ki so za vse uporabnike enaki, se pridobijo na strežniku z direktnim klicem na CobissPlus-API, preostali podatki pa se pridobivajo s klici iz odjemalca (brskalnika) na CobissPlus-API. V aplikaciji smo za usmerjevalnik (ang. router) uporabili Next.js app router, za upravljanje z asinhrono pridobljenimi podatki pa smo uporabili knjižnico Tanstack Query. Uporabniški vmesnik aplikacije je izdelan z ogrodjem Tailwind CSS, generične komponente pa temeljijo na zbirki komponent shadcn/ui. Implementacije teh komponent smo vključili v projekt že na začetku razvoja aplikacije, med razvojem pa smo jih še spreminjali in dopolnjevali.

Za lažji razvoj in sinhronizacijo med CobissPlus-UI in CobissPlus-API smo vse končne točke (ang. end point) na zalednem delu aplikacije opremili z anotacijami Swagger, kar nam omogoča avtomatsko izdelavo specifikacije OpenAPI 3.1 za celotno zaledno aplikacijo. V CobissPlus-UI na podlagi tega z uporabo knjižnice OpenAPI TypeScript generiramo definicije tipov TypeScript.

4 CobissPlus-API

CobissPlus-API omogoča dostop do skoraj 1500 knjižnic (servisi CLib). Prek teh servisov so omogočeni pregled knjižničnega gradiva, rezervacije gradiva, izposoja e-knjig itd. Servisi so dostopni prek lastnega protokola, ki je ovit znotraj HTTPS-ja. Združuje različne mikroservise, in sicer:

- Za vsako knjižnico teče aplikativni strežnik **CLib.**
- Storitev **Marc** omogoča tudi dostop do bibliografskih, normativnih in drugih vrst zapisov iz vzajemne baze in lokalnih baz knjižnic.
- Aplikacija **dCOBISS** je digitalni repozitorij, ki vsebuje polna besedila, slike, videoposnetke in druge digitalne vsebine.
- Jedro sistema je odprtokodni iskalnik **Apache Solr**, v katerem so indeksirane vse bibliografske in druge baze, ki so shranjene v MARC-obliki.
- Storitev **Register** omogoča dostop do metapodatkov same infrastrukture, storitev, podatkov o knjižnicah itd.
- Storitev **Codes** omogoča prevajanje sporočil v različne jezike in šifrante.
- **Online resources** so zunanji viri, kot so e-knjige, zvočne knjige, video posnetki.
- **ADZ (Akademska digitalna zbirka Slovenije)** omogoča integrirano iskanje podatkov po več milijonih različnih elektronskih virov svetovnih založnikov, digitalnih repozitorijev in tiskanih virov, ki so naročeni v slovenskih knjižnicah. Omogočen je dostop do celotnih besedil vsebin v odprtem dostopu, kot tudi licenčnih vsebin, za katere se zahteva avtoriziran dostop na osnovi IP-naslova institucije ali prijave AAI.
- Servis **SSO** omogoča enotno prijavo med vsemi servisi ter omogoča tudi avtentikacijo Shibboleth.

5 Robustnost aplikacije COBISS+

Problem pri takšni arhitekturi je, da lahko servis CobissPlus-API hitro postane nestabilen oz. performančno problematičen zaradi morebitnih problemov na odvisnih mikroservisih. Tudi sam vmesnik je lahko preobremenjen zaradi prevelikega števila zahtev uporabnikov oz. raznih zlonamernih botov. Dodaten problem je iskanje problemov na tako razvejani arhitekturi. Težavo smo poskušali rešiti oz. omiliti z uporabo ogrodja MicroProfile, ki je vgrajeno v aplikacijski strežnik WildFly.

5.1. Ogrodje MicroProfile

Java MicroProfile [3] je odprtokodna iniciativa, namenjena razvoju in izboljšavi mikroservisne arhitekture v okolju Java. Ponuja sklop specifikacij, ki omogočajo enostavno in učinkovito razvijanje ter upravljanje mikroservisov:

- **MicroProfile Config** omogoča dinamično konfiguriranje aplikacij brez ponovnega zagona. Konfiguracijske nastavitve so lahko v različnih podatkovnih virih.
- **MicroProfile Health** omogoča spremljanje stanja mikroservisov, kar olajša odkrivanje in odpravljanje težav.
- **MicroProfile Metrics** zagotavlja zbiranje in objavo podatkov o zmogljivosti in obnašanju aplikacij, kar pomaga pri spremljanju in optimizaciji delovanja.

- **MicroProfile JWT Propagation** omogoča varen dostop do mikroservisov z uporabo žetonov za preverjanje identitete in avtorizacijo.
- **MicroProfile Fault Tolerance** omogoča razvoj zanesljivih aplikacij z uporabo tehnik, kot so ponovni poskusi, časovni izklopi, izolacija krogotokov in razmiki med ponovnimi poskusi.
- **MicroProfile OpenAPI** omogoča generiranje in dokumentiranje API-jev, kar olajša integracijo in uporabo mikroservisov.
- **MicroProfile OpenTracing** omogoča sledljivost klicev znotraj mikroservisov in med njimi, kar pomaga pri diagnostiki in optimizaciji delovanja.
- **MicroProfile REST Client** omogoča enostavno klicanje storitev RESTful znotraj mikroservisov.

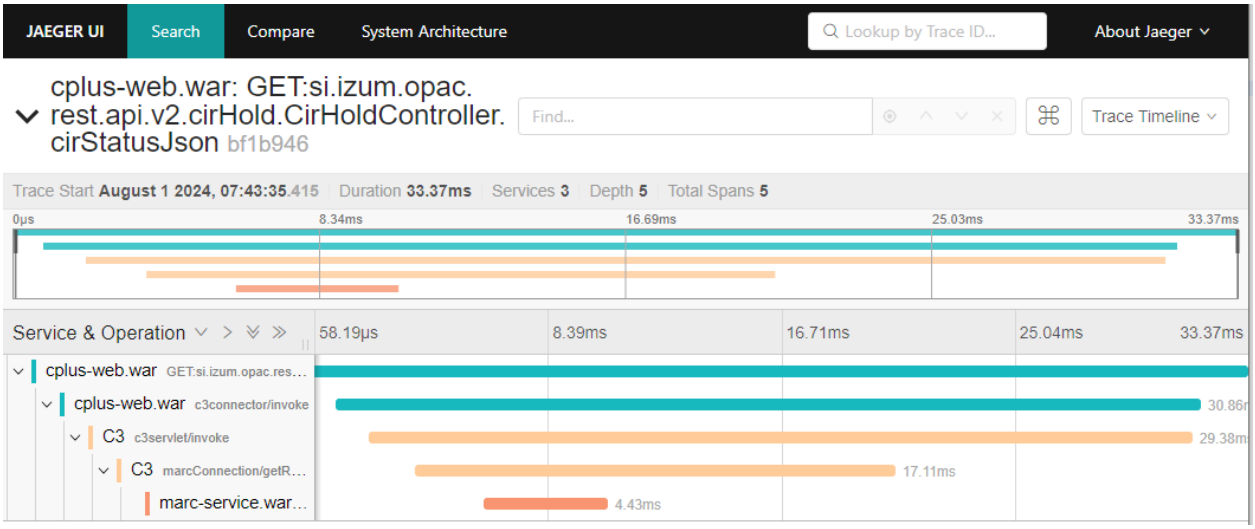
Nekatere od teh specifikacij smo v aplikaciji COBISS+ implementirali oz. nadgradili z lastnimi rešitvami. Rešitve MicroProfile kombiniramo z našimi lastnimi rešitvami za spremljanje in upravljanje infrastrukture COBISS (Monitor manager).

Za opozarjanje na napake oz. probleme uporabljamo odprtokodni sistem Icinga, ki omogoča spremljanje razpoložljivosti in zmogljivosti omrežnih storitev, strežnikov, aplikacij in drugih virov. Periodično se izvaja zdravstveni pregled (ang. Health Check) storitve COBISS+. Izvajamo sistemske in poslovne teste.

Sistemske in poslovne metrične podatke (ang. Metrics) zbiramo z odprtokodnim sistemom Prometheus, ki je bil zasnovan za monitoring in opozarjanje. Prikaz teh podatkov nam omogoča odprtokodno orodje Grafana, ki je namenjeno analitiki in vizualizaciji teh podatkov. Prometheus nam lahko prek Alert Managerja pošilja opozorila o morebitnih problemih aplikacije.

Specifikacija MicroProfile OpenTracing je implementirana z uporabo odprtokodnega programa Jaeger. Program je namenjen sledenju zahtevkov v porazdeljenih sistemih, kar razvijalcem omogoča analizo in odpravljanje težav v mikroservisni arhitekturi. Uporablja se za spremljanje in diagnosticiranje učinkovitosti ter zaznavanje napak v kompleksnih aplikacijah. Podatki se črpajo iz podatkovne baze Elasticsearch. To je zmogljiv in skalabilen odprtokodni iskalnik, ki omogoča shranjevanje, iskanje in analizo velikih količin podatkov v skoraj realnem času.

Na sliki 3 sta prikazana grafični vmesnik Jaeger in primer klica metode na servis CLib in preko tega na servis Marc. Klici metod na različnih strežnikih so prikazani v različnih barvah.



Slika 3: Prikaz klica metode čez različne servise.

V tem prispevku smo se osredotočili predvsem na implementacijo specifikacije MicroProfile Fault Tolerance (Toleranca napak).

5.2. MicroProfile Fault Tolerance

Specifikacija Fault Tolerance [4] zagotavlja nabor programskih modelov in vzorcev, ki razvijalcem pomagajo graditi odporne mikrostoritve, ki lahko obvladujejo napake na eleganten način. Uporablja anotacije za poenostavitev implementacije teh vzorcev. Specifikacija vključuje naslednje funkcije:

- **Timeout**: Funkcija omogoča nastavitev časovne omejitve za izvajanje metode. Če metoda traja dlje od določenega časa, se prekine in vrže TimeoutException.

```
@Timeout(500)
public String doSomething() {
// implementacija metode
}
```

- **Retry**: Funkcija omogoča samodejno ponavljanje izvajanja metode, ko ta ne uspe. Razvijalci lahko določijo število ponovitev, zamik med ponovitvami in pogoje, pod katerimi naj se ponovitve izvedejo.

```
@Retry(maxRetries = 3, delay = 1000)
public String callService() {
// implementacija metode
}
```

- **Circuit Breaker** (varovalka): Vzorec preprečuje storitvi, da bi večkrat poskušala poklicati neuspešno operacijo. Ko doseže določen prag napak, se odklopnik odpre in nadaljnji klici takoj ne uspejo brez poskusa operacije. Po določenem zamiku se lahko odklopnik premakne v pol-odprto stanje, da preizkusi, ali je operacija ponovno na voljo.

```
@CircuitBreaker(requestVolumeThreshold = 4, failureRatio = 0.75, delay = 1000)
public String getData() {
// implementacija metode
}
```

- **Bulkhead** (pregrada): Vzorec omejuje število sočasnih klicev metode, s čimer zagotavlja, da preobremenjeni del aplikacije ne blokira celotnega sistema.

```
@Bulkhead(value = 5)
public String processRequest() {
// implementacija metode
}
```

- **Fallback**: Funkcija omogoča definiranje alternativne metode, ki se izvede, ko glavna metoda ne uspe.

```
@Fallback(fallbackMethod = "fallback")
public String fetchResource() {
// implementacija metode
}
public String fallback() {
return "rezervni odgovor";
}
```

Vse te mehanizme smo vključili v klice naših storitev REST. Pri tem smo morali biti pozorni tudi na nastavitve aplikacijskega strežnika WildFly, saj ima lastne nastavitve za maksimalno število EJB-zrn.

```
<bean-instance-pools>
<strict-max-pool name="mdb-strict-max-pool" derive-size="from-cpu-count" instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
<strict-max-pool name="slsb-strict-max-pool" max-pool-size="50" instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
</bean-instance-pools>
```

Za klic do servisov naših knjižnic (CLib), ki ne poteka po protokolu REST, temveč prek našega lastnega protokola, nismo uporabili mehanizma MicroProfile Fault Tolerance, temveč smo razvili lasten mehanizem, ki deluje na podoben način. Pri tem smo izkoristili možnost specifikacije EJB, ki omogoča klic prestreznikov (ang. interceptor):

```
@AroundInvoke
public Object invoke(final InvocationContext context) throws Exception {
  String className = context.getTarget().getClass().getSimpleName();
  String methodName = context.getMethod().getName();
```

Ko sta MicroProfile Fault Tolerance in MicroProfile Metrics [5] uporabljena skupaj, se metrike samodejno dodajo za vsako metodo, ki je označena z anotacijo @Retry, @Timeout, @CircuitBreaker, @Bulkhead ali @Fallback. Imena za samodejno dodane metrike sledijo doslednemu vzorcu, ki vključuje polno kvalificirano ime označene metode.

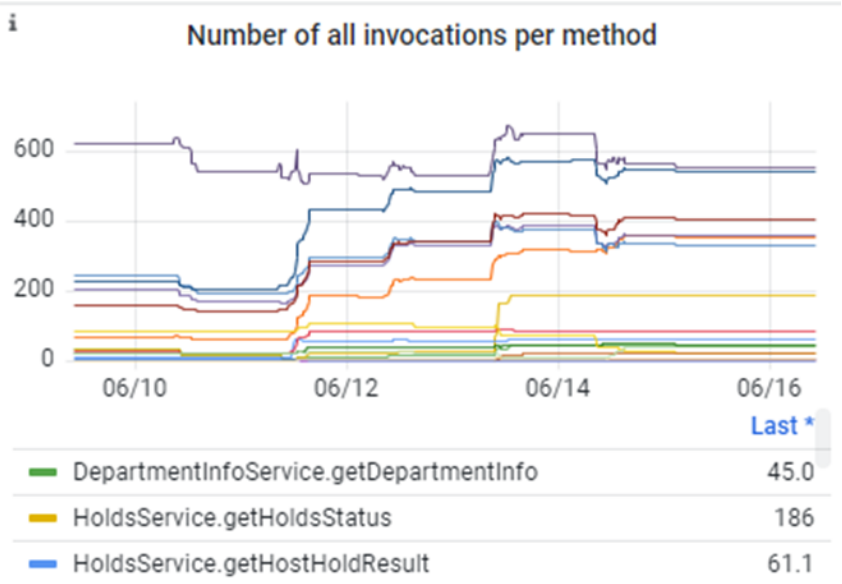
Primer metode in avtomatsko kreiranih metrik:

```
package com.exmaple;
@Timeout(1000)
public class MyClass {
@Retry
public void doWork() {
// work
}
}
```

Avtomatsko kreirane metrike:

```
ft.com.example.MyClass.doWork.invocations.total
ft.com.example.MyClass.doWork.invocations.failed
ft.com.example.MyClass.doWork.retry.callsSucceededNotRetried.total
ft.com.example.MyClass.doWork.retry.callsSucceededRetried.total
ft.com.example.MyClass.doWork.retry.callsFailed.total
ft.com.example.MyClass.doWork.retry.retries.total
ft.com.example.MyClass.doWork.timeout.executionDuration
ft.com.example.MyClass.doWork.timeout.callsTimedOut.total
ft.com.example.MyClass.doWork.timeout.callsNotTimedOut.total
```

Na osnovi teh metrik lahko s pomočjo Prometheusa in Grafane spremljamo odzivnost naše aplikacije oz. morebitne težave v njej. Slika 4 predstavlja primer prikaza matrik.



Slika 4: Prikaz metrik v orodju Grafana.

6 Zaključek

Pri prenovi spletne aplikacije za iskanje knjižničnega gradiva smo želeli predvsem prenoviti uporabniški vmesnik in izboljšati robustnost aplikacije.

Izkušnje pri razvoju uporabniškega vmesnika ločeno od zaledja so pozitivne, predvsem zaradi hitrosti iteriranja. Uporaba ogrodja Next.js pri izdelavi CobissPlus-UI nam je poenostavila razvoj interaktivne spletne aplikacije, težave pa so se pojavljale predvsem pri uporabi Next.js App Router, ki po naših izkušnjah še nima vseh funkcionalnosti, ki bi si jih pri razvoju takšne aplikacije želeli. Zaradi tega moramo včasih najti alternativne poti do rešitve.

Ogrodje MicroProfile Fault Tolerance lahko bistveno pripomore k boljši robustnosti aplikacij, zato nameravamo to ogrodje v prihodnje uporabiti tudi pri drugih aplikacijah v okviru sistema COBISS.

7 Literatura

[1] COBISS+, https://plus.cobiss.net/cobiss/si/sl/bib/search, obiskano 1.8.2024

[2] Next.js, https://nextjs.org/, obiskano 1.8.2024

[3] MicroProfile, https://microprofile.io/, obiskano 1.8.2024

[4] MicroProfile Fault Tolerance, https://download.eclipse.org/microprofile/microprofile-fault-tolerance-4.0/microprofile-fault-tolerance-spec-4.0.html, obiskano 1.8.2024

[5] MicroProfile Metrics, https://microprofile.io/specifications/microprofile-metrics/, obiskano 1.8.2024

OTS 2024 Sodobne informacijske tehnologije in storitve

Zbornik sedemindvajsete 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 sedemindvajsete konference OTS 2024 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 zagotavljanje ustreznega skalabilnega okolja zanje ter avtomatizacijo testiranja, merjenje kakovosti in dostavo s proaktivnim naslavljanjem najpogostejših kibernetskih napadov. Rdečo nit prispevkov predstavljajo podatkovne tehnologije, ki so zastopane v obliki klasičnih podatkovnih baz, podatkovnih jezer ter učinkovitega zbiranja, obdelave in vizualizacije velepodatkov. Prispevki tako še naprej omogočajo boljšo povezanost IT strokovnjakov, informatikov, arhitektov in razvijalcev IT rešitev in storitev, kot tudi 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

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

bintegra
Svetovanje in IT rešitve

COMTRADE GAMING

dem
Dravske elektrarne Maribor
Skupina hse

impol
Aluminium Industry

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.
INFORMATIKA

MIKROpis

msg
PLAUT

nChain

niceHash

NOVUMRGI

Protokol

RAZVOJNI CENTER
IKTS
ŽALEC

RC IRC Celje, d.o.o. / Ltd.

Red Hat

SOFTWARE SYNERGY
softergee

tridens
technology

SAVA
ZAVAROVALNICA

MEDIJSKI POKROVITELJ

Računalniške novice
www.racunalniske-novice.com


ISO 27001, 14001, 9001
Smo INOVA IT
Web & Mobile | IoT | AI | VR | Digital Twins
Ste pripravljeni preseči običajno?
V svet inovacij in prepleta modernih tehnologij ter kreativnosti vabimo vse, ki ste željni izzivov . Sodelujemo z vodilnimi tehnološkimi imeni in skupaj oblikujemo rešitve, ki krojijo prihodnost. Pridružite se našemu dinamičnemu timu, raziskujte neomejene možnosti in prispevajte k tehnološkim inovacijam, ki spreminjajo svet.
APPLY NOW
careers@inova.si
inova.si/careers/

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

Visit our career page:

comtradegaming.com

Follow us

Dravske elektrarne Maribor

Največji proizvajalec električne energije iz obnovljivih virov

Z osmimi velikimi hidroelektrarnami na reki Dravi, s petimi malimi hidroelektrarnami ter petimi sončnimi elektrarnami družba Dravske elektrarne Maribor, d. o. o. proizvede skoraj četrtino vse slovenske električne energije. Povprečna letna proizvodnja družbe, ki znaša 2.800 GWh, predstavlja 80 odstotkov slovenske električne energije, ki ustreza kriterijem obnovljivih virov in standardom mednarodno priznanega certifikata RECS (Renewable Energy Certificates System). Skupna moč na pragu elektrarn družbe je skoraj 600 MW. Učinkovitost, zanesljivost, prilagodljivost, celovitost ter okoljska in družbena odgovornost so temeljne vrednote družbe, ki jim sledimo pri obstoječih zmogljivostih in tistih, ki jih še nameravamo zgraditi.

dem

Dravske elektrarne Maribor

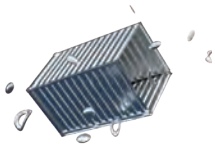
Skupina hse

Z našo novo
blagovno znamko
razvijamo zeleno
prihodnost
v industriji aluminija!

impol Aluminium Industry | InfiniAL

Farmacevtska industrija

Prehrambena industrija



Transport

Elektro-industrija

Avtomobilska industrija

Obnovljivi viri

Potrošniške dobrine

Gradbeništvo

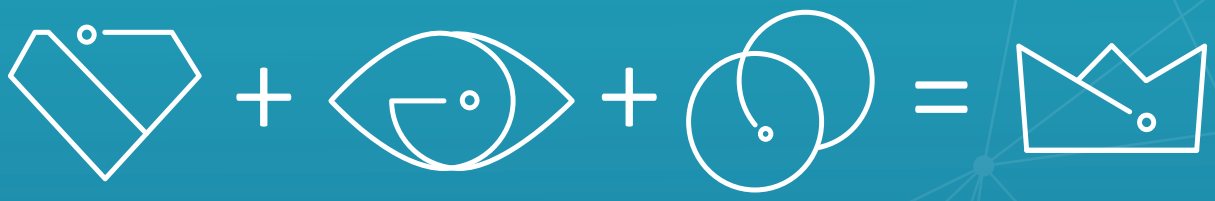
Letalska in vesoljska industrija

Slovenska Bistrica | Slovenija | www.impol.si

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.

INFORMATIKA

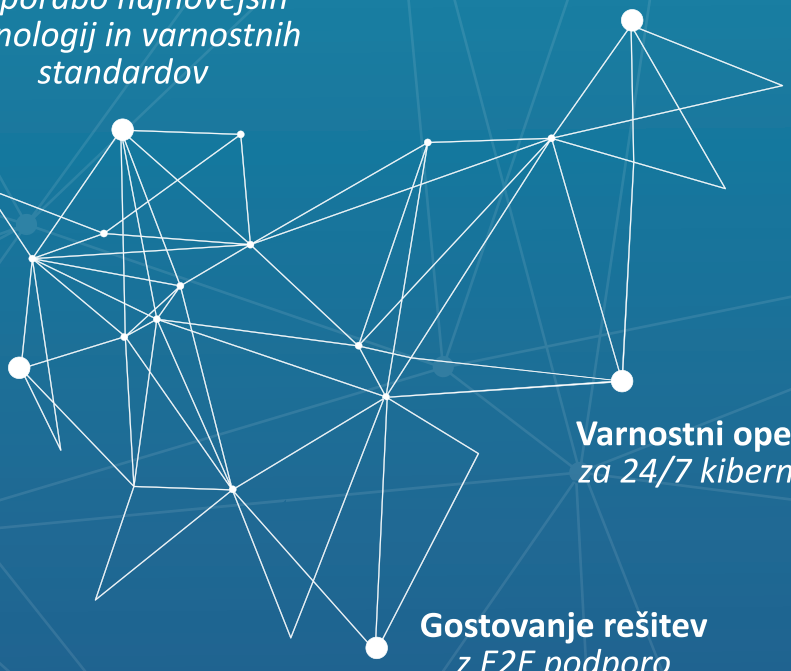
pravi partner za vas



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

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 vsebnikov Docker
- Digitalizacija procesov
- Orodje in specializiranost za migracijo podatkov
- Celostna DevOps storitev

msg.UAP - temeljna rešitev v oblaku za digitalno poslovno prihodnost

- DIGITALIZACIJA ŽIVLJENJSKEGA CIKLA
- OBLAČNA REŠITEV
- ODPRT API
- VISOKE ZMOŽNOSTI AVTOMATIZACIJE
- SAAS ALI NA LOKACIJI
- PREPROSTO IN STROŠKOVNO UČINKOVITO UPRAVLJANJE

+386 2 2356200 www.msg-plaut-uap.com andrej.kline@msg-plaut.com

nChain

5 office locations

Headquarters in Zug, Switzerland, with offices in London, United Kingdom, Maribor and Ljubljana, Slovenia and Manila, Philippines

260+ employees

A multidisciplinary team with one of the largest dedicated teams of blockchain developers

180+ engineers

Developing products your business can rely upon

1000+ research papers

A robust research program covering blockchain technology and applications across industries

#6 ranked globally

For the number of active patents related to blockchain technology

Who we are

nChain is a global leader in blockchain technology, providing products, solutions and IP licensing services.

Making the world-changing potential of blockchain technology accessible to individuals, businesses, and governments.

www.nchain.com

NiceHash prvič predstavlja:

Bitcoin konferenca v Sloveniji!

8.–9. NOVEMBER 2024

Maribor
Hotel Habakuk

Saifedean Ammous
Avtor uspešnice: "Bitcoin standard"

Princ Filip Karađorđević
Vodja strategije pri JAN3

Christian Rau
MasterCard – višji podpredsednik Crypto and Fintech EU

Dr. Timotej Jagrič, CQRM
EPF - predstojnik Inštituta za finance in umetno inteligenco

Konferenčne teme:

- **rudarjenje**
- **energetska učinkovitost**
- **plačila z bitcoini**
- **varnost podatkov**
- **menjalnice za kriptovalute**
- **regulacija**
- **množična uporaba**

Zakaj nas obiskati?

- **100+ vrhunskih strokovnjakov**
- **razstavišče**
- **3 konferenčni odri**
- **poslovna mreženja**
- **plačila z bitcoini**

VSTOPNICE
možnost nakupa z **bitcoini**

NiceHash, vodilni ponudnik platforme za rudarjenje kriptovalut, vas vabi na NiceHashX, mednarodno konferenco posvečeno Bitcoinu. Pridružite se nam na tem edinstvenem dogodku! Za več informacij obiščite **nicehashx.com**.

NiceHash | NiceHash PAY | Habakuk Hotel superior | Maribor

NOVUMRGI

RGI Group

European leader in the **digital transformation** of the insurance sector

Standard **software** and **cloud solutions** for the **international insurance market**

Protokol

Web3 Solutions: Simplified

From Blockchain to NFTs, Digital Product Passports, Smart Contracts and more — Protokol are a specialist web3 development and consulting partner that deliver custom solutions, products and dApps to market.

Find out more at protokol.com

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.

Programske rešitve

Razvoj programske opreme

Vzdrževanje in svetovanje

Informacijska varnost

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

tridens
technology

Tridens je mariborsko IT podjetje, ki je v digitalnem prostoru prisotno že več kot 15 let.

V podjetju se ukvarjamo z razvojem lastnih produktov:

Naša vizija je biti prepoznani kot zaupanja vreden partner za inovativne programske rešitve po celem svetu.

Zaupajo nam:

BYD · TelekomSlovenije · cisco · iskraemeco BY ELSEWEDY ELECTRIC · telemach

Zaposlujemo!

Poglej prosta delovna mesta na naši spletni strani:
https://tridenstechnology.com/careers/
ali poskeniraj QR kodo.


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

Vodilni IKT medij

Računalniške novice

www.racunalniske-novice.com

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

bintegra
Svetovanje in IT rešitve

COMTRADE GAMING

dem
Dravske elektrarne Maribor
Skupina hse

impol
Aluminium Industry

INFORMATIKA

MIKROpis

msg PLAUT

nChain

niceHash

NOVUMRGI

Protokol

RAZVOJNI CENTER IKTS ŽALEC

RC IRC Celje, d.o.o. / Ltd.

Red Hat

SOFTWARE SYNERGY
softergee

tridens
technology

SAVA
ZAVAROVALNICA

MEDIJSKI POKROVITELJ

Računalniške novice
www.racunalniske-novice.com

IoT
Varnost
Kriptografija
Web 3
Podatkovne tehnologije
Umetna inteligenca
Poslovna analitika

INŠTITUT ZA INFORMATIKO

ISBN 978-961-286-893-2

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko