Univerzitetna založba
Univerze v Mariboru

25. konferenca sodobne informacijske tehnologije in storitve

ITS 2022

Urednika:
Marjan Heričko,
Tina Beranič

Zbornik prispevkov

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

OTS 2022

Sodobne informacijske tehnologije in storitve

Zbornik petindvajsete konference, Maribor, 7. in 8. september 2022

Urednika
Marjan Heričko
Tina Beranič

September 2022

Naslov **OTS 2022 Sodobne informacijske tehnologije in storitve**

Podnaslov **Zbornik petindvajsete konference, Maribor, 7. in 8. september 2022**

Title OTS 2022 Advanced Information Technologies and Services

Subtitle Conference Proceedings of the Twenty-fifth Conference, Maribor, September 7th & 8th, 2022

Urednika *Editors*
Marjan Heričko
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Tina Beranič
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Tehnična urednika *Technical editors*
Saša Brdnik
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Jan Perša

Oblikovanje ovitka *Cover designer*
Saša Brdnik
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Grafika na ovitku *Cover graphics* Brdnik, 2022

Grafične priloge *Graphics material* Avtorji prispevkov in pokrovitelji, Brdnik in Kuhar, 2022

Konferenca *Conference* OTS 2022 Sodobne informacijske tehnologije in storitve

Kraj in datum *Location and date* Maribor, 7. in 8. september 2022

Programski odbor *Program committee* Marjan Heričko (predsednik konference, vodja), Tomaž Domajnko, Boštjan Grašič, Tina Beranič, Boštjan Kežmah, Dean Korošec, Luka Pavlič, Boštjan Šumak, Muhamed Turkanović, Bojan Štok, Ivan Lah, Milan Gabor.

Organizacijski odbor *Organizing committee* Tina Beranič (vodja), Katja Kous, Boris Lahovnik, Saša Brdnik, Maja Pušnik, Miha Strehar, Lucija Brezočnik, Saša Kuhar, Alen Rajšp, Tilen Hliš, Viktor Taneski, Patrik Rek.

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, 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

Dostopno na *Availabe at* https://press.um.si/index.php/ump/catalog/book/688

Izid *Published* Maribor, september 2022

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.946.5:004.7(082)(0.034.2)

SODOBNE informacijske tehnologije in storitve (konferenca) (25 ; 2022 ; Maribor)
OTS 2022 [Elektronski vir] : sodobne informacijske tehnologije in storitve : zbornik petindvajsete konference, Maribor, 7. in 8. september 2022 / urednika Marjan Heričko, Tina Beranič. - 1. izd. - E-zbornik. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2022

Način dostopa (URL): https://press.um.si/index.php/ump/catalog/book/688
ISBN 978-961-286-638-9 (PDF)
doi: 10.18690/um.feri.10.2022
COBISS.SI-ID 118638339

ISBN 978-961-286-638-9 (pdf)
978-961-286-639-6 (mehka vezava)
DOI https://doi.org/10.18690/um.feri.10.2022
Cena Brezplačni izvod
Price
Odgovorna oseba založnika red. prof. dr. Zdravko Kačič, rektor Univerze v Mariboru
For publisher

Citiranje Heričko, M., Beranič, T. (ur.). (2022). *OTS 2022 Sodobne informacijske tehnologije in storitve: Zbornik petindvajsete konference, Maribor, 7. in 8. september 2022*. Maribor: Univerzitetna založba. doi: 10.18690/um.feri.10.2022
Attribution

https://www.ots.si

Spoštovane, spoštovani,

strokovna konferenca OTS že petindvajset let uresničuje svoje poslanstvo in zagotavlja platformo za izmenjavo praktičnih izkušenj in spoznanj glede uspešne vpeljave in inovativne uporabe sodobnih tehnologij ter pristopov k razvoju informacijskih rešitev. Le to omogoča napredek na vseh področjih poslovanja organizacij ter delovanja in življenja posameznikov v že skoraj pretirano digitalizirani družbi.

Veseli nas, da že četrt stoletja prispevamo k boljši povezanosti informatikov, arhitektov in razvijalcev naprednih informacijskih storitev in rešitev, računalničarjev in IT strokovnjakov kot tudi akademske sfere in gospodarstva. Ponosni smo na izjemne uspehe sodelujočih podjetij v mednarodnem in domačem prostoru, kar dokazuje izjemno strokovnost in visok nivo znanja, ki ga premoremo v naši regiji, pa tudi sposobnost, da se odlične tehnološke rešitve uspešno vpeljejo in udejanjijo kot dejanska dodana vrednost za končne uporabnike.

Zbornik konference OTS 2022 vključuje prispevke uveljavljenih domačih strokovnjakov, ki predstavljajo svoje dolgoletne izkušnje in, skozi konkretne projekte, dokazano uspešne pristope:

- naslavljanja arhitekturnih izzivov, povezanih z vpeljavo umetne inteligence in strojnega učenja,
- zasnove in vpeljave rešitev poslovne inteligence,
- učinkovite obdelave velepodatkov in podatkovnih tokov,
- apliciranja vzorcev pri zasnovi mikrostoritvenih in brezstrežniških arhitektur,
- uporabe zmožnosti tehnologij veriženja blokov in decentraliziranih aplikacij,
- digitalne preobrazbe in posodobitve IKT rešitev na osnovi funkcionalnosti, ki jih omogočajo digitalne denarnice,
- apliciranja agilnih metod ter izboljšanjem odzivnosti pri zagotavljanju učinkovite podpore poslovnim procesom,
- integracije sistemov in razvoja podpornih storitev pametnega doma, telemedicine in zelenega prehoda,
- razvoja in optimizacije mobilnih, spletnih in oblačnih rešitev ter
- uporabe sodobnih in aktualnih, tudi odprtokodnih, tehnologij pri razvoju naprednih informacijskih rešitev in storitev.

Omenjen širok spekter tematik, obravnavanih na 25. konferenci OTS 2022 Sodobne informacijske tehnologije in storitve, orisuje aktualne izzive, s katerimi se trenutno srečujemo na področju informatike. Za uspešno spopadanje z njimi so potrebni sodobni pristopi ter vpeljava primernih, velikokrat novih, tehnologij. Prav to pa je tisto, kar skozi odlične, s praktičnimi izkušnjami podprte prispevke, v vpogled in kritično ovrednotenje ponujajo avtorji prispevkov v zborniku jubilejne petindvajsete konference OTS.

doc. dr. Tina Beranič
vodja organizacijskega odbora OTS 2022

prof. dr. Marjan Heričko
predsednik 25. konference OTS 2022

KAZALO

25 let razvoja informatike skozi prizmo konference OTS

Luka Pavlič

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

Sinopsis Konferenca OTS (uvodoma »Objektna tehnologija v Sloveniji«, kasneje »Sodobne informacijske tehnologije in storitve«) na nek način predstavlja rdečo nit IT strokovnjakov širše regije že več kot četrt stoletja. Dogodek, kjer strokovnjaki predstavljajo svoje uspešne projekte, izmenjujejo bogate izkušnje iz prakse, identificirajo morebitne prihodnje trende, ter krešejo mnenja o dogajanju v industriji že petindvajsetič, posledično predstavlja pomemben dokument o vsakoletnem stanju informatike v ožji in širši regiji. Konferenca OTS tako skozi svojo zgodovino ponuja ne samo zanimive vpoglede v duh časa, temveč vsakokrat nudi tudi zanimiv in kritičen vpogled v prihodnost področja.

Članek ponuja vpoglede v vsakoletne zbornike, ki preko člankov omogočajo dokaj verodostojno rekonstrukcijo prevladujočih pristopov, orodij, mnenj in trendov skozi čas. V članku skozi analizo zbornikov identificiramo vzpone in padce tehnologij, pristopov in dobrih praks iz industrije. Ugotovili bomo, da smo določene izzive že uspeli rešiti, določeni pa žal ostajajo, čeprav se jih lotevamo vedno znova z drugačnimi pristopi in orodji.

Ključne besede:
informacijske tehnologije
trend razvoja
zgodovinski pregled
trendi

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.1

1 Uvod

Morda se zdi težko razumljivo, a prva konferenca OTS je bila organizirana v času, ko danes vodilne platforme za uporabniške vmesnike informacijski rešitev, pametnih telefonov, niti še ni bilo. Pri Nokii so sicer najavili Communicatorja, ki bi naj bil sposoben poganjati celo neke vrste aplikacij ter bi naj imel dostop do interneta. Skoraj nikomur pa ni bilo prav jasno, kaj bi s tem počeli. Šele ob deseti obletnici konference je bilo slutiti, da se pripravljajo naprave, ki jim danes rečemo »pametni telefoni«. Podjetja, ki ponuja iskalnik Google še ni bilo. Zemljevidi in druge, danes same po sebi umevne oblačne storitve v informacijskih rešitvah so bile izdelane po meri. Vizionarji so nam hiteli razlagati, da bomo v prihodnosti lahko preko interneta izvajali celo plačila.

Operacijski sistem Windows 95 smo po večini nameščali s pomočjo disket, redki celo iz nosilcev CD. Linux jedro smo prevajali pred namestitvijo. Ključ je bil le in zgolj »ključ«, kratica USB je bila neznana. Večslojna arhitektura je po večini preštela le do števila 2. Najnaprednejši arhitekti so zaznali, da bi t.i. vmesni nivo preko standardiziranih posrednikov objektnih zahtev (takrat smo se še prerekali, ali govoriti v objektih ali raje predmetih) utegnil predstavljati preboj. IT oddelki (pravzaprav AOP oddelki – oddelki za avtomatsko obdelavo podatkov) so bili organizirani bistveno drugače kot danes. Besedica »storitev« je svojo domovinsko pravico v svetu IT našla šele nekaj let kasneje.

S pogledom nazaj v tisti prostor in čas je jasno, da je konferenca OTS bila, in je še vedno, eden ključnih kamenčkov v mozaiku preboja informatike v regiji od zgolj navdušenih uporabnikov in sledilcev, do kritičnih strokovnjakov in celo vodilnih ekip v regiji na več IT področjih. Ta članek je zato bil napisan v želji, da preko analize zbornikov 25 let konference ugotovimo, kakšne so bile teme konference v preteklosti. Konferenca OTS predstavlja pomemben vpogled v vsakoletno dogajanje v industriji ter preko tega kritičen indikator trendov informatike v prihodnosti.

1.1. Namen in razvoj konference OTS

Organizatorji prvega strokovnega srečanja »OTS´96 – Objektna tehnologija v Sloveniji« (COT - Center za objektno tehnologijo, UM FERI) v uvodnem nagovoru udeležencem povzemajo namene in cilje srečanja [1]: *»…pospešiti vpeljavo in uporabo objektne tehnologije med slovenskimi informatiki. Z namenom, da razbijemo mit o nezrelosti objektne tehnologije, smo k sodelovanju pritegnili razvijalce informacijskih in programskih rešitev, ki predstavljajo konkretne rezultate uporabe objektnega pristopa. Avtorji podajajo svoje praktične izkušnje pri uvajanju in uporabi objektnih jezikov, objektnih orodij, objektnih podatkovnih baz, objektnih arhitektur, in objektnih razvojnih metodologij.«*

Vsakoletno strokovno srečanje je preraslo v eno najpomembnejših strokovnih konferenc informatikov v regiji. Kljub temu, da je konferenca pripomogla k promociji in ustrezni vpeljavi objektnih pristopov in tehnologij, pa se je izkazalo, da izzivov v informatiki še ne bo zmanjkalo. Ne samo, da je objektna orientacija postala zelo preizkušen pristop, izkazala se je za eno izmed prvih snežnih kep, ki se v npr. komponentnih in (mikro)storitvenih arhitekturah po hribu uspešno kotali in veča še danes. Čeprav je naziv konference zato že zadnjih 15 let »Sodobne tehnologije in storitve«, pa osrednji cilj in poslanstvo, ki ga povzema tudi uvodnik trenutne, ostaja skoraj nespremenjen [25]: *»…platforma za izmenjavo praktičnih izkušenj in spoznanj glede uspešne vpeljave in inovativne uporabe sodobnih tehnologij ter pristopov k razvoju informacijskih rešitev.«* Ter: *»Prispevamo k boljši povezanosti informatikov, arhitektov in razvijalcev naprednih informacijskih storitev in rešitev , računalničarjev in IT strokovnjakov kot tudi akademske sfere in gospodarstva.«*

Sprožitelj prvega strokovnega srečanja ter poslanstvo aktualne strokovne konference torej ostajata:

- osredotočenost na tehnične in organizacijske pristope razvojnih ekip,
- naslavljanje razvojnega cikla IT rešitev (od ustreznega zajema zahtev uporabnikov in poslovnega okolja, preko načrtovanja do implementacije, testiranja in uvedbe rešitev in storitev v poslovanje),
- izmenjava konkretnih (razvojnih) izkušenj iz industrije,

- kritična predstavitev, vrednotenje in demonstracija različnih razvojnih platform in pristopov,
- identifikacija in predstavitev morebitnih prednosti in pasti novosti v industriji,
- na zanimiv in plastičen način prestaviti in promovirati nova spoznanja iz teoretično-akademske sfere in
- usposobiti posameznike in ekipe v različnih tehničnih in organizacijskih veščinah.

Kljub jasni, tehnično-organizacijski razvojno naravnani usmeritvi konference, pa le-ta s prispevki redno ponuja tudi zanimive vsebine, s katerimi se informatiki hote ali nehote srečujejo v podjetjih vsakodnevno:

- izbor, nakup / najem, prilagoditev in vpeljava IT rešitev v poslovanje,
- IT integracije,
- upravljanje in spremljanje IT rešitev in oddelkov,
- optimizacija poslovnih procesov in njihova informatizacija,
- snovanje in uveljavljanje novih poslovnih modelov s pomočjo informatizacije in digitalizacije,
- soustvarjanje in upoštevanje normativnih vidikov informacijskih rešitev in okolij, v katerih delujejo in
- negovanje pristopov, ki identificirajo in naslavljajo t.i. človeški vidik razvoja, vpeljave in uporabe informacijskih rešitev in storitev.

V luči naštetih fokusnih področij, ki ostajajo zadnjega četrt stoletja tako rekoč nespremenjena, zato med udeleženci konference ne najdemo zgolj podjetij, katerih primarna dejavnost je razvoj, vpeljava in upravljanje IT rešitev, temveč tudi podjetja iz drugih domen, ki so v veliki meri odvisna od svojih ali tretjih IT oddelkov. Le-ta primarno skrbijo za nemoteno delovanje informacijskih rešitev v podjetju, njihovo prilagajanje in delno tudi njihovo dograjevanje ter snovanje in razvoj lastnih »in-house« rešitev. Redni udeleženci konference OTS so tudi podjetja, ki so že ugotovila, da informatiki niso le inženirji, ki skrbijo za razvoj in upravljanje informacijskih rešitev, temveč so vedno bolj soustvarjalci novih poslovnih modelov, temelječih na konceptih informatizacije in digitalizacije. Pomembni udeleženci konference OTS so poleg akademske in strokovne javnosti tudi študentje, ki jim organizator ponuja brezplačno udeležbo. To možnost vsako leto izkoristi večje število študentk in študentov ter na takšen način pridobijo bogat vpogled v stanje informatike v regiji in širše, hkrati pa utegnejo vzpostaviti dolgoročne vezi z bodočimi delodajalci ali strankami.

Na konferencah OTS je, poleg vabljenih uglednih govornikov iz tujih inštitucij, do sedaj redno sodelovalo večje število avtorjev iz vsaj 5 različnih slovenskih in tujih univerz. Pomembnejše pa je, da se je v obliki soustvarjalcev vsebine konference do sedaj zvrstilo že več kot 180 domačih in tujih podjetij, katerih zaposleni so pomembno prispevali pri kar 640 strokovnih člankih konference. Skoraj 700 predstavitvam je do sedaj z zanimanjem prisluhnilo več tisoč udeležencev konference. Nekaj zanimivih vpogledov v vključenost podjetij v vsebino konferenc povzema tabela 1.

Tabela 1: Nekaj zanimivejših kazalnikov konference OTS.

Kazalnik – vključenost podjetij v vsebino konferenc OTS	
Vključenost različnih podjetij v vsebino prispevkov na 25 konferencah	180+
Povprečno število različnih podjetij, iz katerih prihajajo avtorji prispevkov konference	18
Število podjetij, katerih zaposleni so s prispevki prisotni na vsaj polovici konferenc	10
Število vseh prispevkov 25 konferenc	640
Povprečno število prispevkov na konferencah	26

Vir: [1-25].

1.2. Metodologija raziskave

Članek želi na kvantitativen, objektiven, podatkovno-usmerjen način postreči z vpogledi v glavne tematike in trende petindvajsetih konferenc OTS. Čeprav rezultati v članku jasno pokažejo, da prispevki v celoti zasledujejo cilje in namen konference, je primarni cilj članka ugotoviti poglavitna področja konference skozi leta ter na tej osnovi pokazati ali so teme konference (bile) zgolj povzetek dogajanja v svetu, skladne z globalnimi trendi, ali celo pred svojim časom.

V ta namen so vsi prispevki konferenc bili ustrezno klasificirani v večje število kategorij, ki so bile združene v več različnih podskupin (od splošnih visokonivojskih kategorij, do konkretnih npr. tehnoloških kategorij). Vsak prispevek je bil razvrščen v večje število kategorij, ki so prevladovale v njihovi vsebini. Med skupinami in kategorijami so (konkretnih kategorij, med katere so bili članki klasificirani, je 47):

- opis tehničnih rešitev (splošno, predstavitev konkretnih aplikacij, predstavitev razvojnih platform, …),
- predstavitev izkušenj in pristopov pri razvoju/prenovi/vpeljavi informacijskih rešitev,
- opis poslovno/organizacijskega vidika snovanja informacijskih rešitev (razvojne prakse, metode klasičnega razvoja, metode agilnega razvoja, vpeljava in uporaba cevovodov dostave informacijskih rešitev, DevOps pristopi, …)
- opis novih smernic razvoja (organizacijskih, tehničnih, novih teoretičnih modelov, pregled aktualnih trendov, primerjava konkretnih pristopov in rešitev, …)
- snovanje novih poslovnih modelov,
- vidiki v IT (človeški, normativni),
- uvedba, upravljanje in spremljanje informacijskih rešitev,
- konkretna področja programskega inženirstva (kibernetska varnost, integracija aplikacij, podpora/informatizacija poslovnih procesov, umetna inteligenca / strojno učenje, prototipiranje rešitev, testiranje in zagotavljanje kakovosti),
- izvor in lastniški modeli informacijskih rešitev (odprta koda, programska oprema kot storitev),
- naslavljanje konkretnih načrtovalsko-razvojnih paradigm (objektna, funkcijska, komponentna, spletne storitve, reaktivno programiranje,…),
- načrtovanje in arhitekturne smernice (v splošnem, storitveno usmerjene arhitekture, večslojne arhitekture, mikrostoritvene arhitekture, računalništvo v oblaku, blockchain in daps, IoT, …),
- pristopi načrtovanja, razvoja in uporabe različnih tipov uporabniških vmesnikov (mobilne in tablične aplikacije, spletne aplikacije, večplatformski razvoj uporabniških vmesnikov, brezstični/glasovni uporabniški vmesniki, navidezna/nadgrajena resničnost, vizualizacija podatkov),
- konkretne tehnološke rešitve in platforme (.NET, Java, Java EE, Go, …),
- podatkovne tehnologije (klasične relacijske podatkovne baze, objektne baze, XML, NoSql, izmenjava podatkov s pomočjo podatkovnih tokov, distribuirani datotečni sistemi, …),
- upravljanje in nameščanje rešitev (oblaki, zabojniki, virtualni stroji),
- konkretne domene informacijskih rešitev (družabna omrežja, digitalni marketing).

V sklopu kategorizacije prispevkov je za vsako konferenco bilo agregirano število prispevkov, ki so spadali v določeno kategorijo. Poleg absolutnega števila prispevkov je bilo izračunano tudi relativno število zastopanosti kategorije glede na število vsej prispevkov konkretne konference. Na tej osnovi so bila za vsako leto identificirana poglavitna področja, in njihova relativna zastopanost glede na področja tistega leta. Na takšen način so bila identificirana področja, ki so na konferenci prisotna skozi vsa leta v večjem ali manjšem obsegu kot tudi trendi posameznih področij (npr. določeno področje postane na konferenci obravnavano šele četrto leto, nato število prispevkov te kategorije narašča, nakar po deseti konferenci pade in področje v nadaljevanju zopet ni omenjeno). V članku so najzanimivejši trendi konference predstavljeni tudi grafično.

Omeniti velja, da so v raziskavo bile vključene le predstavitve in prispevki iz objavljenih zbornikov. Iz analize so tako izpadla vabljena predavanja eminentnih domačih in tujih predavateljev in predavateljic, ki vsakoletno občinstvu predstavijo najnovejše smernice in preboje, povezane z IT. Prav tako v analize niso bile vključene vsakoletne delavnice, ki so na voljo udeležencem konference zadnjih 11 let. Teme delavnic (vsakoletno dve ali tri) so tipično izbrane tako, da naslavljajo trenutne oz. bodoče izzive (npr. upoštevanje smernic GDPR, načrtovanje rešitev z noSql podatkovnimi bazami, potencial pristopa veriženja blokov, vpeljava oblačnih rešitev…) in konkretne tehnološke rešitve (npr. razvoj IoT rešitev, uporaba platforme Docker, razvoj spletnih aplikacij z Angular in ASP.NET, razvoj mobilnih rešitev za Android in iOS, novosti jezika JavaScript, HTML5 …).

Članek je v nadaljevanju sestavljen kot sledi. V drugem poglavju predstavimo glavne teme, ki so na konferencah prevladovale skozi leta, in način predstavitve le-teh. V tretjem poglavju so najbolj zastopana področja konference predstavljena v obliki trenda prispevkov skozi leta. V zaključku poizkušamo na osnovi identificiranih trendov ugotoviti, ali so le-ti skladni z globalnim dogajanjem v informatiki ter predvideti, katere teme bodo prevladovale v naslednjih letih.

2 Glavne in pomožne teme prispevkov skozi leta

Konferenca OTS je na deklarativni ravni namenjena predvsem aktivnostim snovanja, razvoja in vpeljave informacijskih rešitev v poslovanje. Način posredovanja in izmenjave znanj pa naj bi bil preko praktičnih izkušenj. Tabela 1 povzema 15 najpogosteje prisotnih področij konference in navaja, koliko izmed 640 analiziranih in kategoriziranih prispevkov je posamezni kategoriji ustrezalo. V tabeli 1 vidimo, da z naskokom (64% in 35% vseh prispevkov) vodita ravno kategoriji tehničnega vidika in predstavitvi konkretnih izkušenj. Opozoriti velja, da sta v tabeli 1 tudi dve kategoriji, ki sta sicer res kumulativno visoko zastopani, a ju ne najdemo v večini (80%) let konference – »metode razvoja v splošnem« in »način razvoja mobilnih aplikacij«, kar je razumljivo, saj smo pred letom 2007 (dvanajsta konferenca), ko je na trg prišel prvi t.i. pameten telefon, o mobilnih aplikacijah težje in redkeje govorili.

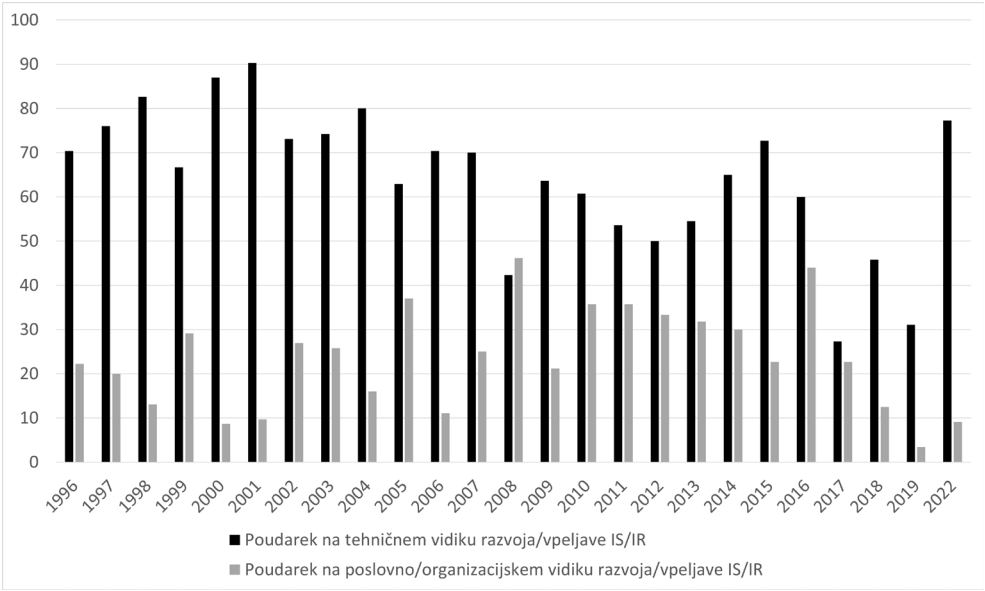
Tabela 2: 15 kumulativno najbolj zastopanih področij konference.

Kategorija prispevka	Število prispevkov
Vsi prispevki	**640**
Poudarek na tehničnem vidiku razvoja/vpeljave IS/IR *	411
Predstavitev izkušenj pri razvoju/prenovi/vpeljavi IS/IR *	223
Poudarek na poslovno/organizacijskem vidiku razvoja/vpeljave IS/IR *	152
Nove razvojne smernice *	145
Teoretične osnove / vpogled v prihodne trende *	128
Nove razvojne platforme *	122
IT arhitekture in načrtovanje IS/IR *	88
Primerjava konkretnih rešitev in pristopov *	82
Načini razvoja spletnih aplikacij *	77
Podatkovne tehnologije / BI *	64
Predstavitev konkretnih rešitev *	50
Predstavitev novih poslovnih modelov *	50
Metode razvoja / SDLC	48
Integracije IS/IR *	47
Način razvoja mobilnih/tabličnih aplikacij	42

* Kategorija je hkrati prisotna na vsaj 80% letih konference

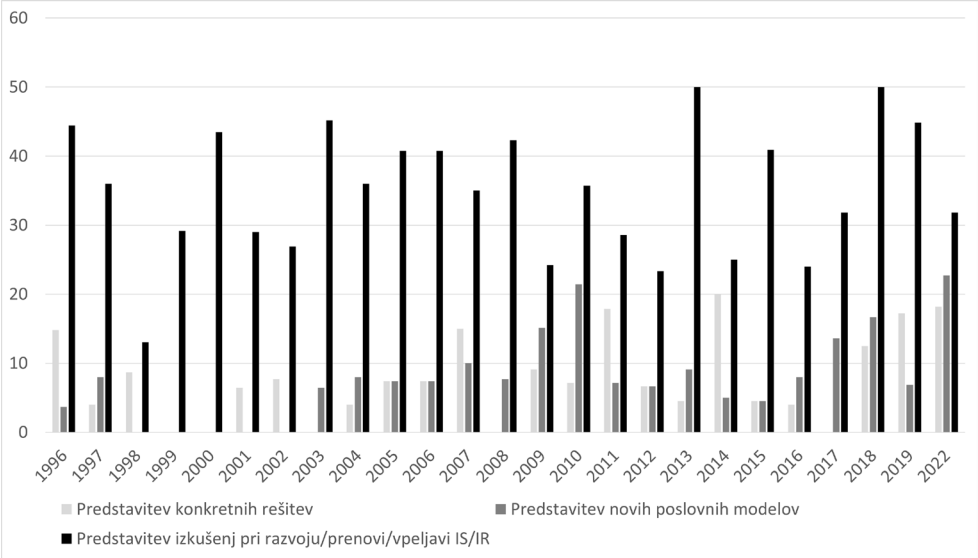
Vir: [1-25].

Gibanje števila prispevkov glede na vodilna poudarka prispevka (kategoriji tehničen in organizacijski vidik) skozi leta je grafično predstavljeno na sliki 1. Na sliki je jasno izstopajoča kategorija tehničnega vidika prispevkov. Ti prispevki tipično predstavljajo konkretne načine razvoja in/ali implementacije neke družine izdelkov, uporabe konkretnih programskih jezikov, ogrodij ipd. Po drugi strani pa kategorija organizacijskega vidika, ki glede na leta izkazuje obratno sorazmernost tehnični, zajema prispevke, ki se osredotočajo predvsem na vidik organizacije razvojnih ekip, uporabe orodij pri tem, načine in postopke vpeljave ali razvoja rešitev ipd.



Slika 1: Poglavitni poudarki prispevkov konferenc OTS.
Vir: lasten.

Ker zgolj tehnična in organizacijska dimenzija razvojnih aktivnosti, sploh glede na področja v tabeli 2, dajejo le delno sliko, si oglejmo vodilne kategorije, ki se ukvarjajo z načinom predstavitve v prispevku. Kot izhaja iz podatkov (glejte sliko 2) vidimo, da je skozi leta v povprečju več kot tretjina (ob določenih letih pa celo polovica) prispevkov temeljila na predstavitvi konkretnih izkušenj pri razvoju, prenovi ali vpeljavi informacijskih rešitev in storitev. Če k temu dodamo še predstavitev konkretnih rešitev (prispevki, ki v podrobnosti predstavijo neko konkretno informacijsko rešitev), lahko zaključimo, da večina prispevkov skozi vsa leta konference temelji na praktičnih izkušnjah, ki jih avtorji delijo z avditorijem. Na sliki 2 vidimo tudi zastopanost prispevkov, ki predstavijo nove poslovne modele, ter jih avtorji tipično nadgradijo s konkretnimi informacijskimi rešitvami. Konstantnost prispevkov, ki se ukvarjajo z informatiko kot načinom snovanja in uveljavljanja novih poslovnih modelov kaže na to, da konferenca OTS vendarle ni le tehnično-razvojno orientiran dogodek, temveč ponuja udeležencem skozi vsa leta tudi vsebine in izkušnje avtorjev, ki verjamejo, da informatika ne le sledi razvoju družbe kot celote, temveč ga aktivno soustvarja.

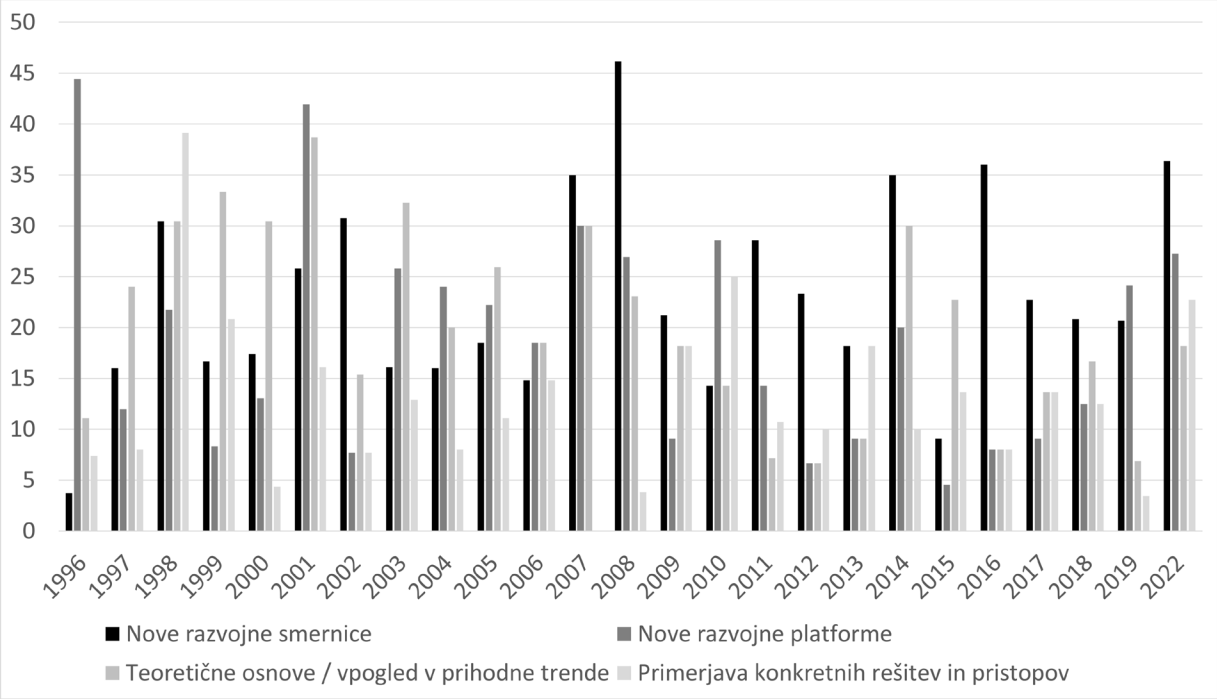


Slika 2: Vodilne tematika prispevkov na konferencah OTS.
Vir: lasten

Graf na sliki 3 področja prispevkov (slika 1) in njihov način predstavitve (slika 2) nadgradi še z glavnimi novostmi, ki jih prispevki promovirajo. Nove razvojne smernice so vodilna kategorija v večini let. Le-ta zajema prispevke, ki se ukvarjajo tako s tehničnimi kot tudi konceptualnimi razvojnimi smernicami – naj bo to spodbujanje ponovne uporabe preko knjižnic, načrtovalskih vzorcev, nove pristope pri hranjenju podatkov, ustrezno organizacijo razvojnih okolij ipd.

Sledijo prispevki (20% vseh prispevkov), ki avditoriju predstavijo abstraktne, teoretične osnove snovanja, razvoja in vpeljave informacijskih rešitev – pa naj bo to ustrezno snovanje objektnih sistemov, uporaba funkcijskega stila programiranja, ustreznega razvrščanje rutin in niti, teoretične osnove kriptografije ipd. Čeprav bi bilo pričakovati, da je večina teh prispevkov nastala v akademskem okolju, jih pomemben delež izhaja iz industrijskega okolja. Povprečno je na tretjem mestu kategorija prispevkov, ki so namenjeni konkretni predstavitvi novih razvojnih platform, jezikov, ogrodij ipd., tipično skozi konkretne primere.

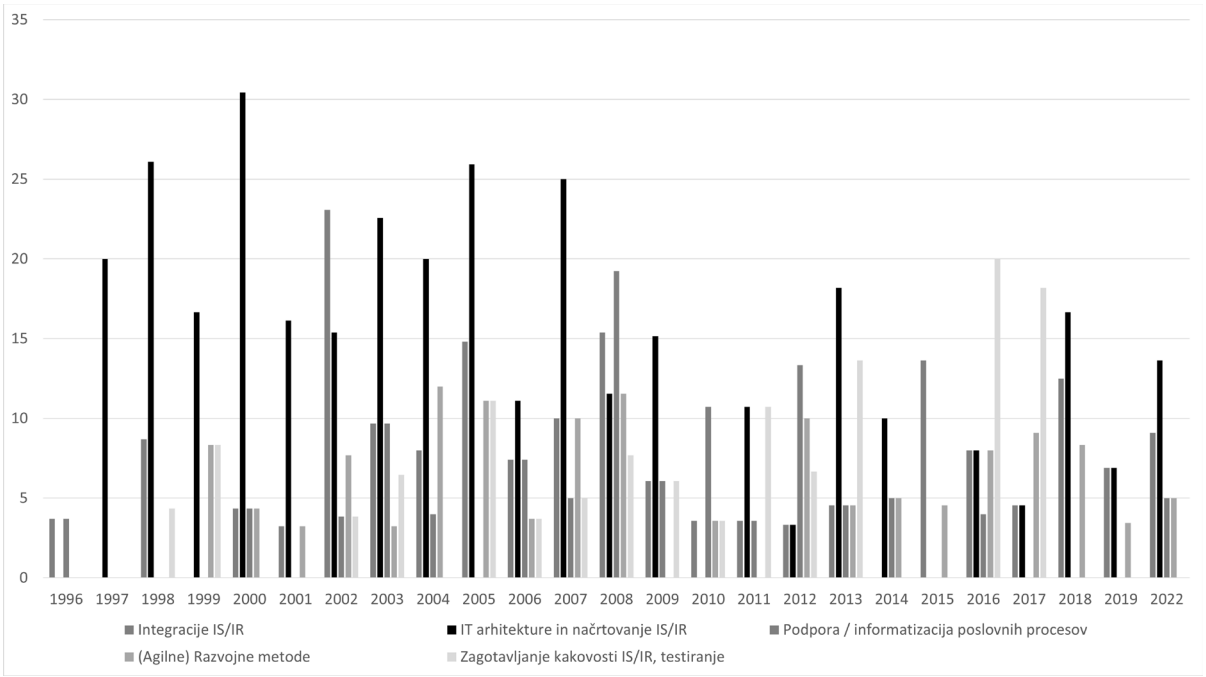
Zelo zastopani, v določenih letih celo tretjinsko, so prispevki, ki se ukvarjajo s sistematičnim primerjanjem več različnih, tipično tekmujočih pristopov in rešitev. Takšni prispevki denimo primerjajo različne metode agilnega razvoja, različne knjižnice za vizualizacijo podatkov, različne načine monetizacije mobilnih aplikacij ipd. ter na takšen način omogočijo posameznikom, da se ustrezno odločijo glede na svoje problemsko področje.



Slika 3: Predstavljene novosti na konferencah OTS.

Vir: lasten.

Če se pomaknemo h konkretnim področjem, o katerih prispevki tipično predstavljajo novosti, teoretične osnove, izkušnje ipd., nam to dopolni slika 4. Iz slike izhaja, da so najpogosteje naslovljene arhitekture informacijskih sistemov ter pristopi njihovega načrtovanja. Med kumulativno najbolj zastopanimi področji še najdemo IT integracije, ki zajemajo slabo desetino vseh prispevkov konferenc OTS. Ostalih treh področij, ki jih prikazuje slika 4, sicer ne najdemo med kumulativno najbolj zastopanimi področji konference, a so umeščeni med vodilna področja konference, ker jih kljub nizki skupni zastopanosti (ranga 5%) najdemo zastopane v večini let konference.



Slika 4: Vodilna področja prispevkov konferenc OTS.

Vir: lasten

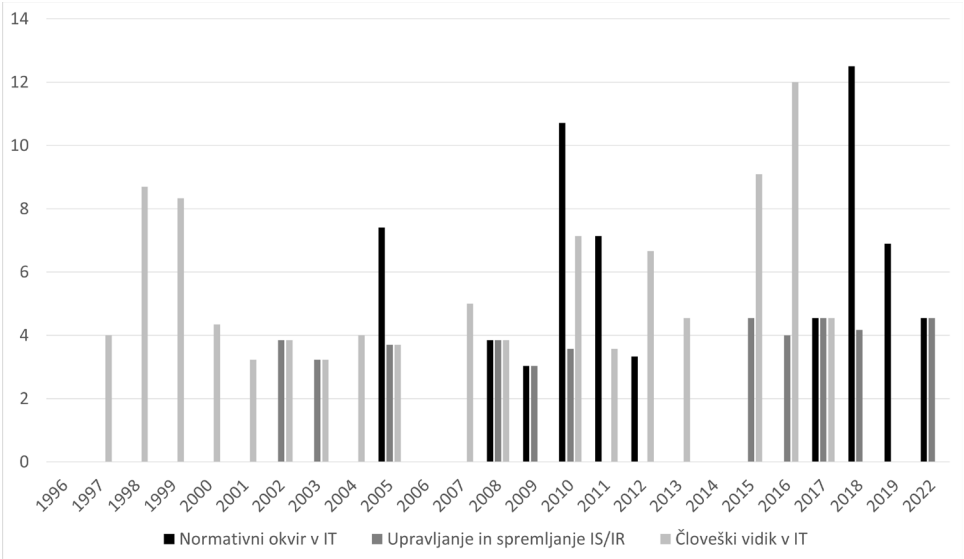
Morebiten razkorak pri sicer kumulativno visoko zastopanih področjih konference, ki pa morebiti niso zastopana skozi večino (80%) let konference, prikazuje tabela 3. V njej najdemo področja konference, ki so bila naslovljena z vsaj enim prispevkom večine let konference. Kategorije so urejene padajoče – na vrhu so področja, ki so največkrat omenjena na največ letih konference.

Tabela 3: Področja konference, zastopana na vsaj 80% letih konference.

Kategorija prispevka
Poudarek na tehničnem vidiku razvoja/vpeljave IS/IR *
Predstavitev izkušenj pri razvoju/prenovi/vpeljavi IS/IR *
Poudarek na poslovno/organizacijskem vidiku razvoja/vpeljave IS/IR *
Nove razvojne smernice *
Teoretične osnove / vpogled v prihodne trende *
Nove razvojne platforme *
Načini razvoja spletnih aplikacij *
Primerjava konkretnih rešitev in pristopov *
Podatkovne tehnologije / BI *
IT arhitekture in načrtovanje IS/IR *
Integracije IS/IR *
Predstavitev konkretnih rešitev *
Predstavitev novih poslovnih modelov *
Metode agilnega razvoja
Kibernetska varnost
Človeški vidiki v IT
* Kategorija je hkrati prisotna med 15 najbolj zastopanimi kategorijami

Vir: [1-25].

Iz tabele 3 izhaja, da so vodilna področja konference sicer tudi res vodilna skozi vsa leta, a bolj zanimiv je spodnji del tabele 3. Iz njega izhaja, da področja »metode agilnega razvoja«, »kibernetska varnost« in »človeški vidiki v IT« sicer niso naslovljeni z izstopajoče visokim številom prispevkov, so pa stalnica skozi skoraj vsa leta konference, zaradi česar jih lahko uvrstimo med pomembna fokusna področja konference OTS. Kot zanimivost na sliki 5 predstavljamo eno izmed pogosteje zastopanih kategorij prispevkov skozi vsa leta (človeški faktor v IT), z v drugi polovici konferenc OTS vedno pogosteje zastopanima kategorijama prispevkov: »upravljanje informacijskih rešitev« in »normativni vidiki v IT«.



Slika 5: Dodatna področja prispevkov konferenc OTS.

Vir: lasten.

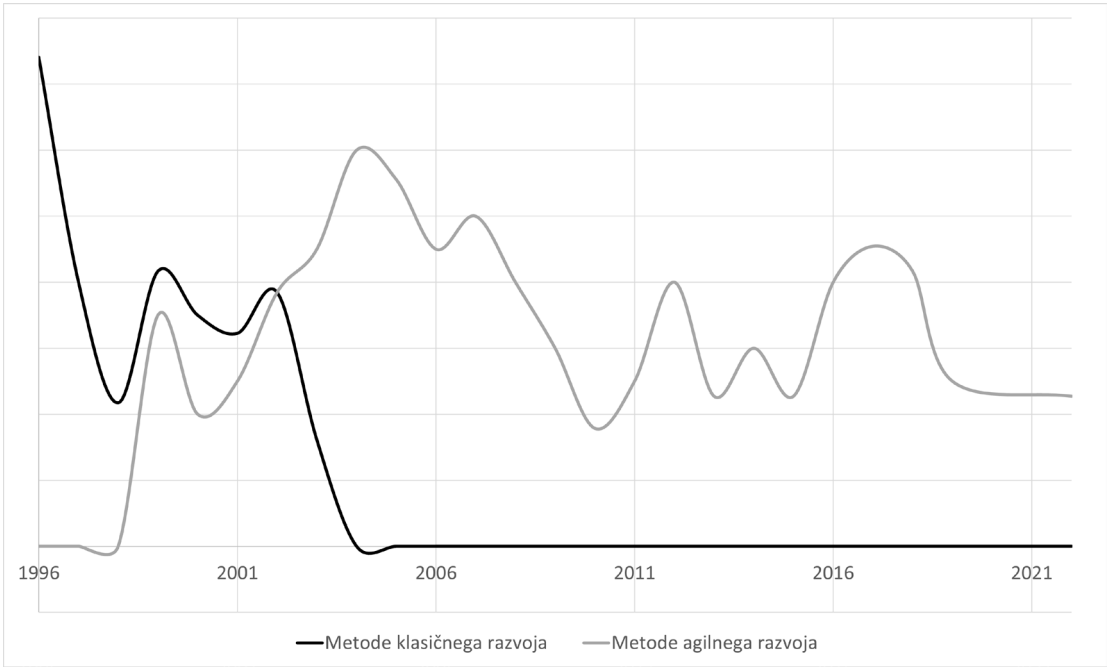
3 Trendi obravnavanih tematik

Uvodoma predstavljena metodologija analize prispevkov konferenc OTS daje tudi možnost analize trendov pojavnosti tematik skozi leta. V tem poglavju predstavljamo le najbolj zanimive in zgovorne primerjave trendov. Zajeli bomo trende pojavnosti razvojnih metod, arhitektur, pristopov k načrtovanju informacijskih rešitev, zelo zanimivi so tudi trendi tipov aplikacij, ki nastopajo v prispevkih, izpostavili pa bomo tudi trende tematik, ki so na konferencah prisotne skozi vsa leta.

Temam, ki so na konferenci tudi bile prisotne, a so izzvenele oz. so postale t.i. horizontalne tematike (implicitno prisotne tako rekoč v vseh prispevkih) v tem poglavju ne posvečamo posebne pozornosti. Primer takšne teme je npr. »odprta koda«. Prvi prispevek, ki se je ukvarjal izključno s fenomenom odprte kode, je bil na konferenci predstavljen že leta 1999. Po štirih letih zatišja so se zopet pojavili prispevki, ki so se prvenstveno ukvarjali z odprtokodnimi pristopi in rešitvami z vrhuncem med leti 2007 in 2010, kjer so se kar štirje prispevki ukvarjali izključno s to tematiko. Z rahlim upadanjem je tema na konferenci izzvenela leta 2012, nato pa samostojno sicer skoraj več ne nastopa (izjema je npr. letošnja konferenca), je pa prisotna kot dodatna tema pri veliko prispevkih.

3.1. Metode razvoja informacijskih rešitev

Metode razvoja informacijskih rešitev in storitev so ena izmed najbolj zastopanih tem (glejte tabelo 1) konference OTS – skoraj 8% vseh prispevkov konferenc se primarno ukvarja s to temo. Prva leta konference, ko je bil poudarek predvsem na objektni tehnologiji, ne čudi, da so bile v ospredju tudi razvojne metode, ki so zelo povezane z objektno orientacijo – OMT (Object Modeling Technology), metoda po Grady-ju Booch-u, tudi RUP (Rational Unified Process) v številnih različicah, nekaj prispevkov je bilo tudi na temo MDD (Model-Driven Development). Že v prvih letih konference so avtorji poročali o svojih izkušnjah pri uvajanju in uporabi današnjih metod agilnega razvoja - ta tema je dobila dodaten zagon ob prelomu stoletja. Formalizacija metod agilnega razvoja se je namreč zgodila leta 2001 z izdajo manifesta agilnega razvoja. Vse od takrat so prispevki, ki predstavljajo prednosti, a tudi pasti, možnosti izbire, dobre prakse ipd. na področju metod agilnega razvoja, stalnica konference.



Slika 6: Trend obravnavanih razvojnih metod v prispevkih konferenc OTS.

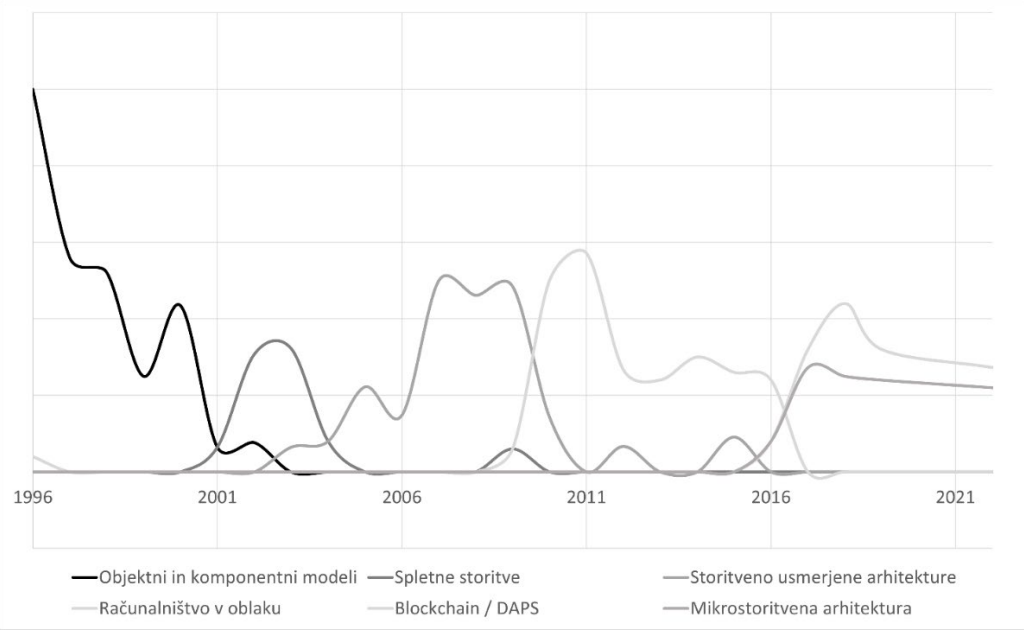
Vir: lasten.

Od leta 2015 dalje tudi redno srečamo prispevke, ki se ukvarjajo z uporabo metod agilnega razvoja v velikem obsegu (npr. LeSS, SAFe ipd.). Upad zanimanja avtorjev za klasične, težje, plansko usmerjene razvojne metode in pojav in rast prispevkov iz metod agilnega razvoja lepo povzema slika 6.

V kontekst agilnega razvoja informacijskih rešitev, predvsem v neiterativnih metodah, kot je npr. Kanban, spadajo tudi pristopi samodejne dostave informacijskih rešitev preko avtomatiziranih CI/CD cevovodov in pristopov DevOps. Tema se je na konferenci pojavila že leta 2015 in je vse odtlej vselej prisotna, a nikoli v prevladujočem odstotku.

3.2. Arhitekturni stili, načrtovanje in zasnova rešitev

Objektna orientacija, ki je bila osrednji sprožilec konference OTS, že v osnovi nosi s sabo zahtevo po ustreznem načrtovanju informacijskih rešitev - ob upoštevanju ustreznih smernic, slogov, vzorcev. Če seštejemo kategorije, ki se ukvarjajo z arhitekturami in načrtovanjem informacijskih rešitev (npr. arhitekture in načrtovanje v splošnem, objektno načrtovanje, komponentni modeli, spletne storitve, storitvene arhitekture, dogodkovno gnane arhitekture ipd.), ugotovimo, da se na konferenci skoraj četrtina prispevkov ukvarja s to tematiko. Bolj zanimivo je opazovati trend gibanja posameznih tem, povezanih z načrtovanjem. Prikazuje jih slika 7. V njej bode v oči visoka zastopanost objektnih in komponentnih modelov v prvi konferenci. Visok odstotek je treba razumeti v kontekstu tega, da je to tudi bila osrednja tema prve konference. Čeprav prispevkov, ki se ukvarjajo z objektnim načrtovanjem, načrtovalskimi vzorci, komponentnimi modeli, kot so DCOM in CORBA, po letu 2002 skoraj več ne zasledimo, pa se kot dodatna tema še vedno pojavljajo v določenih prispevkih. Iz grafa je razvidno, kako so ob prelomu tisočletja pobudo prevzele »klasične« spletne storitve (tehnološki sklad WSDL-SOAP-UDDI), ki pa so v manj kot 5 letih postale le še opcijsko orodje v storitveno usmerjenih arhitekturah, ki so kot tema kraljevale na konferenci do leta 2010. Arhitekturam v klasičnem smislu besede je bilo posvečene manj pozornosti vse do leta 2015, ko so se pojavili prvi prispevki iz področja mikrostoritvene arhitekture, ki predstavlja trend še danes. Vmesno območje, med 2010 in 2017 je bila glavna tema, delno povezana z IT arhitekturami, računalništvo v oblaku. Načrtovalske rešitve decentraliziranih sistemov, ki tipično temeljijo na tehnologijah veriženja blokov, pa sovpadajo s trendom mikrostoritev in mikrostoritvene arhitekture.



Slika 7: Arhitekturni trendi, obravnavani v prispevkih konferenc OTS.

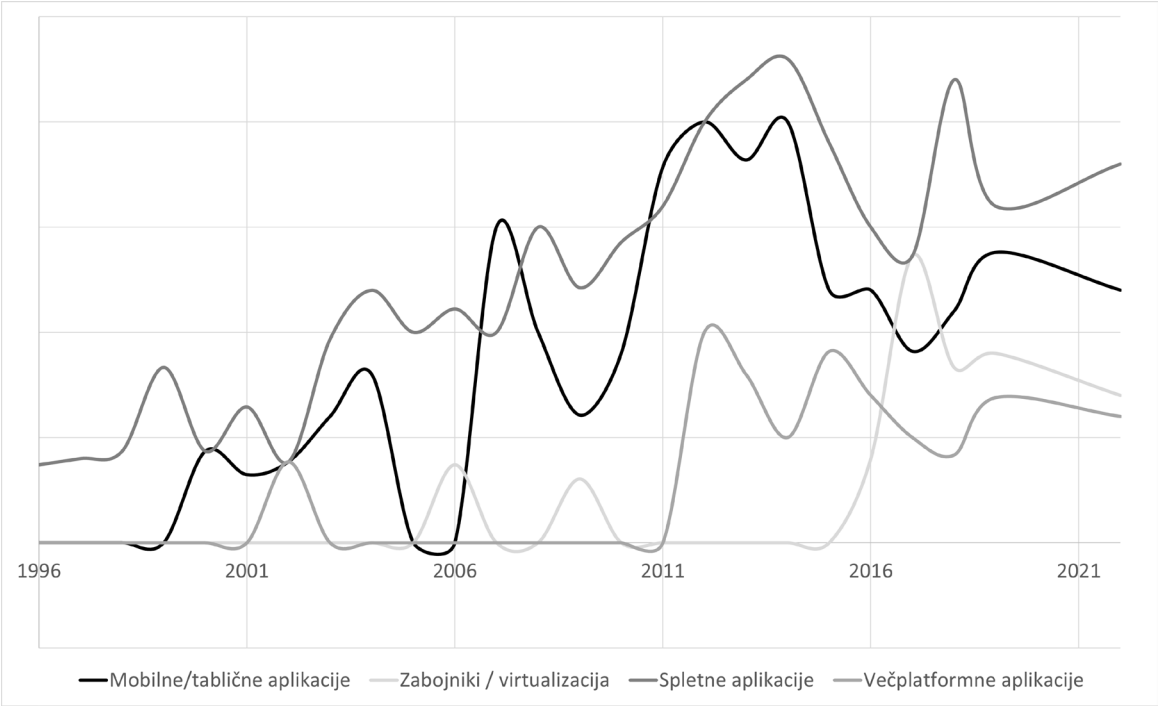
Vir: lasten.

3.3. Tipi aplikacij

Ob analizi prispevkov konference skozi leta se je izkazalo, da se relativno mnogo (ob določenih letih tudi več kot polovica) prispevkov ukvarja s tehnologijami, pristopi, predstavitvijo izkušenj, uporabo ogrodij ipd. iz področja uporabniških vmesnikov oz. ciljnih izvajalnih okolij, kjer aplikacije delujejo oz. za katere so namenjene. Glede na to, da začetek konference OTS sovpada s počasnim zamiranjem zanimanja za namizne uporabniške vmesnike, ne čudi, da v vseh letih ne najdemo prispevka, ki bi se ukvarjal izključno z načinom razvoja oz. prilagajanja klasičnih, namiznih uporabniških vmesnikov. Po drugi strani pa (glejte sliko 8), tudi konferenca OTS dokazuje, da so vodilni uporabniški vmesniki zadnjega četrt stoletja tisti, ki kot platformo aplikacij jemljejo svetovni splet. Več kot 10% prispevkov konferenc OTS se namreč ukvarja z različnimi pristopi, načrtovalskimi prijemi, platformami ogrodji ipd., namenjenimi spletnim aplikacijam. Čeprav se fokus počasi menja (na grafu ga vidimo v obliki valov) od enostavnih CGI skript, preko uporabe strežniških spletnih aplikacij po vzorcu MVC, koncepta AJAX, pa do enostranskih spletnih aplikacij, ki delujejo na odjemalcu, spletni uporabniški vmesniki ostajajo vodilen tip obravnavanih uporabniških vmesnikov na konferenci OTS.

Podoben trend kažejo tudi mobilne oz. tablične aplikacije. To je še eno izmed področij, kjer konferenca kaže svoje vizionarstvo. Zanimiv je »prvi val« prispevkov na temo mobilnih uporabniških vmesnikov med leti 1998 in 2005. Spomnimo, o konceptu mobilne aplikacije, kot jo razumemo danes, se je začelo govoriti šele z letom 2007. Pred tem smo na konferenci OTS bili priča prispevkom, ki so predstavljali smernice in izkušnje z Javo ME, prispevek iz preloma tisočletja pa je celo demonstriral pristop, ki se je čez 20 let izkazal za enega vodilnih: avtorji prikažejo, kako so razvili le eno aplikacijo s HTML uporabniškim vmesnikom ter jo na mobilno napravo namestili v obliki aplikacije, na strežnik pa v obliki spletne aplikacije. Tovrstni prispevki, kategorizirani v »večplatformne aplikacije« so postali trend po letu 2011; področje je preko platform kot so npr. Flutter in React Native, trend še danes.

Morda področje virtualizacije in zabojnikov na prvi pogled ne spada med predstavljene kategorije na sliki 8, a gre za področje, ki narekuje načrtovalske prijeme in standardizacijo pakiranja tudi spletnih, hibridnih in zalednih aplikacij.



Slika 8: Vodilni tipi predstavljenih rešitev v prispevkih konferenc OTS.
Vir: lasten.

Kot zanimivost: na grafu slike 8 bi lahko predstavili tudi številne druge tipe uporabniških vmesnikov. Primeri, ki so na konferenci sicer že bili prisotni nekajkrat, a se prispevki niso izkazali za trendne, so brezstični in/ali glasovni uporabniški vmesniki. Prispevke najdemo že v letih 2001 in 2003, ko je tehnologija VoiceXML še obetala. Ponovno najdemo prispevke na konferencah 2012, 2013 in 2017, ki so se ukvarjali z brezstičnim rokovanjem z informacijskimi rešitvami, temelječih predvsem na nosljivih napravah in kamerah (Xbox/Kinect). Udeleženci ene izmed konferenc so celo tekmovali pri vožnji avtomobilov, kjer so obračali navidezni volan in stopali po navideznih stopalkah za plin in zavore.

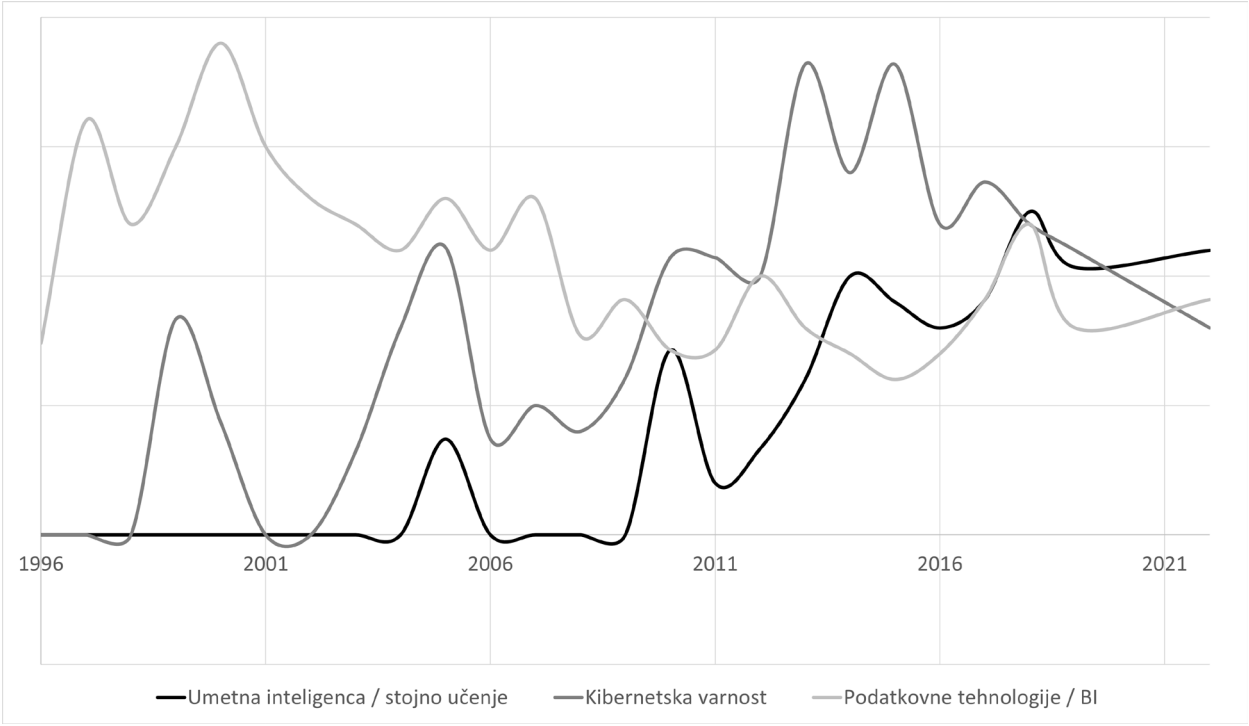
3.4. Dolgoročni trendi področij konference OTS

Če se distanciramo od kategorij prispevkov s konkretnimi tehnologijami in pristopi, ter poizkušamo identificirati področja, ki so oz. postajajo trend konference OTS, gre iskati kandidate predvsem v tabelah 2 in 3. Glede na konstantnost, izrazito naraščajočo pojavnost ter aktualnost izstopajo tri področja (glejte sliko 9).

Področje podatkovnih tehnologij in poslovne inteligence (pa naj gre za relacijske, objektne, dokumentne, noSql ipd. podatkovne baze; tehnologije XML, poslovno analitiko ipd.) je visoko in konstantno zastopano na konferenci večini let.

Kibernetska varnost je vsebinsko področje prispevkov, ki ga npr. v letih 2013 in 2015 naslavlja skoraj 20% vseh prispevkov konference. Sicer pa zaznavamo konstantno rast v valovih. Danes je področje kibernetske varnosti ne samo eno vodilnih področij konference, je tudi izrazito trendno.

Zagotovo pa izstopa področje umetne inteligence in strojnega učenja. Prispevki izključno na to temo so na konferenci prisotni vse od leta 2004, od leta 2008 pa njihovo število zgolj še raste.



Slika 9: Najbolj obravnavana IT področja prispevkov konferenc OTS.
Vir: lasten.

Če poleg, na sliki 9 najatraktivnejših trendnih področij, povzamemo celotne rezultate analize prispevkov konference vseh 25 let, je slika sledeča.

V vrhu tako glede skupnega števila prispevkov kot tudi konstantnosti pojavljanja so naslednja štiri vsebinska področja:

- IT arhitekture in načrtovanje,
- podatkovne tehnologije / BI,
- metode razvoja / SDLC (vključno s pristopi DevOps in CI/CD),
- integracije IS/IR.

Upoštevajoč le drugo polovico izvedb konference OTS, pa se vodilnim vsebinskim področjem pridružujejo še:

- umetna inteligenca / strojno učenje,
- kibernetska varnost,
- načini razvoja spletnih aplikacij,
- načini razvoja mobilnih/tabličnih aplikacij,
- rešitve na osnovi IoT.

Kot dodatek skupno vodilnim vsebinskim področjem lahko izpostavimo še tri, katerih prispevki so bili prisotni na tako rekoč vseh (vsaj 80%) dosedanjih konferenc OTS:

- zagotavljanje kakovosti IS/IR in testiranje,
- podpora / informatizacija poslovnih procesov,
- človeški vidik v IT.

4 Zaključek

V članku so predstavljeni rezultati raziskave, ki je na sistematičen način pristopila k analizi področij, s katerimi so se ukvarjali prispevki na konferenci OTS prvih 25 izvedb. Skozi izpostavljene najzanimivejše tematike konference skozi leta je razvidno, da prednjačijo tehnološko usmerjene vsebine, podane na praktičen način.

Sodobna služba informatika predstavlja zelo široko in izrazito interdisciplinarno delo. Poleg razvoja, uvajanja in vzdrževanja IT rešitev obsega še mnoga druga področja. Med njimi so najpogostejša:

- soustvarjanje politik podjetij pri uvajanju in uporabi informacijskih rešitev pri digitalizaciji in uvajanju novih poslovnih modelov,
- skrb za avtomatizacijo poslovanja, na osnovi informacijskih rešitev in storitev, kjer je to le mogoče,
- spremljanje, presoja, uvajanje in vzdrževanje novih informacijskih tehnologij glede na poslovne potrebe podjetij,
- integracijo različnih informacijskih rešitev v skladno delujočo celoto, ter navsezadnje
- skrb za zagotavljanje ustrezne in pričakovane kakovosti rešitev in storitev v uporabi.

Z veseljem lahko ugotovimo, da med drugim konferenca OTS preko prispevkov avtorjev že leta ustrezno naslavlja vsa zgoraj našteta področja. Dogodek torej ustrezno nadgrajuje osnovno poslanstvo: »…izmenjava praktičnih izkušenj in spoznanj glede uspešne vpeljave in inovativne uporabe sodobnih tehnologij ter pristopov k razvoju informacijskih rešitev.«

V članku pri obravnavi posameznih področij tudi ugotavljamo, da poudarki področij, pa naj bodo tehnična ali organizacijska, ne zaostajajo za vodilnimi trendi v globalnem IT. Prej lahko ugotovimo celo, da so prispevki skladni z vsakoletnim trenutnim stanjem informatike v svetu, ali pa trende s poročanjem o praktičnih izkušnjah celo

prehitevajo. Še posebej veseli, da vizionarski prispevki ne izhajajo le iz akademske sfere, temveč tudi iz industrijskega okolja. To pomeni, da podjetja, ki so s svojim načinom dela in tehnološkimi rešitvami usmerjena daleč v prihodnost, svoje pozitivne in negativne izkušnje t.i. »early-adopterjev« nesebično delijo z ostalimi, ter na takšen način prispevajo k širitvi novosti na ustrezen način v širšo industrijo.

Kot primere lahko navedeno celo vrsto prispevkov, ki so npr:

- se ukvarjali s tehnologijo XML, še preden je le-ta bila sprejeta kot splošen de-facto standard,
- so poročali o večplatformnem razvoju za splet in mobilne naprave, še preden je globalna industrija na vsakdanji besednjak umestila besedno zvezo »mobilna aplikacija«,
- že desetletje pred prvim pametnim telefonom poročali o praktičnih izkušnjah in posebnostih razvoja aplikacij, ki delujejo na mobilnih telefonih,
- poročali o trajni hrambi objektov še preden je industrija prepoznala t.i. objektne podatkovne baze,
- na sistematičen način poročali o uporabi metod agilnega razvoja po principih, kot jih je šele leto za tem Kent (1999) predstavil v metodi XP,
- poročali o programski opremi oz. platformah kot storitvah, še preden je industrija leta 2010 sistematično zaznala področje računalništva v oblaku,
- promovirali (mikro)storitvene arhitekturne stile skladno oz. leto pred globalnim zanimanjem zanju,
- redno poročali o napredku področij kot so tehnologija veriženja blokov, pristopih DevOps in potencialih apliciranja umetne inteligence v oz. pred globalnim trendom,
- …

Ugotovimo lahko, da je konferenca OTS v svojih prvih 25 izvedbah nudila zelo kakovosten, ažuren in celovit forum informatikov pri širjenju znanja in izmenjavi izkušenj. Nepomembno ni niti dejstvo, da konferenca in objavljanje prispevkov poteka v jeziku, ki je globalno gledano s približno dvema milijonoma govorcev, dokaj nepomemben. Na takšen način konferenca bogati tudi strokovne debate in terminologijo v domačem jeziku. Ob podobnem tempu in aktualni vsebinski zasnovi prispevkov konference OTS vse kaže, da se za konferenco ni bati, veseli pa tudi, da v četrt stoletja ostaja fokus dogodka še vedno jasno definiran.

Se nadaljuje… [26]

Literatura

[1] Heričko, M., Rozman, I. (ur.). (1996). *Objektna tehnologija v Sloveniji OTS '96: Zbornik strokovnega srečanja, Maribor, 1996*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[2] Heričko, M., Rozman, I. (ur.). (1997). *Objektna tehnologija v Sloveniji OTS '97: Zbornik drugega strokovnega srečanja, Maribor, 1997*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[3] Heričko, M., Rozman, I. (ur.). (1998). *Objektna tehnologija v Sloveniji OTS '98: Zbornik tretjega strokovnega srečanja, Maribor, 1998*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[4] Heričko, M., Rozman, I. (ur.). (1999). *Objektna tehnologija v Sloveniji OTS '99: Zbornik četrtega strokovnega srečanja, Maribor, 1999*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[5] Heričko, M., Rozman, I. (ur.). (2000). *Objektna tehnologija v Sloveniji OTS'2000: Zbornik pete konference, Maribor, 2000*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[6] Heričko, M. (ur.). (2001). *Objektna tehnologija v Sloveniji OTS'2001: Zbornik šeste konference, Maribor, 2001*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[7] Heričko, M. (ur.). (2002). *Objektna tehnologija v Sloveniji OTS'2002: Zbornik sedme konference Maribor, 2002*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[8] Heričko, M., Živkovič, A. (ur.). (2003). *Objektna tehnologija v Sloveniji OTS'2003: Zbornik osme konference, Maribor, 2003*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[9] Heričko, M., Živkovič, A. (ur.). (2004). *Objektna tehnologija v Sloveniji OTS'2004: Zbornik devete konference, Maribor, 2004*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[10] Heričko, M., Živkovič, A. (ur.). (2005). *Objektna tehnologija v Sloveniji OTS'2005: Zbornik desete konference, Maribor, 2005*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[11] Heričko, M., Živkovič, A. (ur.). (2006). *OTS'2006 Sodobne tehnologije in storitve: Zbornik enajste konference, Maribor, 2006*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[12] Heričko, M., Živkovič, A. (ur.). (2007). *OTS'2007 Sodobne tehnologije in storitve: Zbornik dvanajste konference, Maribor, 2007*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[13] Heričko, M., Živkovič, A., Kous, K. (ur.). (2008). *OTS'2008 Sodobne tehnologije in storitve: Zbornik trinajste konference, Maribor, 2008*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[14] Heričko, M., Živkovič, A., Kous, K. (ur.). (2009). *OTS 2009 Sodobne tehnologije in storitve: Zbornik štirinajste konference, Maribor, 2009*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[15] Heričko, M., Živkovič, A., Kous, K. (ur.). (2010). *OTS 2010 Sodobne in storitve: Zbornik petnajste konference, Maribor, 2010*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[16] Heričko, M., Živkovič, A., Kous, K. (ur.). (2011). *OTS 2011 Sodobne tehnologije in storitve: Zbornik šestnajste konference, Maribor, 2011*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[17] Heričko, M., Kous, K. (ur.). (2012). *OTS 2012 Sodobne tehnologije in storitve: Zbornik sedemnajste konference, Maribor, 2012*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[18] Heričko, M., Kous, K. (ur.). (2013). *OTS 2013 Sodobne tehnologije in storitve: Zbornik osemnajste konference, Maribor, 2013*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[19] Heričko, M., Kous, K. (ur.). (2014). *OTS 2014 Sodobne tehnologije in storitve: Zbornik devetnajste konference, Maribor, 2014*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[20] Heričko, M., Kous, K. (ur.). (2015). *OTS 2015 Sodobne tehnologije in storitve: Zbornik dvajsete konference, Maribor, 2015*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[21] Heričko, M., Kous, K. (ur.). (2016). *OTS 2016 Sodobne tehnologije in storitve: Zbornik enaindvajsete konference, Maribor, 2016*. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[22] Heričko, M., Kous, K. (ur.). (2017*). OTS 2017 Sodobne informacijske tehnologije in storitve: Zbornik dvaindvajsete konference, Maribor, 2017*. Maribor: Univerzitetna založba. doi: 10.18690/978-961-286-040-0.

[23] Heričko, M., Kous, K. (ur.). (2018). *OTS 2018 Sodobne informacijske tehnologije in storitve: Zbornik triindvajsete konference, Maribor, 2018*. Maribor: Univerzitetna založba. doi: 10.18690/978-961-286-162-9.

[24] Heričko, M., Kous, K. (ur.). (2019). *OTS 2019 Sodobne informacijske tehnologije in storitve: Zbornik štiriindvajsete konference, Maribor, 2017*. Maribor: Univerzitetna založba. doi: 10.18690/978-961-286-282-4.

[25] Wiktionary - to be continued. https://en.wiktionary.org/wiki/to_be_continued. Obiskano avgust 2022.

Tehnični izzivi sodobne poslovne analitike

Mateja Verlič Brunčič, Boris Ovčjak

Databox Inc., Ptuj, Slovenija
mateja@databox.com, boris.ovcjak@databox.com

Sinopsis Poslovna analitika lahko pomaga podjetjem sprejeti boljše in informirane odločitve ne glede na velikost podjetja. Na podlagi podatkov podjetja boljše razumejo obnašanje svojih strank ali pridobijo vpogled v konkurenco. Pri tem ne gre zanemariti tehničnih izzivov, ki pridejo s področjem in na finančne, človeške, tehnične, pa tudi strateške omejitve pri reševanju. Podatke je potrebno prečrpati, obdelati, analizirati, vizualizirati in na koncu pripraviti v obliki, ki jo uporabniki razumejo in jo uporabijo kot osnovo za odločanje. Vsak korak predstavlja specifiko in ima svoj nabor omejitev, dodaten vir izzivov pa predstavljajo tudi zunanje storitve, od katerih pridobivamo podatke in nad katerimi načeloma nimamo kontrole. Databox ima več kot deset let izkušenj z reševanjem izzivov na tem področju. V prispevku bomo predstavili, kako smo se lotili njihovega reševanja, da bi uporabniku čim bolj približali podatke na intuitiven in razumljiv način, ki jim daje dodano vrednost, omenili bomo tudi, kakšni izzivi nas čakajo v bližnji in srednji prihodnosti.

Ključne besede:
razvoj digitalnih storitvenih platform
poslovno obveščanje
poslovno usmerjen razvoj

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.2

1 Uvod

Poslovna analitika ni nova stvar, vendar vztrajno pridobiva na pomembnosti in vedno več je podjetij na trgu, ki ponujajo takšno ali drugačno storitev poslovne analitike. Čas Excela in preglednic še sicer ni minil, se pa vse več podjetij zaveda, kako pomembno je spremljati uspešnost, doseganje kzalnikov in vključevati več ljudi iz tima z namenom povečanja transparentnosti, kar pa običajno presega zmožnosti in namembnost trenutnih aplikacij, še posebej namiznih. Srednja in manjša podjetja iščejo SaaS alternative za sodobno poslovno analitiko, saj večina nima ne virov in ne znanja, da bi si sami postavljali podatkovna skladišča in po meri narejeno poslovno analitiko. Na tem mestu vstopi Databox, ki že več kot deset let deluje na tem področju in omogoča manjšim in srednjim podjetjem relativno enostaven način vzpostavitve sodobne poslovne analitike, obdelave podatkov, vizualizacij in poročanja metrik poslovanja.

2 Sodobna poslovna analitika

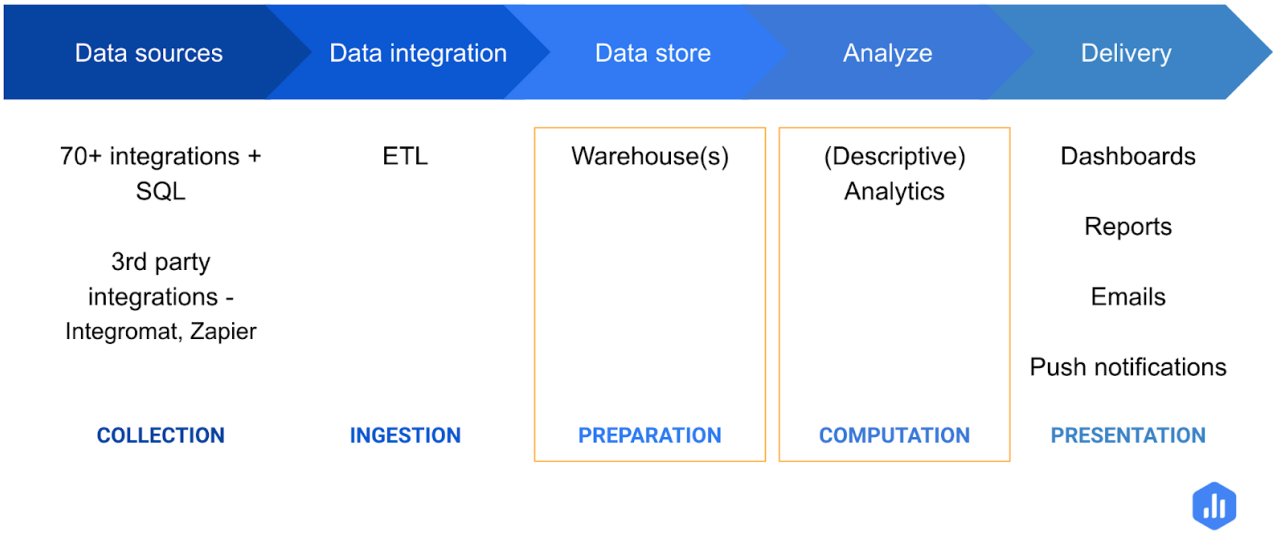
Storitve poslovne analitike se ne dotikajo samo marketinga, ki je sicer tipični uporabnik storitev poslovne analitike, ampak tudi financ, projektnega vodenja, prodaje in še mnogih drugih področij. Za vsako od teh področij najdemo ponudnike storitev s specifičnimi metrikami, lastnimi vmesnike ter drugačnim načinom delovanja. Njihova skupna značilnost pa je to, da jih večina nudi javne programske vmesnike (public API), preko katerih je možno dostopati do podatkov oziroma jih tudi pretakati. Vizualizacija in poročanje sta pomembni komponentni poslovne analitike in Databox omogoča uporabo storitev na poenoten način, agregacijo metrik iz različnih podatkovnih virov in prikaz metrik drugo ob drugi za lažje spremljanje in zaznavo korelacij.

Databox se ukvarja z izzivi in omejitvami v vsakem od korakov podatkovnega toka, pri čemer so ravno omejitve tiste, ki narekujejo način reševanja. Na poslovnem področju imajo podjetja finančne omejitve (stroški), omejitve virov (človeških ali strojnih) in omejitve s strani poslovnih odločitev, ki velikokrat postavijo smernice reševanja določenih težav. V teoriji je možno marsikaj, v praksi ponavadi malo manj. Naš sistem je porazdeljen, dinamičen in asinhron, kar lahko predstavlja dodaten vir izzivov, s katerimi se srečujemo, a hkrati nudi rešitve za to, kar potrebujemo.

3 Podatkovni cevovod in tehnični izzivi

Področje poslovne analitike je nasičeno, v večji meri storitve s tega področja pokrivajo posamezne segmente podatkovnega cevovoda, ena izmed naših prednosti pred konkurenco je pokrivanje celotnega cevovoda, kar pa seboj seveda prinese določene dodatne izzive.

S platformo pokrivamo vseh pet področij (Slika 1): 1) podatkovne vire, 2) integracijo podatkov, 3) shranjevanje podatkov, 4) analizo in 5) dostavo. Pri tem se srečujemo z raznimi tehničnimi izzivi, ki se dotikajo več tehnoloških področij.



Slika 1: Področja podatkovnega cevovoda, ki jih pokriva platforma Databox.
Vir: lasten.

3.1 Pridobivanje podatkov iz podatkovnih virov

Podatkovne vire v našem primeru predstavljajo ponudniki zunanjih storitev (angl. *3rd party service providers*), ki so integrirane v našo platformo in iz katerih črpamo podatke za metrike, ki se potem agregirajo pri nas za lažjo nadaljnjo obdelavo in hranjenje.

Na tem področju se srečujemo z izzivi milijonov dnevnih podatkovnih zahtev z odzivi od nekaj do tudi milijonov vrstic. Zaradi omenjenega števila zahtev se lahko hitro zgodi, da naletimo na limite zahtev, ki jih postavljajo ponudniki in so lahko tako na osnovi uporabnika kot tudi globalne. Ko pridobimo ustrezne odzive, pa je pomembno, da le-te tudi ustrezno shranimo in pripravimo na nadaljnjo obdelavo - za ETL (angl. *extract transform load*).

Soočamo se tudi s specifikami in nekonsistentnostjo različnih ponudnikov podatkov, kjer je pomanjkanje formalnih modelov za predstavitev aktivnosti ETL, ki bi mapirali vhodne podatke virov, da bi bili v primernem formatu za nalaganje v ciljna podatkovna skladišča [1]. Ker je teh veliko se moramo prilagajati različnim aplikacijskim vmesnikom, spreminjajočimi se specifikacijami, omejitvam glede pridobivanja metrik ter limitami ter tudi različnimi časovnimi razponi [2]. Med glavnimi omejitvami so predvsem omejitve glede pridobivanja zgodovine metrik, saj vsi ne omogočajo pridobivanja zgodovine ali pa je le-ta omejena, kar nas naredi dovzetnejše za posledice izgube podatkov pri nas. Pri delu z zunanjimi ponudniki storitev za beleženje analitičnih podatkov se srečujemo z različnimi vrstami, strukturami in formati podatkov, predvsem pa tudi z različnimi interpretacijami pomena posameznih metrik.

3.2 Integracija podatkov

Ko pridobimo podatke s strani ponudnikov, sledi pomemben in hkrati najbolj zahteven del procesa priprave podatkov - ETL. V tej fazi se moramo zavedati različnosti naših podatkovnih virov, nabora ponujnenih metrik ter njihovih specifik in na koncu tudi samih časovnih intervalov podatkov. Pri tem pa zelo pomembno vlogo igrajo tudi sami časovni pasovi, ki se lahko razlikujejo tako med samimi podatkovnimi viri kot pa samim izhodiščnim časovnim pasom naših uporabnikov.

V tej fazi je naša glavna naloga, da znamo pridobljene podatke ekstrahirati iz samih odzivov na kar se da enoten način, poskrbeti da metrike ustrezno obravnavamo (različni tipi metrik), ter jih transformirati v obliko, ki bo olajšala obdelavo, shranjevanje ter omogočila ponovno uporabo tako za platformo kot podatkovno rudarjenje.

3.3 Shranjevanje podatkov

V tej fazi smo že poskrbeli za ustrezno obliko podatkov, primerno za shranjevanje. Potrebno se je zavedati da gre v tem primeru za velike količine podatkov, za katere moramo zagotoviti tako hitro shranjevanje kot tudi kasneje branje. Pri tem pa so pomembne tudi programske optimizacijske tehnike, kjer se je potrebno zavedati narave naših metrik in posledično podatkov. Glede na tip metrike se lahko namreč zgodi, da se vrednosti določenih podatkov lahko s časom spreminjajo, medtem ko so v drugih primerih večni.

Poleh samih podatkov pa se moramo zavedati tudi količine le-teh. S številom uporabnikov in njihovih podatkovnih virov se število podatkov nenehno veča in naša odgovornost je, da skrbimo za različne metode skaliranja podatkovnih shramb, s tem pa ohraniti dobro uporabniško izkušnjo.

3.4 Analiza podatkov

Preden lahko uporabniku dostavimo podatke je potrebno opraviti še enega izmed pomembnejših korakov, in sicer sama analiza podatkov, kalkulacija in priprava le-teh za vizualizacije. Izziv pri tem predstavljajo predvsem raznovrstni tipi metrik, njihove enote ter specifike njihovih podatkovnih virov, kot so različni časovni pasovi ter sam preklop na poletni/zimski čas. Pri tem pa omogočamo pa tudi različne agregacije ter izračunane metrike (iz različnih metrik).

3.5 Dostava (vizualizacija) podatkov

Pri vizualizaciji in poročanju se srečujemo spet z drugim naborom izzivov, ki jih s seboj prinesejo pravočasna priprava podatkov za vizualizacijo, asinhronost sistema, kjer so podatki eventualno konsistentni, zagotavljanje dobre uporabniške izkušnje in pravočasno dostavljanje poročil po različnih distribucijskih kanalih.

4 Rešitve s sklopu platforme Databox

V sklopu platforme smo reševali izzive z vseh prej omenjenih področij, da bi pokrili potrebe naših uporabnikov in zagotovili najboljšo uporabniško izkušnjo. V nadaljevanju na kratko opisujemo pristope k iskanju in uporabi rešitev v najbolj kritičnih zgoraj omenjenih korakih.

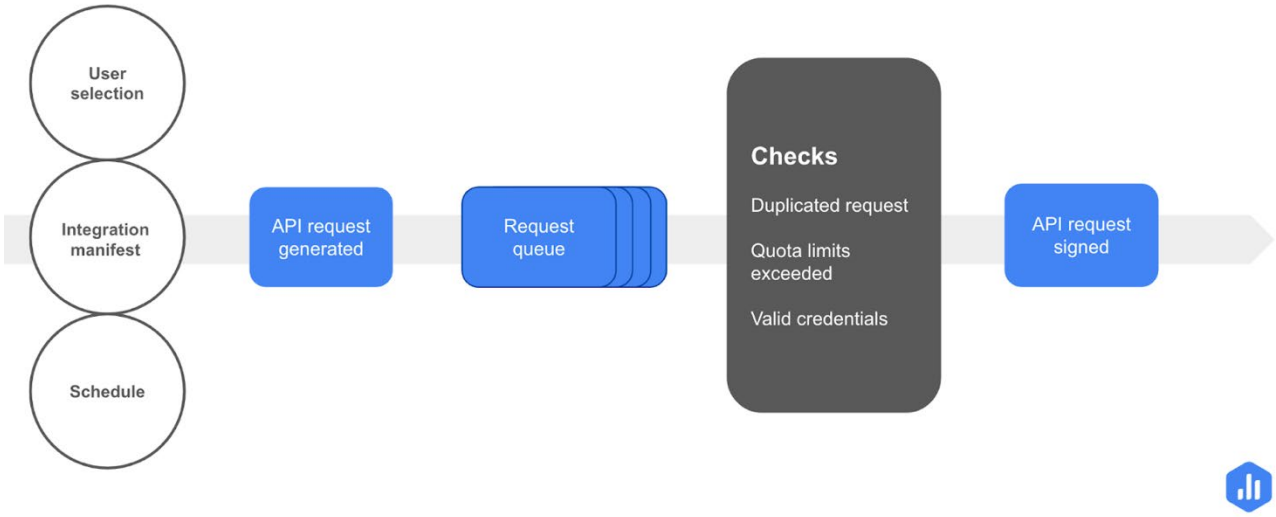
4.1 Pridobivanje podatkov iz podatkovnih virov

Problematike pridobivanja ogromne količine podatkov se lotevamo tako, da pridobivamo oziroma sinhroniziramo le metrike, ki jih je uporabnik izbral in jih dejansko uporablja (pregleduje) v aplikaciji. Na tak način smo zagotovili, da imamo podatke, ki jih uporabnik dejansko potrebuje. S tem sistema ne obremenjujemo po nepotrebnem, hkrati pa skrbimo za optimizacijo stroškov za infrastrukturo. Tudi podatke, ki smo jih že sinhronizirali, ne sinhroniziramo ponovno oz. sinhroniziramo le tiste, katerih narava nam ne omogoča trajne shrambe (metrikam se lahko spreminja zgodovina ali vrednost skozi čas). V ozadju je kompleksen sistem shranjevanja nastavitev posameznih metrik, s katerim smo dosegli relativno optimiziran proces sinhronizacije podatkov.

Poslužujemo se tudi razporejanja in prioritizacije zahtev glede po urniku in glede na izbrane plane naših uporabnikov, običajno so prioritizirani uporabniki, ki plačujejo. Da bi preprečili doseganje limit števila zahtevkov

pri ponudnikih zunanjih storitev, moramo znati zahteve tudi zakasniti in razporediti čez čas, da imajo uporabniki karseda aktualne podatke. S problemom večjih količin odzivov se soočamo tako, da začasno shranjujemo zahteve, dokler ne pridejo na vrsto za obdelavo in shranjevanje v podatkovno skladišče.

Slika 2 prikazuje korake v procesu pridobivanja podatkov in implementirane točke za preverjanje in optimizacijo. Glede na izbiro metrik s strani uporabnika, meta podatki o metriki in podatkovnem virut v integracijskem manifestu in urnikom predvidenega pridobivanja podatkov zgeneriramo zahteve in jih damo v vrsto za izvajanje. Pred dejanskim pošiljanjem zahtev na ponudnike storitev izvedemo preverjanja, da izločimo zahteve, ki jih ni potrebno pošiljati ali jih ne moremo. Če zahteva prestane validacijo, jo ustrezno opremimo in pošljemo naprej do ponudnikov storitev.

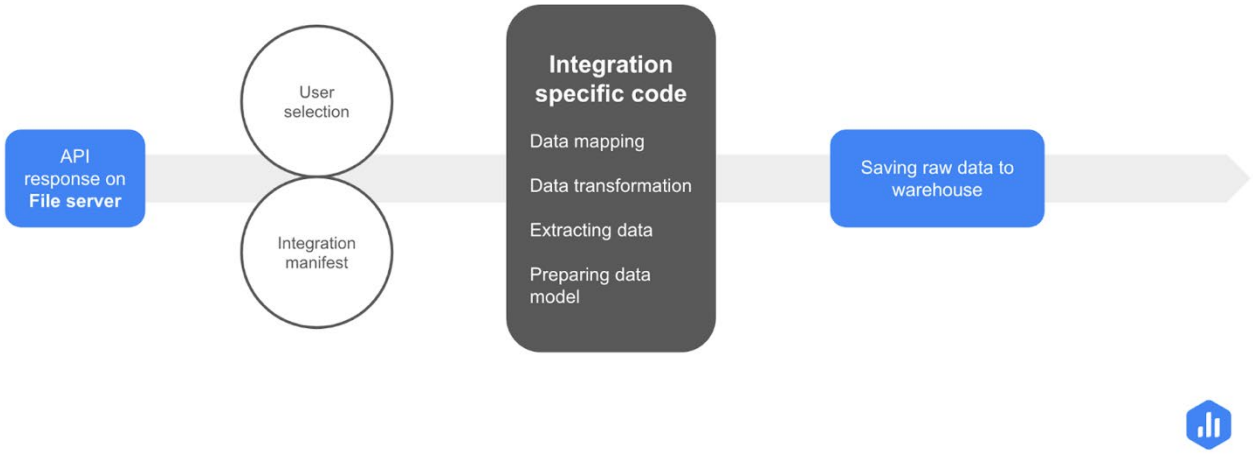


Slika 2: Proces pridobivanja podatkov iz različnih podatkovnih virov in implementiranimi točkami preverjanja. Vir: lasten.

4.2 Integracija podatkov

Raznolikost ponudnikov zunanjih storitev pridobivanja analitičnih podatkov ter njihovih metrik smo rešili z uvedbo integracijskega manifesta, preko katerega poskrbimo za harmonizacijo podatkov. Vsebuje tudi navodila za procesiranje vhodnih odzivov in preslikavo vhodnih podatkov v format podatkov, ki ga uporablja naš sistem. Na tak način smo podatke poenotili in našim uporabnikom omogočili, da metrike iz različnih virov uporabljajo na enak način, jih kombinirajo in relativno enostavno uporabijo na svojih preglednih ploščah z metrikami (angl. *dashboard, databoard*).

Slika 3 ilustrira procesiranje pridobljenega odziva s strani podatkovnega vira, ki ga glede na uporabnikovo izbiro metrik in integracijskega manifesta spustimo skozi kodo, ki je specifična za izbrano integracijo, podatke ustrezno preslikamo, transformiramo, in izluščimo podatke, ki jih nato shranimo v podatkovno skladišče.



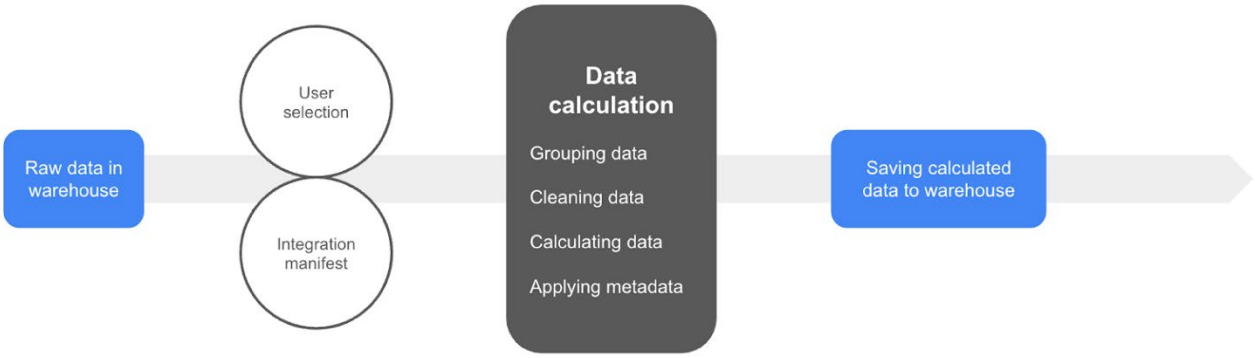
Slika 3: Proces integracije pridobljenih podatkov v platformo.
Vir: lasten.

4.3 Shranjevanje podatkov

Problema reševanja shranjevanja podatkov iz različnih virov smo se lotili z uporabo poenotenega podatkovnega modela, s katerim nam je uspelo vsesplošno raznolikost metrik postaviti na isti skupni imenovalec. To smo uspeli tako, da smo zajeli skupen minimalni nabor podatkov, ki jih potrebujemo, da lahko omogočimo nadaljnjo obdelavo in analizo podatkov na preprost in enoten način.

4.4 Analiza podatkov in vizualizacija

Za pripravo, analizo podatkov in hitrejši prikaz uporabljamo posebno storitev, ki uporabi navodila na podlagi uporabnikove izbire na grafičnem vmesniku, da zbere in pridobi shranjene podatke o metriki, jih primerno združi, izvede potencialne izračune (npr. agregacijo) in vrne izračunane podatke odjemalcu za prikaz (Slika 4). Pri tem upoštevamo metapodatke z informacijo o poteku veljavnosti podatkov, da optimiziramo osveževanje podatkov na podlagi veljavnosti in dejanskega obstoja izračunanih podatkov.



Slika 4: Proces analize podatkov in priprava na vizualizacijo.
Vir: lasten.

Tako pripravljeni podatki so potem uporabljeni s strani odjemalca, ki jih lažje in hitreje vizualizira in se ne rabi ukvarjati še s transformacijo.

5 Odprti izzivi

V sodobni analitiki nikoli ne zmanjka novih. Z zgoraj opisanimi implementiranimi rešitvami smo sicer določene probleme odpravili, druge blažimo oziroma nadzorujemo, imamo pa tudi še nekaj odprtih izzivov, ki so običajno pridruženi rasti podjetja. Mednje sodijo upravljanje naraščajočih obremenitvev sistema zaradi vedno več uporabnikov in novih integracij, skaliranje arhitekture, uvedba in izvajanje procesov odzivanja na nepričakovano delovanje sistema, priprava okvirjev referenčnih performančnih vrednosti, in preprečevanje izgube podatkov. S temi izzivi se ukvarjajo različni inženirski timi na Databoxu, ki skrbijo za posamezne domene.

6 Zaključek

Trenutna platforma za poslovno analitiko v oblaku pokriva večino potreb naših uporabnikov, vendar se nenehno trudimo prehiteti konkurenco z predvidevanjem potreb uporabnikov in uvajanjem sodobnih pristopov, med drugim tudi z apliciranjem podatkovnih ved. Skoraj vsakih nekaj mesecev se pojavi nova storitev, ki uporabnikom omogoča poenostavljeno spremljanje poslovanja, zato je še toliko pomembnejše, da jim zagotovimo dodatno dodano vrednost na podlagi zbranih podatkov, podatkovnega modeliranja, prediktivne analitike in drugih sodobnih pristopov.

Literatura

[1] KIMBALL R., CASERTA J. "The Data Warehouse ETL Toolkit. Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data", Wiley, 2004.

[2] LOSHIN David "Data Integration", MK Series on Business Intelligence: Business Intelligence (Second Edition), Morgan Kaufmann, 2013, 189-210.

Umestitev digitalnih (EU) denarnic v ekosistem sodobnih IKT rešitev

Špela Čučko, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
spela.cucko@um.si, muhamed.turkanovic@um.si

Sinopsis Evropska komisija je junija 2021 predlagala novo uredbo eIDAS 2.0, s katero si prizadeva zagotoviti zaupanja vredno, varno in široko uporabno evropsko digitalno identiteto, ki bo vsem Evropejcem omogočila digitalno dokazovanje svoje identitete z namenom dostopanja do digitalnih storitev po vsej Evropski uniji. V okviru uredbe je izdala zahtevo, da morajo države članice EU do leta 2023 svojim državljanom ponuditi kriptografsko varno digitalno denarnico z EU digitalno identiteto, ki naj bi poleg identifikacije, overjanja, avtorizacije in dostopanja do javnih in / ali zasebnih storitev, omogočala tudi varno shranjevanje, upravljanje in deljenje identitetnih podatkov in poverilnic. Kljub prizadevanju za vpeljavo takšne denarnice obstaja še vedno veliko nejasnosti in odprtih vprašanj glede tehničnih podrobnosti. Poleg tega pa se mnogim postavljajo vprašanja o tem, kaj pravzaprav so digitalne denarnice, komu so namenjene, kakšne scenarije uporabe podpirajo ter kako bodo podpirale digitalne procese in z njimi povezano digitalno preobrazbo v trenutnem, širšem ekosistemu IKT rešitev in storitev. V prispevku bomo tako predstavili tehnično ozadje denarnic digitalnih identitet v okviru Evropske digitalne identitete in posodobljene Uredbe eIDAS. Opredelili bomo osnovne koncepte in tipe denarnic, opisali njihove funkcionalne zahteve, predstavili številne scenarije uporabe ter raziskali interakcije denarnic z obstoječimi IKT rešitvami ob upoštevanju standardov in dobrih praks na področju upravljanja identitet.

Ključne besede:
digitalna identiteta
digitalne denarnice
Evropska komisija
digitalizacija
interoperabilnost
preverljive poverilnice
samoupravljana identiteta

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.3

1 Uvod

Po enoletnem posvetovanju z državami članicami, je junija 2021 Evropska komisija izdala obsežno poročilo o analizi Uredbe eIDAS (storitev elektronske identifikacije in zaupanja), kjer je hkrati naznanila korak naprej k eIDAS 2.0 oz. prenovi in modernizaciji v smeri t. i. ogrodja Evropske digitalne identitete. Posebej velja izpostaviti, da eIDAS 2.0 predvideva zasnovo in nastajanje t. i. digitalnih EU denarnic za upravljanje z dokumenti. Sklenjen je bil tudi akcijski načrt, ki veleva, da morajo do septembra 2023 vse države članice EU svojim državljanom ponuditi digitalno denarnico z EU digitalno identiteto, kar prinaša mnogo prednosti, kot so digitalizacija javnih procesov tudi na nivoju EU, nadzor nad deljenjem lastnih podatkov, preprostejše dokazovanje (digitalne) identitete itn. Pri čemer se postavlja vprašanje o tem, kaj točno so digitalne denarnice oz., kdo, kako in zakaj bi jih uporabljal, kakšne so njihove tehnične podrobnosti ter kako bodo omogočale omenjene prednosti v trenutnem, širšem ekosistemu IKT rešitev in storitev.

V prispevku bomo predstavili tehnično ozadje digitalnih denarnic v okviru Evropske digitalne identitete in posodobljene Uredbe eIDAS. Začeli bomo s predstavitvijo osnovnih konceptov in tipov digitalnih denarnic, s pogledom na funkcionalnosti, ki naj bi jih omogočale (digitalno podpisovanje, overjanje na osnovi QR oznak, selektivno vendar preverljivo deljenje informacij itn.). Poznamo številne tipe digitalnih denarnic, od oblačnih do denarnic, ki temeljijo na konceptu samoupravljanih in decentraliziranih identitet (angl. Self-Sovereign Identity - SSI). Zaradi fokusa na digitalno identiteto morajo digitalne denarnice podpirati upravljanje in nadzorovanje le teh, pri čemer morejo omogočati varno shranjevanje in upravljanje identifikatorjev, zasebnih ključev, podatkov in poverilnic, ki naj bi bile zaščitene in popolnoma pod nadzorom uporabnika. Poleg omenjenega pa naj bi omogočale tudi shranjevanje in upravljanje digitalnih dokumentov (kot so npr. osebna izkaznica, diploma, bančna kartica, vozniško dovoljene, certifikat cepljenja), ki so povezani z digitalnimi identitetami. Dokumenti bodo shranjeni v obliki t. i. preverljivih poverilnic (angl. Verifiable Credentials - VC), ki omogočajo preprost nadzor, preverjanje in selektivno razkritje informacij.

V prispevku bomo predstavili tudi številne scenarije uporabe, ki bodo razsvetlili prihodnje načine digitalnih procesov in s tem povezane digitalne preobrazbe. Prav tako bomo predstavili nujnost posodobitve trenutnih IKT rešitev in storitev z namenom doseganja kompatibilnosti z novimi pristopi (overjanja, avtorizacije, komunikacije, izmenjave dokumentov). Pri tem se bomo sklicevali na obstoječe in nove standarde (DID in VC – W3C) in predloge le teh ter določene nove dobre prakse.

2 Digitalne identitete

Digitalna identiteta predstavlja ključen element digitalnih interakcij. Predstavlja sredstvo za identifikacijo, overjanje in avtorizacijo ter omogoča dostopanje do spletnih storitev. Poznamo različne modele upravljanja identitet (angl. Identity Management - IdM), pri čemer večina vključuje tri entitete, in sicer *uporabnika* oz. imetnika identitete, *ponudnika identitete* (angl. Identity Provider - IdP) oz. ponudnika/izdajatelja identitetnih atributov in *ponudnika storitev* (angl. Service Provider - SP) oz. preveritelja (Slika 1).

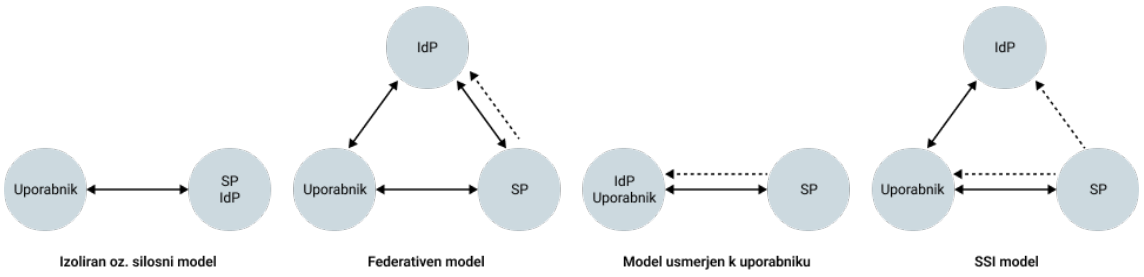
Uporabnik je praviloma fizična oseba z vsaj eno digitalno identiteto, ki želi izvesti digitalno transakcijo in / ali dostopati do storitev. *Ponudnik identitete* je entiteta, ki je zadolžena za administracijo in upravljanje digitalne identitete ter za izvajanje identifikacije in overjanja. Za vsakega novega uporabnika registrira identifikatorje in atribute ter skladno s svojo politiko preveri resničnost podanih identitetnih podatkov (npr. preverjanje osebne izkaznice, dokaz o prejemu elektronske pošte ipd.). *Ponudnik storitev* je entiteta, ki uporabniku zagotavlja storitev in se zanaša na ponudnika identitete za identifikacijo in overjanje uporabnika [1].

Najenostavnejši in najgosteje uporabljen model IdM predstavlja t. i. *izoliran oz. silosni model* (angl. Isolated or Silo Model), ki vključuje zgolj dve entiteti, uporabnika in SP, ki je poleg zagotavljanja storitev, odgovoren tudi za administracijo, shranjevanje ter upravljanje identitetnih podatkov. Uporabnik si mora posledično ustvariti identitetno oz. uporabniški račun pri vsakem izmed SP, kar zahteva večkraten vnos identitetnih podatkov [4],

predstavlja nekonsistentno uporabniško izkušnjo in veliko obremenitev za uporabnika [2] ter prinaša varnostna tveganja, povezana tudi z uporabo podobnih ali enakih gesel z nizko entropijo v različnih sistemih [1, 3, 4].

Rešitev omenjenega problema predstavlja *centraliziran federativen model*, ki uvaja zunanjega IdP, ki služi kot posrednik med uporabnikom in različnimi SP ter zagotavlja funkcionalnosti, povezane z upravljanjem digitalne identitete. Uporabniku omogoča, da si ustvari identiteto pri osrednjem IdP, pri čemer lahko identifikator in pripadajočo poverilnico uporabi pri overjanju z različnimi SP. Z uporabo mehanizma enotne prijave (angl. Single Sign On - SSO) mu je tako omogočen dostop do vseh storitev, ki so odvisne od istega ponudnika identitete [1, 5]. Uporabnik se more posledično registrirati zgolj pri peščici IdP, ki služijo svojemu naboru SP [2]. Ta model je enostavnejši za uporabo in ponuja boljšo uporabniško izkušnjo, vendar centralizacija predstavlja dodatno ranljivost. Razkritje enega identifikatorja in pripadajoče poverilnice namreč zadostuje za dostop do vseh storitev naenkrat [1]. Poleg omenjenega pa neposredna vključenost v proces overjanja, omogoča IdP sledenje uporabnikom in učenje njihovih vedenj [2]. Podobno kot centraliziran, tudi *decentraliziran federativen model* temelji na zunanjih IdP in omogoča implementacijo SSO. Model zahteva vzpostavitev zaupanje med različnimi IdP in SP, ki so združeni v t. i. krog zaupanja (angl. Circle of Trust - CoT), ki pogosto temelji na sporazumih in skupni tehnološki platformi [1]. IdP, združeni v CoT, si lahko medsebojno delegirajo zahtevo za overjanje [2]. Posledično so funkcije IdM [5] in uporabniški podatki porazdeljeni med različnimi ponudniki znotraj CoT, kar zmanjšuje varnostna tveganja. Uporabniki se lahko overijo zgolj pri enem izmed IdP za dostopanje do storitev znotraj CoT [1]. Dobro znan primer tega modela je evropski, interoperabilni okvir eIDAS (angl. electronic IDentification, Authentication and Trust Services), ki združuje nacionalne sisteme IdM držav članic Evropske unije (EU) ter omogoča čezmejno elektronsko identifikacijo, overjanje in storitve zaupanja [2]. Interakcije med IdP in SP ter končnimi uporabniki so bile v *centraliziranem in decentraliziranem federativnem modelu* poenostavljene iz vidika upravljanja identifikatorjev in poverilnic ter standardizirane preko izmenjave žetonov, kot so SAML, OAuth, OpenID Connect (OIDC), itd. [3]. Pri čemer pa nadzor nad uporabniškimi podatki (identifikatorji in atributi) še vedno ostaja na strani IdP in SP, ki identitetne podatke shranjujejo, obdelujejo in tudi distribuirajo. Medtem ko je uporabnik primoran zaupati, da bodo spoštovane njegove pravice in zasebnost [1].

Za razliko od prejšnjih modelov, ki podatke uporabnika shranjujejo centralizirano, bodisi pri IdP ali SP, se v okviru modela, osredotočenega na uporabnika (angl. User-Centric model), podatki shranjujejo v uporabnikovi domeni (npr. na pametni kartici ali na mobilnem telefonu, kjer so zaščiteni z varnostnimi elementi na osnovi strojne opreme), kar povečuje varnost in zasebnost. Podatki so pod uporabnikovim nadzorom in so posredovani SP ob vsakem overjanju. Primeri takšnih rešitev so npr. nacionalne IdM rešitve [2]. eIDAS npr. za uporabo kvalificiranih digitalnih podpisov zahteva uporabo kvalificiranih naprav, kot so kriptografske kartice in USB-ji.



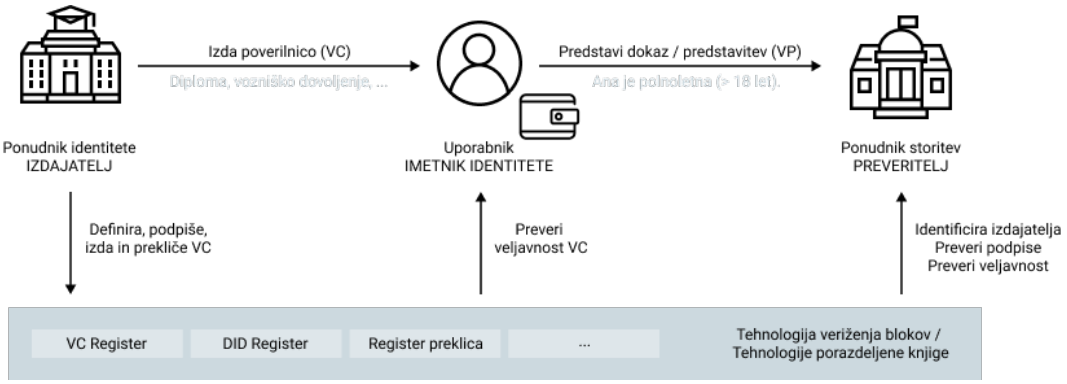
Slika 1: IdM modeli. Neprekinjene črte predstavljajo interakcije, črtkane črte pa zaupanje.
Vir: [3].

2.1 Decentralizirane in samoupravljane identitete

Naslednji korak na področju IdM predstavljajo decentralizirane in samoupravljanje identitete (angl. Self-Sovereign Identity - SSI). Gre za k uporabniku usmerjen decentraliziran pristop, ki entitetam oz. uporabnikom (posameznikom, organizacijam, stvarem) omogoča, da v celoti nadzirajo in upravljajo svojo digitalno identiteto (identifikatorje in povezane identitetne podatke) [13], brez da bi se v interakcijah zanašali na zunanje entitete [14]. Med uporabnikom in drugo entiteto tako ni nobenega posrednika, obstaja zgolj Peer-to-Peer povezava, preko katere si izmenjujeta podatke. Za razliko od ostalih modelov je v okviru modela SSI, vsak uporabnik sam odgovoren za administracijo, shranjevanje, upravljanje in distribucijo identitetnih podatkov. Uporabnik je tako v središču digitalnih interakcij in nadzoruje pretok svojih podatkov ter odloča kdaj, s kom in katere podatke želi deliti. Uporabniku ni potrebno ustvariti računa pri vsakem izmed SP, prav tako ni potrebe po IdP, ki bi uporabniku zagotovili identiteto in njeno upravljanje. Identitetni podatki se shranjujejo v domeni uporabnika, v digitalnih denarnicah, ki so pod popolnim nadzorom uporabnika.

Tudi v SSI modelu so ključne tri entitete, ki nastopajo v vlogi *izdajatelja* identitetnih atributov (angl. Issuer), *uporabnika oz. imetnika identitete* (angl. Identity Holder) in *preveritelja* (angl. Verifier) oz. ponudnik storitev (Slika 2). Pri čemer je potrebno poudariti, da lahko vsaka izmed entitet v različnih kontekstih deluje v drugi vlogi.

Izdajatelj potrdi določene atribute uporabnika oz. izda in digitalno podpiše t. i. *preverljivo poverilnico*, ki vsebuje enega ali več atributov oz. trditev, ki se nanašajo na *imetnika identitete*. Ta je odgovoren za pridobivanje, shranjevanje, upravljanje in deljenje svojih identitetnih podatkov s *preveriteljem*, ki običajno od uporabnika zahteva dokazilo o njegovi identiteti. Slednje omogoča identifikacijo in verifikacijo ter zagotavljanje dostopa do želenih storitev. Proces preverjanja ne zahteva vključitve izdajatelja poverilnic [23], zaupanje se lahko namreč vzpostavi z uporabo *decentraliziranih identifikatorjev* (angl. Decentralized Identifiers - DID), *preverljivih poverilnic* (angl. Verifiable Credentials - VC) in *tehnologije verižnja blokov* (angl. Blockchain, BC) (ali tehnologije porazdeljene knjige (angl. Decentralized Ledger Technology - DLT) ali druge decentralizirane tehnologije), ki predstavljajo ključne komponente SSI arhitekture in bodo predstavljene v nadaljevanju. Poleg omenjenega pa je bistvena komponenta v SSI ekosistemu, *digitalna denarnica*, s pomočjo katere uporabniki upravljajo svoje digitalne identitete in bodo podrobneje naslovljene v poglavju 4.



Slika 2: Interakcije in pretok podatkov med izdajateljem, imetnikom identitete in preveriteljem.
Vir: lasten.

Decentralizirani identifikatorji (DID) so globalni, edinstveni, preverljivi in trajni digitalni identifikatorji, popolnoma neodvisni od centraliziranih registrov, zunanjih entitet ali ponudnikov identitete, ki bi zagotavljala njihovo administracijo, upravljanje in obstoj. Omogočajo samostojno izvajanje operacij registracije, posodabljanja, razrešitve in preklica, brez centralizirane registracije, overjanja in avtorizacije. Omogočajo ustvarjanje edinstvenih, zasebnih in varnih povezav med dvema entitetama. Pri čemer si lahko vsaka entiteta ustvari in upravlja poljubno število DID-ov ter jih ločeno uporablja v različnih digitalnih interakcijah in kontekstih, kar preprečuje korelacijo podatkov. Vsak DID je predstavljen v obliki URI-ja in je povezan z DID subjektom (imetnikom identitete) in

DID dokumentom, ki opisuje subjekt, javne ključe, biometrične podatke in vse druge mehanizme, ki se uporabljajo za preverjanje prisotnosti lastništva nad DID-i [11].

Preverljive poverilnice (VC), lahko podobno kot fizične, vsebujejo informacije, povezane z identifikacijo subjekta oz. imetnika identitete (npr. ime in priimek, identifikacijska številka, fotografija) in nabor trditev o subjektu (npr. državljanstvo, datum rojstva), skupaj z metapodatki, kot so izdajatelj (npr. Upravna enota Maribor), vrsta (npr. osebna izkaznica) in omejitve poverilnice (npr. obdobje veljavnosti, pogoji uporabe itn.), ter podatki o mehanizmu preverjanja in preklica. VC torej vsebuje metapodatke in eno ali več trditev oz. atributov ter je digitalno podpisan s strani izdajatelja, ki s podpisom potrjuje resničnost podatkov v VC. Uporaba kriptografskih mehanizmov, VC ščiti pred nedovoljenimi posegi ter omogoča *kriptografsko preverljivost*. Imetniki VC lahko ustvarijo preverljive predstavitve (angl. Verifiable Presentations - VP) in jih delijo s preveritelji z namenom dokazovanja identitete in identitetnih atributov [10]. Pri čemer lahko z uporabo metod, kot sta *selektivno razkritje* (angl. Selective Disclosure) in *dokaz o ničelnem znanju* (angl. Zero-Knowledge Proof - ZKP) ohranjamo zasebnost v interakcijah z drugimi entitetami. Z uporabo selektivnega razkritja lahko ustvarimo VP, sestavljen zgolj iz podmnožice atributov iz ene ali več poverilnic. Na drugi strani uporaba ZKP omogoča dokazovanje atributov, brez dejanskega razkritja vrednosti. Če moremo na primer dokazati svojo starost, lahko s pomočjo selektivnega razkritja delimo zgolj leto rojstva iz vozniškega dovoljenja, pri čemer pa ne razkrijemo preostalih atributov, kot so dan in mesec rojstva ter naslov itn. Z uporabo ZKP pa lahko dokažemo, da smo npr. starejši od 18 let, ne da bi razkrili datum svojega rojstva. Omenjeno je še posebej koristno v situacijah, ko ne zaupamo preveritelju.

Tehnologija verižnega blokov ali DLT lahko v okviru SSI služi kot preverljiv register podatkov (angl. Verifiable Data Registry - VDR) in vzpostavlja zaupanje med različnimi entitetami. V VDR se lahko shranjujejo (i) javni DID-i, (ii) definicije VC-jev, (iii) sheme, (iv) podatki o stanju VC-jev (register preklica) in (v) dokazi o interakcijah med entitetami. Posledično lahko deluje kot nadomestilo za centralizirani organ za registracijo v tradicionalnih sistemih za upravljanje identitete ter zagotavlja decentralizirano infrastrukturo javnih ključev (angl. Decentralized Public-key Infrastructure - DPKI) [26] ter shranjuje povezavo med identifikatorjem in metodo overjanja [23]. V takšnih DID registrih so shranjeni javni DID-i (angl. public DID) organizacij oz. izdajateljev VC-jev.

3 Evropske digitalne identitete

EU ima splošno znano kompleksno strukturo tako iz pravnih, ekonomskih in drugih sektorjev kakor tudi na področju upravljanja z digitalnimi identitetami (IdM). Vsaka država članica vodi lastno politiko IdM, pri čemer se loteva le te tudi s tehničnega vidika, na različne načine. Določene države, kot so npr. Estonija, Danska in Nemčija, imajo zelo dodelane politike in tehnično podporo IdM, med tem, ko imajo druge to poenostavljeno. Prav tako ima vsaka država članica drugačno filozofijo upravljanja, ki vključuje različne zaupne kvalificirane IdP in SP. Estonci tako ponujajo svojim državljanom številne načine upravljanja digitalnih identitet, kot npr. ID Card, DIGI-ID, RP-card. Med tem so v Avstriji izdali ID Austria, ki omogoča različne funkcionalnosti IdM, podobno novim osebnim izkaznicam v Sloveniji. Slovenija se je s storitvijo SI-PASS pomembno pozicionirala, saj smo pokazali, kako pod eno streho omogočiti različne ravni zaupanja, različne načine identifikacije itn.

Ne glede na dovršenost posameznih pristopov znotraj EU so le ti heterogeni in povzročajo preglavico ideji povezane EU, ki pa to ne more biti le v fizični obliki, temveč nujno tudi v digitalni. Evropska komisija (EK) je tako v prejšnjem desetletju sprejela idejo Enotnega digitalnega trga Evrope (angl. Single Digital Market) ter izdala uredbi eIDAS in SDGR (angl. Single Digital Gateway Regulation).

3.1 Uredba o elektronski identifikaciji in storitvah zaupanja - eIDAS

eIDAS je od leta 2014 glavni zakonodajni akt Evropske unije v zvezi z elektronskim podpisom in elektronskim poslovanjem. Navezuje se na t. i. eIDAS uredbo oz. Uredba (EU) št. 910/2014 - Uredba o elektronski identifikaciji in storitvah zaupanja [19], ki je osnova Zakonu o elektronski identifikaciji in storitvah zaupanja (ZEISZ) v

Republiki Sloveniji. Uredba in zakon urejata elektronsko identiteto posameznika ali podjetja, ki jo posamezna država članica EU dodeli svojim državljanom ali poslovnim subjektom. Del tega je tudi umestitev sredstev za elektronsko identifikacijo in podpisovanje, s katerim se prej omenjena identiteta dokazuje v pravni in tehnični prostor. Poglavitna ideja eIDAS je ta, da lahko ljudje in podjetja znotraj EU uporabljajo lastne nacionalne elektronske identifikatorje (eID) za dostop do (javnih) storitev, ki so na voljo na spletu v drugih državah EU. Primer teh storitev so oddaja davčnih napovedi, vpis na tujo univerzo, odpiranje bančnega računa na daljavo, ustanovitev podjetja v drugi državi članici, preverjanje pristnosti za internetna plačila, spletno zbiranje ponudb, javni razpisi idr. eIDAS tako opredeljuje interoperabilni nivo za združevanje obstoječih nacionalnih rešitev elektronske identifikacije, kot so slovenska SIGEN-CA ali SI-PASS, avstrijska državljanska kartica, belgijska eID kartica itn.

eIDAS prav tako uvaja regulatorni okvir, ki v primeru kvalificiranih digitalnih certifikatov in s tem povezanih podpisov, le te pravno enači z lastnoročnim podpisom. Takšna vrsta pravnih zagotovil je možna zgolj v primeru kvalificiranih digitalnih podpisov (angl. Qualified Electronic Signature - QES), za katere eIDAS zahteva uporabo t. i. kvalificiranih naprav za ustvarjanje kvalificiranega elektronskega podpisa (angl. Qualified Electronic Signature/Seal Creation Device - QSCD). Primer takšnih naprav v primeru lokalne ali centralizirane uporabe so kriptografske kartice in kriptografski USB ter v primeru oddaljene uporabe so to varnostni moduli strojne opreme (angl. Hardware Security Module - HSM). Razen QES uvaja eIDAS tudi napredne digitalne podpise (angl. Advanced Electronic Signature - AdES), ki pa ne zahtevajo uporabe QSCD. S tem povezane so tudi t. i. ravni zanesljivosti (angl. Level of Assurance - LoA), pri čemer definira eIDAS kar tri nivoje: nizka, srednja in visoka raven zanesljivosti. Slednja je definirana ob uporabi kvalificiranih digitalnih podpisov. Prednost različnih ravni zanesljivosti je v tem, da lahko tako uporabnik kot ponudnik storitev, ki ne zahteva visoke ravni zanesljivosti, uporabita tisto z nižjo stopnjo, saj je le ta lažja za pridobiti in upravljati.

Splošna IT arhitektura eIDAS temelji na t. i. nacionalnih eIDAS vozliščih. Vsaka država članica EU upravlja lastno vozlišče, ki je povezano s ponudniki digitalne identitete. Vsako nacionalno vozlišče eIDAS je sposobno identificirati in overiti lastne državljane in sprejemati zahteve za overjanje od SP, ki se nahajajo v isti državi. Da bi omogočili interoperabilnost na nivoju EU, so vsa vozlišča eIDAS povezana ter omogočajo tudi medsebojno avtomatizirano podporo.

EK trenutno ocenjuje eIDAS, pri čemer je leta 2020 že izvedla prvo odprto posvetovanje. Cilj posvetovanja je bil zbrati povratne informacije o gonilih in ovirah za razvoj in uvedbo storitev zaupanja in eID v Evropi, pri čemer se je obravnavala tudi možnost ogrodja za zagotavljanje digitalne identitete EU.

3.2 Ogrodje evropske digitalne identitete

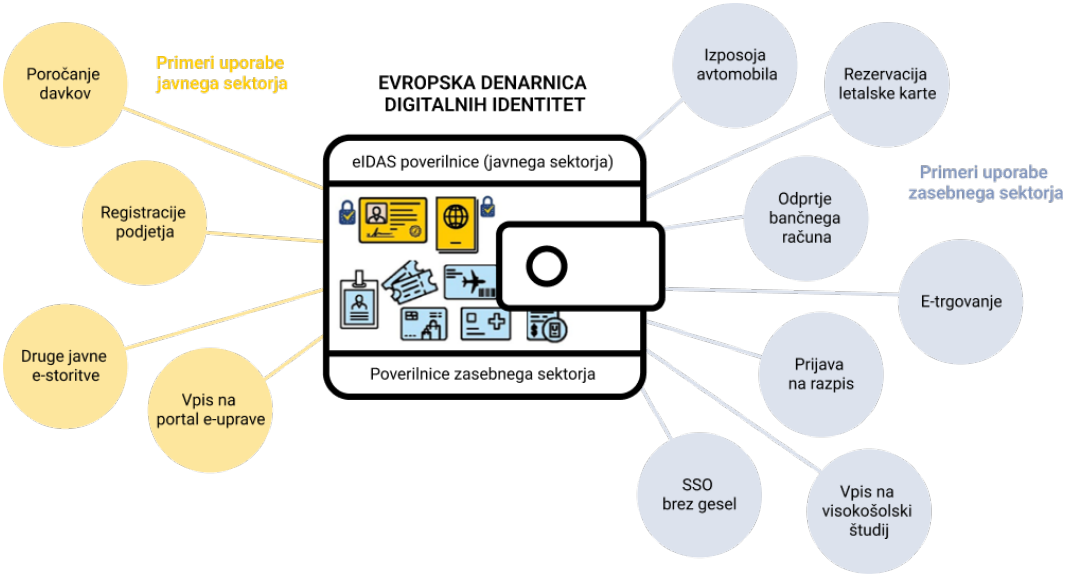
Po letu 2020, ko je EK izvedla širok posvet glede eIDAS, je bilo zaključeno, da je le ta sicer dobro zasnovan in tehnično izpolnjuje svoje osnovno poslanstvo, vendar se na nivoju EU uporablja zgolj 14 % [17]. Razlogov za slabšo sprejetost med prebivalci je mnogo, med njimi tudi dejstvo, da se v določenih državah že nacionalne eID uporabljajo v omejenem obsegu. Ne glede na slednje pa je EK identificirala tudi druge pomanjkljivosti, ki se nanašajo predvsem na vidik uporabniške prijaznosti, nadzora nad lastnimi podatki (tj. zasebnost), ter popolne interoperabilnosti med državami članicami. Slednje je posledica dejstva, da določene države še zmeraj niso oznanile svoje nacionalne sheme (med njimi tudi Slovenija) [18].

Upoštevajoč predstavljena dejstva je EK predlagala novo uredbo, imenovana tudi eIDAS 2.0, za ustvarjanje usklajene zmogljivosti digitalne identitete za vse državljane EU, ki jo je poimenovala tudi Evropske digitalne identitete (angl. European Digital Identity). Predlagana posodobitev si prizadeva uravnotežiti digitalni svet v korist posameznika in manj v korist korporacij, s tem, da ponudi večjo zasebnost, zaupanje in nadzor nad svojimi podatki oz. da se uporabnikom ponudi zaupanja vredno, varno in široko uporabno evropsko digitalno identiteto, ki bo vsem Evropejcem omogočila digitalno dokazovanje svoje identitete z namenom dostopanja do digitalnih storitev

po celotni EU. Ključnega pomena je dejstvo, da se želi graditi na dosežkih in dognanjih eIDAS 1.0 in ne začenjati iz točke nič.

Povzeto so osnovni stebri iskanja izboljšav na osnovi koncepta evropske digitalne identitete: (i) okrepljeni nacionalni sistemi eID znotraj eIDAS; (ii) ponujanje uporabniško nadzorovane digitalne identitete na osnovi digitalnih denarnic; ter (iii) zasebni sektor kot ponudnik digitalnih storitev, ki so povezane z ekosistemom evropske digitalne identitete [20].

Osrednja komponenta nove uredbe so torej t. i. evropske denarnice digitalnih identitet (angl. European Digital Identity Wallet - EUDIW). Te so primarno mišljene kot mobilne denarnice, saj uporabniki po svetu vedno bolj zahtevajo mobilno identifikacijo oz. vedno bolj uporabljajo pametne telefone za različne digitalne storitve (npr. spletno bančništvo, Covid potrdila itn.). Prav tako področje SSI ni ostalo neopaženo s strani EK, saj ponuja morebitno rešitev za številne izzive, ki jih želi EK nasloviti. Izpostaviti je potrebno dejstvo, da je *EK junija 2021 izdala zahtevo, da morajo države članice EU do leta 2023 ponuditi takšno digitalno denarnico vsem svojim državljanom* [22]. S tem želi EK na osnovi eIDAS 2.0 v celoti uresničiti svojo vizijo univerzalne in čezmejne evropske digitalne identitete, pri čemer je tudi zastavljen kazalnik, ki načrtuje, da bo do leta 2030 vsaj 80 % populacije EU imelo registrirano sredstvo elektronske identifikacije [21].



Slika 3: Primeri uporabe EUDIW.
Vir: [15].

EUDIW so torej mišljene kot kriptografsko varne digitalne denarnice, ki bi državljanom omogočile ne zgolj upravljanje svojih digitalnih identitet (kot oblik osebnih izkaznic) ter na osnovi teh identifikacijo, overjanje in avtorizacijo oz. dostop do javnih in / ali zasebnih storitev, temveč tudi varno shranjevanje in upravljanje drugih vrst digitalnih dokumentov oz. poverilnic, kot so vozniška dovoljenja, različne druge potne listine, bančne kartice, klubske izkaznice itn. [17]. V prihodnjih letih bodo EUDIW podpirale vedno večji nabor storitev in primerov uporabe (Slika 3). Že sedaj je jasno, da bo podatkovni model takšnih digitalnih dokumentov temeljil na preverljivih poverilnicah (VC). Dodatna pomembna lastnost EUDIW je tudi omogočanje *selektivnega razkritja atributov*, kot je starost uporabnika. Uporabnik mora imeti možnost s pomočjo EUDIW deliti le izbran, nujen in omejen nabor informacij, ki jih potrebuje potencialen SP. Če na primer uporabljate denarnico za dokazovanje svoje starosti, vam ni potrebno deliti drugih osebnih podatkov, kot so datum rojstva, ime ali naslov. S tehnološkega stališča je takšno funkcionalnost možno doseči z vpeljavo ZKP (angl. Zero-Knowledge Proofs) [16]. Seveda pa bodo EUDIW morale biti skladne tudi z obstoječimi pravnimi okvirji EU, kot so GDPR ter eIDAS, in sicer pri slednjem iz vidika,

da bodo EUDIW morale zagotavljati močno kriptografsko podporo in ponuditi visok nivo zanesljivosti, ki je nuja za zagotavljanje QES ali vsaj AdES.

Države članice si bodo lahko prizadevale za lastno zasnovo, izgled in funkcionalnost EUDIW, pri čemer bo nujna kompatibilnost z osnovnimi zahtevami EK ter posledično popolna interoperabilnost vseh EUDIW, ne glede na državo članico oz. zasebnega ponudnika EUDIW.

4 Tehnično ozadje denarnic digitalnih identitet

Ideja EUIDW ni nova in temelji na *digitalnih denarnicah*, ki že več let predstavlja varno plačilno sredstvo, in posledično, zamenjavo fizične denarnice. Primarno gre za finančne aplikacije za shranjevanje sredstev, izvedbo transakcij in sledenje zgodovini plačil, z uporabo prenosljivih naprav, kot so pametni telefoni, tablice in pametne ure. Omogočajo vnos in varno shranjevanje podatkov o plačilu (npr. podatki o kreditni kartici, debetni kartici ali bančnem računu) in plačevanje brez potrebe po predstavitvi fizičnih sredstev. Poleg omenjenega nekatere denarnice omogočajo tudi digitalizacijo in shranjevanje drugih fizičnih dokumentov, kot so vozniško dovoljenje, osebna izkaznica, razne članske izkaznice, kartice zvestobe, darilne kartice in kupone, pa tudi vstopnice za prireditve, letalske vozovnice in vozovnice za javni prevoz ter hotelske rezervacije [27].

Digitalne denarnice izkoriščajo zmogljivosti mobilnih naprav za izboljšanje dostopa do storitev in produktov ter odpravljajo potrebo po fizičnih denarnicah. Uporabljajo QR oznake ter brezžične tehnologije, kot so Bluetooth, Wifi in magnetne signale (Near-field communication - NFC, Magnetic Secure Transmission - MST), ki omogočajo zanesljive in varne transakcije in s tem prenos plačilnih podatkov med napravami [27]. Poleg znane uporabe plačevanja v trgovinah, restavracijah in na drugih prodajnih mestih, ki uporabnikom omogoča podobno uporabniško izkušnjo, kot pri uporabi brezstične kartice pa nekatere denarnice omogočajo tudi dvig denarja na bankomatih in izvedbo spletnih transakcij (npr. plačevanje preko spleta, potrjevanje spletnih plačil (Two Factor Authentication - 2FA), spletna nakazila ipd.).

Izbira digitalnih denarnic je vedno večja, znana primera sta npr. Google Wallet (Android) in Apple Wallet (iOS), ki podpirata plačevanje na fizičnih plačilnih mestih (NFC), v spletnih trgovinah in aplikacijah, vendar sta omejena na uporabnike pripadajočih operacijskih sistemov. Mobilne denarnice pa ponujajo tudi slovenske spletne banke. Komitentom (i) Nove KBM je na voljo denarnica mDen@rnica, (ii) NLB - NLB Pay, (iii) Banke Intesa Sanpaolo - Wave2Pay in (iv) Sberbank – mBills [28]. Med digitalne denarnice pa spadajo tudi kripto denarnice v obliki strojne ali programske opreme, ki omogočajo shranjevanje javnih in zasebnih ključev, potrebnih za prejemanje in plačevanje s kriptovalutami.

4.1 Osnovni koncepti

Z razvojem tehnologij in identitetnih modelov pa so se pojavile tudi t. i. *denarnice digitalnih identitet* (angl. Digital Identity Wallet - DIW), ki predstavljajo sredstvo, s katerim uporabniki nadzirajo in upravljajo svoje digitalne identitete. Predstavljajo ključen uporabniški vmesnik med končnimi uporabniki in decentralizirano infrastrukturo ter imajo pomembno vlogo pri identifikaciji, overjanju in avtorizaciji uporabnikov ter dostopanju do storitev. Gre za prenosljive in varne osebne repozitorije, običajno v obliki mobilne denarnice ali denarnice v oblaku, ki vključujejo programsko aplikacijo in šifrirano podatkovno bazo, v kateri uporabniki (imetniki identitete oz. upravljavci denarnice) shranjujejo identifikatorje oz. osebne identifikacijske podatke (angl. Personal Identification Data - PID), kriptografski material (zasebni ključi), digitalne dokumente/poverilnice oz. (ne)kvalificirane elektronsko potrjevanje atribute (angl. (Qualified) Electronic Attestation of Attributes - (Q)EAA) in druge občutljive, zasebne podatke. Poleg shranjevanja, morajo takšne DIW podpirati tudi pregled, upravljanje in uporabo oz. deljenje omenjenih podatkov.

Iz trgovin z mobilnimi aplikacijami (Google Play in Apple Store) si lahko že sedaj namestimo DIW, pri čemer mnoge med njimi temeljijo na konceptu SSI in podpirajo (i) ustvarjanje in upravljanje več nepovezanih identifikatorjev za različne interakcije, (ii) povezovanje in komuniciranje z drugimi entitetami, (iii) pridobivanje in shranjevanje identitetnih podatkov v obliki VC-jev od izdajateljev, (iv) ustvarjanje in deljenje VP-jev s preveritelji, ki zahtevajo dokazilo o identiteti za namen preverjanja in zagotavljanja storitev, ter (v) pregled nad podatki in njihovo uporabo. Zagotavljajo varno upravljanje kriptografskih ključev (zasebnih ključev), identifikatorjev in poverilnic ter s tem omogočajo storitve upravljanja identitete, ki je bilo predhodno v domeni IdP. Takšne DIW so npr. VIDwallet, Trinsic Wallet, esatus Wallet, Connect.Me DIW, Gataca, Lissi Wallet, itd. [8]. Pri čemer pa je bil letos objavljen tudi seznam interoperabilnih DIW, kompatibilnih z evropskim omrežjem EBSI (European Blockchain Services Infrastructure) [25].

4.2 Tipi denarnic

Kot omenjeno, DIW predstavljajo orodje, ki omogoča uporabnikom nadziranje in upravljanje digitalnih identitet [6]. Obstajajo različni tipi DIW, ki se razlikujejo glede na uporabljen IdM model in vrsto okolja, ki je lahko bodisi *lokalno* bodisi *oddaljeno*, odvisno od lokacije shranjenih podatkov. V lokalnem okolju uporabniki nadzorujejo in imajo v lasti zahtevano infrastrukturo, medtem ko infrastruktura v oddaljenem, oblačnem okolju ni neposredno v lasti in upravljanju uporabnikov, temveč ponudnikov okolja [2].

Poznamo (i) *namizne denarnice* in (ii) denarnice, ki predstavljajo *razširitev brskalnika* in so naložene na računalnik, (iii) *mobilne denarnice* v obliki mobilne aplikacije, ki jih uporabnik lahko prenese iz trgovin z aplikacijami, (iv) *denarnice v oblaku*, ki temeljijo na oddaljenem shranjevanju v oblaku, ter (v) *denarnice s strojno opremo* (fizične naprave, kot je trdi diska ali USB) [12], ki omogočajo uporabnikom prejemanje in pošiljanje poverilnic le, če je naprava povezana z računalnikom z dostopom do interneta.

Tako poznamo vse od (iv) *oblačnih* denarnic do resnično *SSI denarnic*, ki naj bi bile pod popolnim nadzorom uporabnikov, (iii) na mobilnem telefonu, (i, ii) osebnem računalniku, ali (iv) drugi napravi, izven dosega tretjih oseb. Pri čemer so mobilne denarnice in denarnice v oblaku najbolj enostavne za uporabo, ponujajo dobro uporabniško izkušnjo ter so najbolj prenosljive med omenjenimi možnostmi.

4.3 Funkcionalnosti

Na osnovi analize obstoječih DIW in osnutku arhitekture ter referenčnega okvirja evropske digitalne identitete [24] predstavljamo funkcionalne zahteve DIW, ki naj bi jih izpolnjevale EUDIW namenjene končnim uporabnikom oz. imetnikom identitete.

Slednje morajo v osnovi podpirati funkcionalnosti, povezane *z identifikacijo in overjanjem* ter funkcionalnosti, povezane s *pridobivanjem, shranjevanjem, upravljanjem in izmenjavo* podatkov in poverilnic ter *shranjevanje in upravljanje* kriptografskega materiala. Uporabnikom morajo omogočiti enostavno delovanje, izgradnjo digitalne identitete ter pridobivanje dostopa do (javnih in / ali zasebnih) storitev v digitalnem okolju.

Ključnega pomena je, da DIW zagotavlja *digitalno identifikacijo in overjanje*, ki se lahko izvede z eID, z uporabo kvalificiranih ali nekvalificiranih digitalnih potrdil ali z uporabo DID-ov v kombinaciji s preverljivimi izkaznicami (VID). Slednje je skladno s konceptom SSI in zahteva *obojestransko overjanje*. Uporabnik in ponudnik identitetnih atributov ali storitev morata namreč predhodno vzpostaviti dvosmerno, varno povezavo oziroma komunikacijski kanal med svojima DIW. Vključeni entiteti si morata izmenjati identifikatorje (DID-e) in druge metapodatke, ki so ključnega pomena za overjanje oz. dokazovanje identitete in podatke, pomembne za izvedbo posamezne transakcije. Inicializacija identifikacije in overjanja sta možni na *podlagi generiranja, predstavitve in skeniranja QR oznak*, ki omogoča šifriranje in enostavno posredovanje potrebnih podatkov za izvedbo procesa. Pri tem ni pomembna zgolj identifikacija in overjanje vključenih entitet, ampak tudi njunih DIW, kar povečuje zaupanje in varnost ekosistema ter zagotavlja interoperabilnost z uporabo ustreznih naprav in denarnic.

Uporabniki morajo imeti možnost *zahtevati in pridobiti identitetne atribute*. V okviru EUDIW je govora o t. i. kvalificiranih in nekvalificiranih elektronsko potrjenih atributih (angl. (Qualified) Electronic Attestation of Attributes - QEAA in EAA), ki jih bo najverjetneje možno pridobiti v obliki VC-jev. Posledično mora imeti DIW integrirano funkcionalnost varnega *shranjevanja in upravljanja VC-jev (vključno z brisanjem)*, kar omogoča predstavitev oz. deljenje podatkov na zahtevo, ne da bi uporabnik moral ob vsaki zahtevi pridobiti podatke od ponudnika takšnih atributov. Slednje zmanjšuje možnost sledenja, vpletenost posrednikov v digitalne interakcije ter daje uporabnikom nadzor nad shranjevanjem in deljenjem svojih podatkov. *Shramba DIW* je lahko *lokalna, oddaljena ali hibridna* (z lokalnim shranjevanjem pointerjev do oddaljene shrambe) in mora poleg preverljivih poverilnic, identifikatorjev oz. osebnih identifikacijskih podatkov omogočati tudi shranjevanje kriptografskega materiala, ki omogoča nadzor sredstev in dokazovanje lastništva [7, 9], vključno z elektronsko *identifikacijo, overjanjem in digitalnim podpisovanjem* dokumentov in poverilnic. V primeru uporabe eID, je potrebno zagotoviti hrambo (ne)kvalificiranih digitalnih potrdil. V primeru uporabe DID-ov pa hrambo zasebnih kriptografskih ključev. Kriptografske funkcije so poleg shranjevanja in upravljanja kriptografskega materiala, ključne za večino funkcionalnosti denarnic, kot so npr. kvalificirano digitalno podpisovanje, identifikacija in overjanje, selektivno razkritje ipd. Upravljanje vključuje kreiranje, shranjevanje, uporabo, modificiranje in brisanje kriptografskega materiala, pri čemer lahko vmesnik za zagotavljanje funkcionalnosti izkorišča programske in / ali strojne rešitve.

DIW morajo zagotavljati *deljenje oz. distribucijo* identitetnih atributov, zahtevanih s strani tretjih oseb oz. ponudnikov storitev. Osnovno funkcionalnost *deljenja*, ki omogoča posredovanje podatkov, lahko razširja funkcionalnost *kombiniranja* in *selektivne izbire oz. razkritja ter minimizacije podatkov*, ki omogoča razkritje minimalne količine podatkov, potrebnih za uspešno izvedbo posamezne transakcije. Posledično lahko uporabnik iz nabora atributov v shranjenih poverilnicah, izbere zgolj peščico atributov, pomembnih v specifičnem kontekstu, s čimer pridobi nadzor nad deljenjem lastnih podatkov. Uporabnik mora torej imeti možnost, da selektivno izbere atribute iz enega ali več VC-jev in generira VP ob upoštevanju minimalnega razkritja podatkov ter VP digitalno podpiše z uporabo (ne)kvalificiranih digitalnih potrdil ali zasebnega ključa. DIW morajo posledično zagotavljati funkcionalnost digitalnega podpisovanja dokumentov in poverilnic.

Podatki (identifikatorji, poverilnice in kriptografski material) morajo biti v denarnici vedno dostopni in na razpolago, ko jih uporabnik potrebuje. Omogočen mora biti *hiter in enostaven dostop* ter zagotovljena dobra *uporabniška izkušnja*. Za uspešno upravljanje in deljenje podatkov je ključen intuitiven, nazoren, jasen in nedvoumen prikaz podatkov. Za uporabnike je ključen prikaz pridobljenih identitetnih dokumentov oz. preverljivih poverilnic in prikaz deljenih podatkov iz poverilnic, ter prikaz zgodovine izvedenih transakcij, kar lahko uporabniku olajša nadzorovanje in upravljanje svoje identitete. V okviru koncepta SSI je koristen tudi prikaz kontaktov oziroma t. i. vzpostavljenih DID povezav s tretjimi osebami (ponudniki preverljivih poverilnic in storitev).

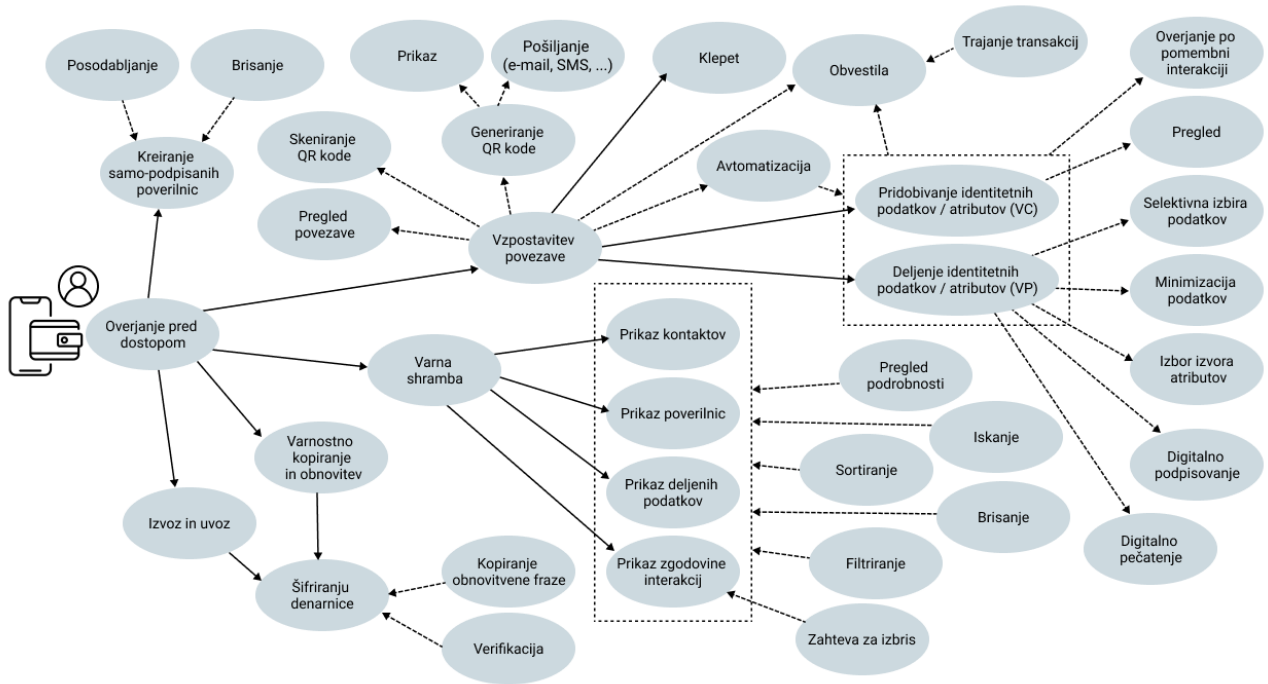
Uporabniki morajo biti *obveščeni* o pomembnih informacijah v povezavi s svojo DIW ter o informacijah in korakih v interakciji z drugimi entitetami. DIW lahko v ta namen uporabljajo opozorilna okna in potisna obvestila, ki so povezana z informiranjem in uspešnimi ali neuspešnimi transakcijami (npr. ob identifikaciji in overjanju, ob prejeti ponudbi za sprejem poverilnic, ob prejeti zahtevi za deljenje podatkov ipd.). Zaradi rokovanja z osebnimi podatki morajo DIW uporabnikom zagotoviti možnost, da podajo zavestno, namerno in dobro razumljivo privolitev za sprejem, uporabo in deljenje podatkov oziroma, možnost zavrnitve in preklica. V ta namen se lahko uporabijo pojavna polja, kot so polja z opozorilom, potrditvijo ali pozivom, ki od uporabnikov zahteva pregled podatkov pred potrditvijo transakcije. Uporabniki morajo biti informirani o identiteti entitet, s katerimi komunicirajo, o razlogu za deljenje podatkov, o vrsti operacije in pravici do varstva podatkov (GDPR) ipd.

Ključnega pomena je tudi funkcionalnost *(avtomatiziranega) varnostnega kopiranja in obnovitve*. V obstoječih DIW omenjena funkcionalnost temelji na šifriranju DIW z obnovitveno frazo, ki je potrebna pri obnovi. Pri tem predstavlja dobro prakso *kopiranje obnovitvene fraze z enim klikom* in njena *verifikacija* pred shranjevanjem. Verifikacija lahko temelji na označevanju navedenih besed ali njihovem vpisu, v ustreznem vrstnem redu, kar zmanjšuje strah pred morebitnimi napakami, ki bi uporabniku preprečile uspešno obnovitev ter s tem povezano izgubo podatkov. Varnostne kopije se shranijo lokalno ali v oblaku pri izbranem ponudniku, odvisno od preferenc uporabnika. Brez ustreznih mehanizmov varnostnega kopiranja in obnovitve lahko v primeru decentraliziranih in samoupravljanih

identitet uporabniki namreč v primeru težav (npr. izgubljen mobilni telefon, pozabljeno geslo za dostop do denarnice itn.) ostanejo brez vseh svojih podatkov ter so primorani začeti s ponovnim zbiranjem in izgradnjo digitalne identitete. V primeru uporabe mobilnih denarnic z lokalno shrambo je ključnega pomena tudi *možnost izvoza in uvoza* podatkov, ki omogoča možnost prenosa podatkov iz ene naprava in / ali denarnice v drugo, kar je skladno z načelom interoperabilnosti. Funkcionalnost, podobno kot varnostno kopiranje in obnovitev temelji na šifriranju najnovejše kopije denarnice, pri čemer mora biti izvedena manualno, na zahtevo uporabnika.

Shranjeni identifikatorji, kriptografski material, osebni podatki, dokumenti in poverilnice morajo biti v DIW pod popolnim nadzorom uporabnika, zaščiteni z eno izmed metod *overjanja*, ki ohranja varnost, zasebnost in preprečuje nepooblaščen dostop do podatkov v DIW ter izvedbo transakcij, ki vključujejo pridobivanje podatkov in poverilnic (VC), njihovo deljenje (VP) ter komunikacijo z drugimi entitetami. DIW lahko zahtevajo *overjanje pred dostopom* do podatkov in *overjanje ob potrditvi* pomembnih interakcij, kar uvaja dodaten sloj zaščite ter preprečuje zlorabe in napake pri rokovanju z občutljivimi podatki. Za overjanje se lahko uporabi biometrija, pin ali alfa-numerično geslo ter drugi mehanizmi overjanja.

Poleg omenjenega so lahko za zagotavljanje boljše uporabniške izkušnje določene akcije *avtomatizirane* na zahtevo uporabnikov. Kljub temu da morajo imeti uporabniki nadzor in možnost podati zavestno, namerno in dobro razumljivo privolitev za overjanje ter prejemanje in pošiljanje podatkov, je lahko izbira atributov in odobritev oz. zavrnitev vsake izmed interakcij z drugo entiteto za uporabnike časovno potratno ter lahko predstavlja nepotreben miselni napor, ki vodi v frustracijo in slabšo uporabniško izkušnjo. Ključnega pomena je tako možnost *avtomatizacije določenih interakcij*, pri čemer je pomembno, da uporabniki ohranjajo nadzor nad stopnjo avtomatizacije z upravljanjem nastavitev. Avtomatizirajo se lahko npr. sprejem povezav in poverilnic ter generiranje in deljenje predstavitev. Uporabniki lahko omogočijo npr. zgolj avtomatiziran sprejem povezav, ki se po skeniranju QR oznak samodejno shranijo v denarnico, ali zgolj sprejem poverilnic od določenih entitet, ki jim zaupajo. Pomembna je tudi avtomatizacija selektivne izbira podatkov, ki omogoča samodejno ustvarjanje predstavitev na podlagi zahtev preveriteljev. Uporabnikom tako ni potrebno iskati zahtevanih poverilnic in atributov, ampak je potreben zgolj njihov pregled in odobritev.



Slika 4: Funkcionalnosti digitalnih identitetnih denarnic.
Vir: lasten.

4.4 Scenariji uporabe

Obstajajo številni primeri uporabe DIW, ki nam omogočajo poenostavljeno identifikacijo, overjanje in dostopanje do različnih produktov in storitev ter nam nasploh olajšajo digitalne interakcije.

V nadaljevanju predstavljamo nekaj scenarijev, ki se osredotočajo na uporabo koncepta SSI ter temeljijo na uporabi DID-ov in VC-jev. Scenariji lahko pomagajo pri razumevanju samega koncepta DIW, prikazujejo uporabo funkcionalnosti ter nakazujejo na široko uporabnost takšnih denarnic.

Zaradi poenostavljene razlage scenarije predstavljamo preko fiktivne persone Ane, ki je zaključila študij na Fakulteti za elektrotehniko, računalništvo in informatiko na Univerzi v Mariboru (FERI UM) in pridobila naziv diplomirana inženirka informatike in tehnologij komuniciranja. Svoj študij želi nadaljevati v tujini, na Tehnični univerzi v Münchnu v Nemčiji (v nadaljevanju univerza). Z uporabo svoje digitalne identitete lahko prijavo opravi hitro in enostavno kar preko spleta, z uporabo svoje DIW, priznane s strani EU, ki si jo je namestila iz ene izmed trgovin z aplikacijami.

Ana preprosto obišče spletno stran univerze v Münchnu in izpolni prijavnico. V primeru izpolnjevanja prijavnice na računalniku se po potrditvi zgenerira QR oznaka, v kateri so šifrirani zahtevani podatki. Ana s svojim mobilnim telefonom skenira QR oznako in na tak način vzpostavi dvosmerno povezavo z univerzo, kar ji omogoči varno komunikacijo in izmenjavo podatkov, vse dokler je povezava vzpostavljena. Po skeniranju je Ana preusmerjena v denarnico, kjer izbere zahtevane dokumente oz. poverilnice, ki jih je predhodno pridobila od drugih (zaupanja vrednih) entitet, in jih sedaj želi deliti z univerzo. Izbere osebno izkaznico, ki jo je izdala Upravna enota Maribor, in diplomo, ki jo je izdal FERI UM ter preprosto potrdi prijavo. Ko univerza prejme Anino prijavo, se lahko začne avtomatiziran proces verifikacije poverilnic, ki vključuje preverjanje, ali so bile vse poverilnice izdane Ani, ali so v izvorni obliki in ali jih je izdala ustrezno akreditirana entiteta, ki ji univerza zaupa.

Univerza, ki nastopa v vlogi preveritelja je torej sposobna preveriti veljavnost poverilnic, njihov status, izdajatelje, imetnika identitete in trditve, ne da bi v sam proces morala vključevati izdajatelje poverilnic (FERI, Upravna enota Maribor).

Po uspešnem preverjanju lahko univerza potrdi Anino prijavo, o čemer je Ana obveščena preko potisnih obvestil, preko katerih je obveščena tudi ob uspešnem vpisu in ostalih pomembnih interakcijah.

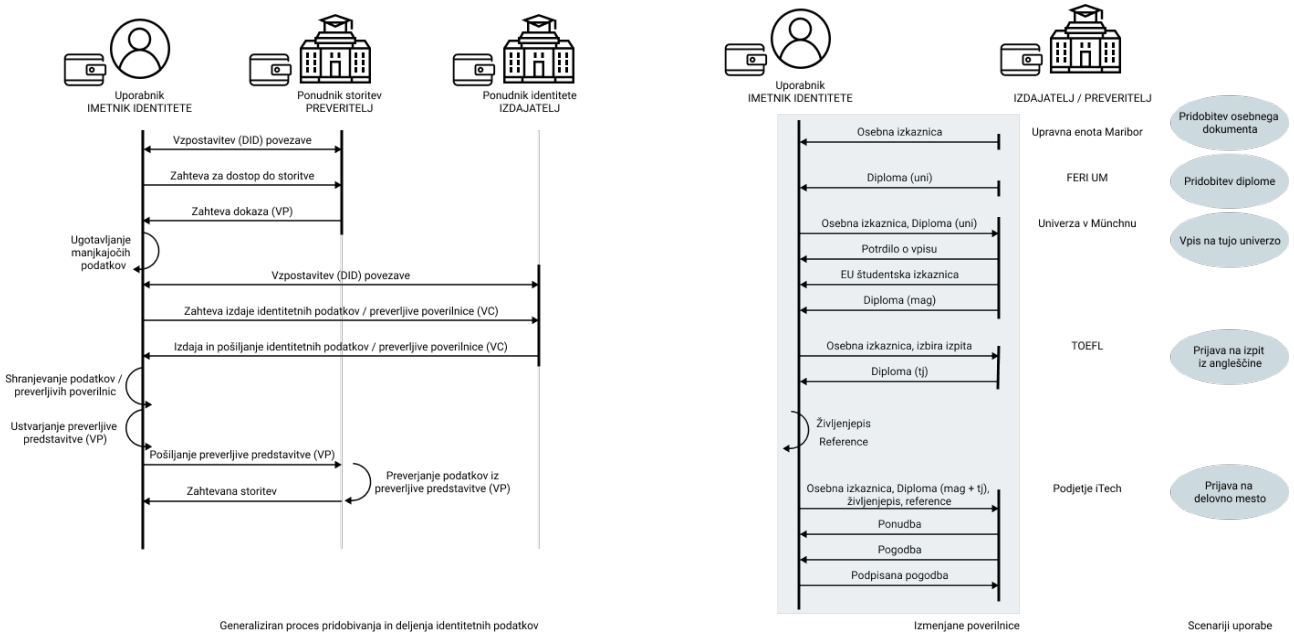
Poleg omenjenega vzpostavljena povezava omogoča tudi, da univerza Ani izda in posreduje različne poverilnice. Ana je tako preko potisnih obvestil obveščena o izdanem potrdilu o vpisu, evropski študentski izkaznici, diplomi itn., pri čemer more ponujene poverilnice pregledati in eksplicitno potrditi ali zavrniti njihov sprejem in shranjevanje v denarnico.

Po zaključku študija se Ana želi prijaviti na delovno mesto razvijalca spletnih aplikacij (v podjetju iTech). Potencialni delodajalec od nje zahteva predložitev dokazila o izobrazbi, dokazilo o opravljenem izpitu iz tujega jezika, življenjepis in reference. Ana, podobno kot pri vpisu na univerzo, obišče spletno stran podjetja, tokrat z mobilnim telefonom, zaradi česar je potreben zgolj klik na gumb za prijavo, ki omogoči vzpostavljanje povezave s podjetjem in Ano preusmeri v denarnico, kjer je obveščena o manjkajočih poverilnicah, ki jih mora pridobiti od ustreznih ustanov.

Ana se odloči za opravljanje spletnega izpita TOEFL. Obišče uradno spletno stran TOEFL in klikne na gumb za začetek procesa prijave, kar vzpostavi povezavo z organizacijo in jo preusmeri v denarnico, kamor prejme zahtevo za deljenje osebnih podatkov iz osebne izkaznice. Po potrditvi zahteve je Ana preusmerjena nazaj na spletno stran, kjer jo čaka izpolnjen prijavni obrazec s podatki iz deljene osebne izkaznice. Ani tako ni potrebno vnašati zahtevanih podatkov, izbrati mora zgolj izpit, ki ga želi opravljati ter potrditi prijavo. Po verifikaciji podatkov je Ana obveščena o terminu in načinu izvedbe izpita. Po uspešno opravljenem izpitu prejme obvestilo in diplomo, ki jo shrani v svojo denarnico.

Pred nadaljevanjem s procesom prijave na delovno mesto naloži še dokument z življenjepisom in ga digitalno podpiše ter samo-potrdi povezavo do spletne strani s svojimi referencami. Sedaj je pripravljena, da nadaljuje predhodno opisanim postopkom prijave. Vzpostavi povezavo, izbere zahtevane poverilnice ter potrdi prijavo.

Po verifikaciji poverilnic in po uspešnem razgovoru Ana prejme obvestilo o ponudbi in po njenem sprejemu še obvestilo o ponujeni pogodbi, ki jo digitalno podpiše, in s pomočjo denarnice posreduje svojemu novemu delodajalcu. Zaradi aktivne dvosmerne povezave lahko tudi v tem primeru od delodajalca prejme različne poverilnice.



Slika 5: V obeh primerih ponudnikov govorimo o klasičnih IT podatkov z uporabo DIW.
Vir: lasten.

V zgoraj opisanih scenarijih lahko opazimo podobnosti, zaradi česar lahko sam proces pridobivanja in deljenja identitetnih podatkov generaliziramo (Slika 5) in posledično apliciramo na mnoge druge primere uporabe.

Pred začetkom interakcije je potrebno vzpostaviti dvosmerno varno povezavo med vključenima entitetama. Ne glede na to, ali želi uporabnik zahtevati izdajo preverljive poverilnice ali želi dostopati do storitve, se mora najprej vzpostaviti komunikacijski kanal med denarnico uporabnika in denarnico druge entitete (izdajatelja ali preveritelja oz. ponudnika storitve).

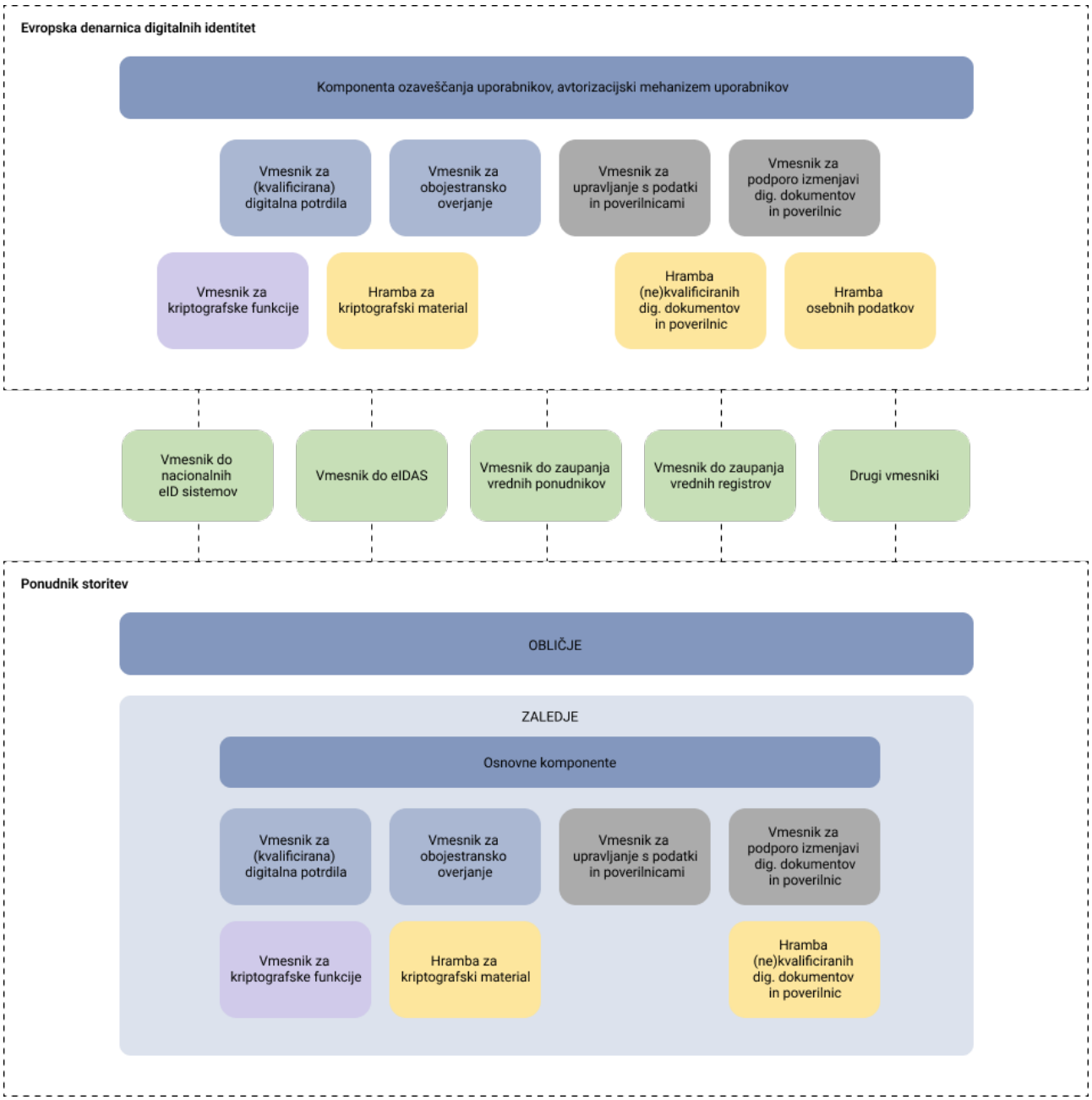
Po vzpostavljeni povezavi lahko uporabnik oz. imetnik identitete (i) zahteva dostop do produkta ali storitve, ali pa (ii) zahteva izdajo preverljive poverilnice, ki vsebuje enega ali več identitetnih atributov. (i) Pri pridobivanju dostopa do produkta ali storitve, tretja oseba oz. preveritelj običajno zahteva posredovanje dokazil/a (npr. potrdilo o izobrazbi oz. diploma itn.), kar omogoča identifikacijo in verifikacijo uporabnika. V primeru, da ima uporabnik v denarnici vse zahtevane poverilnice, lahko nadaljuje s postopkom, tako, da izbere in posreduje zahtevane identitetne atribute ali celotne poverilnice (odvisno od zahtev preveritelja). Uporabnik pravzaprav zgenerira t. i. preverljivo predstavitev (VP), ki omogoča selektivno izbiro in razkritje zgolj minimalne količine podatkov, potrebnih za posamezno interakcijo. (ii) V nasprotnem primeru mora uporabnik pridobiti ustrezne poverilnice (VC) od zaupanja vrednih izdajateljev. Poverilnice se nato shranijo v uporabnikovi digitalni denarnici, skupaj z identifikatorji ter jih je mogoče posredovati tretjim osebam na zahtevo.

5 Interakcija denarnice z obstoječimi IKT rešitvami

Četudi je na nivoju EK veliko fokusa na vpeljavi EUDIW, je razen dobro definiranih funkcijskih zahtev, še zmeraj veliko nejasnosti glede tehničnih podrobnosti. Februarja letos je EK izdala t. i. osnutek tehničnega ogrodja in referenčne arhitekture za Evropsko digitalno identiteto, ki pušča veliko odprtih vprašanj [24]. Osnutek referenčne arhitekture nam je služil kot osnova za visokonivojski pregled komponent in vmesnikov EUDIW ter ponudnikov storitev, ki je prikazan na sliki 6.

Upoštevajoč funkcijske zahteve EUDIW, mora ta zagotavljati v osnovi podporo upravljanju s kvalificiranimi digitalnimi potrdili ter po možnosti tudi nekvalificiranimi. Ideja je, da uporabnikom EUDIW omogoči, da jim le ta postane zamenjava za QSCD, pri čemer omogoča vse s tem že podprte funkcionalnosti, tj. digitalno (kvalificirano) podpisovanje, identifikacijo itn., in sicer tako na nacionalni kot čezmejni (EU) ravni. V ta namen je potrebna integracija vmesnikov za kvalificirana digitalna potrdila ter vmesnikov do nacionalnih eID sistemov in omrežja eIDAS. Kadar govorimo o ponudnikih storitev, velja izpostaviti, da v primeru zahtev po visoki zanesljivosti, ti že sedaj morebiti podpirajo možnosti identifikacije in overjanja na osnovi QR digitalnih potrdil, in sicer v povezavi s prej omenjenimi vmesniki za eID in eIDAS. Primeri takšnih ponudnikov storitev v Sloveniji so npr. AJPES, ki že sedaj omogoča vpis na osnovi SI-PASS. Primer uporabe nekvalificiranih digitalnih potrdil za identifikacijo pa je nekaj, kar je v Sloveniji prav tako že sedaj podprto na osnovi SI-PASS, saj omogoča ta identifikacijo tudi z nižjimi ravnmi zanesljivostmi (npr. na osnovi mreže AAI). Ne smemo pa zanemariti dejstva, da se EK poigrava tudi z idejami vpeljave konceptov SSI, pri čemer bi kvalificiranih ali nekvalificiranih digitalnih potrdil lahko zamenjali DID-i in / ali preverljive izkaznice (angl. Verifiable ID - VID). V tem primeru sta tako na strani EUDIW kot na strani ponudnika storitev nujno aktivirana vmesnika za obojestransko overjanje ter upravljanje s preverljivimi poverilnicami, kakor tudi vmesnik za kriptografske funkcije, ki omogoča upravljanje z DID-i, ter ob izdaji VC-jev tudi njihovo digitalno podpisovanje. S tem je tehnično povezano tudi dejstvo, da se lahko povezava in identifikacija na osnovi DID-ov vzpostavi tudi po protokolu DIDComm (po novem tudi DIDComm2) in v odvisnosti od uporabljene DID metode tudi s tem povezanimi mehanizmi za pridobitev DID dokumentov (npr. iz javnih verig blokov). Prav tako je sama inicializacija identifikacija in overjanja možna na podlagi generiranih in predstavljenih QR oznak, ki hranijo potrebne podatke za izvedbo procesa. Drug način je uporaba protokolov OpenID Connect (OIDC) ali SAML. Pri čemer slednji ne podpira DID-ov, medtem ko jih OIDC v določeni meri omogoča ter nudi tudi podporo izmenjavi VC-jev. Ravno vmesnik za podporo izmenjavi digitalnih dokumentov in VC-jev je tisti, ki nudi omenjeno podporo, pri čemer je v primeru uporabe SAML protokola za identifikacijo in overjanje, nujna namenska podpora izmenjavi na osnovi REST-a ali kaj temu podobnega. V vsakem primeru je ponudnik storitev tisti, ki bo s svojo infrastrukturo in z implementacijo vmesnikov moral podpreti omenjene možnosti.

Upravljanje s preverljivimi poverilnicami (VC) je prav tako ena od osnovnih funkcionalnosti, ki jih EUDIW morajo zadostiti. Za zagotavljanje takšne podpore je potrebno zagotoviti upravljanje z VC-ji na strani uporabnika kakor tudi izmenjavo teh med uporabnikom ter ponudnikom storitev oz. ponudnikom preverljivih poverilnic. V obeh primerih ponudnikov, govorimo o klasičnih IT arhitekturah, ki imajo svoja zaledja in obličja. Tako uporabnik v sklopu EUDIW, kakor tudi ponudniki morajo privzeto podpirati vmesnike za upravljanje z VC-ji in VP-ji. Med tem, ko EUDIW mora uporabniku nuditi možnost izbire VC-jev in generiranje VP-jev ob upoštevanju možnosti selektivnega razkritja, pa mora ponudnik storitev imeti možnost sprejeti VP-je ter jih temu primerno verificirati ter pregledati. Za namen selektivnega razkritja kakor tudi verifikacije VP-jev je potreben primeren vmesnik kriptografskih funkcij, saj se VC-ji in VP-ji primerno digitalno podpisujejo, pri čemer je le to ponovno odvisno od tega, ali bodo za generiranje VC-jev uporabljeni DID-i ali kvalificirani eID-ji kot osnovni identifikatorji. Vmesnik za kriptografske funkcije mora tako na strani ponudnika storitev podpreti preverjanje digitalnih podpisov, pri čemer bo v primeru uporabe DID-ov nujna povezava tudi z vmesniki zaupanja vrednih registrov, kot je npr. evropsko omrežje EBSI ali kaj podobnega. Omeniti je potrebno, da je v odvisnosti od izbire uporabljenih identifikatorjev, kakor tudi drugih komponent, potrebna temu primerna podpora posameznih kriptografskih funkcij, saj si le te niso med seboj kompatibilne. Prav tako je s tem povezana potrebna podpora hrambi kriptografskih materialov, in sicer od zasebnih ključev DID-ov do kvalificiranih digitalnih potrdil, v primeru, da se DID-i ne uporabljajo.



Slika 6: Visokonivojski pregled komponent in vmesnikov EUDIW ter ponudnikov storitev.
Vir: lasten.

Interakcije z EUDIW lahko temeljijo na več možnostih implementacije ob upoštevanju njihovega arhitekturnega ozadja in tehničnih podrobnosti. Te izvedbene možnosti lahko razdelimo na štiri ravni, tj. (1) identiteta, (2) overjanje, (3) komunikacija in (4) podatkovna plast.

(1) Prva raven se ukvarja z *digitalno identiteto* uporabnika EUDIW. Identiteto uporabnika je mogoče obravnavati na dva načina, tj. na osnovi nacionalnih eID ali eIDAS, ali na osnovi decentraliziranih identifikatorjev (DID). Če se uporabijo slednji, uporabnika, kot fizične osebe samega po sebi ni mogoče identificirati, saj zaradi skrbi glede zasebnosti dokumenti DID niso shranjeni na javno dostopni končni točki – v primeru DID-ov so to javno dostopne porazdeljene knjige.

(2) Druga raven je *raven overjanja*, ki se lahko spet razlikuje glede na izbiro izvedbe prve ravni in glede na uporabljeni postopek overjanja. Overjanje je tako mogoče izvesti z uporabo klasičnih protokolov, kot so SAML, ki je bolje integriran v poslovne rešitve ali z uporabo novejših protokolov, kot so OIDC. Izbira je odvisna od ravni zanesljivosti, ki jo postopek oz. ponudnik storitev zahteva. Prvi omogoča višjo raven zanesljivosti, drugi pa nižjo.

Vendar te možnosti niso edine. Če bi bili znotraj prve ravni uporabljeni DID-i, bi lahko overjanje bilo odvisno od dokumentov DID in / ali preverljivih poverilnic (VC). Zgolj uporaba DID-ov zadostuje, če je uporabnik pravna oseba in so njegovi DID dokumenti shranjeni in dostopni v javno dostopni porazdeljeni knjigi (tj. zaupanja vredni registri). DID dokumenti pravnih oseb lahko namreč poleg javnih ključev vsebujejo tudi druge določljive pravne podatke uporabnika. Če pa je uporabnik fizična oseba, mora biti postopek overjanja podprt s preverljivimi izkaznicami (VID), ki so sami po sebi QEAA v obliki VC, ki jih izdajo kvalificirani ponudniki identitete (angl. Qualified Identity Provider - QIdP). Kljub temu je treba upoštevati, da izvedbene možnosti druge ravni niso nujno vezane na izbiro prve ravni.

(3) Tretja raven je *komunikacijska* in ima dve možnosti implementacije, to je klasična SOAP komunikacija s pomočjo HTTP/S, kjer sta zahteva in odziv osnova za komunikacijo, druga možnost pa je DID-komunikacija (DIDComm), ki predstavlja dvosmerni komunikacijski kanal med dvema subjektoma, kjer deležnika poznata DID drug drugega.

(4) Zadnja raven je *podatkovna plast*, ki predstavlja obliko, v kateri se podatki izmenjujejo med obema stranema. V večini primerov je to XML, ki temelji na vnaprej določenih shemah XLS in je usklajen s strukturami SOAP/WSDL. Ta je bolj usklajen s trenutnimi poslovnimi sistemi. Vendar bo verjeten de facto format podatkov v prihodnosti VC v formatu JSON, ki je sam po sebi bolj usklajen s trenutnimi novimi koncepti IKT, kot so REST, mikrostoritve itd. Sam VC je standard W3C in lahko na osnovi definiranih shem podpira veliko podatkovnih vrst dokumentov, kot je QEAA ali kateri koli drug tip dokumentov. Matrica implementacijskih rešitev ni trdno definirana, saj je možnih več izbir glede na možnosti slojev, pri čemer je trenutno verjetno najbolj priročna možnost uporaba eID, SAML ali OIDC in VC.

6 Zaključek

Področje digitalnih identitet je v zadnjih letih ponovno pritegnilo pozornost stroke. Razlog temu so predvsem težnje po večji zasebnosti, kakor tudi uporabniški prijaznosti. V luči teh izzivov in požetih lovorik po vpeljavi GDPR se je EU s predlogom EK po novi uredbi v povezavi z Evropsko digitalno identiteto ponovno samozavestno postavila na globalno prizorišče kot vodilna na tem področju. Izziv, s katerim se EK sedaj sooča, je ta, da je za razliko od predlagane nove uredbe, GDPR sam po sebi ostal na nivoju regulatornih aktov in ni zahteval tehničnih rešitev oz. referenčnih IT arhitektur v že tako kompleksnem prostoru čezmejnih digitalnih identitet. Piko na i pa je dodalo še sunkovito razvijajoče se področje SSI, ki ponuja veliko uporabniško usmerjenih prednosti iz vidika zasebnosti, varnosti in prijaznosti, da se ga ne more ignorirati in ga je kljub nedozorelosti tehnologije, smiselno vključiti v potencialno referenčno IT arhitekturo, in s tem tudi v novo uredbo.

Če bo eIDAS 2.0 dobro implementiran, bo spremenil trenutni pristop k spletni varnosti, zasebnosti in uporabniški izkušnji evropskih državljanov ter se podobno kot GDPR na globalni ravni postavil kot vodilo. Učinkovito vpeljan eIDAS 2.0 bo na račun spodbujanja digitalne preobrazbe prihranil veliko časa in stroškov tako javnemu, kot tudi zasebnemu sektorju. EU si prav tako nadeja, da bo le ta spodbujal nove inovacije in morebitne nove samoroge (tokrat iz EU). Če pa bo eIDAS 2.0 na nivoju EU slabo načrtovan in vpeljan, se ne bo uveljavil, ne bo sprejet med državljani, in borci za zasebnost ga bodo raztrgali na koščke. Takšen morebitni izhod, ki si ga ne želimo, pa je tudi sam Avast slikovito opisal, kot "Velike IT korporacije in njihov nadzorni kapitalizem bosta zapolnila vakuum in prevladala".

Za ponudnike storitev in IKT sektor je smiselno dogajanje in razvoj budno spremljati in se počasi adaptirati v smer podpore EUDIW, ki se bo v roku dveh let izkristaliziral.

Literatura

[1] LAURENT Maryline, DENOUËL Julie, LEVALLOIS-BARTH Claire, WAELBROECK Patrick "1 - Digital Identity," Digital Identity Management, M. Laurent, S. Bouzefrane Ed., Elsevier, 2015, str. 1–45, doi: 10.1016/B978-1-78548-004-1.50001-8.

[2] B. Podgorelec, L. Alber, T. Zefferer, "What is a (Digital) Identity Wallet? A Systematic Literature Review," 2022, doi: 10.1109/COMPSAC54236.2022.00131.

[3] SCHARDONG Frederico, CUSTÓDIO Ricardo, "Self-Sovereign Identity: A Systematic Review, Mapping and Taxonomy," Sensors 2022, letnik 22, številka 15, 2022, doi: 10.3390/s22155641.

[4] SEIGNEUR Jean-Marc, EL MALIKI Tewfiq, "Chapter 17 - Identity Management," Computer and Information Security Handbook, J. R. Vacca, Ed., Boston: Morgan Kaufmann, 2009, str. 269–292, doi: 10.1016/B978-0-12-374354-1.00017-0.

[5] SHENG Quan Z., QIN Yongrui, YAO Lina, BENATALLAH Boualem, "Chapter 14 - Security Issues of the Web of Things," Managing the Web of Things, Q. Z. Sheng, Y. Qin, L. Yao, B. Benatallah, Eds., Boston: Morgan Kaufmann, 2017, str. 389–424, doi: 10.1016/B978-0-12-809764-9.00018-4.

[6] SOLTANI Reza, NGUYEN Uyen Trang, AN Aijun, "Practical Key Recovery Model for Self-Sovereign Identity Based Digital Wallets," 2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), 2019, str. 320–325, doi: 10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00066.

[7] AYDAR Mehmet, CETIN Salih Cemil, AYVAZ Serkan, AYGUN Betul, "Private key encryption and recovery in blockchain," 2019, doi: arXiv:1907.04156v2.

[8] SARTOR Sebastian, SEDLMEIR Johannes, RIEGER Alexander, HEIDI ROTH Tamara, "Love at First Sight? A User Experience Study of Self-Sovereign Identity Wallets," Conference: 30th European Conference on Information Systems (ECIS 2022), 2022.

[9] NAIK Nitin, JENKINS Paul, "Self-Sovereign Identity Specifications: Govern Your Identity Through Your Digital Wallet using Blockchain Technology," 2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2020, str. 90–95, doi: 10.1109/MobileCloud48802.2020.00021.

[10] https://www.w3.org/TR/vc-data-model/, SPORNY Manu, LONGLEY Dave, CHADWICK David, "Verifiable Credentials Data Model 1.0. Expressing verifiable information on the Web," 2021, obiskano 27. 7. 2022.

[11] https://www.w3.org/TR/did-core/, SPORNY Manu, LONGLEY Dave, SABADELLO Markus, REED Drummond, STEELE Orie, ALLEN Christopher, "Decentralized Identifiers (DIDs) v1.0: Core architecture, data model, and representations," 2022, obiskano 27. 7. 2022.

[12] LÓPEZ Marcos Allende, DA SILVA Marcelo, VEGEZZI, Alejandro Pardo, Self-Sovereign Identity: The Future of Identity: Self-Sovereignity, Digital Wallets, and Blockchain, Inter-American Development Bank, 2020.

[13] XU Jie, XUE Kaiping, TIAN Hangyu, HONG Jianan, WEI David S. L., HONG Peilin, "An Identity Management and Authentication Scheme Based on Redactable Blockchain for Mobile Networks," IEEE Transactions on Vehicular Technology, letnik 69, številka 6, str. 6688–6698, 2020, doi: 10.1109/TVT.2020.2986041.

[14] TERZI Sofia, SAVVAIDIS Charalampos, VOTIS Konstantinos, TZOVARAS Dimitrios, STAMELOS Ioannis, "Securing Emission Data of Smart Vehicles with Blockchain and Self-Sovereign Identities,", 2020 IEEE International Conference on Blockchain (Blockchain), 2020, str. 462–469, doi: 10.1109/Blockchain50366.2020.00067.

[15] https://blog.avast.com/eidas-2.0-avast, TOBIN Andy, "eIDAS 2.0: How Europe can define the digital identity blueprint for the world," 2022, obiskano 27. 7. 2022.

[16] PODGORELEC Blaž, TURKANOVIĆ Muhamed, "ZKP (Zero-Knowledge Proof) pod drobnogledom," 2019, doi: 10.18690/978-961-286-282-4.12.

[17] https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/identity/eidas-regulations, Thales, "eIDAS 2: the countdown to a single European Digital ID Wallet has begun," 2022, obiskano 27. 7. 2022.

[18] https://ec.europa.eu/digital-building-blocks/wikis/display/EIDCOMMUNITY/Overview+of+pre-notified+and+notified+eID+schemes+under+eIDAS, KIROVA Marina, EICHHOLTZER Marie, "Overview of pre-notified and notified eID schemes under eIDAS," 2019, obiskano 27. 7. 2022.

[19] COHEN Sara, NUTT Werner, SAGIV Yehoshua, "Deciding equivalences among conjunctive aggregate queries," Journal of the ACM, letnik 54, številka 2, str. 5-es, 2007, doi: 10.1145/1219092.1219093.

[20] https://www.worldbank.org/content/dam/photos/1440x300/2022/feb/eID_WB_presentation_BS.pdf, STEFAN Bogdan, "Get to know ID used across borders in the European Union: An ID4D Webinar Series. A European Framework for Decentralized Digital Identity Wallets," 2022, obiskano 27. 7. 2022.

[21] https://www.cuatrecasas.com/en/latam/article/new-eidas-2-proposal-the-new-paradigm-of-european-digital-identification, MORGADO Claudia, "New eIDAS 2 proposal. The new paradigm of European Digital Identification," 2021, obiskano 27. 7. 2022.

[22] https://www.dw.com/en/eu-unveils-plan-for-new-digital-id-wallet/a-57769145, Deutsche Welle, "EU unveils plan for new digital ID wallet," 2021, obiskano 27. 7. 2022.

[23] MÜHLE Alexander, GRÜNER Andreas, GAYVORONSKAYA Tatiana, MEINEL Christoph, "A survey on essential components of a self-sovereign identity," Computer Science Review, letnik 30, str. 80–86, 2018, doi: 10.1016/j.cosrev.2018.10.002.

[24] https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-architecture-and-reference-framework-outline, European Commission, "European Digital Identity Architecture and Reference Framework – Outline," 2022, obiskano 27. 7. 2022.

[25] https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Conformant+wallets, European Blockchain Partnership, "Conformant Wallets with EBSI," 2022, obiskano 27. 7. 2022.

[26] YUE Liu, LU Qinghua, PAIK Hye-Young, XU Xiwei and CHEN Shiping, ZHU Liming, "Design Pattern as a Service for Blockchain-Based Self-Sovereign Identity," IEEE SOFTWARE, letnik 37, številka 5, str. 30–36, 2020, doi: 10.1109/MS.2020.2992783.

[27] https://www.investopedia.com/terms/d/digital-wallet.asp, KAGAN Julia, "Digital Wallet," 2022, obiskano 27. 7. 2022.

[28] https://www.zps.si/osebne-finance-sp-1406526635/10842-kaj-ponujajo-mobilne-denarnice-slovenskih-bank, MEŠKO Alina, "Kaj ponujajo mobilne denarnice slovenskih bank?", 2021, obiskano 27. 7. 2022.

Razvoj spletnih aplikacij z uporabo spletnih komponent

Andrej Krajnc, Jani Pulko, Andrej Korošec, Bojan Štok

IZUM – Institut informacijskih znanosti, Maribor, Slovenija
andrej.krajnc@izum.si, jani,pulko@izum.si, andrej.korosec@izum.si, bojan.stok@izum.si

Sinopsis Razvoj informacijskih rešitev na osnovi komponent omogoča ponovno uporabo vnaprej pripravljenih gradnikov, ki so dostopni le preko vmesnikov, ni pa vidna njihova notranjost. Razvoj na osnovi komponent je že dolgo uveljavljen pri uporabi objektnih jezikov, ki so zastavljeni tako, da je razvoj komponent precej olajšan. V zadnjem času prevladujejo predvsem spletne aplikacije, kjer pa pri uporabi jezikov HTML in JavaScript razvoj na osnovi komponent ni zaživel v pravi meri. Velik korak naprej v tej smeri predstavljajo spletne komponente (angl. Web Components), ki omogočajo razvoj HTML komponent za spletne aplikacije. Notranja struktura spletnih komponent je skrita za druge, preko vmesnikov pa je omogočena interoperabilnost z drugimi HTML elementi. Implementacija spletnih komponent temelji na uporabi spletnih tehnologij, ki so dobro podprte v sodobnih brskalnikih. Spletne komponente lahko uporabljamo v obstoječih pristopih za razvoj spletnih aplikacij. Tako jih lahko kombiniramo s poljubnimi JavaScript knjižnicami in ogrodji, lahko se uporabljajo tudi v javanskih ogrodjih (npr. JSF), poseben pristop pa predstavlja ogrodje Vaadin, kjer s spletnimi komponentami delamo kot z drugimi javanskimi razredi.

Ključne besede:
splet
komponenta
HTML
DOM
Java

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.4

1 Uvod

Že od nekdaj se pri razvoju programske opreme teži k uporabi komponent. Komponente so ponovno uporabne enote, ki jih razvijamo z namenom, da bodo večkrat uporabljene. Uporaba komponente temelji na konceptu ograjevanja (angl. encapsulation), kjer se za druge skrivajo interne lastnosti in operacije. Komponente so tako dostopne le preko vmesnikov, ni pa vidna njihova notranjost.

Razvoj na osnovi komponent je pridobil na pomenu ob uveljavitvi objektnih jezikov (npr. Java, C+, C#), ki so zastavljeni tako, da je razvoj komponent precej olajšan. Še posebej velik napredek pri uporabi komponent se je zgodil ob uporabi programskega jezika Java. Obstaja namreč veliko javanskih komponent, knjižnic razredov in ogrodij.

V zadnjem času prevladujejo predvsem spletne aplikacije, kjer se objektni jeziki uporabljajo predvsem na strežniku, na odjemalski strani (v brskalniku) pa programska koda temelji na predvsem označevalnem jeziku HTML in programskem jeziku JavaScript. Čeprav se je jezik HTML od pojava leta 1993 veliko razvijal in nadgrajeval, pa dolgo ni omogočal razvoja na osnovi komponent kot ga poznamo v objektnih jezikih.

Pri razvoju na osnovi komponent želimo uporabljati komponente, ki bodo uporabne še čez mnogo let. Žal mnogi pristopi pri razvoju spletnih aplikacij temeljijo na uporabi pristopov, ki se sčasoma precej spreminjajo in z leti komponente sčasoma niso več ponovno uporabne v takšni meri kot bi si želeli. Veliko JavaScript ogrodij ima življenjsko dobo le nekaj let, na drugi strani pa želimo, da naše aplikacije delajo veliko dlje. Zato moramo pri gradnji spletnih aplikacij uporabljati pristope, kjer bomo uporabljali komponente z daljšo življenjsko dobo. Velik korak naprej glede preglednosti JavaScript kode je bil narejen z EcmaScript 6 (ES6).

2 Spletni standardi za spletne komponente

Precejšen korak naprej pri uporabi komponent v spletnih aplikacijah predstavljajo spletne komponente (angl. Web Components). Spletne komponente omogočajo razvoj HTML komponent za spletne aplikacije. Notranja struktura spletnih komponent je skrita za druge, preko vmesnikov pa je omogočena interoperabilnost z drugimi HTML elementi. Implementacija spletnih komponent temelji na uporabi spletnih tehnologij, ki so dobro podprte v sodobnih brskalnikih.

Koncept spletnih komponent je prvi predstavil Alex Russell leta 2011 na konferenci Fronteers Conference. Spletne aplikacije temeljijo na uporabi spletnih standardov, vendar ne obstaja poseben standard samo za spletne komponente, saj spletne komponente povezujejo več drugih spletnih standardov. Veliko spletnih standardov je nastalo znotraj konzorcija za splet (W3C - World Wide Web Consortium) [1], dodatno so različne standarde in specifikacije definirali še Mozilla, Google Web Fundamentals, WHATWG itd.

Trenutno najbolj aktualni standardi so WHATWG standardi [2]. WHATWG (Web Hypertext Application Technology Working Group) je skupnost ljudi, zainteresiranih za razvoj HTML in povezanih tehnologij. Ustanovljena je bila leta 2004, člani pa so Apple, Mozilla, Google, Microsoft itd. Leta 2018 je prišlo do nestrinjanja med W3C in člani WHATWG glede prihodnosti razvoja standardov HTML, leta 2019 pa je bil sklenjen dogovor med W3C in WHATWG, da se bo večina najpomembnejših spletnih standardov razvijala znotraj WHATWG.

Za področje spletnih komponent sta najpomembnejša standarda HTML Living Standard (nadaljevanje W3C HTML standarda) in DOM Living Standard (nadaljevanje W3C DOM standarda). Znotraj HTML Living Standard sta aktualna predvsem standarda HTML custom elements (HTML elementi po meri) in HTML templates (HTML predloge). Znotraj DOM Living Standard je aktualen predvsem standard HTML shadow DOM (HTML senčni DOM).

Spletni standardi so pomembni predvsem zaradi podpore v spletnih brskalnikih. Tipično je potreben določen čas, da brskalniki podprejo nove standarde, stari brskalniki pa imajo pogosto težave z novimi standardi. Vendar pa, ko

so novosti podprte v brskalnikih, pa imajo brskalniki dobro podporo za nazaj. Za razliko od mnogih JavaScript ogrodij se programski vmesniki v brskalnikih ohranjajo skozi mnoga leta.

3 Ključne specifikacije/tehnologije za spletne komponente

Spletne komponente zaobsegajo 4 tehnologije / specifikacije.

Senčni DOM (shadow DOM) je množica JavaScript programskih vmesnikov (API) za pripenjanje ločenega DOM drevesa v glavno DOM drevo oz. element.

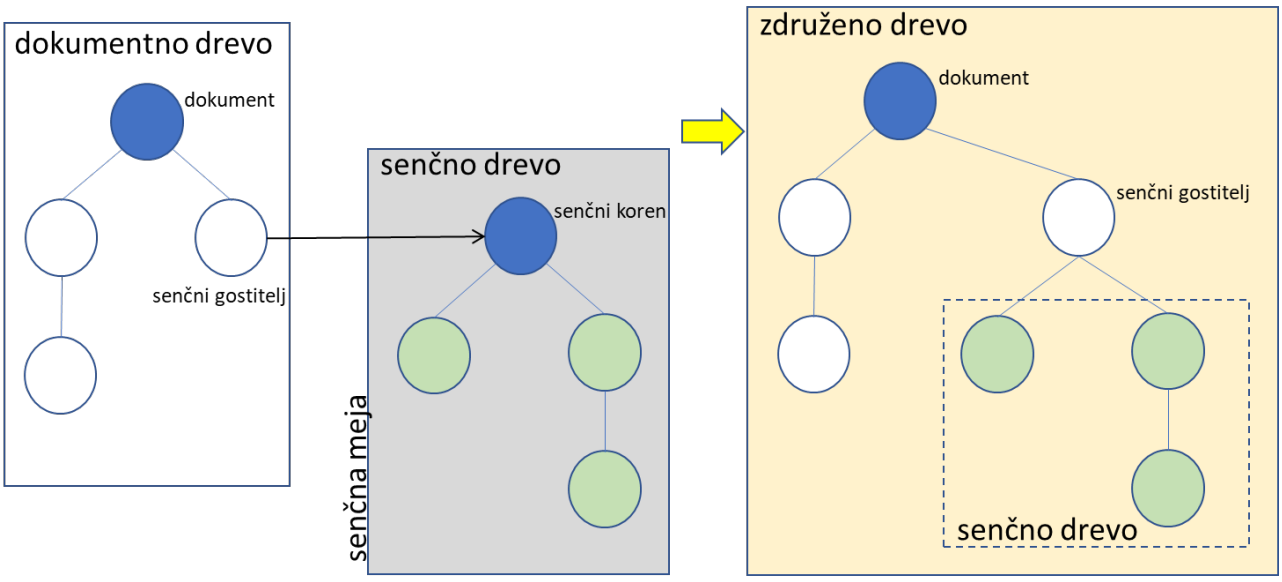
Elementi po meri (custom elements) predstavljajo množico JavaScript programskih vmesnikov (API) za definiranje novih HTML elementov.

HTML predloge (templates) prinašajo HTML elementa <template> in <slot>, ki omogočata izdelavo označevalnih predlog, ki se ne prikažejo na osnovni strani. Predloge so lahko večkrat ponovno uporabljene in predstavljajo osnovo za elemente po meri.

ECMAScript moduli (nodules) omogočajo definiranje modulov v ECMAScript. Ta trenutek moduli še niso tako zaživeli, znajo pa biti zelo pomembni v prihodnosti tudi za spletne komponente.

3.1 Senčni DOM (shadow DOM)

Programski vmesnik, ki ga ponuja senčni DOM, omogoča vpenjanje senčnega drevesa (shadow tree) v glavno drevo. Senčni koren (shadow root) iz senčnega drevesa se pripne v senčnega gostitelja (shadow host), ki je navadno DOM vozlišče (node) v glavnem drevesu. Rezultat je združeno drevo in takšno združeno drevo je upodobljeno (rendered) v brskalniku. Senčni DOM je omejen s senčno mejo (shadow boundary), ki predstavlja mesto, kjer se konča senčni DOM in začne glavni DOM.



Slika 1: Senčni DOM.
Vir: lasten.

V senčnem DOMu je drevo upodobljeno ločeno od glavnega DOM drevesa, zaradi česar so lastnosti elementa zasebne in neodvisne od drugih delov HTML dokumenta. Ta koncept ograjevanja sodi med glavne prednosti uporabe senčnega DOMa. Pri ograjevanju gre za več stvari: ograjevanje kode, ograjevanje stilov in ograjevanje obnašanja. Delo s senčnim DOMom je enako delu z navadnim DOMom, saj lahko dodajamo nova vozlišče v

DOM, spreminjamo atribute, nastavljamo stile itd. Glavna razlika je v tem, da koda, stili in obnašanje v senčnem DOMu nimajo vpliva na obnašanje izven senčnega DOMa.

Med slabosti uporabe senčnega DOMa sodi to, da ne delujejo vse knjižnice s senčnim DOMom. Včasih tudi ne želimo izolacije stilov, temveč želimo globalne stile. V senčni DOM je možno prilepiti zunanji stil, npr.

```
const linkElem = document.createElement('link');
linkElem.setAttribute('rel', 'stylesheet');
linkElem.setAttribute('href', 'style.css');
shadow.appendChild(linkElem);
```

Senčni DOM lahko uporablja na dva načina:

- **odprti način** - iz glavne strani lahko dostopamo do senčnega DOMa
  - senčni DOM dodamo z **attachShadow({mode: 'open'})**, nato lahko senčni DOM pridobimo z **let myShadowDom = myCustomElem.shadowRoot;**
- **zaprti način** - iz glavne strani ne moremo dostopati do senčnega DOMa
  - senčni DOM dodamo z **attachShadow({mode: 'closed'})**, ne moremo pa dostopati do njega, saj **myCustomElem.shadowRoot** vrne **null**

Senčni DOM ni čisto nov koncept v brskalnikih. Že dolgo so v brskalnikih nahajajo elementi, ki skrivajo svojo notranjost. Primer takšnega elementa je implementacija označbe <video>, kjer vsak brskalnik na svoj način implementira predvajalnik videoposnetkov in vsak tak predvajalnik ima svoje elemente (gumbe) za upravljanje predvajanja. V HTML kodi je vidna le označba <video>, elementi predvajalnika pa so navzven skriti.

3.2 HTML elementi po meri

Pri HTML elementih po meri gre za to, da ne uporabljamo standardnih HTML elementov, temveč si ustvarimo lastne HTML elemente po meri. Tako v HTML kodi ne uporabljamo le standardnih HTML označb, temveč tudi naše lastne označbe.

Ločimo dve vrsti HTML elementov po meri

- **avtonomni elementi po meri** so popolnoma ločeni od obstoječih HTML elementov
- **prilagojeni elementi po meri** razširjajo enega od HTML elementov

3.2.1 Avtonomni elementi po meri

Avtonomni elementi po meri so neodvisni od standardnih HTML elementov. Registriramo jih z uporabo **customElements.define**, npr.

```
customElements  .define('my-paragraph1',
  class extends HTMLElement {
    constructor() {
      super();
     // dodatna inicializacija elementa po meri
    }
  }
);
```

V HTML kodi jih uporabljamo kot HTML elemente s HTML oznako, npr.

```
<my-paragraph1></my-paragraph1>
```

ali

```
document.createElement("my-paragraph1")
```

3.2.2 Prilagojeni elementi po meri

Prilagojeni elementi po meri razširjajo enega od standardnih HTML elementov. Registriramo jih z uporabo **customElements.define**, pri čemer se navede oznako razširjenega HTML elementa, npr.

```
class MyParagraph2 extends HTMLParagraphElement {
    constructor() {
      super();
     // dodatna inicializacija elementa po meri
    }
  }
customElements.define('my-paragraph2', MyParagraph2, { extends: 'p' });
```

V HTML kodi jih uporabljamo tako, da uporabimo oznako razširjenega HTML elementa, ime elementa pa navedemo pri lastnosti "is", npr.

```
<p is="my-paragraph2">
```

ali

```
document.createElement("p", { is: "my-paragraph2" })
```

3.3 HTML predloge in reže

Pri spletnih komponentah se pogosto uporabljajo HTML predloge (angl. template) in reže (angl. slot).

HTML označba <template> predstavlja predlogo za HTML element in je na voljo že od ECMAScript 2015 (ES6) dalje. Gre za mehanizem, ki omogoča, da se del HTML kode ne uporabi za prikaz takoj po nalaganju strani (se ne aktivira). Ob nalaganju strani se le validira HTML koda v predlogi. Vse dokler se predloga ne uporabi, se skripte ne izvedejo, slike se ne naložijo, zvok se ne predvaja itd. Pri HTML predlogah gre poudariti, da dostop do elementov v predlogi ni mogoč (npr. preko getElementById ali querySelector).

Predloga se za prikaz uporabi v času izvajanja z JavaScript kodo. Osnovna ideja pri HTML predlogah je večkratna ponovna uporaba predloge. Pogosto želimo doseči ponovno uporabo posameznih delov strani in to lahko naredimo tudi z uporabo HTML predlog.

Primer definiranja in uporabe HTML predloge v spletni komponenti, ki se nahaja v datoteki my-paragraph1.js:

```
customElements.define('my-paragraph1',
  class extends HTMLElement {
    constructor() {
      super();
      let template = document.getElementById('my-paragraph-template');
      let templateContent = template.content;
      const shadowRoot = this.attachShadow({mode: 'open'})
```

```
        .appendChild(templateContent.cloneNode(true));
    }
  }
);

<!DOCTYPE html>
<html>
<head>
  <script src="my-paragraph1.js" defer></script>
</head>
<body>
  <p>Demo Web Components 1</p>
  <template id="my-paragraph-template">
    <style>
      p {
        color: white;
        background-color: rgb(117, 17, 121);
        padding: 15px;
      }
    </style>
    <p>My paragraph</p>
  </template>
  <my-paragraph1></my-paragraph1>
</body>
</html>
```

Ob označbi za predlogo se lahko uporablja HTML označba za režo <slot>. Ideja pri uporabi rež je ta, da se predloga razdeli na manjše dele (reže). Vsak del (reža) ima lahko svoj stil. Primer definiranja in uporabe HTML reže in zgornje spletne komponente.

```
<!DOCTYPE html>
<html>
  <head>
    <script src=my-paragraph1.js" defer></script>
  </head>
<body>
  <p>Demo Web Components 2</p>
  <template id="my-paragraph-template">
    <style>
      p {
        color: white;
        background-color: rgb(117, 17, 121);
        padding: 15px;
      }
```

```
    </style>
    <p><slot name="my-text">My default text</slot></p>
  </template>
  <my-paragraph1></my-paragraph1>
  <my-paragraph1>
    <span slot="my-text">Let's have some different text!</span>
  </my-paragraph1>
  <my-paragraph1>
    <ul slot="my-text">
      <li>Let's have some different text!</li>
      <li>In a list!</li>
    </ul>
  </my-paragraph1>
</body>
</html>
```

4 Podpora za spletne komponente

4.1 Podpora za spletne komponente v brskalnikih

Spletne komponente so že nekaj let dobro podprte v brskalnikih, kot so Chrome, Firefox, Microsoft Edge, Opera itd. Dobra podpora je predvsem v brskalnikih, ki temeljijo na Chromium.

Brskalnik Safari je v zadnjem času zelo izboljšal podporo za spletne komponente, vendar ne podpira spletnih komponent čisto v celoti. Brskalnik Safari ne podpira prilagojenih elementov, ki razširjajo enega od HTML elementov

V primeru brskalnikov, ki ne podpirajo spletnih komponent, je zagotovljena kompatibilnost za nazaj z uporabo kode polyfills [4].

Podporo za spletne komponente v brskalnikih se da preveriti tudi na spletni strani Can I use [4].

4.2 Podpora za spletne komponente v knjižnicah in ogrodjih

Obstaja kar nekaj knjižnic za spletne komponente, kot so FASTElement, snuggsi, X-Tag, Slim.js, Lit, Smart, Stencil, hyperHTML-Element, DataFormsJS, Polymer, Bosonic, Riot.js.

Skupnost za spletne komponente je vedno večja, kar je razvidno tudi na ponudbi spletnih komponent na WebComponents.org [5].

Vedno boljše je podprta tudi uporaba spletnih komponent v navezavi s spletnimi ogrodji, kot so Angular, React, Vue.js itd.

Obstajajo celo ogrodja, ki v celoti temeljijo na uporabi spletnih komponent. Eno takšnih ogrodij je Vaadin [6].

5 Uporaba spletnih komponent v Izumovih spletnih aplikacijah

Institut informacijskih znanosti (IZUM) razvija informacijski servis slovenske znanosti, kulture in izobraževanja. Javnosti najbolj poznan je sistem COBISS, ki predstavlja temelj knjižničnega informacijskega sistema Slovenije in nekaterih drugih držav, ki so povezani v mrežo COBISS.net.

Dosedaj so bile spletne komponente uporabljene v treh spletnih aplikacijah:

- Digitalni repozitorij COBISS (dCOBISS) [7]
- Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS) [8]
- v razvoju je COBISS4 (nova generacija programske opreme za knjižničarje, nadgrajuje COBISS3) [9]

Za razvoj dCOBISS in SICRIS se uporablja programski jezik Java. Ključna ogrodja v dCOBISS in SICRIS so JavaServer Faces (JSF), Krazo (javansko ogrodje za MVC Model-View-Controller) in Bootstrap (CSS ogrodje za razvoj odzivnih spletnih aplikacij). Čeprav gre pri dCOBISS in SICRIS za spletne aplikacije, je klasičnega JavaScripta-a relativno malo. Velik poudarek pa se daje na spletne komponente, na katerih temelji velik del aplikacij.

Pri razvoju COBISS4 se spletne komponente uporabljajo skozi ogrodje Vaadin.

5.1 Uporaba spletnih komponent v dCOBISS

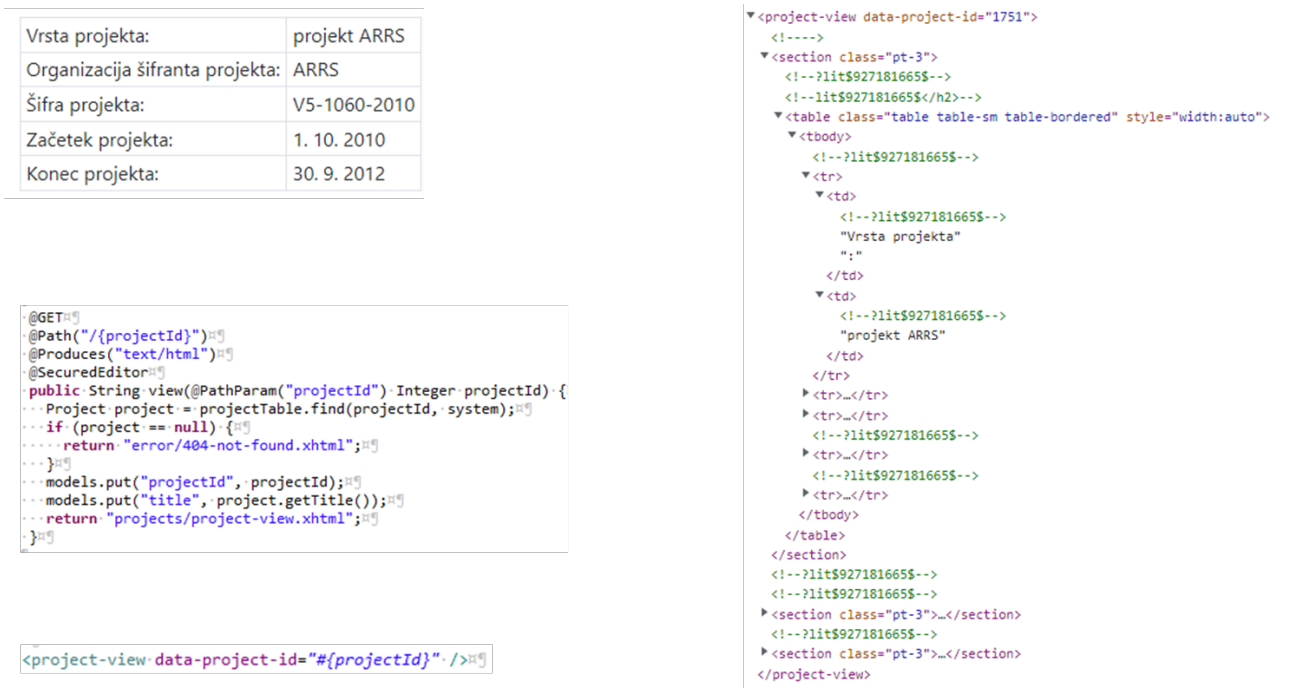
Spletne komponente v dCOBISS se uporabljajo za večino uporabniškega vmesnika dCOBISS (prikaz, urejanje itd.).

Slika 2: Digitalni repozitorij COBISS (dCOBISS).
Vir: [7].

dCOBISS je aplikacija za knjižničarje, ne za končne uporabnike. Zato izgled ni tako zelo pomemben kot pri aplikacijah za končne uporabnike.

HTML struktura je definirana večinoma v sami spletni komponenti, na strežniški strani (Controller, JSF XHTML) se zgolj uporabi ta spletna komponenta.

Primer uporabe spletnih komponent pri prikazu podatkov o projektu v dCOBISS je prikazan na spodnjih slikah.



Vrsta projekta:	projekt ARRS
Organizacija šifranta projekta:	ARRS
Šifra projekta:	V5-1060-2010
Začetek projekta:	1. 10. 2010
Konec projekta:	30. 9. 2012

```
@GET
@Path("/{projectId}")
@Produces("text/html")
@SecuredEditor
public String view(@PathParam("projectId") Integer projectId) {
    Project project = projectTable.find(projectId, system);
    if (project == null) {
        return "error/404-not-found.xhtml";
    }
    models.put("projectId", projectId);
    models.put("title", project.getTitle());
    return "projects/project-view.xhtml";
}
```

```
<project-view data-project-id="#{projectId}" />
```

```
<project-view data-project-id="1751">
  <!---->
  <section class="pt-3">
    <!--?lit$927181665$-->
    <!--lit$927181665$</h2>-->
    <table class="table table-sm table-bordered" style="width:auto">
      <tbody>
        <!--?lit$927181665$-->
        <tr>
          <td>
            <!--?lit$927181665$-->
            "Vrsta projekta"
            ":"
          </td>
          <td>
            <!--?lit$927181665$-->
            "projekt ARRS"
          </td>
        </tr>
        <tr>…</tr>
        <tr>…</tr>
        <!--?lit$927181665$-->
        <tr>…</tr>
        <!--?lit$927181665$-->
        <tr>…</tr>
      </tbody>
    </table>
  </section>
  <!--?lit$927181665$-->
  <!--?lit$927181665$-->
  <section class="pt-3">…</section>
  <!--?lit$927181665$-->
  <section class="pt-3">…</section>
</project-view>
```

Slika 3: Prikaz podatkov o projektu v dCOBISS.
Vir: [7].

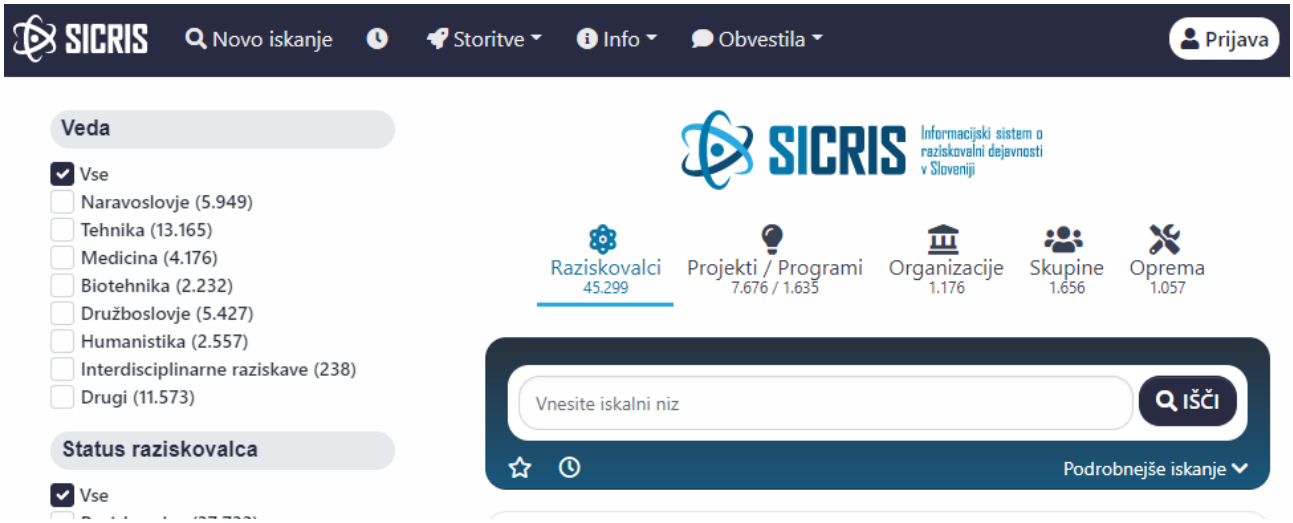
```
class ProjectView extends HTMLElement {
    async connectedCallback() {
        let projectId = parseInt(this.dataset.projectId);
        if (!projectId) {
            logger.warn("project-view.js data-project-id attribute not set");
            return;
        }
        this.project = await getProject(projectId);
        if (!this.project) {
            logger.warn(`project-view.js project ${projectId} not found`);
            return;
        }

        this.render();

customElements.define("project-view", ProjectView);
```

```
render() {
    render(
        html`
            <section class="pt-3">${this.baseDataHtml}</section>
        `,
        this
    );
}
get baseDataHtml() {
    return html`
        <!--${translate("erepo.project.base-data.title")}</h2>-->
        <table class="table table-sm table-bordered" style="width:auto">
            <tbody>
                <tr>
                    <td>${translate("erepo.project.projectType")}:</td>
                    <td>
                        ${translate("erepo.project.type." + this.project.projectType)
                    </td>
                </tr>
```

Slika 4: Spletna komponenta za prikaz podatkov o projektu v dCOBISS.
Vir: [7].

5.2 Uporaba spletnih komponent v SICRIS

Spletne komponente v SICRIS se uporabljajo za večino uporabniškega vmesnika SICRIS. Tako so bile izdelane komponente za prikaz posameznih rezultatov, komponente za prikaz iskalnih rezultatov, vizualizacijo podatkov in filtre, komponente za samodokončanje (autocomplete) in komponente za urejanje podatkov.



Slika 5: Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS).
Vir: [8].

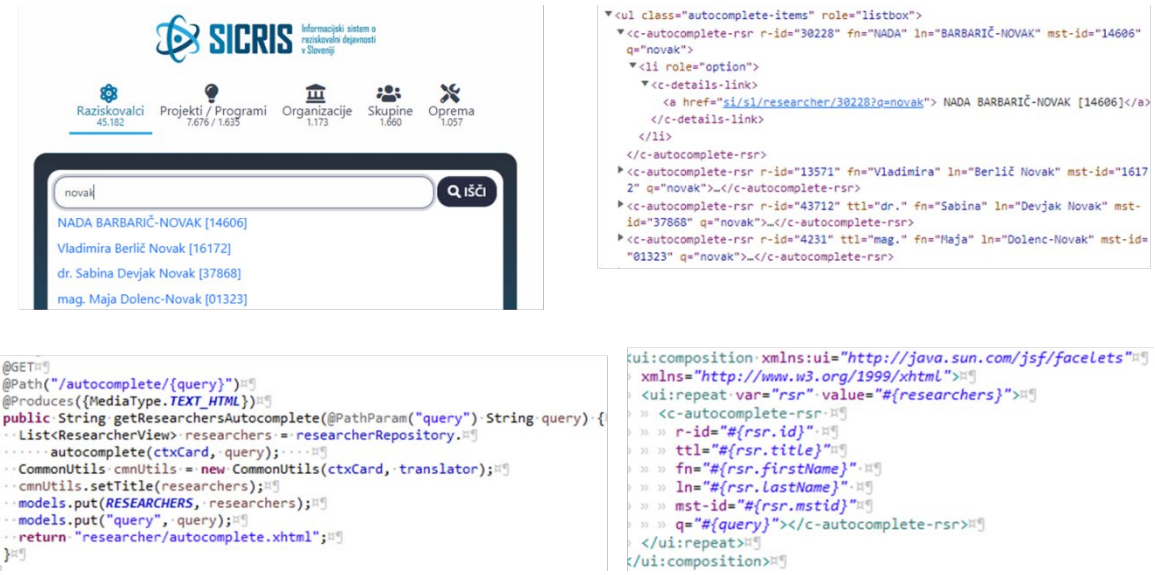
SICRIS je aplikacija za končne uporabnike, zato je izgled zelo pomemben. Pomembno je, da ima oblikovalec aplikacij čimbolj olajšan način oblikovanja aplikacij.

HTML struktura je definirana večinoma na strežniški strani v JSF XHTML. Na odjemalski strani v spletni komponenti ni definirana HTML struktura in tak pristop prijaznejši za oblikovalce uporabniškega vmesnika, ki so navajeni na XHTML datoteke

SICRIS je »single page« aplikacija, ki vsebine pridobiva na asinhroni način preko spletnih komponent. Komponente med seboj komunicirajo preko dogodkov.

Pogosto so onemogočeni privzeti načini pošiljanja podatkov iz obrazcev. V takšnih primerih se proži globalni dogodek, na katerega se odzovejo spletne komponente, ki so poslušalci tega dogodka. Ti tipično pošljejo asinhrone zahteve na strežnik in na ta način se napolnijo posamezni fragmenti strani (rezultati iskanja, filtri, navigacijska vrstica, zgodovina iskanja itd.). Kot fragment se iz strežnika pošljejo kar HTML fragmenti in ne zgolj podatki npr. v JSON obliki.

Primer uporabe spletnih komponent pri prikazu rezultatov samodokončanja (autocomplete) raziskovalcev v SICRIS je prikazan na spodnjih slikah.



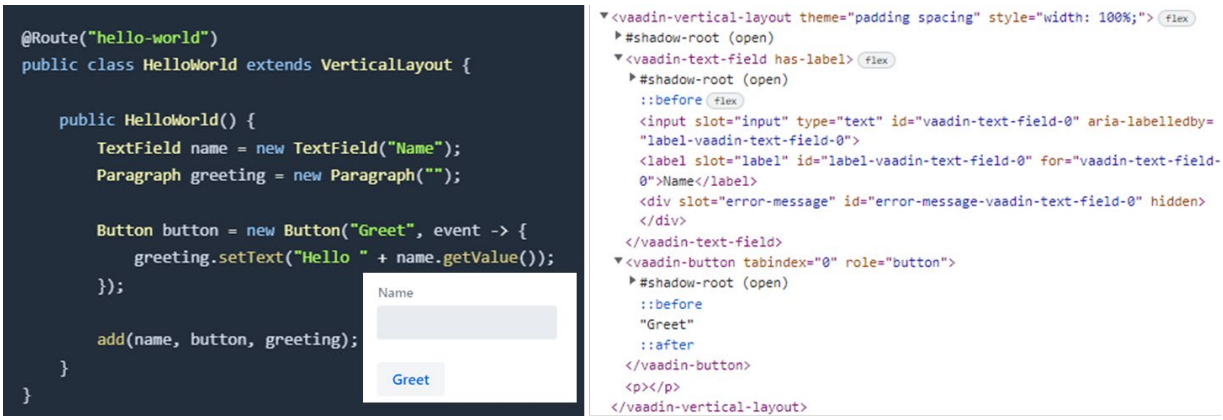
Slika 6: Prikaz rezultatov iskanja raziskovalcev v SICRIS.
Vir:[8].

```
class CAutocompleteRsr extends HTMLElement {
  constructor() {
    super();
  }

  connectedCallback() {
    const id = this.getAttribute("r-id");
    const title = this.getAttribute("ttl") || "";
    const firstName = this.getAttribute("fn");
    const lastName = this.getAttribute("ln");
    const mstid = this.getAttribute("mst-id");
    const q = this.getAttribute("q");
    if (!id || !q) {
      return;
    }
    const li = document.createElement("LI");
    li.setAttribute("role", "option");
    const detailsLink = document.createElement("c-details-link");
    const link = document.createElement("link");
    link.href = `${window.appData.url}/researcher/${id}?q=${q}`;
    link.textContent = `${title} ${boldStart(firstName, q)} ${boldStart(
      lastName,
      q
    )} [${boldStart(mstid, q)}]`;

    detailsLink.appendChild(link);
    li.appendChild(detailsLink);
    li.onclick = () => {
      li.querySelector("a").click();
    };
    this.appendChild(li);
  }
}
```

Slika 7: Spletna komponenta za prikaz rezultatov iskanja raziskovalcev v SICRIS.
Vir:[8].

5.3 Uporaba spletnih komponent v ogrodju Vaadin

Vaadin je odprtokodna platforma za razvoj spletnih aplikacij. Za razvoj spletnih aplikacij v Javi je na voljo ogrodje Vaadin Flow, ki omogoča izgradnjo spletnih aplikacij 100% v Javi. Vaadin Flow omogoča izdelavo spletnih aplikacije brez programiranja v JavaScript in HTML., saj se javanski razredi preslikajo v spletne komponente.



Slika 8: Programiranje v ogrodju Vaadin Flow.
Vir: [6].

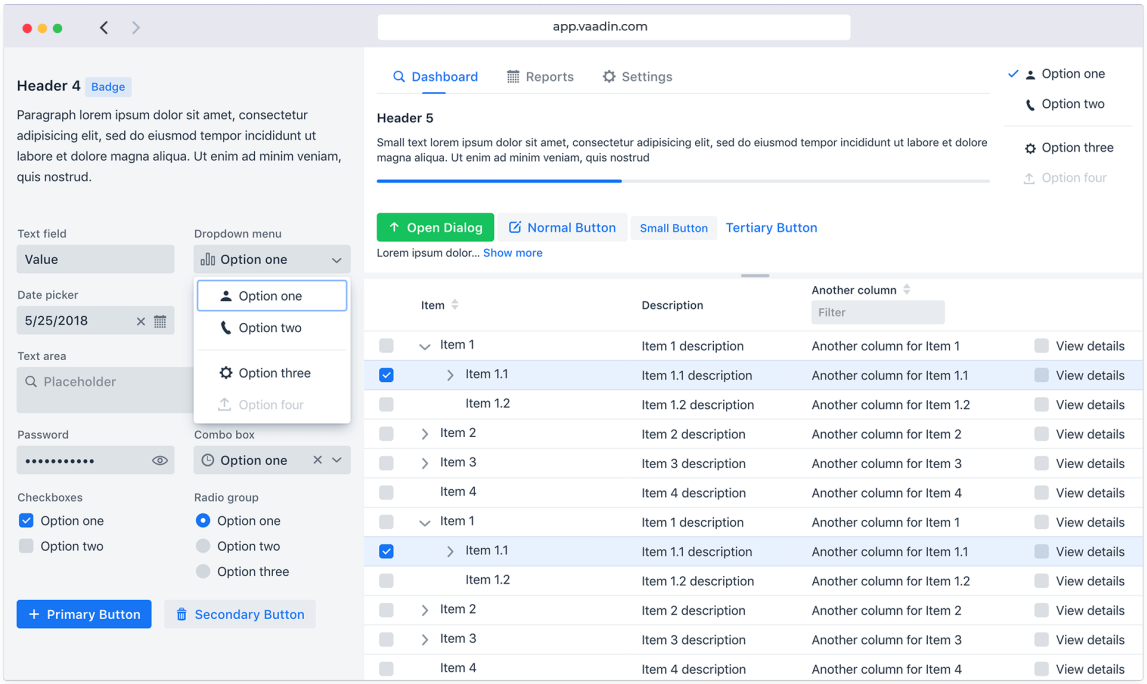
Pri programiranju v Vaadin Flow gre za drugačen pristop k izgradnji spletnih aplikacij. Razvijalci ne razmišljajo o HTTP zahtevah in odgovorih, temveč se podobno kot pri tradicionalnih namiznih aplikacijah (npr. Swing) razmišlja predvsem o izgradnji grafičnega vmesnika na osnovi komponent.

Javanske grafične komponente se v brskalniku prikazujejo v HTML kodi kot spletne komponente v standardni HTML obliki in podobno kot druge spletne komponente delujejo v vseh brskalnikih. Večina logike za grafični vmesnik se izvede na strežniku, s čimer imamo manjšo varnostno izpostavljenost. Komunikacija med odjemalcem

in strežnikom poteka avtomatsko preko ogrodja Vaadin z uporabo WebSocket ali HTTP. Pošiljajo se JSON sporočila, ki posodabljajo tako grafični vmesnik v brskalniku in stanje grafičnega vmesnika na strežniku

Vaadin ponuja veliko spletnih komponent za grafični vmesnik:

- Različna polja za obrazce: Text Field, Number Field, Text Area, Checkbox, Combo Box, Radio Button, Date Picker, Time Picker itd.
- Različni elementi za prikaz: Menu Bar, Context Menu, Button, Dialog, Notification, Progress Bar, Grid, Tabs, Scroller itd.



Slika 9: Komponente v ogrodju Vaadin Flow.
Vir: [6].

Vaadinove spletne komponente temeljijo na modernih konceptih, saj temeljijo na spletnih standardih (W3C/WHATG). V komponentah se uporablja senčni DOM, možno jih je prilagajati z različnimi temami (CSS). Vaadinove komponente je možno uporabljati v praktično vseh sodobnih ogrodjih za front-end. Ponujajo programski vmesnik za Javo in TypeScript. Prijazne so za mobilne naprave, saj so odzivne in optimizirane za različne velikosti zaslonov.

Pomembno je tudi, da podpirajo spletno dostopnost v skladu s standardom WCAG. Še posebej je to pomembno za uporabnike bralnika zaslona in tudi druge ranljive skupine, ki imajo težave pri uporabi spleta.

6 Zaključek

Spletne komponente omogočajo gradnjo ponovno uporabnih enot, do katerih se dostopa preko programskih vmesnikov, notranja implementacija pa je skrita. Na ta način so spletne komponente velik korak naprej k razvoju na osnovi komponent, kar pri HTML prej ni bilo tako prisotno.

Spletne komponente poenostavijo razvoj, omogočajo lažje vzdrževanje na dolgi rok in prispevajo k večji modularizaciji spletnih aplikacij. Temeljijo na spletnih standardih, ki imajo dobro podporo v brskalnikih. Imajo tudi daljšo življenjsko dobo kot marsikatere druge alternativne rešitve.

Spletne komponente lahko uporabljamo na različne načine. Možen je klasični pristop h gradnji aplikacij z generiranjem HTML vsebine na strežniku oz. uporabo predlog (npr. z uporabo JSF), javanski pristop (Vaadin) ali pa z generiranjem HTML na odjemalcu (JavaScript).

Naše izkušnje so pozitivne in z uporabo spletnih komponent nameravamo nadaljevati tudi v prihodnosti.

Literatura

[1] World Wide Web Consortium (W3C), https://www.w3.org/, obiskano 28. 7. 2022

[2] Web Hypertext Application Technology Working Group (WHATWG), https://whatwg.org/, obiskano 28. 7. 2022

[3] polyfills, https://github.com/webcomponents/polyfills/tree/master/packages/webcomponentsjs, obiskano 28. 7. 2022

[4] Can I use web components?, https://caniuse.com/?search=web%20components, obiskano 28. 7. 2022

[5] WebComponents.org, https://www.webcomponents.org/, obiskano 28. 7. 2022

[6] Vaadin, https://vaadin.com/, obiskano 28. 7. 2022

[7] Digitalni repozitorij COBISS (dCOBISS), https://d.cobiss.net/repository/, obiskano 28. 7. 2022

[8] Informacijski sistem o raziskovalni dejavnosti v Sloveniji (SICRIS), https://www.sicris.si/, obiskano 28. 7. 2022

[9] Programska oprema COBISS3, https://www.cobiss.si/c3/, obiskano 28. 7. 2022

Primerjalna analiza: ali naj pisanje predlog CSS ostane ločeno od programske kode JavaScript?

Gregor Jošt, Luka T. Korošec, Matej Žvan

3fs, d.o.o., Kranj, Slovenija
gregor.jost@3fs.si, luka.korosec@3fs.si, matej.zvan@3fs.si

Sinopsis Definiranje stilnih predlog CSS v datotekah JavaScript, oz. CSS-v-JS (angl. CSS-in-JS) predstavlja sodoben pristop k oblikovanju spletnih strani, ki ne zahteva uporabe zunanjih datotek CSS, ampak celotno oblikovanje CSS definiramo v programskem jeziku JavaScript. Pri tem ohranimo vso sintakso, pravila in razumevanje CSS, le implementacijo prestavimo v datoteko JavaScript. Ogrodja za razvoj odjemalskega dela spletnih strani ne podpirajo CSS-v-JS pristopa, zato imamo v ta namen na voljo več kot 50 knjižnic.
Po drugi strani ima običajen pristop oblikovanja spletnih aplikacij še vedno jasne prednosti. Uporaba različnih metodologij ali tehnologij ostaja pogost pristop oblikovanja spletnih strani, ki ne zahteva učenja novega ogrodja. CSS-v-JS se vse pogosteje uveljavlja kot alternativa, saj ima CSS svojevrstne težave, ki se kažejo v prekrivanju stilov, pomanjkanju izolacije in v pomanjkanju tehnologije za odstranjevanje neuporabljenih stilov (angl. dead code elimination).
Glede na vse večjo priljubljenost in uporabnost pristopa CSS-v-JS smo se v sklopu članka osredotočili na primerjavo omenjenega pristopa z običajnim načinom. V primerjalni analizi se bomo omejili na knjižnico React, za vključitev CSS-v-JS pa bomo uporabili knjižnico styled-components, saj ta velja za eno bolj priljubljenih knjižnic za rabo v ta namen.

Ključne besede:
CSS
CSS-v-JS
React
styled-components

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.5

1 Uvod

Tehnika pisanja stilov CSS v jeziku JavaScript ni nova, saj se diskusija o smiselnosti in uporabnosti tovrstnega pristopa odvija že vrsto let. Spletne strani Reddit, Atlassian, IMDb in Auth0 [1] [2] CSS-v-JS (angl. CSS-in-JS) omenjeno tehniko že uporabljajo, kar zagotovo priča v prid očitnim prednostim tehnike pred bolj tradicionalnem pristopom, tj. Pred pisanjem stilov v ločenih datotekah CSS.

Pogosto se tehnika CSS-v-JS zamenjuje z dodajanjem stilov CSS neposredno na elemente DOM (stili-v-vrsti, angl. Inline-styles), vendar ta predstavlja popolnoma drugačen pristop. Kljub definiranju stilov v datoteki JavaScript večina knjižnic, ki podpira CSS-v-JS, stile CSS prevede iz kode JavaScript, s čimer ustvari unikatna, samodejno generirana imena razredov, ki jih nato pripne elementom DOM [3]. Z uporabo takšnega načina zagotovimo, da se razredi stilov CSS ne prepisujejo (ali pa, da se sploh ne uporabljajo kljub morebitni definiciji v datoteki CSS). Obenem s tem omejimo tudi apliciranje stilov zgolj na komponento, ki jo določen stil potrebuje. Ravno (pravilno/konsistentno) poimenovanje razredov in prepisovanje stilov veljata za pogosta izziva pri pisanju predlog CSS. Pomembno je izpostaviti, da tehnika CSS-v-JS običajno ne vpliva na hitrost delovanja aplikacij. Še več, CSS-v-JS lahko celo izboljša delovanje, saj spletna stran naloži le stile, ki jih posamezne komponente potrebujejo.

Tehnika CSS-v-JS ima tudi svojevrstne izzive. Dejstvo, da se pisanje stilov CSS prestavi v jezik JavaScript, lahko razvijalcem predstavlja omejitev (še posebej, če so ti vajeni pisati stile CSS v temu namenjeni datoteki). Obenem lahko tvegamo izgubo dodatne sintakse in prednosti SCSS ter podobnih generatorjev CSS, ki jih knjižnice CSS-v-JS načeloma ne podpirajo.

Literatura na področju primerjave tehnike CSS-v-JS in pristopa pisanja stilov CSS v ločeni datoteki je omejena, zato smo v sklopu tega prispevka opravili primerjalno analizo omenjenih pristopov. Omejili smo se na knjižnico React, za tehniko CSS-v-JS pa uporabili priljubljeno knjižnico styled-components.

Članek je sestavljen iz naslednjih poglavij. Najprej bomo v sklopu drugega poglavja predstavili sodobne načine pisanja predlog CSS, pri čemer se bomo osredotočili na module CSS, metodologije, SCSS in spremenljivke. V tretjem poglavju bomo predstavili tehniko CSS-v-JS in si podrobneje pogledali knjižnico styled-components. Sledili bosta poglavji primerjave stilov CSS s tehniko CSS-v-JS, kjer bomo najprej opredelili primer in ga razvili z uporabo obeh pristopov, opisali orodja, s katerimi si bomo pomagali pri pridobivanju podatkov, potrebnih za primerjavo (četrto poglavje) in nato podali še rezultate analize (peto poglavje). V šestem poglavju bomo povzeli ugotovitve in podali zaključne misli.

2 Sodobni načini pisanja predlog CSS

Predloge CSS v osnovi vsebujejo skupek deklaracijskih blokov. Pri pisanju uporabljamo selektorje in jim nastavljamo različne vrednosti za lastnosti, pri tem pa moramo biti pozorni na specifičnost selektorjev in njihov vrstni red. Selektorji lahko poljubno spreminjajo lastnosti, dodajajo nove ali spreminjajo obstoječe, ne morejo pa izbrisati že napisanih lastnosti.

Brskalniki pri izrisu elementov najprej upoštevajo specifičnost selektorjev. V kolikor se pojavita dva identična selektorja, brskalnik uporabi tistega, ki je napisan nižje (angl. Cascading-style-sheet). Pri tem velja omeniti, da obstajajo manjše razlike kako različni brskalniki implementirajo jezik CSS.

Upoštevajoč ta pravila ugotovimo, da lahko implementacija in vzdrževanje predlog CSS predstavljata številne izzive. Slednji pridejo še posebej do izraza, če razvijamo obsežnejše aplikacije v skupinah več razvijalcev. Pri tem so nam lahko v pomoč različna orodja, s katerimi predlog CSS ne pišemo, ampak generiramo. Tak pristop posledično omogoči hitrejši razvoj z manj napakami.

2.1 Generatorji CSS

Generatorji CSS (angl. CSS preprocessors) omogočajo, da namesto neposredne definicije predlog CSS pišemo kodo, ki se bo prevedla v ustrezno predlogo CSS [4]. Med najbolj razširjena generatorja CSS štejemo SCSS in LESS.

Generatorji CSS omogočajo uporabo pogojnih stavkov, zank, funkcij, predpripravljenih blokov (angl. Mixinov) in spremenljivk za optimizirano pisanje predlog CSS. Pri tem lahko poljubno nastavimo spremenljivke, ki omogočijo bolj poenoten končni izdelek. Z uporabo spremenljivk, funkcij in predpripravljenih blokov lahko kodo napišemo enkrat in jo nato večkrat uporabimo (princip DRY), kar omogoča lažje vzdrževanje in spreminjanje predlog.

Številne prednosti generatorjev lahko po drugi strani vodijo v površnost in zlorabo moči generatorjev. Velikokrat se izkaže, da je generirana predloga večja od zapisa brez generatorjev, saj ti omogočajo gnezdenje selektorjev. Slednje lahko hitro privede do nujnega upoštevanja preveč specifičnih pravil, ki jih moramo ponoviti, ne moremo pa jih uporabiti drugje.

2.2 Metodologije

Glavni izziv pri pisanju predlog CSS predstavlja razumevanje zapisa. Običajno ne želimo, da potrebuje uporabnikov brskalnik več različnih predlog, zato te združimo v čim manj dokumentov, kar lahko zaradi velikosti datotek vodi v nepreglednost in težjo razumljivost predlog CSS. V praksi so se zato uveljavile različne metodologije, ki poizkušajo olajšati razumevanje napisanih predlog. Med slednje štejemo BEM, OOCSS, AMCSS, SMACSS in SUITCSS [5]. Z uporabo omenjenih metodologij imamo definirana jasna pravila reševanja problemov. Velikokrat metodologija določi tudi način poimenovanja selektorjev in prav z uporabo strukturiranega pristopa pridobimo na razumevanju napisanega.

Če se osredotočimo na BEM (angl. Block-element-modifier), lahko opazimo, da je osrednji cilj metodologije ponovna uporaba kode (angl. Code reuse). Slednje dosežemo s konsistentnim poimenovanjem in omejitvijo obsega. Po metodologiji BEM ni nič globalno, vsaka komponenta je namreč zapisana v svojem dokumentu in nima vpliva na druge komponente.

Obseg predloge CSS omejimo tako, da uporabljamo izključno razrede CSS, pri čemer vsak razred dobi imenski prostor. BEM predpiše poimenovanje selektorjev kot »blok__element–različica«, vsak izmed treh elementov pa ima svoj pomen. Blok predstavlja imenski prostor komponente, zato mora vsak selektor na začetku vsebovati ime bloka, za katerega nastavlja pravila. Element je podenota komponente, ki ne more delovati samostojno, različica pa je sprememba na bodisi bloku bodisi elementu. V kolikor se striktno držimo metodologije, lahko komponente premikamo med različnimi projekti in ohranjamo enoten izgled.

2.3 Moduli CSS

Moduli CSS se uporabljajo v okoljih JavaScript oz. TypeScript, kjer v kodo vključimo predlogo CSS. Cilj modulov je omejitev obsega delovanja napisanih stilov CSS, pri čemer se zanašamo na to, da bo orodje naše selektorje preimenovalo tako, da ne bodo imeli zunanjega vpliva [6].

2.4 Orodje Postcss

Orodje Postcss služi transformaciji predlog CSS z uporabo jezika JavaSript, kjer lahko CSS poljubno spreminjamo, dodajamo ali brišemo. V praksi se uporabljajo različni, že napisani vtičniki za postcss, na primer postcss-inline-svg[1], ki vse slike SVG doda neposredno v predlogo kot »data:image…«. Vtičnik autoprefixer[2] pa stile CSS transformira tako, da določenim lastnostim doda predpone za brskalnike (npr. -ms, -webkit).

2.5 Spremenljivke CSS

Spremenljivke CSS predstavljajo entitete, ki jih definirajo razvijalci, vsebujejo pa lahko različne vrednosti [7]. Te entitete so nato dosegljive kjerkoli v stilih CSS, običajno pa se uporabljajo, ko moramo določene vrednosti lastnosti spreminjati med delovanjem spletnih strani, na primer pri izdelavi različnih tematik. Obenem tudi omogočajo, da se izognemo ponavljanju določenih vrednosti.

Spremenljivke CSS nastavimo s specifično notacijo (primer: »main-color: black;«) in do njih dostopamo z rezervirano besedo »var« (primer: »color: var(--main-color);«).

3 Definicija stila CSS v jeziku JavaScript (CSS-v-JS)

CSS-v-JS predstavlja tehniko, s katero se oblikovanje komponent definira v kodi JavaScript (oz. TypeScript) in ne v zunanjih datotekah CSS [8]. S poznavanjem sintakse JavaScript torej ustvarimo, dodajamo in upravljamo s stilom CSS. Kljub temu omenjena tehnika ne nadomesti poznavanja sintakse CSS; še vedno moramo razumeti načine dodajanja stilov na elemente DOM, dedovanje in poznati vrste lastnosti, ki jih podpira sintaksa itd.

Kljub dejstvu, da imamo trenutno za razvoj CSS-v-JS na voljo več kot 50 JavaScript knjižnic [3], vse izmed njih posedujejo nekatere skupne značilnosti:

- Omejen doseg CSS, kar podpirajo tudi moduli CSS, omenjeni v prejšnjem poglavju. Knjižnice CSS-v-JS običajno ustvarijo unikatna imena razredov, ki se dodajo na element DOM [3]. Na takšen način je doseg stilov CSS omejen (običajno na eno komponento), prav tako pa je odveč skrb, da bi prišlo do prekrivanja imen razredov CSS.
- Knjižnice skrbijo za samodejno dodajanje predpone (angl. Vendor prefix) lastnostim CSS, ki so še v fazi razvoja ali niso podprte na vseh brskalnikih (primer: -webkit-transition).
- Vse sodobne knjižnice ne dodajajo stilov CSS neposredno na elemente DOM (stili-v-vrsti, angl. Inline-styles), saj ima tovrstno dodajanje stilov precej omejitev, npr. Pomanjkanje podpore psevdo-razredom, psevdo-elementom in medijskim poizvedbam (angl. Media-query) [9], zato je večina knjižnic takšen pristop v celoti umaknila.
- V povezavi z zgornjo točko, knjižnice za razvoj CSS-v-JS podpirajo celoten nabor sintakse CSS, vključno s psevdo-razredi, medijskimi poizvedbami itd., saj se zanašajo na dinamično ustvarjanje razredov.

Knjižnice za razvoj CSS-v-JS se medsebojno razlikujejo, saj so nekatere namenjene specifično določenemu ogrodju JavaScript, medtem ko druge niso odvisne od ogrodja [9]. Med slednje spadajo Emotion, Treat in JSS, medtem ko sta knjižnici styled-components in Stitches namenjeni ogrodju oz. Knjižnici JavaScript (v tem primeru gre za knjižnico React). V prispevku smo se omejili na knjižnico React, zato smo za potrebe raziskave izbrali eno bolj priljubljenih knjižnic, in sicer styled-components.

[1] https://github.com/TrySound/postcss-inline-svg
[2] https://github.com/postcss/autoprefixer

3.1 Knjižnica styled-components

Knjižnica styled-components je namenjena razvoju aplikacij React oz. React Native in skladno z definicijo CSS-v-JS omogoča definiranje stilov na nivoju komponente. Ta knjižnica predstavlja kombinacijo jezika JavaScript (oz. TypeScript) in CSS. Osnovni koncept za pisanje stilov predstavljajo označevalne predloge (angl. Tagged Template Literals) ECMAScript 2016 [2].

Osnovna ideja styled-components je v odstranjevanju preslikave med komponentami in stili. Namesto predloge CSS tako implementiramo komponento React, znotraj katere s pomočjo označevalne predloge definiramo stile CSS. Knjižnica nato ustvari predlogo CSS in razrede pripne vozliščem DOM preko »className« lastnosti elementov React. Predlogo CSS nato vključi na konec elementa »head« spletne strani med izvajanjem (angl. Runtime). Osnovna sintaksa komponent, implementiranih z razširitvami knjižnice styled-components, je naslednja:

```
const ComponentName = styled.element`
      css_property: css_value;
`;
```

Iz zgornjega izseka kode je razvidno, da objekt »styled« podpira vse elemente HTML, znotraj označevalne predloge (vse, kar je znotraj dveh ostrivcev, ``) pa lahko stile CSS definiramo enako kot bi jih v običajni datoteki CSS. Specifičen primer takšne komponente prikazuje naslednji izsek programskega koda:

```
import styled from 'styled-components';

const Container = styled.div`
  margin: 2.5em auto;
  border: 1px solid #dadada;
  padding: 1em;
  min-height: 25em;
`;

const Button = styled.button`
  border: 0.1em solid #999999;
  cursor: pointer;
`;
```

Kot je razvidno, smo definirali dve komponenti; prvo, poimenovano »Container« (predstavlja vsebnik za spletno stran, razširi obstoječ element HTML »div«) in drugo z imenom »Button« (predstavlja generičen gumb, razširi element HTML »button«). Obe komponenti lahko uporabimo na enak način kot običajno komponento React, na primer:

```
function App() {
 return (
   <Container>
     <Button>Hello world</Button>
   </Container>
 );
}
```

Knjižnico styled-components odlikujejo tudi druge lastnosti in funkcionalnosti:

- Ob možnosti razširitve obstoječih elementov DOM imamo tudi možnost razširitve lastnih, obstoječih komponent React.

- Dostop do lastnosti »props« znotraj stilov CSS, kar velja za eno ključnih prednosti knjižnice styled-components. Na takšen način lahko glede na lastnosti komponente ustrezno (in dinamično) definiramo stil CSS. Znotraj označevalne predloge imamo namreč dostop do lastnosti »props«, ki so lahko kakršnega koli tipa.
- Podpora temam: knjižnica podpira definiranje različnih tem in v ta namen ponuja specifično komponento. Vsi elementi znotraj komponente imajo nato dostop do stilov, opredeljenih v temi. Na takšen način je zelo preprosto definirati na primer svetlo oz. Temno temo spletne strani.
- Knjižnica podpira definicijo globalnih stilov s pomočjo temu namenjene funkcije. Komponento, ki predstavlja globalne stile CSS, enkrat vključimo na najvišjem nivoju strukture aplikacije, podobno kot v primeru globalne datoteke CSS.

Podobno kot razvoj običajnih aplikacij React tudi v primeru styled-components struktura ni vnaprej definirana in strogo predpisana. Običajno imamo dva pristopa, in sicer: 1) komponente styled-components so v istem imeniku kot običajne komponente React (npr. Komponenta »Header.tsx« bi imela v istem imeniku še komponento »Header.styled.tsx«) ali 2) komponente styled-components so znotraj istega imenika (običajno poimenovanega »styles«) razdeljene na več datotek (npr. »Header.styled.tsx«, »Footer.styled.tsx« in »Button.styled.tsx«).

Kljub jasnim prednostim, kot so ponovna uporaba kode, dostop do lastnosti »props«, dinamično oblikovanje, podpora temam in zmanjšanju napak med razvojem, ima uporaba knjižnice tudi določene slabosti oz. Izzive. Med slednje lahko štejemo težje razhroščevanje (zaradi potencialnega gnezdenja stilov znotraj ene komponente), povečanje velikosti virtualnega DOM-a knjižnice React, prav tako pa izgubo podpore razširitvam CSS, kot je npr. SASS.

4 Primerjava stilov CSS s pristopom CSS-v-JS

Za relevantno primerjavo različnih stilov pisanja CSS smo potrebovali dve primerljivi aplikaciji. Za namen objektivnosti primerjave smo se najprej odločili za implementacijo aplikacije, ki uporablja CSS, in jo nato prepisali v obliko CSS-v-JS.

Za primerjavo bomo razvili preprosto aplikacijo, ki bo vsebovala dovolj funkcionalnosti, da bomo zajeli različne lastnosti CSS, saj želimo ugotoviti, ali sta pristopa pisanja CSS primerljiva. V ta namen bomo razvili skrajševalnik URL-jev, aplikacija pa bo vsebovala več strani:

- stran za vnos URL-ja in njegovo okrajšavo,
- stran za prikaz okrajšanih URL-jev,
- stran, s predstavitvijo podjetja 3fs, na kateri bo obsežno besedilo in nekaj slik,
- nalagalnik (angl. Loader) in
- stran za primer napake (404).

Z zgoraj omenjenimi stranmi bomo pokrili animacije, vnosna polja in gumbe, tabele, besedilo, različne pisave ter rastrske in vektorske fotografije. Obenem bomo poskrbeli, da bodo vse strani prilagodljive tako majhnim, vmesnim in velikim zaslonom.

Prepis aplikacije v CSS-v-JS bo potekal sistematično, da bosta končna produkta ekvivalentna tako v smislu uporabljenih stilov kot v izgledu spletne aplikacije. Za sledenje spremembam v kodi bomo uporabili GIT, za prepis pa ustvarili novo vejo. Obe aplikaciji bomo namestili na spletno platformo Heroku, kjer bomo kasneje izvedli analizo.

Pri primerjavi obeh pristopov smo se osredotočili na merljive in neposredno primerljive metrike. Najprej bomo primerjali število uporabljenih knjižnic in velikost imenika »node_modules«, za kar bomo uporabili orodje za upravljanje paketov yarn in klasično orodje Linux du. Nato se bomo z odprtokodnim orodjem cloc[3] posvetili razliki v številu datotek in vrstic kode, saj to orodje v svojem poročilu prikaže tudi tip datoteke in tip vrstice v datotekah (tj. Prazna, komentar, koda). Sledi merjenje hitrosti izgradnje (angl. Build) celotne aplikacije z orodjem yarn in klasičnim orodjem Linux time[4] ter primerjava velikosti končnega programskega paketa. Za konec si bomo prihranili primerjavo hitrosti nalaganja obeh spletnih strani, ki jo bomo izmerili z orodjema Google Chrome Lighthouse in SiteSpeed.io.

Analiza različnih metrik se bo izvajala na istem računalniku ob istem času, s čimer zmanjšamo vpliv zunanjih dejavnikov na rezultate analize. V izogib enkratnim odstopanjem v merjenju bomo opravili več meritev in nato izračunali povprečje ter mediano, kjer je to primerno (npr. Pri primerjavi števila knjižnic ni potrebno opraviti več meritev).

5 Rezultati analize

Skladno z opredeljenim v prejšnjem poglavju bomo v tem sklopu izvedli primerjavo in podali rezultate analize.

5.1 Velikost imenika »node_modules« in število uporabljenih knjižnic

Velikost imenika »node_modules« in število uporabljenih knjižnic vpliva na diskovni prostor, zato smo ti dve metriki združili v skupno podpoglavje. Pri izbiri načina pisanja CSS je namreč vredno upoštevati tudi ta vidik.

Pri sodobnem pisanju spletnih aplikacij, ki temeljijo na jeziku JavaScript, je veliko kode, ki poganja aplikacijo, že spisane v obliki različnih knjižnic. Z uporabo orodij za upravljanje paketov, kot sta prej omenjena Yarn in NPM (Node Package Manager), na enostaven način pridobimo knjižnice in jih uporabimo v svoji aplikaciji. Dodane knjižnice tudi same pogosto uporabljajo kakšno drugo knjižnico, tako da lahko število vseh uporabljenih knjižnic naše aplikacije hitro naraste. Vse te knjižnice so lokalno nameščene v imenik »node_modules«.

5.1.1 Velikost imenika »node_modules«

Z uporabo prej omenjenega orodja du lahko hitro ugotovimo velikost imenika »node_modules«. Z ukazom `rm -rf node_modules && yarn install && du -hd 0 node_modules` najprej izbrišemo celoten imenik, nato namestimo vse potrebne knjižnice, na koncu pa pogledamo še velikost novonastalega imenika. Zastavica `-h` pomeni, da je izpis, ki je privzeto v bajtih, pretvorjen v človeku prijazno obliko (megabajti), medtem ko zastavica `-d 0` orodju pove, da nas zanima samo velikost imenika na nivoju 0 (tj. »node_modules«). Postopek ponovimo na obeh vejah, pri CSS in CSS-v-JS.

Rezultat postopka je pokazal, da je velikost imenika na veji CSS 362 Mb, medtem ko je imenik na veji CSS-v-JS velik 373 Mb. Slednji je tako za 3,0 % ali 11 Mb večji.

5.1.2 Število uporabljenih knjižnic

Pri pridobivanju števila uporabljenih knjižnic na prvem nivoju (angl. Direct dependencies) si lahko pomagamo z vpogledom v datoteko package.json, ki se uporablja za beleženje dodanih knjižnic. Upoštevali bomo tako tiste, ki

[3] https://github.com/AlDanial/cloc
[4] https://www.gnu.org/software/time/

so uporabljene samo pri razvijanju aplikacije (razdelek »devDependencies«), kot tudi tiste, ki so v programski paket vključene kasneje (angl. Bundle) (razdelek »dependencies«).

Za pridobivanje števila knjižnic na vseh nivojih (angl. Indirect dependencies) moramo uporabiti orodji yarn in wc[5]. Ukaz, ki ga poženemo v ukazni vrstici, je `yarn list | wc -l`. Prvi del izriše drevesno strukturo vseh knjižnic na način, da je v vsaki vrstici izpisana po ena knjižnica. S preštevanjem števila vseh vrstic dobimo skupno število vseh uporabljenih knjižnic. V ta namen uporabimo orodje wc z zastavico `-l`, ki orodju pove, naj prešteje vse vrstice. Nato odštejemo 3 vrstice, ki jih doda yarn, in dobimo realno število. Tabela 1 prikazuje rezultat analize.

Tabela 1: Primerjava števila uporabljenih knjižnic.

	Število uporabljenih knjižnic na prvem nivoju	Število knjižnic na vseh nivojih
CSS	20	21
CSS-v-JS	35	40
Razlika	21,7 %	2,8 %

Kot je razvidno iz zgornje tabele, uporabljamo 23 knjižnic pri osnovnem pristopu CSS, medtem ko pri CSS-v-JS uporabljamo 28 knjižnic. Število le-teh se tako poveča za 21,7 %.

Število vseh uporabljenih knjižnic pri CSS pristopu je 3926, medtem ko jih pri pristopu CSS-v-JS uporabljamo 4035, kar je za 2,8 % več.

5.2 Število datotek in vrstic kode

Z večanjem števila datotek se obvladljivost projekta otežuje, zato želimo obdržati kar se da nizko število, podobno je tudi s številom vrstic kode. Če bi za ekvivalenten rezultat potrebovali dvakrat več kode, je tak pristop manj uporaben. V tem poglavju bomo analizirali, kako se število datotek in vrstic kode spremeni, ko aplikacijo prepišemo iz CSS v ekvivalenten pristop CSS-v-JS.

Uporabili bomo odprtokodno orodje cloc (angl. Count lines of code). Le-to bo z uporabo zastavice `--vcs git` preštelo in analiziralo le datoteke, ki so vključene v sistem sledenja spremembam git, ignoriralo pa bo datoteke, ki jih ta sistem ignorira (npr. Imenik »node_modules«).

Z vpisom ukaza `cloc --vcs git .` v ukazno vrstico pridobimo tabelo, ki število datotek in vrstice kode združi glede na programski jezik (npr. TypeScript, SCSS, JSON, itd). Za vsak jezik tako pridobimo število datotek, praznih vrstic, komentarjev in število vrstic dejanske kode.

Analiza je pokazala, da se s prepisom aplikacije v pristop CSS-v-JS pravzaprav zmanjša število datotek iz 85 na 70, kar predstavlja 17,6 % razliko. Slednje lahko pripišemo dejstvu, da v primeru CSS-v-JS ne potrebujemo predpripravljenih blokov (angl. Mixin), saj imamo možnost pisanja funkcij neposredno v komponenti CSS-v-JS. Obenem se je število vseh vrstic dejanske kode s prepisom zmanjšalo za 0,7 %, iz 3389 (veja CSS) na 3367 (veja CSS-v-JS). Velikost razlike priča o tem, da moramo v obeh pristopih še vedno pisati isti CSS, spremeni se le lokacija.

Rezultati povedo, da lahko z uporabo pristopa CSS-v-JS pričakujemo manjše število datotek, pri čemer se število potrebnih vrstic kode bistveno ne spremeni.

[5] https://linux.die.net/man/1/wc

5.3 Hitrost graditve aplikacije

Aplikacijo je pred namestitvijo na strežnik potrebno zgraditi. Za gradnjo poskrbi orodje react-scripts, ki je ključen del razvojnega okolja React, aktiviramo pa ga preko ukaza `yarn build`. Za meritev bomo upoštevali čas, ki ga poroča orodje time. Prav tako bomo opravili 5 meritev in izračunali povprečje, s tem pa zmanjšali vpliv ostalih procesov na računalniku na hitrost gradnje. Za meritev bomo v ukazno vrstico vpisali ukaz `rm -rf build && time yarn build`. Ukaz bo najprej odstranil rezultat gradnje, tako da bo morala vsaka gradnja aplikacijo zgraditi od začetka, nato pa bo orodje time gradnjo zagnalo in merilo.

Povprečje petih meritev na veji CSS znaša 5,52 sekund, na veji CSS-v-JS pa 4,81 sekund. Vidimo lahko, da se z uporabo pristopa CSS-v-JS čas gradnje zmanjša za 12,7 % ali za 0,71 sekunde. Predpostavljamo, da je čas krajši zaradi manjšega števila datotek kot tudi zaradi tega, ker obdelovanje datotek CSS ni potrebno, potrebujemo le obdelati datoteke JavaScript, kar zmanjša število korakov.

5.4 Hitrost delovanja in velikost spletne strani

Ko je stran zgrajena, jo namestimo na strežnik in tako postane dostopna vsem na spletu. Hitrost je pri tem ključnega pomena. Aplikacija se mora naložiti dovolj hitro, saj to vpliva na uporabniško izkušnjo. V tem poglavju bomo primerjali rezultate testov, opravljenih z orodjema Lighthouse in Sitespeed.io.

Najprej smo obe veji, CSS in CSS-v-JS, naložili na spletno platformo Heroku. Vsaka aplikacija je nato dostopna na svojem spletnem naslovu, kjer sta neodvisni ena od druge. V podpoglavjih sledi merjenje hitrosti in velikosti spletne strani s prej omenjenima orodjema.

5.4.1 Analiza z Lighthouse

Lighthouse je orodje, vgrajeno v spletni brskalnik Google Chrome. Za izvedbo testa zadostuje obisk spletne strani, ki jo želimo analizirati z zagonom analize Lighthouse (najdemo jo med zavihki v razvijalskih orodjih brskalnika).

Na obeh vejah so vsi glavni indikatorji (hitrost, dostopnost, dobre prakse in optimizacija za iskalnike) pokazali popolno oceno (vrednost 100/100), zato lahko trdimo, da med stranema ni bistvene razlike.

Manjše razlike se kažejo v podrobnostih. Čas do interaktivnosti (angl. Time to interactive) je na veji CSS znašal 0,5 sekunde, na veji CSS-v-JS pa 0,3 sekunde. Prav tako je razlika vidna pri največjem vsebinskem prikazovanju (angl. Largest contentful paint), in sicer se le-ta zgodi pri 0,8 sekunde na veji CSS in pri 0,6 sekunde na veji CSS-v-JS. Slednje pripisujemo manjšemu številu datotek, ki jih mora brskalnik prenesti za ogled strani, saj datotek CSS ne rabi prenašati. Z uporabo http/2 protokola bi se zahtevki za datoteke izvajali vzporedno, kar bi pospešilo delovanje. Predpostavljamo, da bi to vplivalo na rezultate v prid pristopa CSS.

5.4.2 Analiza s Sitespeed.io

SiteSpeed.io je odprtokodno orodje za analizo in spremljanje spletnih strani. V primerjavi z Lighthouse zajame več metrik, hkrati pa podpira analizo več strani hkrati in zaporednih testiranj, ki jih uporabi za izračun povprečij.

Uporaba orodja je nekoliko bolj zahtevna kot pri Lighthouse. Najlažje ga je zagnati z uporabo virtualizacijskega orodja Docker, za katerega SiteSpeed.io ponuja vsebnik (angl. Container). Orodje zaženemo z vpisom ukaza `docker run –rm -v »$(pwd):/sitespeed.io« sitespeedio/sitespeed.io:25.5.1 https://<naslov-nase-spletne-strani>/`. Po zagonu orodje v trenutnem imeniku ustvari poročilo HTML, ki ga lahko odpremo v brskalniku in na tak način preverimo rezultate. Na voljo imamo širok nabor metrik, zato izberemo tiste, ki so najbolj relevantne in ilustrativne za prikaz razlik med različnima načinoma pisanja CSS, ali tiste, ki so se med vejama najbolj razlikovale.

Tabela 2 prikazuje rezultate analize, v kateri so zbrane povprečne vrednosti tekom treh meritev na petih podstraneh naše spletne aplikacije in absolutna in relativna razlika med povprečjema na različnih vejah.

Tabela 2: Rezultati analize s Sitespeed.io.

Angleško ime metrike	CSS	CSS-v-JS	Absolutna razlika	Razlika v odstotkih
Coach performance score	81	84	+3	+3,7%
Best Practice score	99	100	+1	+1,0 %
Total requests	12	10	-2	-16,7 %
HTML size	1,3 KB	1,1 KB	-0,2 KB	-15,4 %
CSS size	3,2 KB	0 b	-3,2 KB	-100 %
Javascript size	70,2 KB	86,3 KB	+16,1 KB	+22,9 %
Size SUM	74,7 KB	87,2 KB	+12,5 KB	+16,7 %
First Paint	396 ms	402 ms	+6 ms	+1,5 %
Fully Loaded	1557 ms	1286 ms	-270 ms	-17,4 %
Largest Contentful Paint	718 ms	683 ms	-35 ms	-4,9 %
First Contentful Paint	396 ms	402 ms	+6 ms	+1,5 %
First Visual Change	421 ms	417 ms	-4 ms	-1,0 %
Speed Index	696 ms	664 ms	-32 ms	-4,6 %
CPU Style Layout	19 ms	20 ms	+1 ms	+5,3 %
CPU Script Evaluation	33 ms	45 ms	+12 ms	+36,4 %

Vidimo lahko, da je celotna ocena strani za 3,7 % višja pri pristopu CSS-v-JS, kar predstavlja povprečje vseh meritev na spletni strani. Število zahtevkov za datoteke je pričakovano višje pri pristopu CSS. Prav tako je pričakovana velikost CSS datotek pri pristopu CSS-v-JS 0 b, medtem ko se velikost datotek JavaScript poveča za kar 22,9 %. V povprečju je velikost vseh datotek pri pristopu CSS-v-JS večja za 16,7 %. Posledično (a morda ne intuitivno) je tudi čas nalaganja spletne strani pri CSS-v-JS manjši, saj mora ta kljub večji velikosti datotek le-teh prenesti več, kar prinese 17,4 % pospešitev. Indeks hitrosti kaže, da je pristop CSS v povprečju za 4,6 % počasnejši od pristopa CSS-v-JS. Pričakovano se poveča čas, ki ga procesor porabi za evaluacijo datotek JavaScript, in sicer kar za 36,4 %.

5.4.3 Velikost programskega paketa

Ob gradnji aplikacije orodje react-scripts priročno prikaže velikost izdelanega programskega paketa. Ta paket ne vsebuje slik, tekstovnih (razne licence) in razhroščevalnih datotek, ki pomagajo pri razvoju aplikacije. Paket smo pretvorili v obliko gzip.

Na veji CSS velikost programskega paketa znaša 89 KB, medtem ko je na veji CSS-v-JS paket velik 104 KB, kar je za 16,9 % ali 15 KB več od paketa na veji CSS.

6 Zaključek

V prispevku smo opravili primerjavo pristopa CSS-v-JS in pisanja stilov CSS v ločeni datoteki. Najprej smo predstavili sodobne pristope pisanja običajnih stilov CSS, nato smo opisali pristop CSS-v-JS, ki združuje prednosti komponentnega razvoja in pisanja stilov CSS.

Sledila je primerjalna analiza obeh pristopov na preprostem primeru, ki je vključeval različne aspekte spletnih aplikacij. Ugotovili smo, da sta pristopa z vidika hitrosti delovanja precej primerljiva. Pričakovana odstopanja smo zaznali pri velikosti programskega paketa, vendar le-ta ne predstavlja razloga za uporabo enega pristopa pred drugim. Po drugi strani je število paketov pri nalaganju spletne strani v primeru pristopa CSS-v-JS nižje kot v primeru pristopa CSS, kar ni bilo pričakovano.

Pri interpretaciji rezultatov moramo upoštevati nekaj omejitev. Kompleksnost in velikost aplikacije, ki smo jo razvili, ni primerljiva z običajnimi poslovnimi aplikacijami. Poudarek smo dali tudi čim bolj podobnemu razvoju obeh aplikacij, zato nismo posebej optimizirali določenega pristopa. Obenem smo izpustili tudi subjektivne metrike, predvsem hitrost pisanja kode, preglednost, težavnost vzdrževanja, razširljivost itd. Nenazadnje se omejitev pojavi tudi v izbiri knjižnice za pristop CSS-v-JS, pri čemer smo v našem primeru izbrali styled-components.

Očitnih prednosti tehnike CSS-v-JS ali slabosti običajnega pristopa pisanja stilov CSS ni zaznati, zato je končna odločitev o izbiri pristopa odvisna zgolj od preferenc razvojne skupine, znanja jezika JavaScript in nenazadnje tudi od obsega projekta, ki ga razvijamo.

Literatura

[1] C. Meade, "A Lukewarm Approval of CSS-in-JS," 20 10 2021. [Spletna stran]. Dosegljivo na: https://sparkbox.com/foundry/css_in_js_overview_css_in_js_pros_and_cons.

[2] styled-components. [Spletna stran]. Available: https://styled-components.com/.

[3] A. Pfeiffer, "A Thorough Analysis of CSS-in-JS," 3 6 2021. [Spletna stran]. Dosegljivo na: https://css-tricks.com/a-thorough-analysis-of-css-in-js/.

[4] MDN, "CSS preprocessor," 10 8 2021. [Spletna stran]. Dosegljivo na: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor.

[5] "BEM," [Spletna stran]. Dosegljivo na: https://en.bem.info/methodology/faq/#how-does-bem-differ-from-oocss-amcss-smacss-suitcss.

[6] R. Rendle, "What are CSS Modules and why do we need them?," 10 8 2021. [Spletna stran]. Dosegljivo na: https://css-tricks.com/css-modules-part-1-need/.

[7] MDN, "Using CSS custom properties (variables)," 10 7 2022. [Spletna stran]. Dosegljivo na: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties.

[8] A. Rudenko, "CSS-in-JS support in DevTools," [Spletna stran]. Dosegljivo na: https://developer.chrome.com/blog/css-in-js/. [Accessed 26 2 2021].

[9] A. Monus, "An Introduction to CSS-in-JS: Examples, Pros, and Cons," 9 9 2019. [Spletna stran]. Dosegljivo na: https://webdesign.tutsplus.com/articles/an-introduction-to-css-in-js-examples-pros-and-cons--cms-33574.

Praktični primeri pristopov za izboljšavo hitrosti in delovanja React aplikacij

Leon Pahole

Povio Inc., San Francisco, United States
leon.pahole@poviolabs.com

Sinopsis Knjižnica React omogoča implementacijo sodobnih interaktivnih spletnih aplikacij. Temelji na komponentah, ki predstavljajo vizualne elemente v aplikaciji. Komponente se povezujejo v hierarhijo in definirajo logiko ter vizualno predstavitev aplikacije. Skozi proces razvoja lahko večanje števila komponent in njihove kompleksnosti vodita v upočasnjeno delovanje aplikacije. V prispevku smo predstavili pristope za analizo in izboljšavo delovanja spletnih aplikacij v knjižnici React. Razložili smo uporabo in interno delovanje knjižnice React ter predstavili tehnike za izvedbo performančnih analiz aplikacij. Na podlagi tega smo pripravili več praktičnih primerov optimizacij. Na koncu smo podali lastno mnenje in nekaj nasvetov na temo optimizacij v procesu razvoja spletnih aplikacij.

Ključne besede:
React
spletne aplikacije
performančna analiza
optimizacija delovanja

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.6

1 Uvod

Napredek na področju spletnih tehnologij je v zadnjih letih povzročil rast števila spletnih aplikacij. Na to nakazuje tudi dejstvo, da so spletne tehnologije JavaScript, HTML in CSS na vrhu lestvice najpopularnejših tehnologij v letu 2022 [1,2].

Podjetja vse več svojih produktov razvijajo v obliki spletnih aplikacij, saj lahko na tak način dosežejo širšo publiko, hkrati pa so aplikacije dostopne kadarkoli, na katerikoli napravi. Takšne aplikacije po večini niso namenjene zgolj statični predstavitvi informacij, temveč služijo kot interaktiven produkt s pripadajočo poslovno logiko in profesionalnim izgledom.

Grajenje takšnih aplikacij zahteva previdno planiranje, fleksibilnost logike in enostavno programsko kodo, saj je razvoj dolgotrajen in se velikokrat nikoli ne zaključi. Hkrati je treba poskrbeti za sprejemljivo hitrost izvajanja aplikacije, predvsem, če je ta namenjena javni uporabi. Statistike obnašanj uporabnikov na spletu namreč konsistentno kažejo, da hitrost nalaganja in delovanja strani vplivata na njeno kredibilnost in sposobnost zadrževanja uporabnikov (angl. user retention) [3].

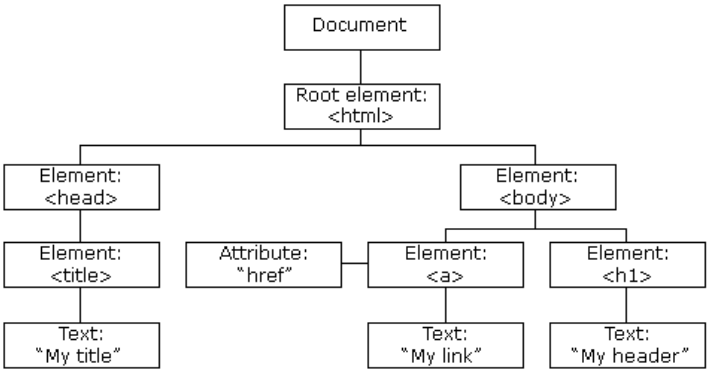
V tem prispevku bomo predstavili knjižnico React.js (v nadaljevanju React) kot eno izmed rešitev za grajenje interaktivnih, hitrih in za vzdrževanje prijaznih spletnih aplikacij. Predstavili bomo uporabo in interno delovanje knjižnice, nato pa na podlagi primerov razložili nekaj tehnik za optimizacijo hitrosti delovanja React aplikacij.

2 O knjižnici React

Želimo implementirati interaktivno spletno aplikacijo. To zahteva manipulacijo z elementi na spletni strani, ne da bi pri tem le-to osvežili - npr. ob kliku na gumb odpremo pojavno okno in skrijemo gumb.

2.1 Vmesnik DOM

Struktura spletne strani je definirana s pomočjo označevalnega jezika HTML. Brskalniki v osnovi ponujajo programski vmesnik za upravljanje s strukturo strani, imenovan DOM (document object model) [4]. V DOM je struktura strani modelirana kot drevo elementov (slika 1). Vsak element ima lahko nič, enega ali več otrok in, razen korena drevesa, natanko enega starša. S pomočjo programskega jezika JavaScript lahko komuniciramo z vmesnikom DOM in manipuliramo z drevesno strukturo; npr. pridobimo podatke o enem ali večih elementih, jim spreminjamo lastnosti ter jih dodajamo ali brišemo.



Slika 1: Vizualizacija drevesa DOM.
Vir: [5].

Zgornja leva stran slike 2 prikazuje primer programske kode, ki komunicira z DOM. Tak pristop se odsvetuje za razvoj kompleksnih spletnih aplikacij iz naslednjih razlogov:

- Koda je manj berljiva in strukturirana. Razvijalec več časa posveti temu, *kako* (imperativno) razviti določen vmesnik, kot temu, *kaj* (deklarativno) naj ta vmesnik vsebuje.
- Če nismo pozorni na implementacijske podrobnosti, je lahko pristop počasen. V primeru, da sprememba v DOM zahteva spremembo v postavitvi strani ali izgledu elementov, bo brskalnik sprožil procesa prilagoditve strukture (angl. reflow) ali izrisa (angl. repaint), ki sta časovno potratna [6]. Z določenimi tehnikami [7] lahko sicer optimiziramo to obnašanje, vendar te optimizacije kodo še dodatno obtežijo, kar nas vodi nazaj v problem zmanjšane berljivosti in nestrukturiranosti.

Slika 2: Različni načini ustvarjanja komponent v spletni aplikaciji.
Vir: lasten.

2.2 Zakaj React?

React je odprtokodna knjižnica za enostavno grajenje uporabniških vmesnikov. Knjižnica ni omejena zgolj na razvoj spletnih aplikacij z delom z DOM v brskalnikih, je pa za ta namen najpogosteje uporabljena [8]. S knjižnico React lajšamo težave, ki jih prinaša neposredno upravljanje z DOM. React delo z DOM abstrahira in programerju omogoči prijaznejši, deklarativni vmesnik za implementacijo spletnih aplikacij. Poleg tega React uporablja algoritme, ki optimizirajo komunikacijo z DOM tako, da so spremembe čim manj potratne [9].

2.3 Komponente

Osnovni gradniki v Reactu so komponente, ki predstavljajo vizualne elemente v aplikaciji [10]. V praksi vmesnik razdelimo na več manjših komponent, kjer ima vsaka komponenta svojo (idealno le eno) nalogo. Komponente lahko velikokrat tudi ponovno uporabimo. Aplikacijo tvori več hierarhičnih komponent tako, da nastane drevo komponent. Komponente se v večini primerov preslikajo v več DOM elementov, kar pomeni, da drevo komponent ni enako drevesu DOM.

Slike 3, 5 in 7 prikazujejo primere komponent. Komponenta je zgolj JavaScript funkcija[1] (ti. funkcijska komponenta). V hierarhiji lahko otrok prejme lastnosti (angl. props) od starša. Komponente tipično opravijo določen izračun ali pripravijo podatke, nato pa vrnejo nekaj, kar izgleda kot HTML. V resnici gre zgolj za

[1] Alternativno se komponente lahko implementira tudi v obliki ES6 razredov.

poenostavljeno sintakso (angl. syntactic sugar), imenovano JSX, ki se med procesom grajenja (angl. build) prevede v klice funkcije *createElement*, ki je del knjižnice React [11].

Slika 2 prikazuje JSX sintakso (levo v sredini) in pripadajoče klice *createElement* (levo spodaj). Opazimo, da se za vsak element v JSX uporabi en klic s parametri *(značka, props, otroci)*. Klici so gnezdeni, s čimer se definira hierarhija (npr. *div* je starš *span*). Rezultat klica *createElement* je React element. Vsaka komponenta vrne en korenski React element in poljubno število otrok. Desna stran slike 2 prikazuje rezultat klica *createElement*. Vidimo lahko, da je React element v ozadju zgolj JavaScript objekt.

Komponente so same po sebi zgolj funkcije. Da povežemo komponente z React-om, je potrebno uporabiti funkcijo *createRoot*, kateri pošljemo korensko komponento v hierarhiji (v našem primeru *Parent*) [12]. S tem bo React izrisal podano hierarhijo komponent v izbran element.

2.4 Izris komponent

Rezultat klica funkcijske komponente je objekt, ki predstavlja React element. Komponente združimo v hierarhijo in jih pošljemo v React s pomočjo *createRoot*. Kako React iz hierarhije komponent izriše strukturo strani?

2.4.1 Prvi izris

Najprej bomo opisali postopek za prvi izris - ob nalaganju strani [13].

Pretvorba v React elemente. React najprej pridobi React elemente za vsako izmed komponent v hierarhiji. To stori tako, da rekurzivno evalvira (angl. render[2]) vsako funkcijo, ki predstavlja funkcijsko komponento, začenši s korensko komponento. Iz vsake komponente pridobi en React element.

Gradnja virtualnega DOM. React iz pridobljenih React elementov zgradi drevesno strukturo, imenovano virtualni DOM (angl. virtual DOM). Virtualni DOM ni dejanski DOM v brskalniku, temveč interna podatkovna struktura v Reactu v obliki JavaScript objekta. S takšno strukturo je delo lažje, kot z dejanskim DOM.

Generiranje mutacij (angl. mutation). React bo dobljen virtualni DOM pretvoril v zaporedje mutacij (sprememb), ki se morajo izvesti na DOM, da bo doseženo želeno stanje. Ker gre za prvi izris, bo potrebno ustvariti prav vse elemente v hierarhiji. Poudariti je treba, da React sam po sebi ne kliče DOM vmesnika brskalnika, temveč zgolj pripravi seznam mutacij v neodvisni obliki.

Manipulacija DOM v brskalniku. V tem delu se seznam mutacij pretvori v dejanske klice DOM (kot na sliki 2, zgoraj levo). Mutacije so aplicirane na optimalen način. Dejanske klice DOM generira knjižnica, ločena od Reacta, imenovana ReactDOMClient[3]. To pomeni, da je React dejansko neodvisen od okolja, v katerem se izvaja. Poleg uporabe v spletnih aplikacij je npr. popularen tudi za razvoj mobilnih aplikacij (React native).

2.4.2 Stanje komponent

Zgoraj opisani postopek velja za prvi izris. Namen interaktivnih aplikacij je, da se skozi čas stanje v komponentah spreminja; npr. ob kliku na gumb se v tabeli prikažejo podatki iz zalednega sistema. Poskusimo implementirati komponento, ki šteje, kolikokrat se je gumb kliknil. Leva stran slike 3 prikazuje naš prvi poskus.

[2] V angleški terminologiji prihaja do nekaj konfliktne terminologije glede besed render in rerender. Ti besedi namreč lahko označujeta tako klice funkcij, ki jih opravlja React, kot tudi izris vsebine na zaslon brskalnika. V tem prispevku v prvem kontekstu uporabljamo besedo evalvacija [komponente], v drugem pa izris.
[3] Pred React verzijo 18 je to bila knjižnica ReactDOM [14].

Ta koda ne deluje, saj React ob izrisu (tako prvem, kot vseh naknadnih) funkcijske komponente kliče (ti. evalvacija), kot se kličejo navadne funkcije. To pomeni, da se bo spremenljivka *count* ob vsakem izrisu ponastavila na 0. Poleg tega React ne bo zaznal spremembe tako deklarirane spremenljivke, zato koda v funkciji *onClick* v bistvu ne naredi ničesar.

Potrebujemo način, da Reactu sporočimo, da želimo določeno spremenljivko ohraniti skozi izrise, saj takšna spremenljivka predstavlja stanje naše aplikacije (v tem primeru števec na gumbu). Hkrati želimo, da sprememba stanja povzroči ponoven izris. V Reactu se takšno stanje imenuje state. V funkcijskih komponentah ga lahko deklariramo s pomočjo vgrajene funkcije *useState*. Funkcija *useState* je ena izmed funkcij, ki služi kot vmesnik z Reactom (angl. hook). Prejme začetno vrednost stanja in vrne dva podatka - prvi je trenutna vrednost stanja, drugi pa funkcija, ki nam omogoča spremembo stanja in posledično proženje procesa ponovnega izrisa. Desna stran slike 3 ponazarja delujoč primer števca.

```
function Counter() {
  let count = 0;

  const onClick = () => {
    count++;
  };

  return (
    <div>
      <button onClick={onClick}>Clicked: {count}</button>
    </div>
  );
}
```

```
function Counter() {
  const [count, setCount] = useState(0);

  const onClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <button onClick={onClick}>Clicked: {count}</button>
    </div>
  );
}
```

Slika 3: Levo primer napačne uporabe stanja, desno primer pravilne uporabe stanja.
Vir: lasten.

2.4.3 Ponovni izris

Ko se aplikacija prvič izriše na zaslonu, postane interaktivna. V poglavju 2.4.2 smo implementirali enostaven števec na klik. Kaj se zgodi, ko se gumb izriše in nanj kliknemo? Postopek je podoben, kot pri prvem izrisu, in se imenuje postopek ponovnega izrisa (angl. rerender) [13].

Označevanje sprememb. Ob kliku se pokliče funkcija *setCount* z novo vrednostjo števca, kar označi komponento *Counter* kot spremenjeno (angl. dirty). React bo najprej preveril, ali se je stanje dejansko spremenilo (na primer, če bi klicali *setCount(0)*, bi stanje ostalo enako in bi komponenta ostala nespremenjena).

Pretvorba spremenjenih komponent v React elemente[4]. React bo v hierarhiji komponent poiskal vse spremenjene komponente. Za vsako izmed teh komponent bo evalviral pripadajočo funkcijsko komponento. V našem primeru se bo evalvirala funkcija za komponento *Counter*, kjer pa bo spremenljivka counter nastavljena na posodobljeno vrednost (ob prvem kliku bo to 1). Rezultat bo React element, kjer bo edina razlika vsebina elementa *button*. Na tak način se evalvirajo vse komponente, ki so označene za posodobitev. Iz rezultatov se zgradi nov virtualni DOM (nespremenjene komponente ohranijo enake elemente, kot v prejšnjem izrisu).

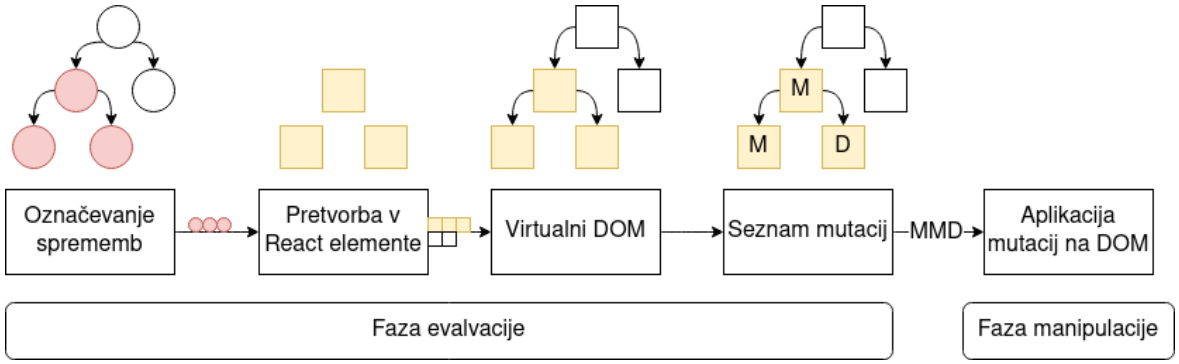
Primerjava Virtualnih DOM dreves in generiranje mutacij. React si interno hrani Virtualni DOM iz prejšnjega izrisa. V tem koraku bo primerjal prejšnji virtualni DOM s trenutnim. Ker je poseganje v dejanski DOM brskalnika potratno, je cilj ugotoviti minimalni nabor mutacij, ki bodo ustrezno pretvorile prejšnje drevo v trenutno drevo. React ta proces imenuje usklajevanje (angl. reconciliation) [15]. Algoritem za iskanje sprememb na drevesih (angl. diffing algorithm) je hevristično s kompleksnostjo O(n). Usklajevalnik (angl. reconciler), uporabljen v Reactu, se

[4] React bo, če je le možno, več sprememb stanja vključil v en cikel ponovnega izrisa (angl. batching). V React 18 je podpora za batching postala še boljša [17].

imenuje Fiber. V primeru komponente *Counter* bo edina sprememba v drevesu to, da se spremeni vsebina elementa *button*.

Manipulacija DOM v brskalniku. Postopek je enak, kot pri prvem izrisu, vendar se tokrat spremeni občutno manj elementov. Postopek usklajevanja je eden izmed glavnih razlogov, da je React tako hiter, saj je število dragih mutacij v DOM minimizirano.

Zgoraj opisani postopki izrisa se pogosto konceptualno ločujejo na dve fazi: fazo evalvacije (angl. render phase) in fazo manipulacije (angl. commit phase). V fazo evalvacije sodijo vsi koraki do vključno z izračunom mutacij, v fazo manipulacije pa apliciranje generiranih manipulacij na DOM brskalnika [16]. Slika 4 prikazuje povzetek postopka za izris.



Slika 4: Postopek izrisa v knjižnici React.
Vir: lasten.

2.4.4 Rekurzivna evalvacija otrok

V poglavju 2.4.3 smo zapisali, da React pri ponovnem izrisu evalvira le komponente, ki so označene kot spremenjene. Kaj se pri tem zgodi z otroki komponente? Oglejmo si na primeru komponente *Counter* z dvema otrokoma (slika 5).

```
function Counter() {
  const [count, setCount] = useState(0);

  const onClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <button onClick={onClick}>Click</button>
      <DisplayText />
      <CounterText count={count} />
    </div>
  );
}

function DisplayText() {
  return <p>Hello, OTS 2022!</p>;
}

function CounterText(props) {
  return <p>Clicked: {props.count}!</p>;
}
```

Slika 5: Komponenta z dvema otrokoma.
Vir: lasten.

Ali se bosta otroka ob spremembi stanja *count* v *Counter* izrisala ponovno? Lahko bi predvidevali, da se *DisplayText* ne bo izrisal, saj ni odvisen od stanja starša, medtem ko se *CounterText* bo, saj je odvisen od stanja *count* v staršu. Ali je to pravilno razmišljanje? Odvisna je od tega, kaj razumemo pod besedo "izris". Pojdimo skozi postopek.

Označevanje sprememb. Ob klicu *setCounter* se bo komponenta *Counter* označila kot spremenjena.

Pretvorba spremenjenih komponent v React elemente. Komponenta *Counter* se bo evalvirala. Proces evalvacije bo evalviral tudi vse otroke *Counter* ne glede na to, ali so dejansko odvisni od spremenjenega stanja. Otroci bodo evalvirani rekurzivno, kar pomeni, da bi se evalvirali tudi morebitni otroci komponent *DisplayText* in *CounterText*. Rezultat tega koraka so torej evalvacije komponent *Counter*, *DisplayText* in *CounterText*. Rekurzivna evalvacija poteka samo po hierarhiji navzdol, zato se komponenta, ki je v hierarhiji nad *Counter*, ne bi evalvirala.

Primerjava Virtualnih DOM dreves in generiranje mutacij. Po ustvarjanju virtualnega DOM bo React generiral zgolj dve spremembi: spremembo vsebine v komponentah *Counter* in *CounterText*.

Opazimo, da je pri *DisplayText* prišlo do zanimivega pojava: komponenta se je evalvirala v fazi evalvacije, vendar v njej ni bilo sprememb, zato ni prispevala k seznamu mutacij za fazo manipulacije. Tak pojav imenujemo nepotrebna evalvacija (angl. unnecessary render), saj evalvacija komponente ni uvedla sprememb v DOM [18].

3 Optimizacija delovanja v knjižnici React

V poglavju 2 smo opisali postopek izrisa v knjižnici React. Pri optimizaciji delovanja React aplikacij je cilj bodisi skrajšati dolžino postopka enega izrisa, ali pa zmanjšati veliko število izrisov skozi čas, v kolikor je to smiselno. Preden si ogledamo primere, predstavimo tehnike za merjenje hitrosti delovanja React aplikacij.

3.1 Tehnike za merjenje hitrosti delovanja in razhroščevanje postopka izrisa

Za merjenje hitrosti je uporabno orodje React Developer tools (slika 6), ki je na voljo v obliki binarnega paketa ali razširitve za brskalnik. Na podlagi izkušenj predlagamo razširitev za brskalnik Chrome.

Orodje React Developer tools uporabimo tako, da za določen čas snemamo podatke za profiliranje [19]. Med snemanjem aplikacijo uporabljamo, s čimer se zajemajo podatki o izrisih. Po snemanju so nam na voljo podatki:

- seznam izrisov in porabljen čas,
- graf, ki za vsak izris prikazuje hierarhijo komponent, skupaj s podatki o tem, kako dolgo je trajal izris posamezne komponente in ali se je komponenta sploh spremenila v DOM (angl. flamegraph),
- pogled na individualno komponento in vzrok, zakaj se je evalvirala.

Slika 6: Uporaba React Developer tools.
Vir: lasten.

Pri uporabi orodja React Devtools ponavadi snemamo del aplikacije, ki je na vizualni pogled najpočasnejša. Ko posnamemo dovolj podatkov, v seznamu izrisov poiščemo vrhove in velike čase (npr. nad 30 milisekund). S pomočjo grafa nato ugotovimo, katera komponenta je vzela največ časa in zakaj se je evalvirala.

Poleg grafa je uporabna funkcionalnost orodja React Developer tools tudi možnost označevanja izrisov (angl. highlight updates when components render), ki vizualno označi komponente, ko se izrisujejo. Na tak način lahko identificiramo okvirne dele aplikacije, ki predstavljajo morebitna ozka grla, nakar jih kvantificiramo s snemanjem.

Poleg orodja React Developer tools so pogosto uporabni tudi izpisi v kodi komponent (angl. logs). Na tak način se bo v brskalnik izpisal niz znakov ob evalvaciji komponente. Po izkušnjah je tak pristop primeren za iskanje točnega vzroka počasnega delovanja, ko s pomočjo grafa ugotovimo, katere komponente so problematične.

3.2 Primeri vzorcev optimizacij

Primeri, ki sledijo, so namenjeni demonstraciji delovanja postopka izrisa in tehnik za optimizacijo.

3.2.1 Primer: enakost stanja

Imamo predvajalnik videov in komponento *VideoTime*, v kateri bi želeli prikazovati trenutni čas videa v sekundah. Predvajalnik nam trenutni čas sporoča v enoti milisekunda vsakih 50 milisekund. Ob zagonu predvajalnika si bodo časi torej sledili tako: 0, 50, 100, 150 itn. Leva stran slike 7 prikazuje neoptimalno implementacijo.

```
function VideoTime() {
  const [time, setTime] = useState(0);

  const onTimeChange = (timeMs) => {
    setTime(timeMs);
  };

  useVideo(onTimeChange);

  return <p>Time: {Math.floor(time / 1000)}</p>;
}
```

```
function VideoTime() {
  const [time, setTime] = useState(0);

  const onTimeChange = (timeMs) => {
    setTime(Math.floor(timeMs / 1000));
  };

  useVideo(onTimeChange);

  return <p>Time: {time}</p>;
}
```

Slika 7: Levo primer neoptimalne, desno primer boljše rešitve.
Vir: lasten.

S pomočjo označevanja evalvacij v orodju React Developer tools ali s pomočjo izpisov lahko opazimo, da se komponenta *VideoTime* evalvira vsakih 50 milisekund.

Razmislimo, kaj se dogaja v postopku izrisa za časa 0 in 50 milisekund. Ob klicu *setTime(0)* se komponenta označi kot spremenjena, zato jo React v naslednjem ciklu evalvira. Postopek evalvacije izračuna tudi vrednost *Math.floor(0 / 1000)*, ki milisekunde pretvori v sekunde in rezultat zaokroži navzdol. Rezultat je vrednost 0. Ker gre za prvi izris, se bo komponenta v celoti izrisala na DOM v fazi manipulacije (poglavje 2.4.1).

Ob klicu *setTime(50)* se ponovi podoben postopek. V tem primeru se evalvira izraz *Math.floor(50 / 1000)*, katerega rezultat je ponovno 0. Pri postopku primerjave virtualnega DOM-a bo React ugotovil, da ni potrebno ničesar spremeniti, saj je element *p* že v prejšnjem izrisu imel vsebino 0. Podoben postopek se bo ponavljal vse do klica *setTime(1000)* čez eno sekundo, kjer se bo vsebina spremenila na 1.

Opazimo, da imamo vsako sekundo le eno manipulacijo na DOM in 19 nepotrebnih evalvacij, ki sicer spremenijo stanje *time*, ampak ne povzročijo izrisa na DOM. Rešitev je enostavna - klic *Math.floor* prestavimo iz funkcije v klic *setTime* (desna stran slike 7).

Sedaj se bo pri časih 0, 50 in 100 *setTime* klical s parametri 0, 0, 0 in ne 0, 0.05 in 0.1. React bo pri klicu *setTime* zaznal, da se stanje ni spremenilo (saj je 0 enako 0) in ne bo evalviral komponente. Komponenta se bo sedaj evalvirala le enkrat na sekundo[5].

Primer lahko še dodatno optimiziramo tako, da pred klicem *setTime* primerjamo zaokroženo vrednost s trenutno vrednostjo *time*. Priporočamo takšno rešitev, kljub temu pa je na prejšnji rešitvi dobro ponazorjen efekt spremembe stanja na enako vrednost.

3.2.2 Primer: otrok v lastnostih

Kot opisano v poglavju 2.4.4, se ob evalvaciji komponente rekurzivno evalvirajo tudi vsi njeni otroci. Želeli bi se izogniti nepotrebnim evalvacijam otrok, ko na otroka sprememba stanja v staršu ne vpliva.

Manj znan pristop, s katerim lahko to dosežemo, je pošiljanje otroka v starša preko lastnosti (props). V tem primeru potrebujemo še eno starševsko komponento *CounterParent* (slika 8).

```
function CounterParent() {
  return <Counter textComponent={<DisplayText />} />;
}

function DisplayText() {
  return <p>Hello, OTS 2022!</p>;
}

function Counter(props) {
  const [count, setCount] = useState(0);

  const onClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <button onClick={onClick}>Click {count}</button>
      {props.textComponent}
    </div>
  );
}
```

Slika 8: Uporaba otroka v lastnostih za preprečevanje nepotrebnih evalvacij.
Vir: lasten.

Ko se v *Counter* spremeni *count*, se starš *CounterParent* ne bo evalviral. Hkrati vemo, da komponenta ne more spreminjati svojih lastnosti. Iz tega sledi zaključek, da se lastnost *textComponent* pri spremembi stanja v komponenti *Counter* zagotovo ni spremenila. React lahko torej varno izpusti evalvacijo komponente *CounterText*.

3.2.3 Primer: preprečevanje nepotrebnih evalvacij

Rešitve v prejšnjem primeru ni vedno možno uporabiti, poleg tega pa lahko pristop hierarhijo komponent naredi manj razumljivo. V tem primeru bomo razložili alternativo s pomočjo komponente *Counter*, ki vsakih 50 milisekund inkrementira števec (slika 9). Poleg tega ima komponenta 4 otroke:

- *Text* zgolj izriše statičen tekst. Ta komponenta ima še enega otroka *TextChild*, ki prav tako samo izriše statični tekst.

[5] React pravzaprav bo evalviral komponento prvič, ko bo sprememba stanja enaka. To je varnostni mehanizem [20].

- *ResetButton* ima preko funkcije *onClick*, poslane kot lastnost, povezavo v starša, s katerim se stanje števca ponastavi ter inkrementira število klikov.
- *ComputedText* prejme in izriše objekt s številom klikov na gumb ter fakulteto tega števila.
- *CounterText* izriše vrednost števca.

S pomočjo orodja React Developer tools takoj opazimo, da se izrisujejo prav vsi otroci zaradi rekurzivne evalvacije. Preden nadaljujemo, analizirajmo optimalne čase izrisa za vse komponente:

- *Text*, *TextChild* in *ResetButton* niso odvisni od stanja v komponenti *Counter*, zato bi se v optimalnem primeru izrisali samo enkrat, na začetku.
- *ComputedText* je sicer odvisen od stanja v *Counter*, ampak le od števila klikov *clickCount*, zato bi se v optimalnem primeru izrisal le ob spremembi *clickCount* (tj. ob kliku na gumb *ResetButton*).
- *CounterText* je odvisen od stanja števca *count*, zato se mora izrisati vsakič, ko se spremeni stanje *count*.

```
const Counter = () => {
  const [count, setCount] = useState(0);
  const [clickCount, setClickCount] = useState(0);

  const onIncrement = () => {
    setCount((count) => count + 1);
  };

  const onReset = () => {
    setCount(0);
    setClickCount((clickCount) => clickCount + 1);
  };

  useIncrement(onIncrement);

  const data = {
    clickCount,
    factorial: factorial(clickCount),
  };

  return (
    <div>
      <Text />
      <ResetButton onClick={onReset} />
      <ComputedText data={data} />
      <CounterText count={count} />
    </div>
  );
};
```

```
const Text = () => {
  return (
    <div>
      <span>Hello,</span>
      <TextChild />
    </div>
  );
};

const TextChild = () => {
  return <span>OTS 2022!</span>;
};

const ResetButton = (props) => {
  return <button onClick={props.onClick}>Reset</button>;
};

const ComputedText = (props) => {
  return <p>{JSON.stringify(props.data)}</p>;
};

const CounterText = (props) => {
  return <p>Counter: {props.count}</p>;
};
```

Slika 9: Komponenta *Counter* za primer preprečevanja nepotrebnih evaluacij.
Vir: lasten.

Optimizacija komponent *Text* in *TextChild*. Komponenta *Text* se izrisuje kljub temu, da ni odvisna od stanja *Counter*. Ko se evalvira *Counter*, se evalvira *Text*, in rekurzivno tudi *TextChild*. Gre za nepotrebno evalvacijo, saj se vsebina komponent ne spreminja. React ponuja mehanizem *React.memo*, s katerim lahko preprečimo evalvacijo komponente, v kolikor so izpolnjeni pogoji [21]:

- evalvacija je sprožena iz starša,
- stanje komponente se ni spremenilo,
- lastnosti (props) komponente se niso spremenile.

Komponento *Text* objamemo (angl. wrapping) s funkcijo *React.memo* (slika 11), kar vrne novo komponento. Ko se evalvira komponenta *Counter*, se komponenta *Text* ne bo, saj so izpolnjeni vsi pogoji za preprečevanje evalvacije. Poudarimo, da komponente *TextChild* ni potrebno objeti z *React.memo*, saj bo veriga rekurzivne evalvacije ustavljena že na komponenti *Text*.

Optimizacija komponente *ResetButton*. Podobno kot pri komponenti *Text*, se tudi lastnosti komponente *ResetButton* ne spreminjajo, ko se evalvira *Counter*, torej lahko ponovno uporabimo *React.memo*. Ugotovili bomo, da se bo kljub uporabi *React.memo* komponenta *ResetButton* še vedno evalvirala ob vsaki evalvaciji *Counter*. To pomeni, da eden izmed pogojev za preprečevanje evalvacije ni dosežen. V tem primeru je lahko to zgolj pogoj, da se lastnosti komponente niso spremenile.

Komponenta *ResetButton* ima le eno lastnost - funkcijo *onClick*. Na prvi pogled bi lahko rekli, da se ta funkcija ob evalvacijah *Counter* ne spreminja, vendar temu ni tako. V komponenti *Counter* deklariramo funkcijo *onReset*, ki

predstavlja lastnost *onClick* za *ResetButton*. Ko se *Counter* evalvira, se bo izvedel tudi ta del kode, kar pomeni, da se bo ob vsaki evalvaciji *Counter* ustvari nova funkcija, ki se priredi spremenljivki *onReset*.

React.memo[6] pri detektiranju sprememb uporablja JavaScript funkcijo *Object.is* [22]. Ta funkcija preveri referenčno enakost (angl. referential equality) parametrov. V primeru enakih vrednosti primitivnih tipov v jeziku JavaScript [23] bo funkcija vrnila pričakovani rezultat (leva stran slike 10), pri objektih pa le, če gre za referenci na enak objekt (desna stran slike 10). Enako pravilo velja za polja in funkcije, ki so v jeziku JavaScript pravzaprav le posebne vrste objektov.

```
Object.is(1, 1); // true
Object.is(1, 2); // false
Object.is("OTS", "OTS"); // true
Object.is("", ""); // true
Object.is(null, undefined); // false
```

```
Object.is({}, {}); // false - različna referenca!

const arr = [];
const arr2 = arr;
Object.is(arr, arr2); // true - enaka referenca!
```

Slika 10: Demonstracija delovanja *Object.is* funkcije.
Vir: lasten.

V našem primeru se ob vsaki evalvaciji *Counter* ustvari nova instanca funkcije *onReset*, ki ni enaka instanci iz prejšnje evalvacije, zaradi česar *Object.is* v *React.memo* zazna spremembo lastnosti, kar evalvira funkcijo *ResetButton*.

Potrebujemo mehanizem, s katerim lahko skozi evalvacije komponente *Counter* ohranimo enako referenco na funkcijo *onReset*, tako da bo *Object.is* ustrezno prepoznal referenčno enakost. Rešitev je funkcija *useCallback* (slika 11), ki izvede tako imenovano memoizacijo (angl. memoization) nad funkcijo *onReset* in skozi evalvacije ohranja njeno referenco.

Optimizacija komponente *ComputedText*. Pri optimizaciji komponente *ComputedText* pridemo do podobnega zaključka, kot pri komponenti *ResetButton*. Tudi v tem primeru *React.memo* sam ne bo preprečil nepotrebnih evalvacij, saj se objekt *data* ponovno ustvari ob vsaki evalvaciji komponente *Counter*. Razlika je zgolj v tem, da v tem primeru želimo ohraniti referenco na objekt, namesto na funkcijo. Ekvivalent *useCallback* za objekte je funkcija *useMemo*. Funkcija *useMemo* bo ohranjala vrednost poljubne spremenljivke (v našem primeru je to objekt *data*) skozi evalvacije, dokler se ne spremeni ena izmed spremenljivk, naštetih v polju, ki je podan kot drug parameter v klic *useMemo*. Ob spremembi se bo ponovno evalvirala funkcija, podana kot prvi parameter. V našem primeru se bo to zgodilo ob kliku na gumb *ResetButton*. V tem primeru bo prišlo do evalvacije *Counter* in *ComputedText* (zaradi spremenjene reference na objekt *data*), kar pa je v tem primeru pravilno obnašanje, saj je *ComputedText* odvisen od števila klikov v *Counter*.

Optimizacija komponente *CounterText*. Komponenta *CounterText* je odvisna od spremenljivk stanja *count* in *clickCount* v komponenti *Counter*. Pri tem se spremenljivka *count* spreminja najpogosteje. To pomeni, da se komponente ne splača optimizirati, saj se bodo v vsakem primeru njene lastnosti spremenile ob vsaki evalvaciji.

[6] Enak algoritem se uporablja tudi pri detektiranju spremenjenih stanj v komponentah v prvem koraku postopka izrisa.

```
const Counter = () => {
  const [count, setCount] = useState(0);
  const [clickCount, setClickCount] = useState(0);

  const onIncrement = () => {
    setCount((count) => count + 1);
  };

  const onReset = useCallback(() => {
    setCount(0);
    setClickCount((clickCount) => clickCount + 1);
  }, []);

  useIncrement(onIncrement);

  const data = useMemo(
    () => ({
      clickCount,
      factorial: factorial(clickCount),
    }),
    [clickCount]
  );

  return (
    <div>
      <Text />
      <ResetButton onClick={onReset} />
      <ComputedText data={data} />
      <CounterText count={count} />
    </div>
  );
};
```

```
const Text = React.memo(() => {
  return (
    <div>
      <span>Hello,</span>
      <TextChild />
    </div>
  );
});

const TextChild = () => {
  return <span>OTS 2022!</span>;
};

const ResetButton = React.memo((props) => {
  return <button onClick={props.onClick}>Reset</button>;
});

const ComputedText = React.memo((props) => {
  return <p>{JSON.stringify(props.data)}</p>;
});

const CounterText = (props) => {
  return <p>Counter: {props.count}</p>;
};
```

Slika 11: Optimizirana komponenta *Counter*.
Vir: lasten.

3.2.4 Primer: optimiziranje faze manipulacije

Vse dozdajšnje optimizacije smo izvajali v sklopu faze evalvacije. V tem poglavju bomo predstavili dva primera optimizacije v fazi manipulacije. V poglavju 2.4.3 smo omenili, da je algoritem usklajevanja hevristično, in sicer za optimalnost izvajanja naredi dve predpostavki [9]. Razumevanje obeh predpostavk je ključnega pomena za optimizacijo faze manipulacije.

Predpostavka 1: elementa z različnima tipoma generirata drugačna drevesa.

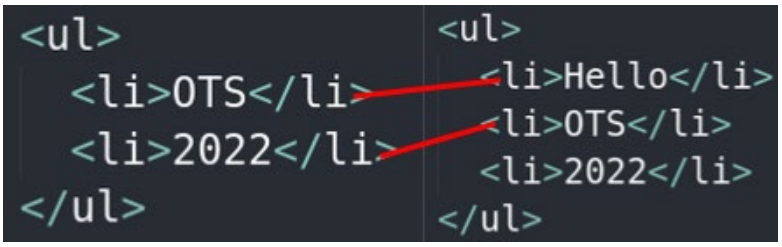
Imejmo komponento, ki ob prvem izrisu ustvari React element z elementom *div* in tremi otroci *span*, ob naslednjem izrisu pa *div* zamenja z *section* (otroci ostanejo enaki). React bo v procesu usklajevanja ugotovil, da se je tip starša spremenil iz *div* v *section*, zato bo vse otroke rekurzivno izbrisal in jih ponovno ustvaril v DOM kljub temu, da se niso spremenili. To je posledica predpostavke 1.

Rekurzivni izbris vseh otrok v hierarhiji je zelo počasna operacija, predvsem, če ima starš veliko število otrok. Da se izognemo počasnemu delovanju, se v praksi skušamo izogniti spremembam tipov starševskih komponent.

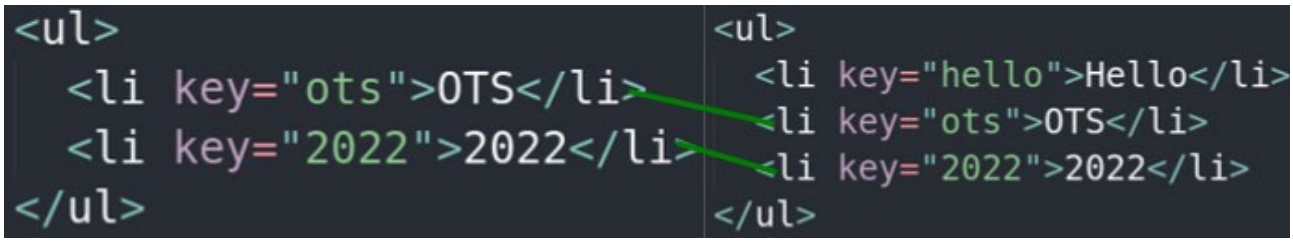
Predpostavka 2: razvijalec lahko z lastnostjo *key* označi enake komponente v ponovnem izrisu.

Imejmo komponento, ki vrne element *ul* in tri otroke *li*. V naslednjem izrisu dodamo še enega otroka *li* na konec seznama. V procesu usklajevanja bo React primerjal elemente po vrsti od zgoraj navzdol in pravilno generiral eno manipulacijo: dodajanje novega otroka *li* na konec seznama.

Oglejmo si, kaj pa se zgodi, če element *li* dodamo na začetek seznama. V tem primeru bo primerjanje elementov od zgoraj navzdol pri vsakem elementu *li* zaznalo spremembo, zato bo React zgeneriral spremembo prvih dveh elementov *li* in dodajanje zadnjega elementa *li* (slika 12).



Slika 12: Brez uporabe lastnosti *key* - React ne najde ujemanja med elementi.
Vir: lasten.



Slika 13: Z uporabo lastnosti *key* - React najde ujemanje.
Vir: lasten.

Težavo lahko rešimo s pomočjo lastnosti *key*, ki predstavlja identiteto komponente v seznamu (slika 13). V kolikor elementom podamo unikatno lastnost *key*, bo react pare iskal po lastnosti *key*, namesto od zgoraj navzdol. V prejšnjem primeru bo tako pravilno našel zgolj eno spremembo: dodajanje elementa *li* na prvo mesto seznama.

Priporočljivo je, da za lastnost *key* uporabimo unikaten identifikator. Indeksi polja niso dobri kandidati za vrednost *key*, saj pri vstavljanju na sredino ali pri menjavi elementov v seznamu elementi menjajo lastnost *key* [24].

3.3 Primer iz produkcije

Predstavljajmo si aplikacijo, ki predvaja video posnetek. Uporabniki lahko na poljubni točki v posnetku dodajo komentar. Komentarji se prikažejo v seznamu poleg posnetka v času, ko so bili dodani in do 5 sekund po tem času. Komentarje lahko brišemo. Prikazujemo tudi lastno časovnico videa, ki se posodablja ob predvajanju videa.

V aplikaciji se moramo povezati na čas videa. Video nam čas sporoča v obliki milisekund približno vsakih 10 milisekund. V polju hranimo podatke o dodanih komentarjih. Ob vsaki spremembi časa je potrebno izračunati komentarje, ki se prikazujejo na zaslonu. Prav tako je treba prilagoditi časovnico.

Pri planiranju smo ugotovili, da se bodo vsi komentarji velikokrat nepotrebno izrisali, saj se čas videa spreminja okoli 50-krat na sekundo. V ta namen smo uporabili *React.memo*. Poleg tega smo morali uporabiti še *useCallback*, da smo zagotovili nespremenljivost funkcijskih referenc.

Kljub temu, da so se ponekod v aplikaciji še pojavljale nepotrebne evalvacije, z optimizacijo nismo nadaljevali, saj nadaljnje optimizacije ne bi prinesle večje vrednosti za produkt.

3.4 Kdaj izvesti optimizacije?

Kdaj se lotiti izvajanja optimizacij? Mnenj na to temo je veliko, nekateri razvijalci so zagovorniki optimizacij šele, ko je opazno počasno delovanje, spet drugi pa optimizirajo že v začetku razvoja. Na koncu je najpomembneje, da se odločimo po lastni presoji glede na situacijo.

Naše mnenje je, da je potrebno najti balans med izvedbo optimizacij v razvojnem procesu in vrednostjo, ki jo to prinaša za produkt.

V kolikor pišemo prototip ali dokaz koncepta, potem je vrednost optimizacij praktično ničelna, zato optimizacij ne izvajamo. V kolikor pa razvijamo produkt v produkciji, pa optimizacije vkomponiramo v razvojni proces, vendar na način, ki ni časovno potraten in je osredotočen le na pomembne stvari. V fazi planiranja razvoja funkcionalnosti identificiramo komponente, ki bi se lahko pogosto izrisovale. Med razvojem pogosto uporabljamo funkcionalnost vizualizacije izrisov, s katerimi dobimo hiter vizualni feedback o morebitnih ozkih grlih. Na tak način lahko že v zgodnji fazi preprečimo počasno delovanje.

Dobro se je zavedati, da nepotrebnih evalvacij ni potrebno vedno naslavljati. V večini primerov gre za komponente, katerih evalvacije so dovolj enostavne ali pa dovolj redke, da ne prispevajo veliko k času izrisa. V teh primerih lahko optimizacije celo poslabšajo hitrosti komponent, saj ima vsaka optimizacija v ozadju določeno režijo, ki prav tako zavzame nekaj časa v izrisu.

Optimizacije se najbolj splačajo na komponentah, ki se pogosto evalvirajo z enakimi lastnostmi. V teh primerih bo večina evalvacij komponente nepotrebnih, tako da bo *React.memo* v kombinaciji z *useMemo* in *useCallback* drastično znižal število evalvacij komponente.
Velikokrat se optimizacije lahko izvede brez uporabe eksplicitnih orodij za optimizacijo, npr. s tehniko otroka v lastnosti ali z premestitvijo hierarhije komponent.

Priporočamo, da se po vsaki optimizaciji s pomočjo React Developer tools preveri, ali je optimizacija dejansko pospešila delovanje.

4 Zaključek

V prispevku smo predstavili React kot orodje, ki poenostavi grajenje modernih interaktivnih aplikacij s pomočjo deklarativne sintakse in strukturiranih komponent. S pomočjo analize internih algoritmov v knjižnici React smo ugotovili, zakaj je React že v izvoru zelo hiter. Z znanjem o internem algoritmu in pravilih za izris smo nato pripravili primere vzorcev optimizacij ter predstavili en primer iz produkcijskega okolja. Na koncu smo podali nekaj praktičnih izkušenj in napotkov za optimizacije v knjižnici React.

Literatura

[1] https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages, 11 Most In-Demand Programming Languages in 2022, obiskano 27. 7. 2022.

[2] https://survey.stackoverflow.co/2022/#most-popular-technologies-language, Stack Overflow Developer Survey 2022, 2022, obiskano 27. 7. 2022.

[3] Kristen Baker, https://blog.hubspot.com/marketing/page-load-time-conversion-rates, 11 Website Page Load Time Statistics You Need, 2022, obiskano 27. 7. 2022.

[4] https://developer.mozilla.org/en-US/docs/Glossary/DOM, DOM (Document Object Model), 2022, obiskano 27. 7. 2022.

[5] https://www.w3schools.com/js/js_htmldom.asp, JavaScript HTML DOM, 2022, obiskano 27. 7. 2022.

[6] Anshul Parmar, https://www.linkedin.com/pulse/reflow-repaint-manipulating-dom-responsibly-anshul-parmar, Reflow and Repaint: Manipulating DOM responsibly, 2019, obiskano 27. 7. 2022.

[7] Gopalakrishnan, https://dev.to/gopal1996/understanding-reflow-and-repaint-in-the-browser-1jbg, Understanding Reflow and Repaint in the browser, 2020, obiskano 27. 7. 2022.

[8] https://www.geeksforgeeks.org/reactjs-reactdom/, ReactJS | ReactDOM, 2021, obiskano 27. 7. 2022.

[9] https://reactjs.org/docs/reconciliation.html, Reconciliation, obiskano 27. 7. 2022.

[10] https://reactjs.org/docs/components-and-props.html, Components and Props, obiskano 27. 7. 2022.

[11] https://reactjs.org/docs/introducing-jsx.html, Introducing JSX, obiskano 27. 7. 2022.

[12] https://reactjs.org/docs/react-dom-client.html#createroot, ReactDOMClient, obiskano 27. 7. 2022.

[13] https://blog.isquaredsoftware.com/2020/05/blogged-answers-a-mostly-complete-guide-to-react-rendering-behavior/, A (Mostly) Complete Guide to React Rendering Behavior, 2020, obiskano 27. 7. 2022.

[14] https://reactjs.org/blog/2022/03/08/react-18-upgrade-guide.html#updates-to-client-rendering-apis, How to Upgrade to React 18, obiskano 27. 7. 2022.

[15] Kishan Sheth, https://hackernoon.com/virtual-dom-reconciliation-and-diffing-algorithm-explained-simply-ycn34gr, Virtual DOM, Reconciliation And Diffing Algorithm Explained Simply, 2021, obiskano 27. 7. 2022.

[16] Bryan Almaraz, https://dev.to/thee_divide/reconciliation-react-rendering-phases-56g2, 2020, obiskano 27. 7. 2022.

[17] Lakindu Hewawasam, https://blog.bitsrc.io/automatic-batching-in-react-18-what-you-should-know-d50141dc096e, 2022, obiskano 27. 7. 2022.

[18] https://www.debugbear.com/blog/react-rerenders, Optimizing React performance by preventing unnecessary re-renders, 2021, obiskano 27. 7. 2022.

[19] https://www.geeksforgeeks.org/react-developer-tools/, React Developer Tools, 2021, obiskano 27. 7. 2022.

[20] https://reactjs.org/docs/hooks-reference.html#bailing-out-of-a-state-update, Bailing out of a state update, obiskano 27. 7. 2022.

[21] https://reactjs.org/docs/react-api.html#reactmemo, React.memo, obiskano 27. 7. 2022.

[22] Valentino Gagliardi, https://www.valentinog.com/blog/react-object-is/, Demystifying Object.is and prevState in React useState, 2021, obiskano 27. 7. 2022.

[23] https://developer.mozilla.org/en-US/docs/Glossary/Primitive, Primitive, obiskano 27. 7. 2022.

[24] Robin Pokorny, https://robinpokorny.medium.com/index-as-a-key-is-an-anti-pattern-e0349aece318, Index as a key is an anti-pattern, 2015, obiskano 27. 7. 2022.

Kaj je Blazor in kako se primerja z JavaScript ogrodji?

Matjaž Prtenjak
Matjaž Prtenjak s.p., Laško, Slovenija
matjaz@mnet.si

Sinopsis Nekako smo se že navadili, da se spletne aplikacije razvijajo v programskem jeziku JavaScript. Tukaj govorim o programski kodi, ki teče v brskalniku, kajti sodobna spletna aplikacija potrebuje tudi zaledje, podatkovne baze in podobno. Vsa ta zaledna koda je pisana v različnih programskih jezikih. Koda, ki teče v brskalniku in torej uporabnik z njo neposredno kontaktira, pa je v večini primerov klasična JavaScript koda.
Kaj pa, če bi želeli izdelati spletno aplikacijo v kakšnem drugem programskem jeziku? Kaj pa, če sicer uporabljate Microsoftova razvojna okolja in bi želeli spletno aplikacijo, ki teče v poljubnem brskalniku, napisati v programskem jeziku C#?

Leta 2018 je bil sprejet standard poimenovan WebAssembly [1], ki ga podpirajo vsi novejši brskalniki. Kot že ime implicira gre za neke vrste strojni jezik (assembly) brskalnika. In če naredimo prevajalnik za ta strojni jezik, se potem lahko ta koda izvaja v brskalniku.
To je Blazor [2] in to je tematika tega prispevka.

Ključne besede:
Blazor
WASM
.NET
spletne aplikacije
JavaScript
VUE
C#

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.7

1 Uvod

Leta 2018 je bila sprejeta prva verzija standarda WebAssembly [1], ki so ga podprli vsi izdelovalci brskalnikov.

WebAssembly (WASM) je binarni format navodil za virtualni stroj. WASM je zasnovan tako, da omogoča izvajanje programske kode (v peskovniku) tako na odjemalcih, kot tudi na strežnikih.

Microsoft je vzpon JavaScript-a zamudil. Pred leti je poskušal razviti okolje, ki bi bilo bližje povprečnemu MS razvijalcu/razvijalki. Naredili so Silverlight [3], ki pa je v brskalniku potreboval vtičnik in ni nikoli zares zaživel.

Hkrati pa MS ima močno orožje, ki nenehno tekmuje z Java svetom in to je seveda .NET, ki ga je v zadnjih letih povsem prenovil in tudi odprl svetu, saj je ustanovil .NET Foundation [4] in je zatorej odprtokodno orodje.

Vendar to ni tematika tega prispevka. To je samo osnova, ki nam pomaga razumeti, kaj se je zgodilo, ko sta se združili ideji odprtega .NET razvojnega okolja in odprtega WASM. MS je dobil možnost, da veliko količino programske kode prevede v WASM ali povedano drugače, dobil je možnost razvoja .NET programov, ki tečejo v brskalniku.

Blazor je odprtokodno razvojno okolje podjetja Microsoft, kjer lahko program razvijamo v C# programskem jeziku, prevajalnik pa ga prevede v WASM. Takšen program se torej lahko izvaja v poljubnem brskalniku (vsi novejši).

2 Blazor

2.1 Kratka zgodovina

Definicija imena Blazor, ki je zapisana v prejšnjem odstavku, nam ne pove kaj dosti, zato si na kratko poglejmo razvojno pot Blazor-ja in posledično razvojno pot podjetja Microsoft v spletne aplikacije.

Podobno kot siceršnji razvoj, lahko tudi spletni razvoj razdelimo na dva dela. Na strežniški del (angl. back-end) in na uporabniški del (angl. front-end).

2.1.1 Strežniški del

Seveda se je najprej razvijal strežniški dal, saj moramo najprej nekaj imeti, da to lahko pokažemo uporabniku. Uporabnik oz. brskalnik je torej na začetku pridobival statične HTML strani, ki pa so kmalu postale dinamične v smislu, da jih je strežnik pred-obdelal, preden jih je poslal uporabniku.

Tipičen primer je recimo spletna stran za prikaz detajla artikla. Če imamo 100 artiklov, ni potrebno napisati 100 HTML strani, temveč napišemo eno HTML stran, ki ima dinamične komponente in preden brskalnik dobi takšno stran, strežnik vanjo vrne podatke konkretnega artikla. Programska jezika PHP in RUBY sta bila med pionirji takšnega načina izdelave spletnih strani.

Ta vlak je MS malce zamudil, toda navkljub vsemu je dokaj hitro izdal programski jezik ASP (Active Server Pages). Njegova težava nikoli ni bila neuporabnost ali zapletenost ali kaj podobnega, temveč precej bolj banalna - denar. Vse ostale rešitve so lahko tekle na zastonjskih (ali pa zelo poceni) Linux strežnikih, za ASP (in kasneje ASP.NET) rešitve, pa je bilo potrebno v ozadju imeti MS infrastrukturo, ki pa je seveda plačljiva.

Kakorkoli, MS je imel rešen strežniški del, manjkalo pa jim je orodje na uporabniški strani.

2.1.2 Uporabniški del

Prvi resni vstop v svet uporabnikov je MS naredil leta 2007, ko je izdal orodje Silverlight. Tedaj je na brskalnikih kraljeval Adobe Flash [5] in Silverlight je bil Microsoftov poskus vstopa v ta segment. Oba sta bila vtičnika in oba je bilo potrebno najprej instalirati v brskalnik. V nekem trenutku so vtičniki postali nekaj slabega (upravičeno ali ne) in uporabniki jih niso več želeli, zato so se poslovili.

Javascript se je pojavil seveda še pred Flash-om in je bil vseskozi prisoten, največja težava pa je bila v neusklajenosti brskalnikov pri podpori JavaScript. Tako so/smo morali razvijalci razvijati različne funkcije za različne brskalnike. Toda, ker to ni prispevek o JavaScript, bom ta del preskočil in preprosto skočil v čas po Flash-u in po JQuery knjižnici [6].

JQuery je sicer knjižnica, ki je prva poenotila način delovanja JavaScript v različnih brskalnikih. Razvijalci smo imeli možnost razvoja JavaScript programske kode, ki je v 95% primerov enako tekla na vseh brskalnikih.

Spet je MS ostal zadaj in spet je zamudil vlak. Na njegovo srečo pa so se v WWW konzorciju uspeli dogovoriti in vzpostaviti standard WebAssembly, ki predstavlja neke vrste strojni jezik brskalnika. Nek program lahko torej prevedemo v WebAssembly (popularno imenovan WASM) in tekel bo v vseh novejših brskalnikih (torej v brskalnikih izdanih v zadnjih nekaj letih).

Po drugi strani pa je Microsoft začel odpirati njegovo paradno okolje .NET in naredil .NET Core. Ker to spet ni prispevek o MS, bom preskočil tudi ta del in preprosto prišel na ključni del:

1. WWW konzorcij je standardiziral WebAssembly.
2. MS je odprl .NET ter naredil .NET Core, ki ni več vezan na MS, temveč odprtokoden.
3. Točki 1 in 2 sta omogočili, da je MS prevedel .NET Core knjižnice v WebAssembly in sedaj lahko vsaka koda, ki sicer teče v .NET Core teče tudi na brskalniku, saj se brskalnik obnaša kot operacijski sistem z .NET knjižnicami.

Torej sedaj imamo produkt, ki lahko .NET program prevede v WebAssembly in stvar teče v brskalniku. In to je Blazor.

Pravzaprav je to del Blazor-ja. Blazor se namreč ravno tako deli na strežniški in uporabniški del. Medsebojno sta neodvisna in seveda lahko sodelujeta. Pomembno pa je, da lahko uporabniški del Blazor aplikacije teče povsem neodvisno od strežniškega dela.

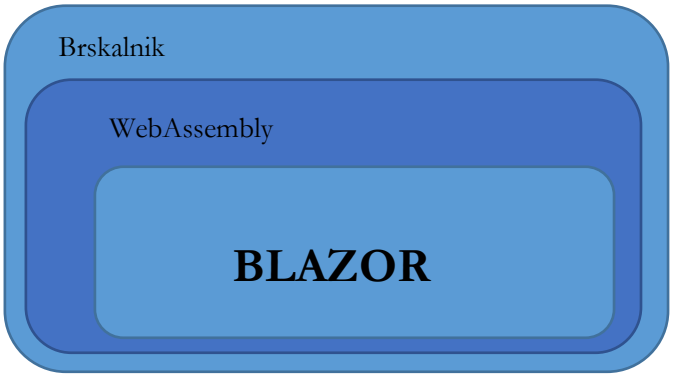
To slednje pa pomeni, da za delovanje ne potrebujemo MS infrastrukture!

In to je bistvo tega prispevka. V njem bom predstavil uporabniški del Blazor aplikacije, ki teče v WebAssembly-u in ne potrebuje MS infrastrukture, temveč v celoti teče v brskalniku, aplikacijo pa lahko streže poljuben strežnik - recimo Apache.

2.2 Dva tipa Blazor aplikacij

2.2.1 Blazor Web Assembly

Ta v celoti teče na brskalniku in to je tudi model, ki ga bom opisoval.



Slika 1: Blazor WebAssembly.
Vir: lasten.

Prednosti:

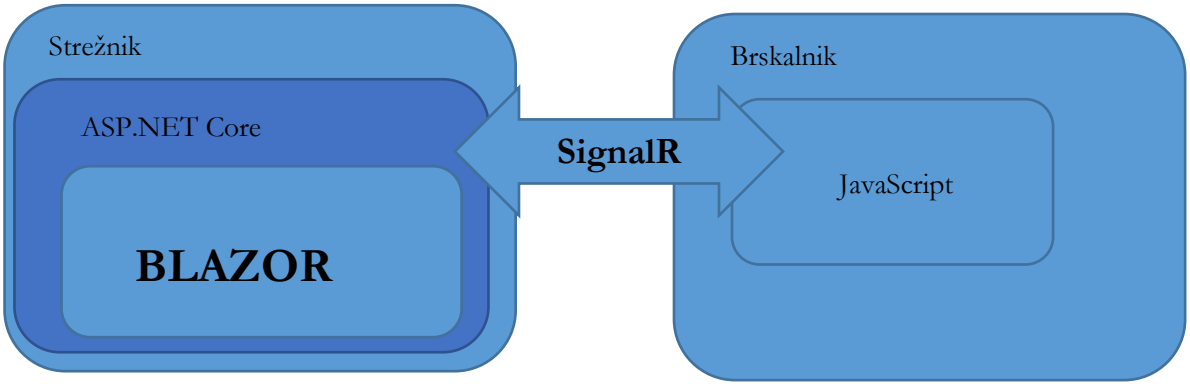
- Teče v brskalniku, neodvisno od strežnika (razen seveda, da potrebujete spletni strežnik za osnovni zagon).
- Lahko pa razvijete tudi t.i. PWA (Progressive Web App), ki pa je nisem preizkusil.

Slabosti:

- Osnovna slabost tega načina je daljši zagonski čas, saj mora strežnik najprej na uporabnika poslati datoteko blazor.webassembly.js, ki potem posledično s strežnika pretoči vse potrebne .DLL datoteke, kar lahko traja dlje časa, v kolikor je povezava slabša.

2.2.2 Blazor Server Side

Tega nisem preizkušal in tukaj opisujem samo v toliko, da podam celotno sliko o Blazor tehnologiji. Razlog ne-ukvarjanja s tovrstno rešitvijo pa je zopet v dejstvu, da zanjo potrebuješ MS infrastrukturo, medtem ko mene zanimajo odprtokodne rešitve, ki lahko tečejo na različnih spletnih strežnikih.



Slika 2: Blazor Server Side.
Vir: lasten.

Prednosti:

- Ni dolgega zagonskega časa.
- Na uporabnika se prenaša samo HTML in JavaScript.
- Ker ni potrebe po WASM, lahko rešitev deluje tudi v starejših brskalnikih.

Slabosti:

- Aplikacija za komunikacijo med uporabniki in strežnikom uporablja SignalR, kar pomeni, da se mora strežnik zavedati VSEH trenutnih uporabnikov in to lahko povzroči veliko porabo sistemskih sredstev na strežniku.
- Zelo veliko komunikacije med uporabnikom in strežnikom, saj mora uporabnik vsak dogodek (preko SignalR) poslati na strežnik, ki ga (v .NET okolju) obdela in rezultat (delni HTML/CSS) pošlje nazaj uporabniku.
- Potrebujemo MS strežniško infrastrukturo.

2.3 Kako razviti Blazor WASM aplikacijo?

Najprej moramo seveda instalirati .NET Core razvojna orodja, ki so brezplačno dosegljiva na MS spletnih straneh.

Nadalje moramo izbrati nek urejevalnik, ki je seveda lahko poljuben, čeravno se s .NET (seveda!) najbolje razumeta MS urejevalnika Visual Studio Code (ki je brezplačen) in Visual Studio (ki obstaja v več različicah, tudi brezplačni).

Ko torej imamo razvojna orodja in urejevalnik:

- Odpremo nov BLAZOR WASM projekt.
- Izdelamo program.
- Ga prevedemo.
- Prepišemo na spletni strežnik.
- Ko spletna aplikacija teče, spletni strežnik v celotnem procesu deluje zelo malo, saj gre za statično podajanje spletnih strani in se aplikacija prenese na brskalnik, kjer se tudi izvaja.

Ker ta prispevek nikakor ni namenjen učenju programiranja ali čemu podobnemu, postopka ne bom nadalje in podrobneje opisoval, saj so postopki lepo opisani na mnogih spletnih straneh. Najlažje pa je začeti recimo na spletni strani: https://blazor-university.com/.

2.4 Primerjava med JavaScript knjižnico VUE in Blazor WASM

VUE je JavaScript knjižnica in lahko jo vključite v vsako HTML stran ter na njej uporabite poljubne VUE konstrukte. Torej brez kakršnekoli instalacije, brez prevajanja, brez "aplikacij" in podobnega. Preprosta JavaScript koda, ki jo vključite v HTML in uporabite.

Blazor WASM tega ne omogoča. V tem primeru imate vedno aplikacijo, ki jo je potrebno razviti, prevesti, prepisati na strežnik in uporabiti.

Primerjamo lahko torej samo VUE CLI aplikacije in Blazor WASM aplikacije. VUE CLI aplikacija je standardna VUE aplikacija, ki je v splošnem sestavljena iz več datotek izvorne kode in zato se najprej prevede v standardno JavaScript kodo in nato prepiše na spletni strežnik.

Kot že rečeno, to ni tečaj programiranja, zato bom razlike med okoljema prikazal tako, da bom vzel par primerov iz VUE tečaja (Intruduction), na VUE spletni strani [7].

V razlagi bom uporabljal besedo RAZOR, ki je zelo podobna besedi BLAZOR in to namenoma. RAZOR je namreč MS programski jezik za »nadzor« HTML kode in BLAZOR za obdelavo HTML uporablja RAZOR. Od tod v resnici tudi izhaja njegovo ime. RAZOR je namreč MS že imel in potem si je nad njim pač izmislil BLAZOR.

2.4.1 Ustvarjanje aplikacije

VUE (potrebujemo node)	*Blazor (potrebujemo .NET CORE)*

`npm install -g @VUE/cli`
`VUE create pozdrav`
`cd pozdrav`
`npm run serve`
	`dotnet new blazorwasm -n pozdrav`
	`cd pozdrav`
	`dotnet run`

Kot je vidno tukaj, razlike ni. Oba ukaza pripravita vse potrebno za razvoj aplikacije v VUE oz. Blazor okolju.

2.4.2 Prikaz podatkov

VUE	*Blazor*

`<template>`
`  <div>`
`  {{ message }}`
`  </div>`
`</template>`
 	
`<script>`
`  export default {`
`  data () {`
`  return {`
`  message: 'Hello VUE!'`
`  }`
`  }`
`  }`
`</script>`
	`<div>`
	`  @message`
	`</div>`
	
	`@code {`
	`  private string message = "Hello Blazor!";`
	`}`

Kot je vidno iz primera je programska koda pri Blazor aplikaciji manjša, predvsem pa je pomembno, da BLAZOR uporablja RAZOR programski jezik, kar pomeni, da spremenljivke izpisuje z uporabo afne (@message), VUE pa uporablja Mustashe, kar pomeni, da spremenljivke izpisuje z uporabo dvojnih zavitih oklepajev ({{ message }})

2.4.3 Prikaz podatkov - uporaba znotraj HTML atributov

VUE	*Blazor*

`<template>`
`  <div>`
`  <span v-bind:title="message">`
`  Hover your mouse over me for a few seconds`
`  to see my dynamically bound title!`
`  </span>`
`  </div>`
`</template>`
 	
`<script>`
`  export default {`
`  data () {`
`  return {`
`  message: 'You loaded this page on '`
	`<div>`
	`  <span title="@message">`
	`  Hover your mouse over me for a few seconds`
	`  to see my dynamically bound title!`
	`  </span>`
	`</div>`
	
	`@code {`
	`  private string message =`
	`  $"You loaded this page on`
	`  {DateTime.Now.ToLongDateString()}`
	
	`{DateTime.Now.ToLongTimeString()}";`
	`}`
	`]]>`

```
          + new Date().toLocaleString()
      }
    }
  }
</script>
```

VUE za spremembo lastnosti HTML elementov uporablja v-bind, medtem ko Blazor še vedno ohranja poenoten način z uporabo afne. Gornji primer uporabniku izpiše trenutni datum in čas, v kolikor se z miško dovolj dolgo zadrži na DIV elementu.

2.4.4 Pogojni prikaz

VUE

```
<template>
  <div @click="ToogleMessage">
    <span v-if="seen">Now you see
me</span>
    <span v-else>Click Me!</span>
  </div>
</template>

<script>
  var app = new VUE({
    el: "#app",
    data: {
      seen: true
    },
    methods: {
      ToogleMessage() {
        this.seen = !this.seen;
      }
    }
  });
</script>
```

Blazor

```
<div @onclick="ToogleMessage">
  @if (seen)
  {
    <span>Now you see me</span>
  }
  else
  {
    <span>Click Me!</span>
  }
</div>

@code {
  private bool seen = false;

  private void ToogleMessage() =>
            seen = !seen;
}
```

Tukaj pa se vidi prva slabost RAZOR programskega jezika, saj ne pozna posebnih HTML lastnosti, s pomočjo katerih bi lahko nadzirali posamezne elemente. VUE tako pozna ukaz v-if (tudi v-show), kar je super orodje za pogojno prikazovanje elementov. Po drugi strani pa RAZOR ohranja koncept programskega jezika, zato je pač potrebno kodo zapreti v pogojni stavek. Resnici na ljubo lahko prikaz elementa kontroliramo tudi z uporabo CSS lastnosti, vendar tukaj ni moj namen prikazati različne opcije, temveč preprosto razliko med obravnavo vejitev v HTML kodi.

2.4.5 Zanke

VUE

```
<template>
  <div>
    <ol>
      <li v-for="todo in todos">
        {{ todo.text }}
      </li>
    </ol>
  </div>
</template>

<script>
  export default {
    data () {
```

Blazor

```
<div>
  <ol>
    @foreach(var item in todos)
    {
      <li>@item</li>
    }
  </ol>
</div>

@code {
  private string[] todos = {
    "Learn C#",
    "Learn .NET Core",
```

```
      return {
        todos: [
          { text: 'Learn JavaScript' },
          { text: 'Learn VUE' },
          { text: 'Build something
awesome' }
        ]
      }
    }
  }
</script>
```

```
    "Build something awesome"
  };
}
```

Tudi pri zankah je (meni osebno) VUE lepši, saj zopet uporabi vgrajeni konstrukt v-for, medtem ko Blazor ohranja klasično zanko, ki se grdo zareže v sicer lepo HTML kodo.

2.4.6 JavaScript in Blazor

Iz Blazor aplikacije lahko seveda uporabljate in kličete JavaScript funkcije in knjižnice. V primeru VUE je to seveda samoumevno, saj je VUE napisan v JavaScript. V primeru Blazor aplikacije pa to ni tako samoumevno.

Kličete lahko tako JavaScript kodo in Blazor aplikacije, kot tudi Blazor kodo iz JavaScript aplikacije. Povezljivost je torej 100%.

Na žalost pa zadeva ne deluje tako, da bi lahko programer znotraj Blazor aplikacije preprosto pisal v JavaScript, temveč je potrebno napisati JavaScript funkcijo v neki .js datoteki, ki se naloži in potem jo lahko Blazor pokliče preko vgrajenega objekta IJSRuntime.

Recimo primer iz testne aplikacije:

```
js.InvokeAsync<string>("highlightCode", code, language);
```

V tem primeru je js objekt tipa JSRuntime, funkcija highlightCode, pa je JavaScript funkcija, ki sprejme dva parametra in vrača string:

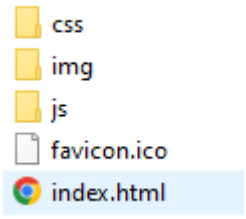
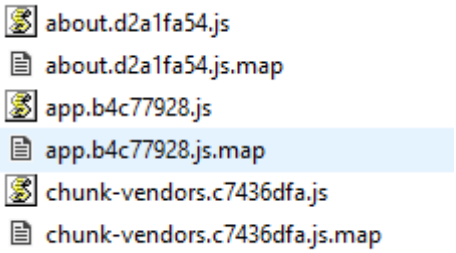
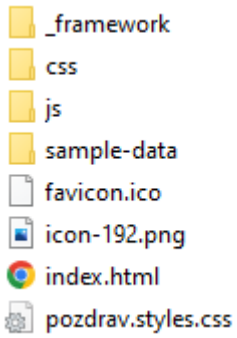
```
function highlightCode(code, language)
{
  let result = hljs.highlight(code, { language, ignoreIllegals: true })
  return result.value
}
```

V tem prispevku bom s programskimi primeri končal, saj sem jih napisal samo v toliko, da dobite občutek in vidite, da je zadeva zares zelo podobna. Bi pa v nadaljevanju pokazal, kaj dobimo, ko projekt prevedemo.

2.5 Prevod projekta

Ko smo s programiranjem zaključili in želimo naš produkt postaviti na spletno stran, ga je potrebno seveda najprej prevesti. VUE v ta namen uporablja klasičen nabor Node orodij in programsko kodo prevede s pomočjo orodja Webpack, Blazor pa uporabi .NET Core prevajalnik, ki ga razvijalec dobi z instalacijo .NET Core razvojnih orodij.

Toda ne glede na prevajalnik, slednji pač pripravi neko prevedeno kodo, ki jo je potrebno prepisati na strežnik. In vprašanje je, kaj pripravi kateri prevajalnik.

VUE	*Blazor*
VUE pripravi **dist** podmapo, ki ima lahko več podmap (odvisno od velikosti in kompleksnosti projekta), zagotovo pa je med podmapami tudi **js** podmapa, kamor **VUE** prevede program:  **js** podmapa torej vsebuje več ali manj **.js** datotek, spet seveda odvisno od velikosti in kompleksnosti projekta:  	Blazor lahko projekt prevede tako, da je direktno pripravljen za določene platforme, kot sta recimo Azure ali IIS. Ima pa tudi možnost prevoda v mapo na disku, kar lahko potem prepišemo na spletni strežnik. To je enakovredno VUE in zato prikazujem to opcijo. Blazor pripravi mapo **wwwroot**, ki ima, podobno kot v primeru VUE, več ali manj podmap, zagotovo pa ima podmapo **_framework**  in podmapa **_framework** je pravzaprav največja težava Blazor aplikacije. V njej se namreč nahajajo vse .NET osnovne knjižnice (DLL-i), katerih število je odvisno od kompleksnosti projekta, jih je pa v vsakem primeru veliko

Slika 3: Blazor Server Side.
Vir: lasten.

Blazor pripravi knjižnice v nestisnjeni obliki, s končnico **.dll**, kakor tudi stisnjene različice **.br** in **.gz.**

2.5.1 Uporaba Blazor WASM aplikacije na Apache spletnem strežniku

Na žalost ni dovolj, da na Apache spletni strežnik prepišemo samo wwwroot podmapo, temveč je seveda potrebno nastaviti tudi slavno Apache **.htaccess** datoteko. Osnovni primer takšne .htaccess datoteke je preprosto najti, je pa smiselno Apache prepričati, naj uporabi **.br** oz **.gz** datoteke (stisnjene datoteke), v kolikor jih uporabnik sprejme in večina uporabnikov jih sprejme. Na ta način lahko potrebno število prenesenih kB zmanjšamo kar za kakšnih 60%!

Tukaj prilagam **.htaccess** datoteko, ki jo sam uporabljam na Apache strežniku. V njej sem z odebeljeno pisavo označil element **podmapa-programa,** saj je to neka poljubna podmapa aplikacije na Apache strežniku. Smiselno je namreč, da Apache gosti več kot eno aplikacijo:

```
<IfModule mod_headers.c>
    # V kolikor je možno, uporabi brotli datoteke
    RewriteCond "%{HTTP:Accept-encoding}" "br"
    RewriteCond "%{REQUEST_FILENAME}\.br" "-s"
    RewriteRule "^(.*)\.(js|json|css|dll|dat|blat|wasm)$" "$1\.$2\.br" [QSA]

    # Pošlji ustrezen tip in prepreci dvojno stiskanje
    RewriteRule "\.css\.br$" "-" [T=text/css,E=no-brotli:1]
```

```
    RewriteRule "\.js\.br$"  "-" [T=text/javascript,E=no-brotli:1]
    RewriteRule "\.json\.br$"  "-" [T=application/json,E=no-brotli:1]
    RewriteRule "\.dll\.br$"  "-" [T=application/x-msdownload,E=no-brotli:1]
    RewriteRule "\.dat\.br$"  "-" [E=no-brotli:1]
    RewriteRule "\.blat\.br$"  "-" [E=no-brotli:1]
    RewriteRule "\.wasm\.br$"  "-" [E=no-brotli:1]

    <FilesMatch
"(\.js\.br|\.css\.br|\.json\.br|\.dll\.br|\.dat\.br|\.blat\.br|\.wasm\.br)$">
      Header append Content-Encoding br
      Header append Vary Accept-Encoding
    </FilesMatch>
</IfModule>

<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /podmapa-programa/
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /podmapa-programa/index.html [L]
</IfModule>
```

2.5.2 Verzioniranje

S sodobnimi brskalniki naletimo še na eno težavo in to je predpomnilnik. Brskalniki si namreč ob obisku spletne strani zapomnijo naložene datoteke in v kolikor je lokacija datoteke enaka neki prejšnji, je pač ne zahtevajo s strežnika, temveč jo vzamejo iz predpomnilnika.

Kje je težava? Težava se pojavi, ko mi programsko kodo popravimo, Blazor pa posamezne datoteke še vedno enako poimenuje. To pa pomeni, da uporabnik, čeravno smo mi prepisali novo verzijo, lahko še vedno pridobiva stare datoteke iz lastnega predpomnilnika. Ali pa se zgodi še hujša nočna mora in uporabnik dobi nekaj starih datotek iz predpomnilnika in nekaj novih s strežnika.

VUE to lepo rešuje z dodajanjem številk na konec datotek. Na prejšnji strani se lepo vidi, da je VUE eno izmed JavaScript datotek poimenoval app.b4c77928.js, kjer je b4c77928 pač »hash vrednost«, ki se bo ob naslednjem prevodu spremenila. V originalu se datoteka namreč imenuje app.js. Tega, po mojem vedenju, v tem trenutku, Blazor še nima zadovoljivo rešenega.

V dosedanjih projektih je zadostovalo, da sem napisal majhen pomožni programček, ki datoteko index.html popravi za prevajalnikom. Prevajalnik torej pripravi datoteko index.html, jaz pa nadalje popravim tako, da povezavam dodam še edinstven atribut.

V index.html tako recimo piše:

```
<link href="css/main.css" rel="stylesheet">
```

pomožni programček pa to popravi v:

```
<link href="css/main.css?cache_version=1.1.44" rel="stylesheet">
```

kjer je številka 1.1.44 nekaj, kar sam kontroliram. Na ta način dosežem, da strežnik vedno posreduje datoteke uporabniku, v kolikor se številka pač spremeni. Programsko kodo in primere lahko najdete na **GitHub** spletišču: https://github.com/MPrtenjak/OTS2022-Blazor

3 Zaključek

S programiranjem se ukvarjam profesionalno 25 let, sicer pa seveda še dobrih 10 let več. Moji začetki segajo v čase vzpona objektnega programiranja in jezika C++. Osebno se torej dobro počutim v statičnih jezikih z znano strukturo in čim manj dinamičnosti.

Preveč dinamičnosti mi ne diši, saj dnevno vzdržujem starejšo programsko kodo (tudi več kot 20 let staro!) in ne predstavljam si, kako bi to počel, če bi bila koda dinamična in bi me prevajalnik ne mogel opozoriti na določeno vrsto napak.

Ta uvod je pomemben, saj je to tudi razlog, da mi je Blazor tehnologija resnično všeč in se v njej počutim bolje, kot ob razvoju z JavaScript orodji. Vsaka izmed opcij seveda ima določene prednosti in slabosti, izmed katerih sem jih nekaj omenil v tem prispevku, nekaj pa v samem predavanju na konferenci.

Toda če nekako rešimo največjo težavo BLAZOR WASM aplikacije in to je dolgi začetni zagon (veliko DLL-ov) in posledično tudi veliko kB prenesenih podatkov, potem zame sploh ni nikakršne dileme in vedno bi raje izbral Blazor WASM pred neko JavaScript knjižnico.

Sam osebno vidim rešitev v tem, da bi se .NET WebAssembly knjižnice instalirale v brskalnik podobno kot v računalnik in potem bi lahko VSE aplikacije uporabljale tiste skupne knjižnice. Upam, da bodo sčasoma torej obstajali neki nabori standardnih knjižnic, ki jih bo lahko uporabnik dodal brskalnikom, WASM programi pa jih bodo lahko uporabili, ne da bi jih morali sami vleči s spleta.

Dokler pa bo torej Blazor imel težavo z veliko količino prenesenih podatkov, pa bom pri rešitvah uporabljal tudi VUE oz. bom med njima preklapljal glede na konkretne zahteve in potrebe.

Literatura

[1] WabAssembly, https://webassembly.org/, obiskano 15.07.2022

[2] Blazor, https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor, obiskano 15.07.2022

[3] Microsoft Silverlight, https://www.microsoft.com/Silverlight/, obiskano 15.07.2022

[4] .NET Foundation, https://dotnetfoundation.org/, obiskano 15.07.2022

[5] Adobe Flash Wiki, https://en.wikipedia.org/wiki/Adobe_Flash, obiskano 15.07.2022

[6] JQuery knjižnica, https://jquery.com/, obiskano 15.07.2022

[7] VUE Introduction, https://v2.VUEjs.org/v2/guide/index.html, obiskano 16.07.2022

Moderni trendi pri integracijah sistemov: agilne integracije, osredotočene na API-je

Dušan Rauter, CEO

Bintegra, d.o.o.,
Maribor, Slovenija
dusan.rauter@bintegra.com

Sinopsis Poslovni uspeh vse bolj temelji na sposobnosti podjetja, da se odzove na spremembe. Ker novi moteči akterji vstopajo na trge in tehnologija spreminja pričakovanja potrošnikov, se morajo organizacije razvijati, da bi te spremembe obravnavale v veliko krajših ciklih, kot kdaj koli prej. Sodobne arhitekture in procesi programske opreme lahko naredijo organizacije učinkovitejše pri soočanju s spremembami in nastopajo kot zmagovalci na svojih trgih.

Tehnologija je preoblikovala celotne industrije. Trend velikih sprememb vodi do tega, da organizacije radikalno preoblikujejo svoja IT okolja, da bi lahko ponudile nove digitalne storitve, ki jih stranke zahtevajo, bolje in hitreje kot njihovi konkurenti.

Sposobnost integracije aplikacij in podatkov je ključnega pomena za uresničitev različnih poslovnih ciljev in zagotavljanje konkurenčnih storitev. Nove in vse težje zahteve se postavljajo k starim pristopom, saj digitalne inovacije in motnje postajajo glavni normativ. Pred nami so novi izzivi, kot je povečana uporaba hibridnih aplikacij v oblaku, IT okolja v oblaku, potreba po razširitvi sistemov za nove storitve za partnerje in stranke ter povpraševanje po sodobnih aplikacijah. Zaradi tega je integracija podjetja še pomembnejša in zagotavljanje storitev na hitrejši in neprekinjen način še bolj kritično. Verjamemo, da je boljši način za reševanje teh novih in hitro naraščajočih izzivov integracija različnih aplikacij in informacijskih sistemov, uporaba strategije agilne integracije. Agilna integracija združuje tri zmogljive arhitekturne principe: porazdeljeno integracijo z mikro storitvami, vmesnike za programiranje aplikacij (API) in vsebnike (kontejnerje). Namen tega spodbujanje agilnosti, razvoj novih poslovnih procesov in končno zagotavljanje konkurenčne prednosti.

Ključne besede:

integracije sistemov
agilne integracije sistemov
mikro storitve
vsebniki (kontejnerji)
Red Hat rešitve

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.8

1 Uvod - Živimo v povezanem svetu

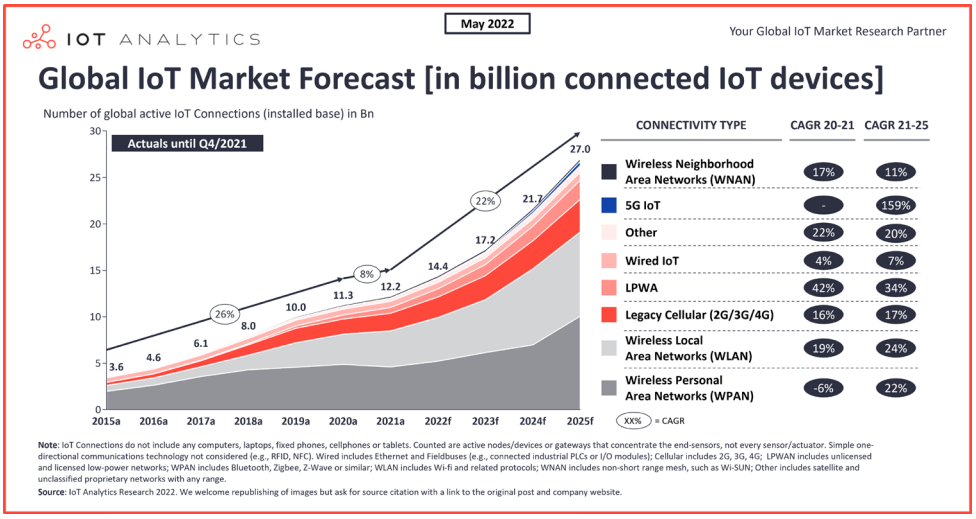
Danes živimo v vedno bolj povezanem svetu. Svet poslovnih integracij in povezovanje poslovnih rešitev v storitve se spreminja z neverjetno naglico. Če pogledamo samo podatke statistične urada Republike Slovenije vidimo, da je na primer situacija v Slovenije glede povezanosti na internet naslednja:

- 93% gospodinjstev ima dostop do interneta,
- 85% oseb uporablja internet vsak da ali skoraj vsak dan,
- 24% podjetij ima implementirano prodajo preko spletnih strani,
- 92% podjetij ima mobilni širokopasovni dostop do interneta,
- 71% prebivalcev uporablja splet za nakupe (e-kupci).

1.1 Področje IoT

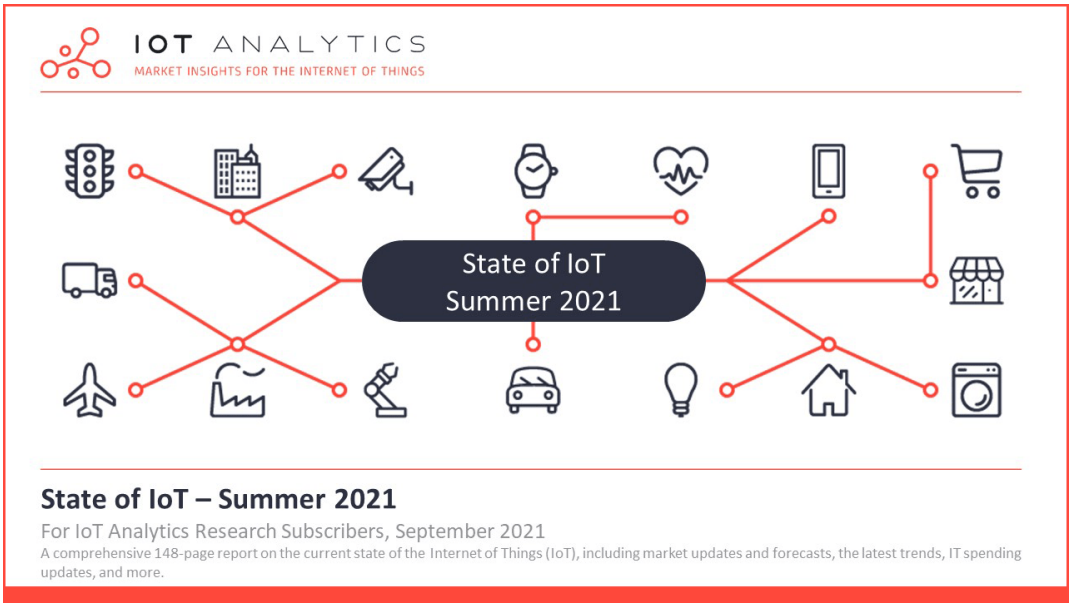
Če pogledamo področje uporabe IoT naprav na svetu ugotovimo naslednje:

- Stanje IoT 2021: število povezanih naprav interneta stvari raste za 9 % na 12,3 milijarde po vsem svetu
- Trg IT storitev za IoT bo leta 2025 predstavljal priložnost v vrednosti 58 milijard dolarjev.



Slika 1: Predvidevanja rasti IoT trga.
Vir: https://iot-analytics.com/wp/wp-content/uploads/2022/05/Global-IoT-Market-Forecast-in-billion-connected-IoT-devices-min.png

Število naprav, ki so povezane v internet neprestano raste, prav tako tudi načini uporabe le teh:



Slika 2: Stanje povezav IoT naprav.
Vir: https://iot-analytics.com

1.2 Big Data in spremembe v energetiki

Pričakuje se, da bo trg analitike velikih podatkov v energetskem sektorju v napovedanem obdobju 2020-2025 rasel s 10,22 %. Analitika velikih podatkov v energetskem sektorju igra ključno vlogo pri zmanjševanju porabe energije in izboljšanju energetske učinkovitosti.

Pomanjkanje fosilnih goriv povzroča uporabo nadomestnih virov energije, kot so sončne, valovne in vetrne turbine, pri katerih uporaba narašča z veliko hitrostjo. Tako je postala nujna uporaba naprednih orodij, ki uporabljajo analitična orodja, ki temeljijo na velikih podatkih, za razumevanje obnašanja ali prilagajanja teh virov energije.

Energetski sektor zahteva visoko vzdrževanje za spremljanje strojev in opreme, zaradi česar ima analitika velikih podatkov pomembno vlogo.

Pametno merjenje v analitiki velikih podatkov vključuje komponente, kot so delovanje omrežja, terenske storitve, načrtovanje virov, uporabniška izkušnja in skladnost z zakonodajo. Pomaga napovedati povpraševanje na podlagi zbranih podatkov, kar bo spodbudilo rast trga.

1.3 Uporaba API-jev strmo narašča

Raziskave poročajo o naslednjem (vir: https://nordicapis.com/20-impressive-api-economy-statistics/)

- Več kot 90 % razvijalcev uporablja API-je.
- 69 % jih uporablja API-je tretjih oseb.
- 20 % uporablja interne ali zasebne API-je.
- Razvijalci porabijo 30 % svojega časa za kodiranje API-jev.
- Potovalni podatki so najpomembnejša potreba po integraciji.
- API Management Market je do leta 2023 ocenjen na 5,1 milijarde dolarjev.
- 91 % organizacij je imelo API varnostni incident leta 2020.

- Na GitHubu je več kot 2 milijona repozitorijev API-jev.
- Več kot 93 % ponudnikov komunikacijskih storitev uporablja OpenAPI.
- OpenBanking bo do leta 2024 imel 130 milijonov uporabnikov.
- Avtomatizacija v oblaku bo do leta 2023 postala 623,3 milijarde dolarjev vredna industrija.
- 83 % vsega internetnega prometa pripada storitvam, ki temeljijo na API-jih.

Glede na poročilo „State Of The API Economy" 56 % razvijalcev meni, da API-ji pomagajo zgraditi boljše digitalne izdelke. Prav tako menijo, da API-ji:

- Pospešijo inovacije (52 %),
- Pospešijo integracije sistemov (40 %),
- Ustvarjajo poslovno vrednost (36%),
- Ali so izdelki sami po sebi (22%).

2 Agilnost kot pogoj za hiter razvoj in kompetenčne prednosti

Agilna podjetja odlikujejo naslednje značilnosti:

- Hitro se odzivajo na priložnosti in grožnje.
- Manj planirajo in več poskusijo narediti.
- Poskušajo in prototipirajo ideje ter se ne bojijo neuspeha.
- Naredijo razliko skozi inovativne aplikacije in rešitve.

Sposobnost integracije aplikacij in podatkov je ključnega pomena za uresničitev različnih poslovnih ciljev in zagotavljanje konkurenčnih storitev. Nove in vse težje zahteve se postavljajo k starim pristopom, saj digitalne inovacije in motnje postajajo glavni normativ.

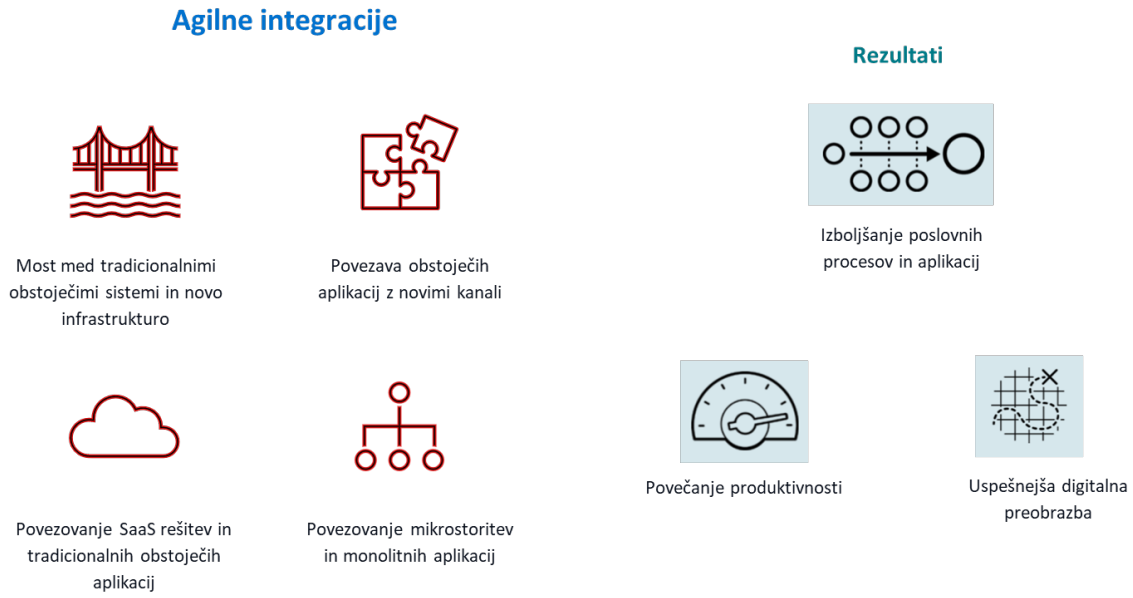
Pojavljajo se novi izzivi, kot je povečana uporaba hibridnih aplikacij v oblaku, IT okolja v oblaku, potreba po razširitvi sistemov za nove storitve za partnerje in stranke ter povpraševanje po sodobnih aplikacijah. Zaradi tega je integracija podjetja še pomembnejša in zagotavljanje storitev na hitrejši in neprekinjen način še bolj kritično. Verjamemo da je boljši način za reševanje teh novih in hitro naraščajočih izzivov integracija različnih aplikacij in informacijskih sistemov uporaba strategije agilne integracije.

2.1 Agilnost + Integracije = Agilne integracije

Izraz "agilna integracija" se običajno nanaša na stalno integracijo/neprekinjen razvoj in dostavo posodobitev (CI/CD), ki predstavljajo integracijo ali združevanje različnih razvojnih procesov v neprekinjen proces. Agilna integracija temelji na sodobnih platformah, procesih in tehnologijah, ki so za to primerno hitre in prilagodljive rešitve. Agilni pristopi k integracijam lahko strankam pomagajo vključiti svoje integracijske storitve kot del procesov CI/CD.

Agilno integracijo definiramo kot arhitekturni pristop, ki se posebej nanaša na integracijske tehnologije in procese – z uporabo prednosti agilnih metod in fleksibilnih arhitektur mikrostoritev – tako da je mogoče aplikacije in

podatke v več sistemih ter storitve kar najhitreje integrirati in prilagajati, da izpolnijo hitro spreminjajoče se zahteve digitalnega poslovanja.



Slika 3: Značilnosti agilnih integracij

2.2 Potreba po integraciji, kjer je nameščeno

Vedno bolj se pri integracijah soočamo s potrebo, da se zadeve integrirajo tam, kjer so nameščene.

Pri tem se srečujemo z naslednjim:

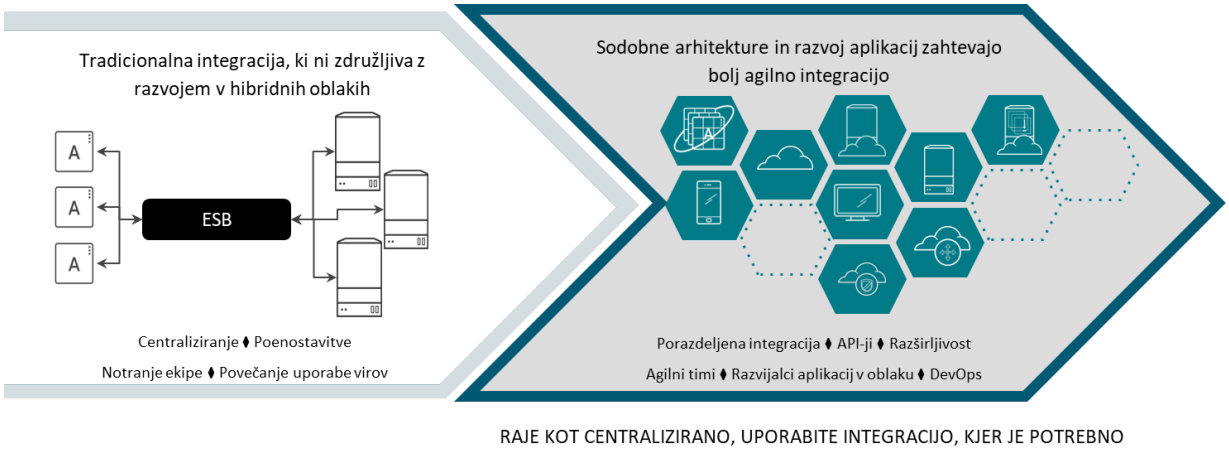
- Vedno več aplikacij je nameščenih v oblakih,
- Ne obstaja eden dominanten oblak,
- Medtem ko večina organizacij ne preskoči z lokalnih namestitev na več oblačnih ponudnikov naenkrat, je 93 odstotkov podjetij zgradilo strategijo v več oblakih (multi-cloud),
- Na organizacijo se uvaja ali testira povprečno 3,4 javnih oblakov in 3,9 zasebnih oblakov, kar jim omogoča, da svoje zmogljivosti v oblaku prilagodijo svojim zahtevam v oblaku.

2.3 Zakaj agilna integracija?

Enterprise Service bus (ESB) in druge tradicionalne integracijske tehnologije zagotavljajo bistvene zmogljivosti (kot so transformacija, usmerjanje, orkestracija in povezljivost), ki so potrebne za integracijo in povezavo različnih aplikacij. ESB-ji v kombinaciji z arhitekturnimi vzorci, kot je storitveno usmerjena arhitektura (SOA) zagotavljajo platformo za enkapsulacijo integracijske logike kot storitve za večkratno uporabo. SOA omogoča prednosti modularnih poslovnih funkcij in možnost ponovne uporabe teh storitev. Ampak tak pristop tudi postavlja izzive, kot so: zapletenost tehnologije in upravljanja, dolgotrajni izvedbeni cikli.

Podjetja že leta želijo ublažiti težave monolitne interoperabilnosti aplikacij, ki prinašajo eksponentno povečanje števila povezav med aplikacijami. Rešitev tega problema je bila integracija vseh aplikacij v eno samo poslovno vodilo. Vendar pa težava z več povezavami ni izginila in namesto tega je kompleksnost teh povezav bila omejena v eno samo škatlo (ESB), ki se je lahko spreminjala le navpično in je sama postala monolitna aplikacija. Ta arhitektura je zahtevala centralizirano upravljanje za nadzor povezav znotraj ESB. Cilj zmanjšanja kompleksnosti

s prisilnim povezovanjem vseh aplikacij v eno osrednjo »povezavo« ni uspelo doseči. Ta rešitev je zmanjšala agilnost razvoja novih aplikacij.

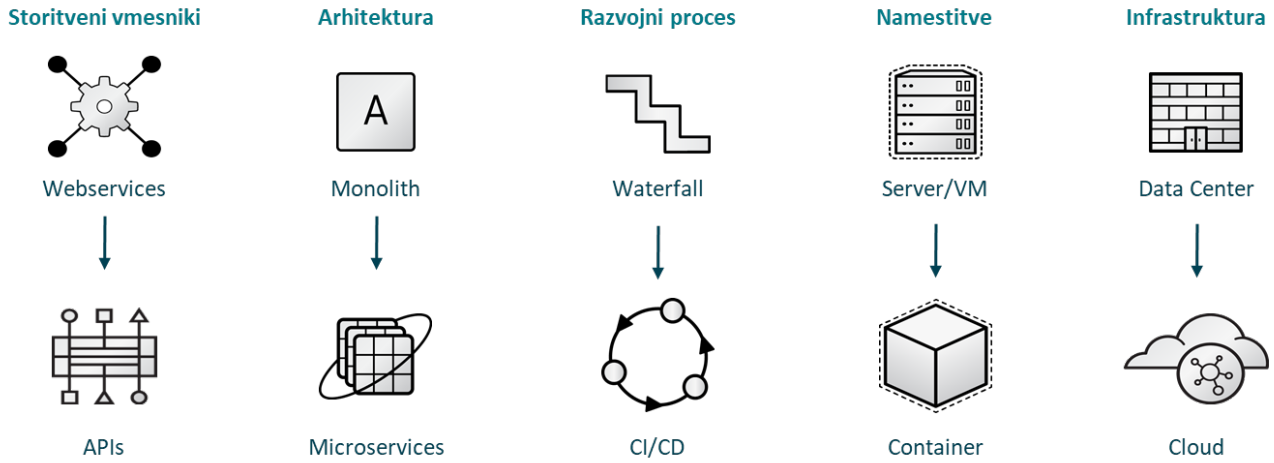


Slika 4: Primerjava tradicionalnega in agilnega pristopa pri integracijah

Arhitektura v slogu mikrostoritev zagotavlja bolj agilen pristop k razvoju aplikacij z oblikovanjem in razvojem funkcionalnosti aplikacij kot storitev, ki jih je mogoče samostojno razmestiti.

Arhitekture mikrostoritev olajšajo gradnjo agilnih poslovnih sistemov, ki omogočajo hitrejše spreminjanje poslovanja, gradnja novih funkcionalnosti, eksperimentiranje in boljšo pripravljenost na obravnavanje motenj.

Da bi podjetja ostala konkurenčna, potrebujejo integracijsko platformo, ki lahko podpira trenutno in naslednjo generacijo arhitekture.



Slika 5: Podjetja gredo skozi velike spremembe

Agilni pristop integracij temelji na platformah, procesih in tehnologijah, ki so bolj primerne za prilagodljive rešitve. Z agilnim pristopom so integracije lahko del aplikacije vključno z arhitekturo mikrostoritev, ki zagotavljajo večjo agilnost. Za integracijo bi morala biti ključna zmožnost porazdeljenih ekip, zadolženih za zagotavljanje novih inovativnih sistemov in rešitev. Združevanje tehnoloških zmogljivosti z različnimi organizacijskimi in procesnimi pristopi omogoča resnične spremembe.

2.4 Trije stebri agilne integracije

Tri glavne tehnologije, ki podpirajo agilni pristop integracije:

- **Distribuirana integracija**: nekaj deset vzorcev integracije na visoki ravni odraža delo v podjetju in tokove podatkov. Ko so ti integracijski vzorci nameščeni v vsebnikih, jih je mogoče razmestiti v obsegu in na lokaciji, ki sta potrebna za posebne aplikacije in ekipe. Ta pristop predstavlja porazdeljeno integracijsko arhitekturo in ne tradicionalno centralizirano integracijsko arhitekturo in omogoča posameznim ekipam, da z agilnostjo definirajo in uvedejo integracijske vzorce, ki jih potrebujejo.
- **API-ji:** Stabilni, dobro upravljani API-ji imajo velik vpliv na sodelovanje med ekipami, razvoj in delovanje. API-ji zavijejo ključna sredstva v stabilne vmesnike za večkratno uporabo, ki jih je mogoče uporabiti in ponovno uporabiti kot gradnike v celotni organizaciji, s partnerji in strankami. API-je je mogoče namestiti skupaj s kontejnerji v različna okolja, kar različnim uporabnikom omogoča interakcijo z različnimi nizi API-jev.
- **Kontejnerji**: za tehnologije API in porazdeljene integracije delujejo kontejnerji kot osnovna platforma za uvajanje. Omogočajo razmestitev storitve v določenem okolju na način, ki je enostaven in dosleden za razvoj, testiranje in vzdrževanje. Ker so kontejnerji prevladujoča platforma za okolja mikrostoritev, uporaba kontejnerjev kot integracijske platforme ustvarja veliko bolj pregleden in sodelovalen odnos med razvojnimi in infrastrukturnimi ekipami.

Slika 6: Tri ključne zadeve, potrebne za arhitekturni pristop agilnih integracij.

3 Red Hat-ov pristop do Agilnih integracij

3.1 Distribuirane integracije

Ker uporabniki vse bolj sodelujejo prek digitalnih kanalov (mobilnih, družabnih, sporočil in spleta) se je programska oprema premaknila k modelu, ki je bolj osredotočen na uporabnika, pri čemer je povpraševanje po funkcijah in storitvah prihaja od zunaj navznoter, namesto da bi bila prihajalo od znotraj navzven. To v kombinaciji z enostavnim dostopom do uporabniku prijaznih programskih orodij in storitev v oblaku, je povzročilo premik v vlogi IT-ja na tisto, ki je bolj za sodelovanje in omogočanje poslovanja, manj pa za centraliziran nadzor in varovanje pred vdori.

Zaradi teh tržnih in organizacijskih premikov se mora IT pri integraciji zrahljati v bolj modularen in porazdeljen model, medtem ko se še vedno držijo varnosti in upravljanja. Integracijski kompetenčni centri, ki so bili de-facto centri odličnosti za najboljše prakse za integracijo podjetij se zdaj razvijajo v bolj porazdeljen integracijski in

poslovno usmerjen model. Velike integracijske ekipe v IT organizaciji se prepuščajo manjšim in bolj prilagodljivim ekipam, ki se lahko odzovejo z večjo agilnostjo.

3.2 Kontejnerji

Sodobne aplikacije se morajo pogosto razširiti na stotine tisoč ali milijone transakcij, pogosto na nenačrtovan in elastičen način. V večini primerov je treba te aplikacije povečati neodvisno druga od druge za zagotavljanje ustreznih podatkov na zahtevo. Pogosto so tudi predmet nenehnih posodobitev, da bi zadostili hitremu digitalnemu povpraševanju. V vedno bolj povezanem in podatkovno vodenem svetu, so razširljivost in upravljanje osnovne infrastrukture, hitrost dostopa do podatkov ter stalen razvoj in cikli dostave kritični za uspeh.

Agilna integracija, ki jo omogočajo kontejnerske tehnologije, igra osrednjo vlogo pri tem. Kontejnerji so temelj za porazdeljeni integracijski model, saj se izognejo ozkim grlom, povezanim s togim in centraliziranim ESB-jem. Zagotavljajo sredstva za ustvarjanje lahke, vendar popolnoma preizkušene in potrjene enote uvajanja, ki jih je mogoče neodvisno povečati na zahtevo.

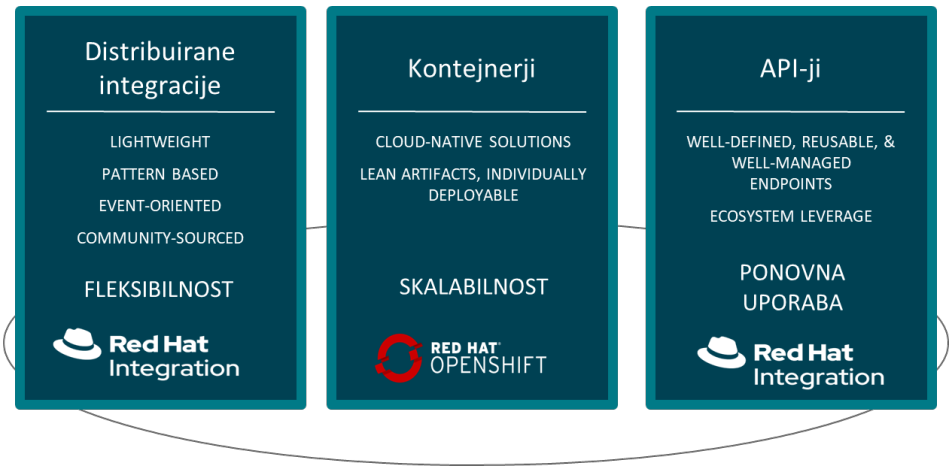
Zagotavljajo tudi odlično paradigmo za razdelitev kompleksnih sistemov na manjše delujoče enote, mikro storitve. Uvajanje storitev v kontejnerjih omogoča neodvisnim ekipam, da uvedejo in pospešiti dostavo storitev. Predvsem pa zagotavljajo pomembno zmogljivost, ki omogoča doslednost upravljanje različic in razširljivost posameznih gradnikov sistema.

3.3 API-ji

Programska oprema, ki podpira splet in mobilne naprave, je postala norma. Ker se integracijske točke in uporabniški vmesniki množijo, integracija od točke do točke ni več trajnostna. Namesto tega postajajo API-ji sprejemljivo sredstvo za povezovanje poslovnih sredstev – IT sistemov, internih in zunanjih, odjemalskih aplikacij in stranke.

Prednost API-jev je, da lahko pomagajo velikim, tradicionalnim podjetjem delovati kot manjša, bolj okretna podjetja, s povečanjem njihove agilnosti. Nasprotno pa lahko API-ji omogočijo start-up organizacijam, da razširijo svojo prisotnost hitro čez nova ozemlja. Z zmanjšanjem kompleksnosti integracije in pospeševanjem ustvarjanja aplikacij, lahko API-ji spodbujajo notranje inovacije, dosežejo nove stranke, razširijo izdelke in storitve ter ustvariti živahne partnerske ekosisteme.

Odpiranje API-jev običajno omogoča organizacijam zagotavljanje enotnih podatkovnih in transakcijskih vmesnikov notranjim in zunanjim razvijalcem, partnerjem in strankam za izboljšan dostop do podatkov in transakcije. Takšne organizacije lahko razvijajo tudi programske aplikacije za dostop do teh API-jev in ustvarijo novo funkcionalnost in vrednost tako zase kot za širši svet.



Slika 7: Implementacija agilnih integracij z Red Hat Integration.

3.4 Red Hat Open Shift- Hibridne integracijske platforme in kontejnerji

Enotna platforma in veriga orodij v oblačnih okoljih zagotavljata doslednost in prilagodljivost za trenutne in prihodnje načrte uvajanja aplikacij in rešitev.

Slika 8: Red Hat Open Shift.

3.5 Red Hat FUSE- Distribuirana integracijska rešitev prilagojena oblačnim rešitvam

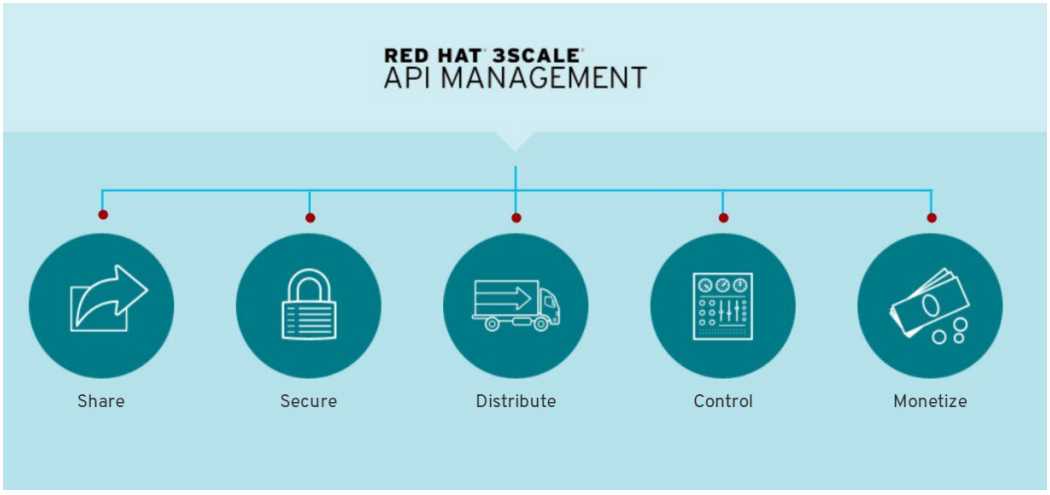
Slika 9: Lastnosti Red Hat FUSE

3.6 Red Hat 3Scale API management- upravljanje API-jev

Red Hat 3Scale API management je platforma za skupno rabo, distribuiranje, nadzor, monetiziranje in zagotavljanje varnosti API-jev.

Ko imate API-je nameščene v vašem okolju, postane ključnega pomena, da upravljate, kdo jih lahko uporablja in za kakšen namen. Prav tako morate začeti spremljati uporabo teh različnih uporabnikov, da boste vedeli, kdo je/ne uspešen pri njihovi uporabi.

Red Hat ponuja eno vodilnih orodij za upravljanje API-jev, ki zagotavljajo storitve upravljanja. Rešitev 3scale API Management vam omogoča hitro in enostavno zaščito in upravljanje vaših API-jev. Red Hat 3scale API Management nam omogoča tudi spremljanje varnosti. Če je za vas pomembno tudi pridobivanje prihodkov iz API-jev, vam 3scale omogoča monetizacijo vaših API-jev z vgrajenim sistemom zaračunavanja.



Slika 10: Red Hat 3Scale API Management.

3.7 Red Hat AMQ - Dogodkovno zasnovana arhitektura in prenos sporočil

Red Hat AMQ – ki temelji na odprtokodnih skupnostih, kot sta Apache ActiveMQ in Apache Kafka – je prilagodljiva platforma za sporočanje, ki zanesljivo zagotavlja informacije, omogoča integracijo v realnem času in povezovanje interneta stvari (IoT).

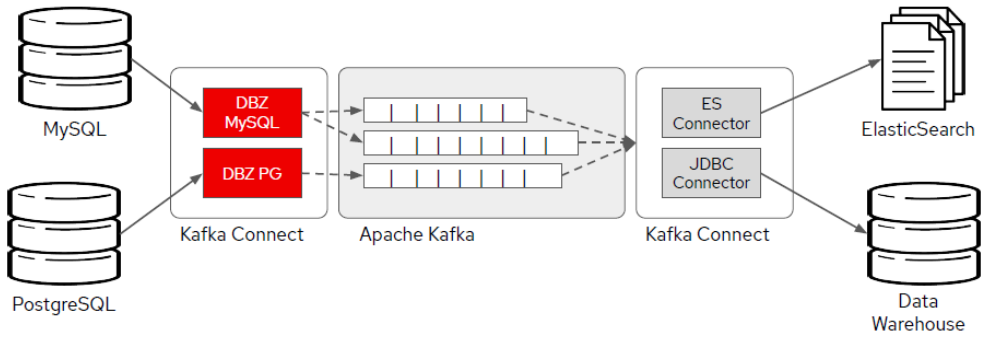
Slika 11: Trije načini uporabe Red Hat AMQ

3.8 Podatkovna integracija (Debezium)

Uporaba Debeziuma omogoča replikacijo podatkov in širjenje podatkov med storitvami brez povezovanja.

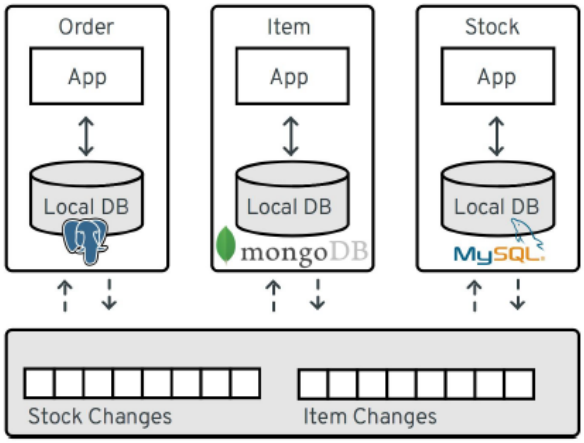
Uporaba za replikacijo podatkov:

- Zajem sprememb podatkov zagotavlja pretvorbo sprememb iz različnih baz podatkov v dogodke, ki lahko tečejo tam, kjer so potrebni, ko so potrebni:
  - Replikacija podatkov v drugo bazo podatkov.
  - Hranjenje podatkov DWH.
  - Podpira MySQL, Postgres, SQL Server, MongoDB, DB2.



Slika 12: Replikacija podatkov

Odlično primeren je tudi za arhitekture mikro storitev: širjenje podatkov med storitvami brez povezovanja.



Slika 13: Mikro storitve.

3.9 Varnost API-jev: Red Hat SSO

Red Hat SSO je brezplačen dodatek k paketu Red Hat Integrations in:

- Bazira na Keycloaku,
- Upravitelj odprtokodnega dostopa in identitet,
- Posredovanje identitete,
- Povezava uporabnikov z imeniškimi storitvami, ki temeljijo na LDAP,
- Odjemalske knjižnice za JavaEE, Spring, NodeJS, JS in več.

4 Zaključek

Sposobnost integracije aplikacij in podatkov je ključnega pomena za uresničitev različnih poslovnih ciljev in zagotavljanje konkurenčnih storitev. Nove in vse težje zahteve se postavljajo k starim pristopom, saj digitalne inovacije in motnje postajajo glavni normativ. Zaradi tega je integracija podjetja še pomembnejša in zagotavljanje storitev na hitrejši in neprekinjen način še bolj kritično. Verjamemo da je najboljši način za reševanje teh novih in hitro naraščajočih izzivov integracija različnih aplikacij in informacijskih sistemov uporaba strategije agilne integracije. Pri tem vam lahko pomagajo tudi rešitve podjetja Red Hat, opisane v tem članku.

Literatura

[1] Red Hat Partner Content Hub

Omejevanje frekvence zahtevkov na odjemalcu

Aljaž Mislovič

Podatkovni inženiring & zaledni sistemi, Databox, Ptuj
aljaz.mislovic@outlook.com

Sinopsis Strežniki so pogosto tarča namernega ali nenamernega povišanja frekvence zahtevkov v kratkem časovnem obdobju. Z različnimi metodami omejevanja zato strežniki ščitijo svoje sistemske vire, podatkovne baze in izboljšujejo uporabniško izkušnjo. V članku smo reševali realni problem produkcijskega okolja Databox, ki v vlogi odjemalca z različnih strežnikov črpa podatke za svoje uporabnike. V prispevku smo opisali, kako smo zasnovali in implementirali programsko rešitev, ki uspešno omejuje frekvenco zahtevkov na razne strežnike z različnimi metodami omejevanja. Na koncu smo predstavili tudi rezultate testiranja na več sto milijonov izvedenih zahtevkov na produkcijskem okolju, ter podali rezultate zmogljivosti.

Ključne besede:
frekvenca zahtevkov
omejevanje zahtevkov
odjemalec
drsno okno
API

Opomba: Prispevek temelji na: Mislovič, A. (2022). Sistem za omejevanje frekvence zahtevkov na odjemalcu : diplomsko delo, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor: A. Mislovič.

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.9

1 Uvod

Najbrž smo vsi kdaj stopili v dvigalo in opazili opozorilni napis za maksimalno število ljudi oziroma maksimalno skupno težo. Te omejitve proizvajalci nastavijo, da zaščitijo dvigalo in ljudi v njem. Spletne storitve uporabljajo podobno omejitev, ki jo imenujemo *omejevanje frekvence zahtevkov*. S tem zaščitijo sebe in svoje uporabnike.

Z napredkom tehnologije in povezovanjem različnih sistemov na spletu hitro narašča tudi količina prenesenih podatkov. Velik delež teh podatkov se prenaša s pomočjo spletnih strežnikov. Spletne storitve množični dostop do podatkov zagotavljajo preko aplikacijskega programskega vmesnika (angl. Application Programming Interface, API). Povečana količina poizvedb pa zahteva tudi določeno mero zaščite pred različnimi namernimi ali nenamernimi zlorabami kot je npr. prevelika frekvenca zahtev. Spletne storitve se pred slednjo zaščitijo s pomočjo sistemov za omejevanje frekvence zahtevkov, ki so lahko implementirani z uporabo različnih algoritmov. Odjemalci, ki pošiljajo veliko količino zahtevkov na isti strežnik, morajo tako sami poskrbeti za omejevanje.

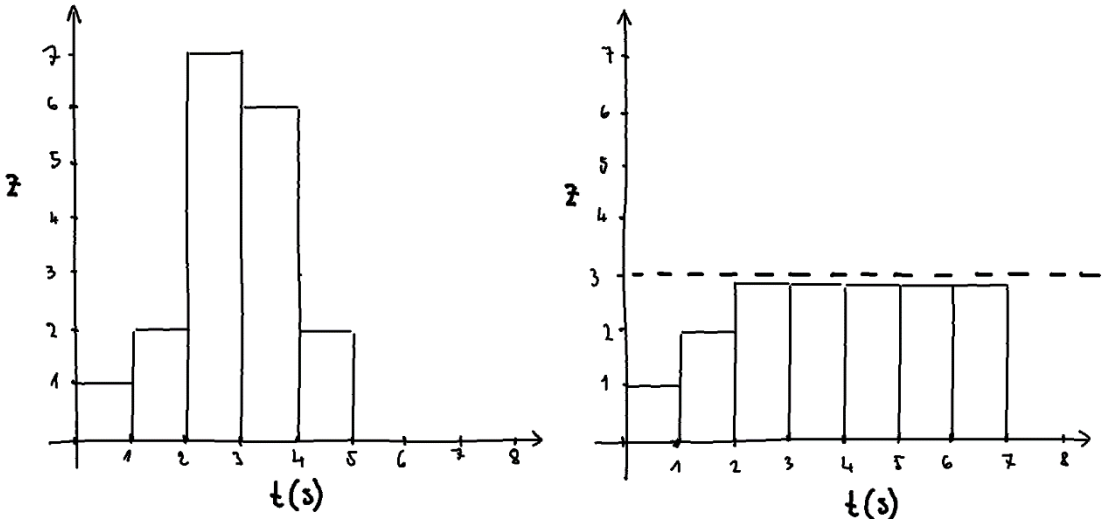
V tem članku, ki temelji na diplomskem delu [5], primarno opisujemo reševanje realnega problema na produkcijskem okolju podjetja Databox [Databox, glej diploma]. Databox podpira pridobivanje podatkov iz več kot 70-ih spletnih storitev, več kot polovica le-teh pa implementira eno ali več različnih metod omejevanja zahtevkov. Na spletne strežnike z omejitvami dnevno izvede okrog 17 milijonov zahtevkov. Razporejanje teh zahtevkov znotraj omejitev je zahtevno opravilo z vidika sistemskih virov, zato smo želeli implementirati sistem, ki bo kos nalogi.

V nadaljevanju članka smo podrobneje predstavili kontekst problema. Teoretično smo razdelali nekaj najpogostejših metod omejevanja frekvence zahtevkov, s katerimi se redno srečujemo. Opisanim metodam smo nato poiskali skupni imenovalec, ter pripravili načrt za implementacijo. S psevdokodo smo implementacijo tudi predstavili. Ob koncu pa smo podali še nekaj rezultatov s produkcijskega okolja.

2 Omejevanje zahtevkov na strani odjemalca

Omejevanje zahtevkov se običajno implementira na strani ponudnika storitve in predstavlja učinkoviti način za preprečevanje širjenja napak v porazdeljenih sistemih. Ena izmed primerov uporabe omejevanja zahtevkov je zaščita pred napadi onemogočanja storitve (angl. Denial of Service), s katerim nepridipravi preplavijo nek strežnik z zahtevki. Podobno lahko tudi odjemalci nenamerno preplavijo sistem. To se lahko zgodi zaradi napak v kodi ali pa s kodo, ki istočasno pošilja ogromne količine zahtevkov. Drug primer uporabe so spletne storitve za prenos reprezentativnega stanja (angl. representational state transfer, REST), ki uporabljajo podatkovno bazo, za zaščito le-te pa razvijalci storitve implementirajo omejevanje zahtevkov. Brez omejevanja bi lahko storitev, ki je zmožna sprejeti več zahtevkov hkrati, povzročila ogromno količino sočasnih zahtevkov, podatkovne baze pa niso opremljene z jasnim načinom omejevanja frekvence zahtevkov. Vse to lahko upočasni spletno storitev, poveča odzivni čas ali pa spletno storitev naredi začasno nedostopno [1].

Spletne storitve, ki omejevanja zahtevkov nimajo implementiranega, torej nimajo kontrole nad uporabo le-te s strani odjemalcev, s tem odjemalcem dovolijo prosto uporabo in so dovzetna za zgoraj opisane situacije. Pomembnost omejevanja zahtevkov smo ponazorili s spodnjo sliko. Slika 1 na levi prikazuje primer števila zahtevkov brez omejevanja, na desni pa pa enako količino zahtevkov, ki se zaradi omejitve treh zahtevkov na sekundo razporedijo skozi čas.



Slika 1: Primerjava zahtevkov na spletno storitev brez in z omejevanjem zahtevkov.
Vir: lasten.

Drugi, manj tipičen način omejevanja je omejevanje zahtevkov na strani odjemalca, ki pa je v našem primeru potreben. Več kot 70 različnih podatkovnih virov se na različne načine integrira v Databox platformo. Vsaka od integracij podpira številne metrike in omogoča pretok veliko podatkov, kar se preslika v ogromno količino zahtevkov, ki jih izvedemo.

Omejevanje zahtevkov je mogoče izvajati čisto reaktivno ali proaktivno. Primer reaktivnega omejevanja zahtevkov je odziv na informacijo klicane spletne storitve, da zahtevke pošiljamo s preveliko frekvenco - torej po tem, ko se zgodi napaka. V tem primeru moramo velikokrat počakati, da mine določen čas, ki je specificiran s strani ponudnika kot omejitev, zato da ne pride do prevelike obremenitve storitve. V veliko primerih ta potreben čas čakanja ni podan v povratni informaciji ob napaki. V našem primeru smo se osredotočili predvsem na drugi, proaktivni način, s katerim poskrbimo, da strežnikov prekomerno ne obremenjujemo in da do napake sploh ne pride.

V Databoxu integracije implementirajo integracijski inženirji, ki morajo pred dejansko implementacijo vsako integracijo podrobno analizirati, raziskati, ter najdene metode omejevanja tudi opisati v primernem formatu, da ga lahko strojno obdelujemo in uporabimo. Najpogosteje najdene metode smo predstavili v nadaljevanju.

3 Metode omejevanja zahtevkov

Kot smo omenili že prej, je namen omejevanja zahtevkov med drugim tudi zaščita same spletne storitve ter njenih sistemskih virov. Najpogosteje razvijalci to storijo z implementacijo ene ali več metod omejevanja frekvence zahtevkov, opisanih v nadaljevanju. Preden pa se razvijalci omejevanja frekvence zahtevkov na spletnih storitvah lotijo izbire prave metode, morajo razumeti, zakaj omejitve implementirajo. Ključnega pomena je, da razumejo svoj sistem in se vprašajo, po katerem ključu (angl. limiting key) bodo implementirali omejitve. Namreč to informacijo morajo razumeti tudi odjemalci.

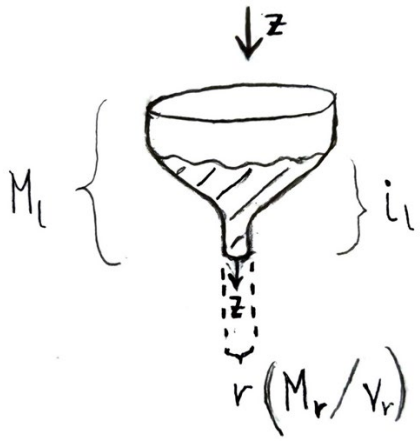
Strežniške sistemske vire si v večini primerov deli več odjemalcev oziroma uporabnikov. Povišano število zahtevkov enega odjemalca lahko nenamerno (ali namerno) vpliva na ostale odjemalce. V ta namen je izbira ključa omejevanja pomembna za zagotavljanje enakomerne in poštene delitve sistemskih virov. Omejitve oziroma kvote razvijalci nastavijo na število zahtevkov v nekem časovnem obdobju. Pri nekaterih strežnikih so rezervirane ali porabljene kvote lahko tudi plačljive [1].

Primeri ključev so identifikator (angl. identifier, ID) odjemalca (angl. client ID), ID uporabnika (angl. user ID), naslov spletnega protokola (angl. internet protocol address, IP address), ID vira, za katerega zahtevamo podatke, itd. Ko je ključ izbran, lahko začnemo slediti uporabi servisa za vsak ključ in izberemo metodo za omejevanje frekvence zahtevkov [1]. V nadaljevanju smo opisali tri najpogostejše metode, s katerimi se integracijski inženirji na Databoxu najpogosteje srečujejo.

3.1 Puščajoče vedro

Puščajoče vedro (angl. *Leaky Bucket*) je metoda, ki jo lahko opišemo z analogijo vedra ali lijaka. Lijak prepušča neko maksimalno količino vode v določenem času. Prav tako ima lijak maksimalen volumen vode, ki je lahko istočasno v lijaku. Na zgornji strani v lijak prilivamo vodo. Če vodo prilivamo hitreje, kot jo lijak prepušča, bo lijak preplavljen.

V primeru strežniških zahtevkov lahko algoritem puščajočega vedra implementiramo kot vrsto FIFO (angl. first in, first out). Odjemalec zahtevke iz vrste procesira z ritmom r, ki ga definiramo s številom zahtevkov M_r v časovni enoti v_r. Vrsta ali lijak lahko naenkrat hrani neko maksimalno število zahtevkov M_l, prav tako pa hrani števec, označen z i_l. Števec nam pove trenutno količino zahtevkov, shranjenih v vrsti. Zahtevek, ki ga želimo shraniti v vrsto, smo označili s črko z. Ponazoritev lahko vidimo na sliki 2.



Slika 2: Ponazoritev metode »puščajoče vedro«.
Vir: lasten.

Sliko 2 smo ponazorili še s spodnjo psevdokodo.

Psevdokoda 1: Omejevanje zahtevkov z metodo »puščajoče vedro«

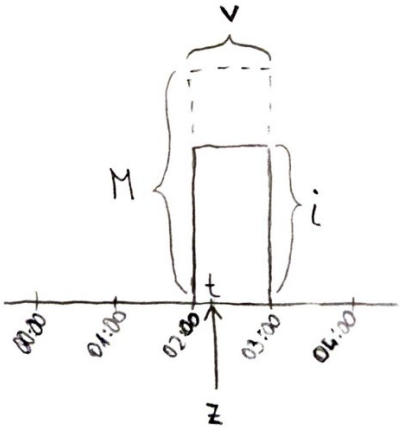
```
če i + z <= M
    zahtevek shrani v vrsto
sicer
    zahtevek zavrzi ali počakaj, da nekaj zahtevkov iz vrste zaključi s
procesiranjem
```

Prednost te metode je, da se zahtevki iz vrste pošiljajo v procesiranje s približno isto hitrostjo, ne glede na to, če zahtevke shranjujemo s konstantno ali povišano hitrostjo. Vendar pa ima ta metoda tudi pomanjkljivost, saj ne zagotavlja, da se procesiranje zahtevka zaključi v nekem konstantnem času. Shranjevanje povišanega števila zahtevkov v vrsto v kratkem času pomeni, da bo vsak naslednji zahtevek na procesiranje čakal dlje, saj je v vrsti ostalo nekaj starejših zahtevkov [2].

3.2 Fiksno okno

Metoda za omejevanje zahtevkov fiksno okno (angl. fixed window) uporablja fiksno velikost časovnega okna oziroma časovni interval, v katerem definiramo maksimalno število zahtevkov. Za lažje razumevanje lahko tudi to metodo opišemo s terminologijo iz prejšnje metode, terminologijo vedra. Pri metodi fiksnega okna vedro ne pušča, ampak se sprazni ob določenem času – na primer, vodo ves dan zlivamo v vedro, ki ga spraznimo vsako jutro ob točno določenem času. Če v nekem dnevu v vedro zlijemo več vode, kot je njegov volumen, bo voda tekla čez rob [1].

Strežniki za implementacijo te metode najprej definirajo velikost okna ***v***. Velikosti so najpogosteje nastavljene v človeku berljivi obliki, kot so na primer 1 minuta, 1 ura ali 1 dan. Prav tako definirajo maksimalno število zahtevkov ***M***, ki jih lahko v tem oknu izvedemo na strežnik. Tudi pri tej metodi je implementiran števec ***i***, ki nam pove število zahtevkov, ki jih je uporabnik izvedel v določenem oknu. Zahtevek, ki ga odjemalec pošlje ob nekem času ***t***, smo označili s črko ***z***. Metodo na časovnici prikazuje slika 3.



Slika 3: Ponazoritev metode fiksno okno.
Vir: lasten.

Odločitveni model za sprejemanje ali zavračanje zahtevkov se glasi:

Psevdokoda 2: Omejevanje zahtevkov z metodo fiksno okno.

```
če i + z <= M(t)
    zahtevek sprejmi
sicer
    zahtevek zavrzi in počakaj na naslednje časovno okno
```

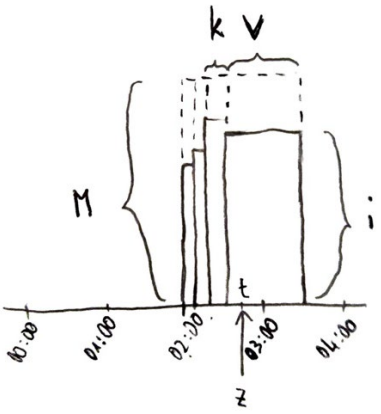
Prednost te metode je enostavna implementacija, ki zahteva zelo malo sistemskih virov, vendar pa nas ta metoda še vedno ne obvaruje pred povišanim številom zahtevkov v zelo kratkem času, ki lahko preobremenijo sistem. Dnevno fiksno okno odjemalcem namreč dovoli, da lahko celotno dovoljeno število zahtevkov izvedejo že v prvi minuti dneva [1].

3.3 Drsno okno

Namen metode drsno okno (angl. sliding window) je čim bolj zgladiti število zahtevkov na strežnik skozi čas. Podobno kot pri metodi fiksno okno tudi drsno okno uporablja velikost časovnega okna oziroma interval, v katerem definiramo maksimalno število zahtevkov. Za razliko od fiksnega okna pa se drsno okno ne »izprazni« le na koncu časovnega okna, temveč se skozi čas pomika oziroma drsi z določenim korakom.

Konec časovnega okna je torej vedno ta trenutek, zaokrožen navzgor na neko časovno enoto, ki jo definira velikost koraka (milisekunde, sekunde, minute, itd.).

Konfiguracija vsebuje – enako kot pri fiksnem oknu – velikost časovnega okna ***v***, le da je z namenom glajenja zahtevkov skozi čas večinoma podana v manjših enotah (1 s, 10 s, 60 s, 100 s ..). Časovno okno skozi čas drsi s korakom ***k***. V časovnem oknu je prav tako definirano maksimalno število zahtevkov ***M***, ki jih strežnik dovoljuje izvesti v časovnem oknu, števec ***i*** pa shranjuje število zahtevkov, ki jih odjemalec naredi v posameznem časovnem oknu. Zahtevek ***z***, ki ga odjemalec naredi ob času ***t***, torej pade v več oken istočasno (odvisno od velikosti časovnega okna in velikosti koraka), kar prikazuje spodnja slika.



Slika 4: Ponazoritev metode drsno okno.
Vir: lasten.

Omejevanje s to metodo smo opisali s psevdokodo 3.

Psevdokoda 3: Omejevanja zahtevkov z metodo drsno okno.

```
če i + z <= M(t)
    zahtevek sprejmi
sicer
    zahtevek zavrzi in počakaj na zdrs časovnega okna, dokler zahtevki niso pod
omejitvijo
```

Implementacija te metode je nekoliko zahtevnejša, saj je treba časovno okno zamikati in s tem ustrezno posodabljati tudi števec zahtevkov. Časovno okno z drsenjem odjemalce spodbuja k enakomernejšemu pošiljanju zahtevkov na strežnik.

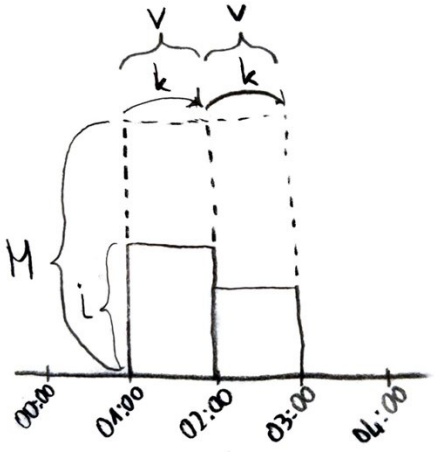
3.4 Skupni imenovalec opisanim metodam

Med raziskavo metod za omejevanje frekvence zahtevkov je postalo jasno, da imajo vse izmed njih nekaj skupnih točk. Med načrtovanjem rešitve smo se začeli spraševati, ali lahko vse tri naštete metode podpremo z eno samo rešitvijo, zato smo jih poskušali dati na skupni imenovalec. Ugotovili smo, da je vsem skupno to, da vsaka izmed njih dovoljuje določeno število zahtevkov v nekem časovnem oknu.

3.4.1 Primerjava metod drsno okno in fiksno okno

Metodi imata za posamezno okno teoretično enako opisano velikost okna ***v***, enako opisano maksimalno kapaciteto okna ***M*** ter števec trenutnega števila zahtevkov ***i***. Razlika je le v drsnem koraku ***k***. Korak nam pove, za koliko enot na časovnici okno drsi. Pri razlagi metode fiksnega okna koraka sicer nismo omenili,

povedali pa smo, da se naslednje okno začne na koncu prejšnjega okna (npr. naslednjo uro ali naslednji dan). Na osnovi tega lahko trdimo, da je korak k pri metodi fiksnega okna enak velikosti okna v. To je možno razbrati tudi s slike 5.



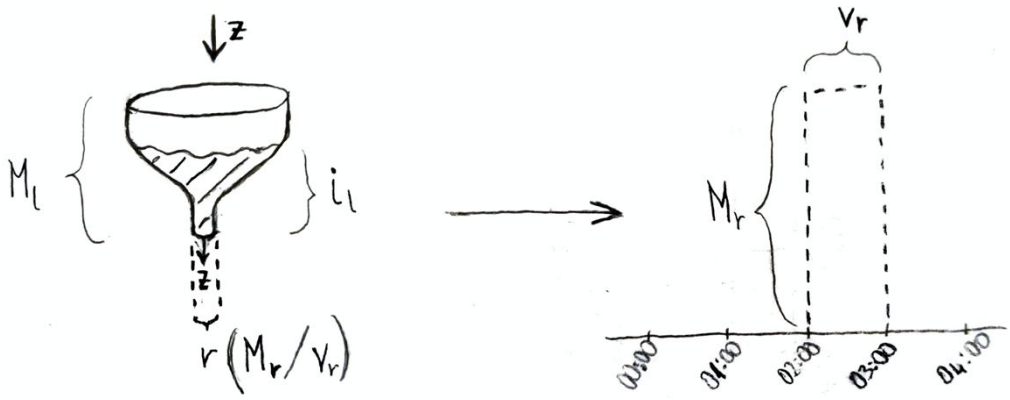
Slika 5: Primerjava metod fiksno okno in drsno okno.
Vir: lasten.

Tako smo ugotovili, da lahko obe metodi damo na skupni imenovalec, če na strani odjemalca implementiramo metodo drsno okno.

3.4.2 Primerjava metod drsno okno in puščajoče vedro

Metoda puščajoče vedro se od metode drsno okno razlikuje predvsem po tem, da ima puščajoče vedro vrsto. Kot smo navedli v opisu metode, je namen te vrste sprejeti večje število zahtevkov v kratkem času, brez vpliva na aplikacijski del strežnika. Procesiranje zahtevkov iz te vrste je določeno z ritmom r. Pomislili smo na primer, ko na strežnik pošiljamo maksimalno število zahtevkov M_r v enaki časovni enoti (oknu) v_r, kot določa ritem r. To pomeni, da vsi zahtevki takoj zapustijo vrsto, saj ne čakajo na nobene predhodne zahtevke. Če ostanemo znotraj teh omejitev, potem vrste oziroma vedra ali lijaka ne potrebujemo.

Naloga naše programske rešitve je, da zna zahtevke pošiljati znotraj nekih omejitev. Če metodi puščajoče vedro v implementaciji odstranimo vrsto, lahko metodo opišemo zgolj z ritmom. Kot smo omenili, je ritem definiran z nekim številom zahtevkov z v neki časovni enoti, na primer 10 z/1 s, 1200 z/60 s, 5000 z/3600 s. Na sliki 6 smo ritem r preslikali na časovnico.



Slika 6: Preslikava metode puščajoče vedro na časovnico.
Vir: lasten.

Ritem r metode puščajoče vedro nas hitro spomni na ponazoritev metode fiksno okno, ki smo jo ponazorili na sliki 3. V prejšnjem podpoglavju smo primerjali metodi drsno okno in fiksno okno ter ugotovili, da je metodo fiksno okno mogoče opisati z metodo drsno okno, menimo, da je tudi metodo puščajoče vedro možno opisati z metodo drsno okno. Z drugačnim oziroma bolj omejenim opisom metode puščajoče vedro smo torej to metodo dali na skupni imenovalec z metodo drsno okno. S tem smo dosegli podporo večim različnim metodam omejevanja z eno samo programsko rešitvijo. V nadaljevanju smo torej načrtovali le podporo metodi drsno okno.

4 Načrtovanje programske rešitve

V prejšnjem poglavju smo navedli, da je možno tri najpogosteje uporabljene metode omejevanja frekvence zahtevkov opisati z metodo drsno okno. Metod za omejevanje obstaja več, kot smo jih našteli. Ena izmed možnosti je bila raziskati vse metode in poiskati skupni imenovalec vseh. To bi povečalo kompleksnost rešitve, podpora vsem pa bi najverjetneje vpeljala dodatno kompleksnost ter dodatne omejitve. V izogib temu smo se odločili, da naša rešitev implementira zgolj metodo drsno okno. Vseh prej omenjenih 70+ integracij, ki jih Databox podpira, so integracijski inženirji uspeli opisati s to metodo.

Pred implementacijo smo iz preteklih izkušenj in implementacij definirali še nekaj drugih predpostavk in omejitev:

- k števcu zahtevkov vedno prištejemo en zahtevek naenkrat (to nam omogoča natančnejše omejevanje);
- predvidevamo, da strežnik in odjemalec števce zahtevkov prištevata v trenutku, ko odjemalec zahtevek izvede (zanemarimo omrežno latenco oz. čas, ki ga zahtevek potrebuje, da doseže strežnik);
- časovno okno v mora biti naravno število;
- časovno okno v je podano v sekundah;
- maksimalno število zahtevkov M mora biti naravno število;
- zgornje meje velikosti časovnega okna ne bomo nastavljali, ker ne poznamo vseh primerov uporabe; dodali jo bomo po potrebi.

4.1 Konfiguracija omejitve frekvence zahtevkov

Naša rešitev mora za delovanje razumeti, s kakšnimi omejitvami dela. Glede na te omejitve smo kasneje implementirali posamezne števce ter integracijskim inženirjem zagotovili, da danih omejitev ne prekoračijo. V konfiguracijo omejitve smo zajeli dva tipa spremenljivk - osnovne in identifikacijske spremenljivke. Osnovne so tiste, ki jih sistem potrebuje za omejevanje.

Ponovno moramo poudariti, da sistem istočasno omogoča uporabo več različnih omejitev in za vsako omejitev več različnih števcev. Opisne spremenljivke tako potrebujemo, da znamo za posamezno integracijo in posameznega uporabnika voditi pravi števec, kjer se nahaja trenutno stanje omejitve.

Končni nabor zahtevanih vhodnih parametrov v sistem za omejevanje je sledeč:

- *id* – spremenljivka, ki služi kot identifikacija posameznega števca omejitve;
- *interval* – spremenljivka, v kateri navedemo velikost časovnega okna v v sekundah;
- *requests* (slov. zahtevki) – spremenljivka, v kateri navedemo maksimalno število dovoljenih zahtevkov v tem časovnem oknu M.

Metoda drsno okno pa za delovanje potrebuje še korak k. Med raziskavo spletnih storitev smo ugotovili, da dokumentacije velikosti koraka ne podajo. Nekatere ne podajo niti algoritma, ki se skriva za omejitvijo. Uporabniku smo želeli poenostaviti uporabo naše rešitve, zato smo se odločili, da bo korak k izračunala koda sama.

4.2 Izračun velikosti koraka

Korak je ključna spremenljivka, ki bo definirala natančnost naše programske rešitve. Pri definiranju le-tega smo se seveda vprašali, kakšna je smiselna in zadovoljiva natančnost. Velikost koraka prav tako določa število zdrsov, ki ga posamezno časovno okno naredi. Več zdrsov pomeni tudi zahtevnejše procesiranje.

Omejitve, ki jih uporabljajo naše integracije, uporabljajo velik nabor intervalov. Najmanjši interval, ki ga med integracijami najdemo, je 1 sekunda, največji pa 1 dan. Iz tega razloga fiksne velikosti koraka za vse omejitve ni bilo smiselno nastaviti. Po preračunavanju smo prišli do zaključka, da bomo velikost koraka računali približno sorazmerno z velikostjo intervala. Pravila za izračun velikosti koraka glede na časovno okno oziroma interval smo zapisali v tabeli 1.

Tabela 1: Velikost koraka glede na velikost intervala.

Interval (od, do]	Korak	Najmanj zdrsov	Največ zdrsov
0 s – 10 s	1 cs	100	1000
10 s – 60 s (1 min)	1 ds	110	1000
60 s (1 min) – 3600 s (1 h)	1 s	61	3600
3600 s (1 h) – 86400 s (1 dan)	1 min	61	1440
86400 s (1 dan) – ∞	1 h	25	∞

4.3 Drsni števci zahtevkov

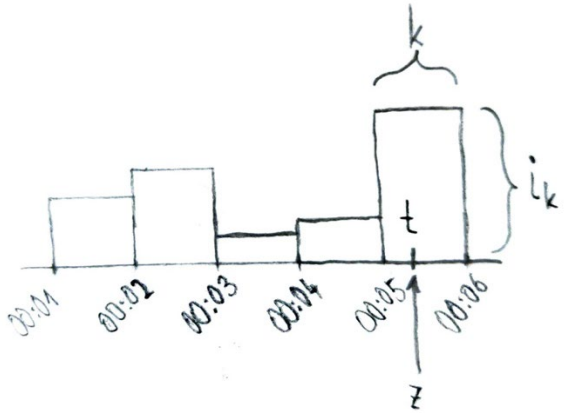
Pri načrtovanju smo izhajali iz razlage za metodo drsno okno v podpoglavju 3.3, kjer smo navedli, da časovno okno v, prej opisano s spremenljivko interval, drsi skozi čas s korakom k. Vsa časovna okna so zaokrožena na velikost koraka, kar definira tudi njihovo natančnost. V nadaljevanju smo posamezno časovno okno opisali z matematično notacijo odprtega in zaprtega intervala – »[začetek okna, konec okna)«. Časovno okno se začne z zaprtim intervalom, kar pomeni, da čas, ki opisuje začetek okna, še zajema

zahtevke, izvedene v tistem trenutku. Konec časovnega okna pa smo opisali z odprtim intervalom. To pomeni, da časovno okno vključuje zahtevke, izvedene vse do trenutka, ki opisuje konec časovnega okna. Zahtevki, izvedeni točno v času konca okna, pa v časovno okno več ne spadajo.

Pri načrtovanju smo našli več načinov implementacije, a na koncu izbrali slednjega: **vodimo števce korakov k_i ter iz njih izračunamo stanje omejitve na zahtevo**. S to rešitvijo ob vsakem zahtevku izvedemo zgolj eno prištevanje. Trenutno stanje omejitve dobimo tako, da seštejemo vse števce korakov, ki spadajo v trenuten interval. Podrobnejši opis in načrt te rešitve smo razdelili na dva dela, to sta prištetje posameznega števca koraka in izračun trenutnega stanja časovnega okna oziroma trenutnega stanja omejitve.

4.4 Prištetje števca koraka

Števec koraka nam pove, koliko zahtevkov je bilo narejenih v časovnem oknu, ki je enako velikosti koraka. Iz časa zahtevka t in velikosti koraka k bo naša rešitev sama določila, kateri števec koraka i_k mora povečati za 1. Na časovnici smo to ponazorili z naslednjo sliko.



Slika 7: Prištetje števca koraka, v katerem je bil zahtevek izveden.
Vir: lasten.

S spodnjo psevdokodo 4 smo opisali celoten postopek prištetja števca koraka.

Psevdokoda 4: Postopek prištetja števca koraka

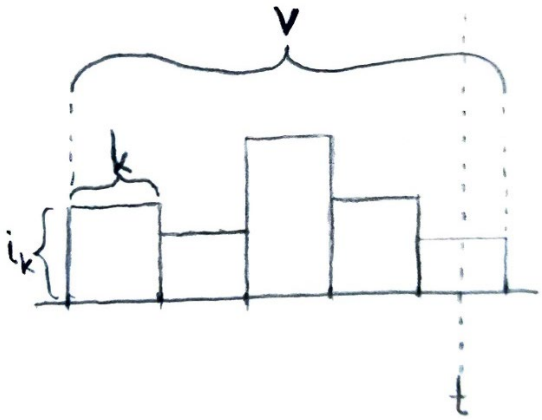
```
t = časZahtevka

v = pridobiIntervalOmejitve() // glej poglavje 4.1
k = izračunajVelikostKoraka(v) // glej poglavje 4.2

idŠtevcaKoraka = izračunajIdPripadajočegaŠtevecaKoraka(k, t)
če !številecKorakaŽeObstaja(idŠtevcaKoraka)
    inicializirajŠtevecKoraka(idŠtevcaKoraka)

povečajŠtevecKoraka(idŠtevcaKoraka)
```

S tem smo dosegli, da naša rešitev zna voditi števce korakov. Uporabnik mora zgolj poklicati metodo vedno, ko izvede zahtevek, in kot parameter podati čas zahtevka. Naslednji del naše rešitve pa je ključen za

omejevanje frekvence zahtevkov. V naslednjem podpoglavju smo opisali, kako naša rešitev izračuna trenutno stanje omejitve.

4.5 Izračun trenutnega stanja omejitve

Trenutno stanje omejitve smo v prejšnjih poglavjih opisali s črko ***i***, ki pomeni trenutno število zahtevkov, narejenih v intervalu omejitve oz. v časovnem oknu ***v***. Ta informacija kasneje Databox platformi pove, če lahko v tem trenutku izvede še kak zahtevek, ali je omejitev že dosežena in moramo še počakati. Za izračun stanja zahtevkov bomo iz konfiguracije omejitve potrebovali velikost časovnega okna ***v***. Prav tako bomo ponovno uporabili prej opisane metode za izračun velikosti koraka ter pripadajočega okna zahtevka. Slika 8 prikazuje števce korakov i_k, ki spadajo v neko časovno okno ***v***, s črko ***t*** pa smo označili trenuten čas, ob katerem izračunavamo stanje omejitve.



Slika 8: Števci koraka v časovnem oknu oziroma intervalu.
Vir: lasten.

Po tej formuli moramo za izračun števca zahtevkov ***i*** sešteti vse števce korakov i_k, ki so element časovnega okna ***v*** v trenutku ***t***. To smo zapisali tudi s psevdokodo 5.

Psevdokoda 5: Izračun trenutnega stanje omejitve.

```
t  = trenutenČas
v = pridobiIntervalOmejitve() // glej poglavje 4.1
vZačetek = izračunajZačetekIntervalaGledeNaČas(v, t)
vKonec = izračunajKonecIntervalaGledeNaČas(v, t)
števciKoraka = pridobiŠtevceKorakaMed(vZačetek, vKonec)
i = seštejŠtevceKoraka(števciKoraka)
```

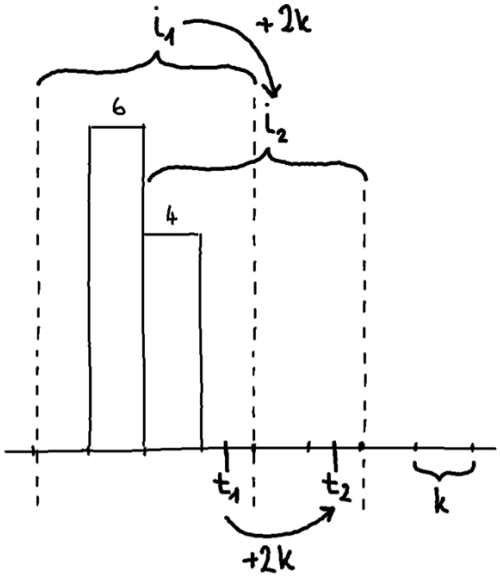
S tem naši rešitvi omogočimo pridobivanje stanja omejitve za trenutni čas. Ko ta seštevek primerjamo z maksimalnim številom zahtevkov ***M***, lahko ugotovimo, kako je stanje posamezne omejitve.

4.6 Izračun naslednjega prostega časovnega okna za izvedbo zahtevka

Z informacijo stanja omejitve se lahko uporabnik v nekem trenutku odloči, če bo izvedel nek zahtevek. Če je odgovor na to vprašanje negativen, mora uporabnik to vprašanje ponavljati, dokler mu izračun stanja ne pove, da lahko zahtevek izvede. Zato smo se odločili našo programsko rešitev izboljšati z odgovorom na vprašanje: »Kdaj lahko izvedem naslednji zahtevek?« V podpoglavju 4.5 smo specificirali metodo, ki zna izračunati trenutno stanje zahtevkov, kot parameter pa prejme čas t. Če ugotovimo, da trenutno stanje

zahtevkov i v časovnem oknu v za čas t ne omogoča izvedbe naslednjega zahtevka, lahko čas t zamaknemo za korak k v prihodnost. Če ta čas podamo v metodo, se začetek časovnega okna na časovnici prestavi za en korak naprej. To efektivno pomeni, da prvi števec koraka odstranimo od prej izračunanega seštevka. Ker števci koraka v prihodnosti ne obstajajo oziroma je njihovo stanje 0, pa ničesar ne dodamo.

Na sliki 9 smo izračun naslednjega prostega časovnega okna za izvedbo zahtevka prikazali na primeru omejitve **10 zahtevkov / 4 časovne enote**. Stanje števca i_1 za trenuten čas t_1 je enako 10, kar pomeni, da je omejitev dosežena. Novih zahtevkov v času t_1 ne smemo izvajati. Ugotovili smo, da moramo t_1 zamaknit za 2 koraka k v prihodnost (t_2), saj bo takrat stanje števca i_2 enako 4, kar pomeni, da lahko izvedemo nove zahtevke.



Slika 9: Zamik okna v prihodnost in iskanje naslednjega okna, v katerem bo možna izvedba zahtevka.
Vir: lasten.

Prej omenjeni postopek zamika smo opisali še s psevdokodo spodaj.

Psevdokoda 6: Izračun naslednjega prostega časovnega okna za izvedbo zahtevka.

```
t  = trenutenČas
v = pridobiIntervalOmejitve() // glej poglavje 4.1
M = pridobiŠteviloDovoljenihZahtevkovOmejitve() // glej poglavje 4.1
k = izračunajVelikostKoraka(v) // glej poglavje 4.2
i = izdačunajStanjeOmejitveVČasu(t) // glej poglavje 4.5
dokler i >= M
    t = t + k
    i = izdačunajStanjeOmejitveVČasu(t) // glej poglavje 4.5
časNaslednjegaOknaZaIzvedboZahtevka = t
```

S tem diagramom poteka smo zaključili načrtovanje naše programske rešitve. Na Databoxu smo to rešitev implementirali kot ločen strežnik, ki hrani informacije o vseh števcih. Strežnik ponuja dve končni točki.

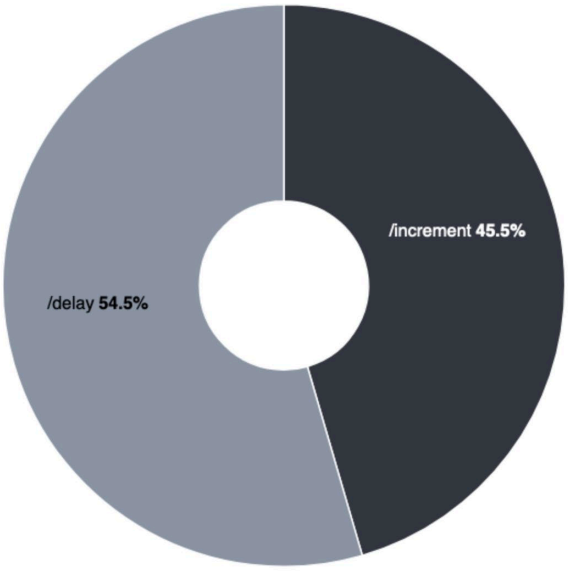
Končno točko za prištevanje zahtevka k števcu zahtevkov smo poimenovali */increment*. Klicati jo je potrebno z metodo *POST*. Uporabnik jo pokliče, kadar izvede zahtevek na neko spletno storitev. Zraven klica na to končno točko mora uporabnik podati parameter, ki pove točen čas izvedbe zahtevka. S tem sistem ve, kateri števec koraka mora prišteti.

Končno točko za izračun naslednjega prostega okna za izvedbo zahtevka smo poimenovali ***GET /delay***. Uporabnik jo pokliče z metodo ***GET***, preden želi zahtevek izvesti. Kot odgovor dobi točen čas naslednjega prostega okna za izvedbo zahtevka. Če ta čas ni v prihodnosti to pomeni, da lahko zahtevek izvede takoj.

5 Rezultati

Programsko rešitev smo po implementaciji testirali lokalno. Testi so bili uspešni, vendar je za realne podatke potrebno večjo število zahtevkov. V nadaljevanju smo predstavili nekaj rezultatov strežnika v polni obremenjenosti v produkcijskem okolju podjetja Databox [3].

Za vse zahtevke na naš strežnik smo beležili odzivni čas in s pomočjo brezplačnega orodja Kibana [4] to tudi vizualizirali. Rezultate smo zajeli za obdobje sedmih dni, natančneje od 14. 7. 2022 do vključno 20. 7. 2022. V tem času smo na strežnik naredili približno **311,43 milijonov zahtevkov**, kar je povprečno **44,49 milijonov na dan**.
Zanimivo je tudi razmerje zahtevkov med obema končnima točkama (/delay in /increment). Pričakovano je zahtevkov na končno točko za izračun naslednjega prostega okna za izvedbo zahtevka več. Namreč, če je potrebno z zahtevkom zaradi visoke porabe omejitve počakati dlje časa, smo končno točko za izračun prostega okna poklicali večkrat. Razmerje lahko podrobneje vidimo na sliki 10.



Slika 10: Razmerje zahtevkov med končnima točkama.
Vir: lasten

Prav tako smo raziskali odzivni čas strežnika na posamezni končni točki. Zavedali smo se, da je povprečen odzivni čas v nekaterih primerih slabo merilo delovanja, saj lahko nekaj počasnih zahtevkov v ključnih trenutkih pokvari uporabniško izkušnjo. V ta namen smo z zabeleženimi odzivnimi časi, z uporabo prej omenjenega orodja, izračunali še nekaj percentilnih vrednosti. Izbrali smo 95. in 99. percentil. Rezultati so prikazani v tabeli 2.

Tabela 2: Odzivni časi na posamezni končni točki.

	Povprečje (ms)	95. percentil (ms)	99. percentil (ms)
/delay	4	7	12
/increment	3	7	11

Razlika v odzivnem času je pričakovana, saj je pri izvedbi slednjega potrebno več procesiranja. Za lažje razumevanje smo enega izmed zgornjih podatkov opremili še s kratko razlago. 95. percentil na končni točki za izračun naslednjega prostega okna za izvedbo zahtevka /delay pomeni, da se 95 odstotkov vseh zahtevkov izvede v največ 4 ms. Velika večina zahtevkov se torej izvede zelo hitro.

6 Zaključek

V tem članku smo razdelali nekaj teorije metod omejevanja frekvence zahtevkov in predstavili načrtovanje ter implementacijo tega sistema na odjemalcu. Želeli smo zagotoviti enostavno uporabo in s tem dobro uporabniško izkušnjo uporabnikov in razvijalcev, ki to programsko rešitev uporabljajo. Implementacija v obliki strežnika, na katerega izvajamo zahtevke, omogoča uporabo iz kateregakoli programskega jezika. Programsko rešitev smo preizkusili tudi v realnem produkcijskem okolju in podali performančne rezultate. Percentili, ki smo jih podali, pa so nam pokazali še nekoliko jasnejšo sliko. Naša programska rešitev ima še vedno nekaj prostora za izboljšave. Predvidevamo, da je večina zahtevkov z višjim odzivnim časom povezana s sistemskimi viri na strežniku v času izvedbe zahtevkov, v nekaterih pa se najverjetneje skriva tudi kakšna napaka v naši kodi.

Literatura

[1] Rate-limiting strategies and techniques, https://cloud.google.com/architecture/rate-limiting-strategies-techniques, obiskano 21. 7. 2022.

[2] Guanlan Dai, How to Design a Scalable Rate Limiting Algorithm, https://konghq.com/blog/how-to-design-a-scalable-rate-limiting-algorithm, obiskano 22. 7. 2022.

[3] Databox, https://databox.com, obiskano 29. 7. 2022.

[4] Kibana, https://www.elastic.co/kibana, obiskano 29. 7. 2022.

[5] Aljaž Mislovič, Sistem za omejevanje frekvence zahtevkov na odjemalcu, diplomsko delo, 2021.

Prehod na realno-časovno obdelavo podatkovnih tokov s pomočjo Kafka Streams in KSQL/ksqlDB

Martina Šestak

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

Sinopsis V sodobnih arhitekturah IKT rešitev se pogosto srečamo z obdelavo dogodkov, ki jih proizvajajo poslovni procesi podjetij. Potreba po takojšnji obdelavi posameznih dogodkov oz. podatkovnih tokov (angl. stream), ki jih le-ti tvorijo, namesto paketov (angl. batch) je zadnjih nekaj let v ospredje postavila rešitve za pretočno obdelavo podatkov. Ena izmed pogosto uporabljanih rešitev za pretočno obdelavo podatkov je platforma Apache Kafka, ki ponuja širok nabor orodij in rešitev za realno-časovno obdelavo podatkov. V prispevku bomo podrobneje predstavili platformo Kafka iz vidika realno-časovne pretočne obdelave podatkov. Predstavili bomo razlike med rešitvama Kafka Streams in KSQL/ksqlDB ter njuno umeščenost v ekosistem Kafka oz. v ekosistem IKT rešitev, ki uporabljajo Kafko za dogodkovno orientirano obdelavo podatkov. Izpostavili bomo ključne prednosti Kafke pred klasičnimi sporočilnimi sistemi, kot je npr. RabbitMQ. Pri tem bomo poskusili biti objektivni ter v razpravo vključili tudi morebitne konkurenčne sisteme, kot je Apache Flink s svojo nadgradnjo Flink SQL. S tehničnega vidika pa se bomo predvsem osredotočili na KSQL oz. ksqlDB. Opisali bomo podobnosti in razlike s klasičnim poizvedovanjem po podatkovnih bazah. Dotaknili se bomo tudi scenarijev povezovanja tabel s tokovi ter kompleksnejših primerov in izzivov uporabe KSQL/ksqlDB pri realno-časovni obdelavi podatkov.

Ključne besede:
podatkovni toki
realno-časovna obdelava podatkovnih tokov
pretočna obdelava
Kafka
Kafka Streams
KSQL
ksqlDB

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.10

1 Uvod

Dogodkovno orientirana miselnost procesiranja sveta okrog nas postavlja v ospredje dogodke (angl. event) kot atomarno osnovo obdelave podatkov v modernih IKT rešitvah. Dogodke lahko tako metaforično predstavimo kot posamične atome vode, ki skupaj tvorijo potok, ki ima svoj izvir in po navadi tudi ponor, pri čemer lahko trenuten način obdelave teh dogodkov enačimo z naključnim zajemom vode v dlan in požirkom med sprehodom po gozdu. Platforma Kafka se je že vrsto let nazaj izkazala kot odlična osnova za pretočno obdelavo podatkov, pri čemer lahko le to, s svojo osnovno gručo Kafka, metaforično predstavimo kot vodno strugo.

V trenutnih IKT rešitvah in s tem povezanih arhitekturah IT/IS se že opaža pogostejša uporaba Kafke, vendar z osredotočanjem na razvoj Kafka proizvajalcev (angl. producer) in potrošnikov (angl. consumer) ter uporabo Kafke kot sporočilnega sistema. Prihajamo pa vedno bolj do situacij, kjer želimo dogodke neprekinjeno in v celoti spremljati in ne zgolj sporadično. Na primeru našega potoka, si zahtevo lahko predstavimo kot nenadno zastrupitev vode, kjer je potrebno neprekinjeno in detajlno spremljati celoten tok na določeni točki in iskati morebitne škodljive primesi. Za ta namen je nujno potrebno začeti Kafko uporabljati kot platformo za realno-časovno obdelavo podatkovnih tokov, in sicer z uporabo komponent, kot sta Kafka Streams in/ali KSQL oz. ksqlDB. Prva komponenta ponavadi zahteva višji nivo programerskih veščin, vendar omogoča izvedbo bolj kompleksnih analiz nad podatkovnimi toki. Slednja komponenta pa je bolj prijazna do končnega uporabnika, saj omogoča izvedbo povpraševanj s pomočjo sintakse, ki temelji na poizvedovalnem jeziku SQL in pri uporabniku spodbuja občutek, kot da je v interakciji s klasično podatkovno bazo.

V nadaljevanju bomo predstavili osnovne lastnosti pretočne obdelave podatkov in platforme Kafka. Podrobno bomo predstavili komponenti Kafka Streams in KSQL/ksqlDB ter razlike v njuni uporabi, ki lahko vplivajo na izbiro ene izmed teh dveh rešitev za realno-časovno obdelavo podatkovnih tokov. Slednji komponenti ekosistema Kafka bomo predstavili tudi na praktičnem primeru, s pomočjo katerega bomo potem argumentirali izzive in možnosti njune uporabe.

2 Pretočna obdelava podatkov

Izbira načina obdelave domenskih podatkov direktno vpliva na zmogljivost in možnosti analiz, ki jih lahko izvajamo nad zbranimi podatki. Podjetja, ki želijo imeti vpogled v podatke takoj ko le-tisti pridejo v sistem ker jim hitra analiza omogoča, da iz podatkov izvlečejo največjo vrednost za njihovo poslovanje, se pogosto odločajo za pretočno obdelavo podatkov. Obstajajo pa še vedno primeri "tradicionalnega" pristopa k obdelavi podatkov, kjer se podatki v sistemu analizirajo v množicah, ki jim rečemo "paketi" (angl. batches).

Pretočna obdelava podatkov temelji na tehnologiji "pretakanja" (angl. streaming) podatkov. V tem primeru, sistem, ki omogoča pretakanje podatkov, vsebuje stroj za obdelavo podatkov, ki je načrtovan z neskončnimi nabori podatkov v mislih [1]. Pri paketni obdelavi je stroj za povpraševanje (angl. query engine) naravnan tako, da periodično izvede povpraševanja nad končnim naborom podatkov, ki so v sistemu zabeleženi od časa zadnjega povpraševanja, medtem ko ta stroj neprekinjeno izvaja povpraševanja nad neskončnimi podatkovnimi toki pri pretočni obdelavi.

Kot največja prednost pretočne obdelave podatke se v literaturi izpostavlja možnost obdelave podatkov z zelo nizko zakasnitvijo (angl. latency) [2]. Ravno je zakasnitev najbolj relevantno merilo za določanje zmogljivosti sistema za realno-časovno obdelavo podatkov. Namreč, ponudniki rešitev za pretočno obdelavo podatkov pogosto trdijo, da omogočajo realno-časovno obdelavo podatkov, vendar temu v praksi ni vedno tako. Glede na vpliv zakasnitve na IKT sistem znotraj katerega se izvaja, razlikujemo tri pristopa za realno-časovno obdelavo podatkov [3]:

- Trda realno-časovno obdelava podatkov (angl. hard real-time) – zakasnitev se meri v milisekundah in sistem ima nulto toleranco na zakasnitev, saj so posledice lahko katastrofalne (npr. zavorni sistem v avtomobilu),
- Mehka realno-časovna obdelava podatkov (angl. soft real-time) – zakasnitev se meri v sekundah in nima katastrofalnega učinka na sistem, vendar vpliva na učinkovitost sistema (angl. sistem za trgovanje na borzah), in
- Obdelava podatkov skoraj v realnem času (angl. near real-time) – zakasnitev se meri v minutah in ne povzroča večje škode na delovanje sistema (npr. sistemi za poročanja).

V naslednjem poglavju bomo predstavili platformo Kafka ter analizirali možnosti realno-časovne obdelave podatkov, ki jih ta platforma ponuja.

3 Platforma Kafka

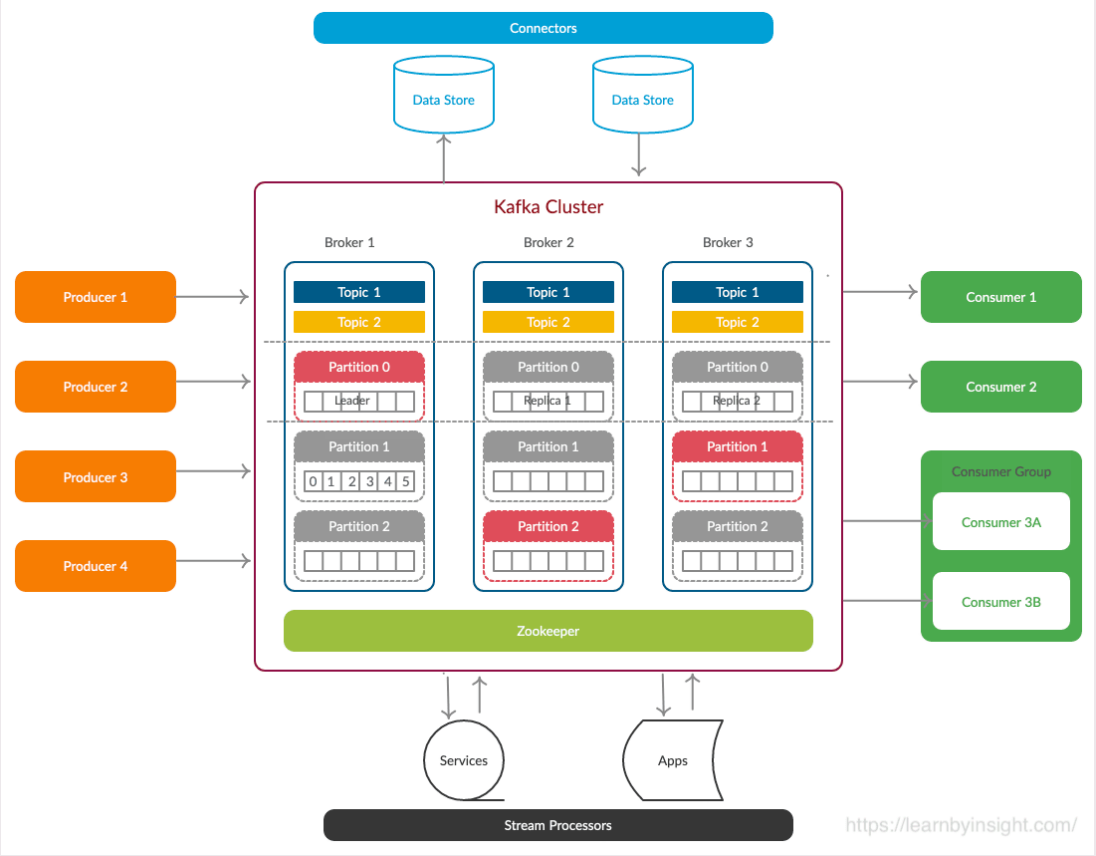
3.1 Osnovni koncepti in arhitektura

V literaturi se pogosto srečamo z definicijo platforme Kafka kot sporočilnega sistema, ki je odgovoren za prenos podatkov iz ene aplikacije v drugo. Danes se to sporočanje s pomočjo Kafke dosti krat izvaja v porazdeljenem okolju, vendar Kafka še zmeraj zagotavlja realno-časovno obdelavo podatkov ob nizkih zakasnitvah in visoki prepustnosti. V nadaljevanju bomo na kratko predstavili arhitekturo platforme Kafka, ki je tudi prikazana na Sliki 1. Sporočila so v Kafki razvrščena v *teme* (angl. topic), ki predstavljajo tok (angl. stream) podatkovnih zapisov, ki so kategorizirane v skup. Vsak tok podatkov je neomejen in kontinuiran pretok podatkov v realnem času. Teme definirajo razvijalci preden ko začnejo pošiljati sporočila v sistem. Najbližja analogija teme je tabela v podatkovnih bazah. Vsaka Kafka tema je razdeljena v *particije* (angl. partition), ki predstavljajo dejansko enoto za shranjevanje podatkovnih zapisov v obliki dnevniške datoteke (angl. log file), medtem ko tema predstavlja le logični koncept v ekosistemu Kafka. Teme so shranjene na strežnikih, ki se imenujejo *posredniki* (angl. brokers) in tvorijo Kafka gručo s katero upravlja Zookeeper in znotraj katere se podatki shranjujejo v porazdeljeni obliki (particije na različnih posrednikih).

Primarna vloga tem v Kafki je zmanjšanje odvisnosti med *proizvajalci* (angl. producers) in *potrošniki* (angl. consumers) [5]. To pomeni, da s tem lahko zagotavljamo, da npr. počasni potrošniki ne vplivajo na proizvajalce, dodajanje potrošnikov ali napake potrošnikov ne vplivata na ekosistem, ali da se potrošniki lahko razširijo brez vpliva na ekosistem. V ozadju, Kafka sporočilni sistem sledi vzorcu *objava-naročnina* (angl. publish-subscribe), pri čemer je ena aplikacija *proizvajalec*, ki pošilja sporočila v določeno Kafka temo. Z druge strani, ko se naroči na to isto Kafka temo, druga aplikacija, ki se ji reče *potrošnik,* sproti sledi prispelim zapisom v to Kafka temo in je v realnem času o tem obveščena. Proizvajalci in potrošniki so instance programske kode, ki jih razvijamo po želji glede na poslovno logiko. Kafka je s stališča proizvajalcev in potrošnikov agnostična, kar pomeni, da je neodvisna od platforme in programskega jezika v katerem se le-ti implementirajo. Za namene prebiranja novih sporočil, potrošniki hranijo zadnji odmik (angl. offset) na podlagi katerega, posredniki vedo, katera nova sporočila potrošnik še ni prebral oz. obdelal. Potrošniki lahko tudi tvorijo potrošniške skupine (angl. consumer groups), ki predstavljajo eno končno aplikacijo/sistem. Ko jih preberejo, potrošniki procesirajo prispele podatkovne zapise kot pare ključ-vrednost (angl. key-value pairs), kar pomeni, da je za namene obdelave potrebno pretvoriti vsaki podatkovni zapis iz polja bajtov (angl. byte array) v par ključ-vrednost.

Funkcionalnosti platforme Kafka danes na trgu ponuja več ponudnikov. Pri tem je večina ponujenih rešitev nastala na podlagi implementacije odprtokodne platforme Apache Kafka in se majhne razlike izmed ponudnikov pojavljajo predvsem pri implementaciji določenih funkcionalnosti platforme zaradi različnih namenov (pretakanje, analiza, infrastruktura ali nekaj tretjega). Razen Apache Kafka, najbolj znane variante različnih ponudnikov platforme Kafka so Confluent Kafka, Cloudera Hortonworks Kafka, Red Hat Kafka in Amazon MSK. V tem

prispevku bomo praktične primere implementirali s pomočjo platforme *Confluent Kafka*, ki se osredotoča na pretakanje dogodkov in vključuje dodatne možnosti za shrambo in obdelavo podatkov.

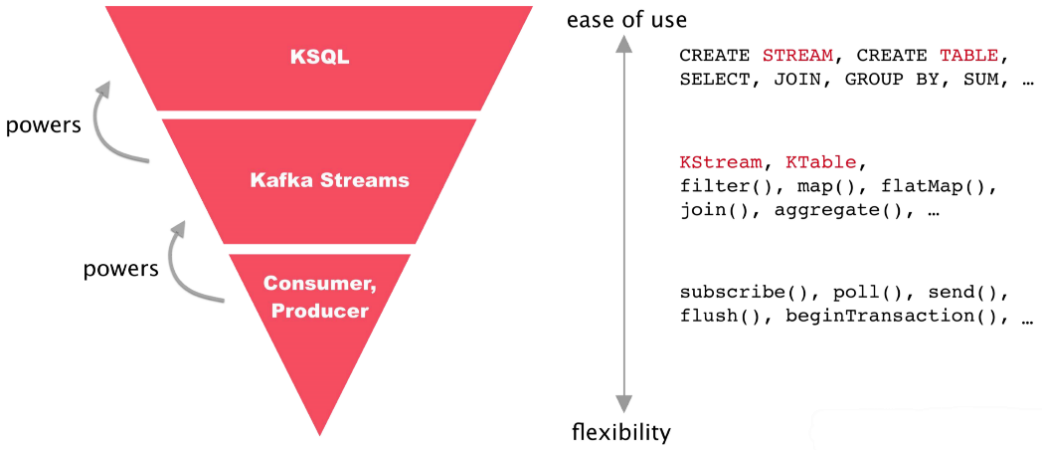


Slika 1: Arhitektura platforme Kafka.
Vir: [4].

V osnovi, platforma Kafka vključuje pet osrednjih programskih vmesnikov, ki omogočajo enostavnejšo upravljanje in uporabo Kafka ekosistema. Obdelava podatkov v realnem času, in sicer neprekinjeno, sočasno in na način, da se obdeluje vsak posamezen podatkovni zapis (dogodek), se izvaja s pomočjo *Kafka Streams API* ali s pomočjo *KSQL/ksqlDB*. Pri tem realno-časovna obdelava običajno vključuje branje neprekinjeno prihajajočih podatkovnih zapisov določenega toka/teme, izvajanje nekaterih analiz ali preoblikovanja podatkov in nato zapisovanje rezultatov tega v drug ali nov tok/temo.

Za izvedbo realno-časovne obdelave s platformo Kafka lahko uberemo več načinov:

- Klasičen način – vključuje razvoj potrošnika, ki podatke obdela in jih nato s pomočjo proizvajalca pošiljamo naprej v nov tok. Takšen način je zelo zahteven, saj je za namene stanovitne obdelave potrebno veliko namenskega procesiranja.
- Način kjer uporabimo polnopravni okvir za obdelavo realno-časovnih tokov (angl. full-fledged stream processing framework), kot so Spark Streaming, Flink, Storm itd.
- Domorodni način za Kafko - vključuje uporabo Kafka Streams API ali KSQL.



Slika 2: Primerjava med različnimi možnostmi platforme Kafka za realno-časovno obdelavo podatkov. Vir: [6].

V odvisnosti od poslovnih potreb naše aplikacije lahko uporabimo bodisi preproste Kafka programske elemente, kot so proizvajalci in potrošniki, ali pa preidemo na uporabo Streams API ali KSQL, ki nam na različnih nivojih abstrkacije pomagajo pri realno-časovni obdelavi podatkov (Slika 2). V nadaljevanju bomo prikazali domorodni način za realno-časovno obdelavo, in sicer z uporabo Kafka Streams API-ja ter KSQL/ksqlDB.

3.2 Kafka Streams

Kafka Streams je osrednja komponenta, ki omogoča realno-časovno obdelavo pretakajočih se podatkov. Gre se za odjemalsko knjižnico za gradnjo aplikacij in mikrostoritev, kjer so vhodni in izhodni podatki shranjeni v Kafka posrednikih. Takšne programske nadgradnje omogočajo višji nivo abstrakcije ter transformacijo in obogatenje podatkov med pretakanjem. Apache Kafka Streams se kot knjižnica izvaja v sklopu aplikacije in ne znotraj posrednikov.

Med ostalim, Apache Kafka Streams omogoča [7]:

- Procesiranje vsakega posameznega podatkovnega zapisa med pretakanjem z milisekundno zakasnitvijo,
- Grupiranje toka po ključu,
- Združevanje več tokov,
- Neprekinjeno transformacijo,
- Nestanovitno (angl. stateless) in stanovitno (angl. stateful) obdelavo glede na to, ali se nov podatkovni tok procesira neodvisno od prejšnjih tokov ali ne,
- Filtriranje, mapiranje, združevanja, agregacije podatkov,
- Časovno obdelavo (angl. windowing).

Znotraj Kafka Streams aplikacije za obdelavo se vsak korak obdelave izvaja po določenem vrstnem redu, ki sestavlja *topologijo* procesorja. Topologija procesorja definira logiko obdelave podatkov, ki jo mora izvesti aplikacija za obdelavo tokov (angl. streams application). V topologiji procesorja pretoka obstaja vozlišče, ki ga imenujemo procesor toka, ki predstavlja korak obdelave za pretvorbo podatkov v tokove s sprejemanjem enega vhodnega podatkovnega zapisa od svojih procesorjev v topologiji navzgor. Prav tako naknadno izdela enega ali več izhodnih zapisov za svoje nadaljnje procesorje.

Kafka Streams API ponuja podporo obdelave tokov kakor tabel, in sicer s pomočjo svojih osnovnih abstrakcij *KStream* in *KTable*. KStream je objekt, ki se uporablja za abstrakcijo toka (teme), s katerim lahko izvajamo določene

operacije, kot sta filtriranje in mapiranje (nestanovitni operaciji). Uporabljamo ga, kadar na dogodke v toku gledamo kot neodvisna nespremenljiva dejstva, ki se konstantno posodabljajo (npr. beleženje dogodkov kot so gol, podaja, kot na tekmi). Dogodki se konstantno dogajajo, pri čemer nov dogodek ne prepiše prejšnjega. KTable je abstrakcija toka dnevnika sprememb, kjer vsak podatkovni zapis predstavlja posodobitev. Natančneje, vrednost v podatkovnem zapisu se razlaga kot posodobitev "UPDATE" zadnje vrednosti za isti ključ zapisa, če ustrezen ključ še ne obstaja, se posodobitev šteje kot vnos "INSERT". V primerjavi s tokom dogodkov tabela predstavlja stanje sveta v določenem trenutku, običajno najbolj sveže stanje (npr. statistika nogometne tekme). Tabela je pogled toka dogodkov in ta pogled se nenehno posodablja, ko je zajet nov dogodek. Nad KTable objektom izvajamo določene operacije, kot sta združevanje (angl. join) in agregiranje (stanovitni operaciji), pri čemer ustvarjamo t. i. obogatene tokove (angl. enriched streams).

3.3 KSQL/ksqlDB

KSQL je zgrajen na Kafka Streams in predstavlja robusten okvir za obdelavo tokov, ki so del Kafke. Ponuja nam poizvedbeni sloj za izdelavo aplikacij za pretakanje dogodkov na Kafkine teme. Medtem ko Kafka Streams omogoča pisanje nekaterih zapletenih topologij in za to zahteva nekaj bistvenega znanja programiranja, želi KSQL to zapletenost abstrahirati tako, da nam zagotovi vsem znano semantiko SQL, pri čemer ne smemo pozabiti, da ne govorimo o paketnem SQL-u, temveč pretočnem SQL-u, tj. izvedbi povpraševanj med pretakanjem podatkov. Podobno kot Streams API, KSQL omogoča višji nivo abstrakcije ter transformacijo in obogatenje podatkov med pretakanjem. Kot naslednik KSQL-a je nastal ksqlDB in se danes ta dva koncepta oz. rešitvi pogosto uporabljata izmenično.

KSQL oz. ksqlDB lahko uporabljamo za številne operacije nad tokovi, kot so [8]:

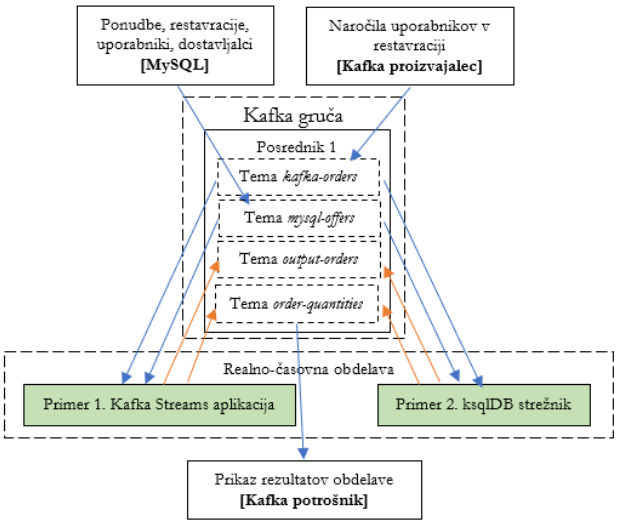
- Raziskovanje podatkov shranjenih v Kafka temi,
- Pretočni ETL (angl. Extract-Transform-Load),
- Zaznavanje anomalij,
- Realno-časovno spremljanje dogodkov.

V razliko od Streams API, ki se izvaja v sklopu naše aplikacije, se ksqlDB izvaja kot podatkovna baza, ki se paralelno namesti ob Kafka gruči in je z njo v tesni povezavi. Ker se ksqlDB samodejno izvaja na platformi Apache Kafka, je potrebna torej prvo namestitev Kafke, ki je konfigurirana za uporabo ksqlDB.

4 Realno-časovna obdelava podatkovnih tokov na praktičnem primeru

4.1 Opis primera

V praktičnem primeru smo vzpostavili arhitekturo s pomočjo platforme Kafka, ki omogoča neprekinjeno branje naročil v restavracijah, ki jo preko spletne strani lahko oddajo uporabniki (Slika 3). Naročila kot podatkovne toke vsakih deset sekund v temo *kafka-orders* pošilja Kafka proizvajalec implementiran v programskem jeziku Java, medtem ko se s pomočjo knjižnice Kafka Connect v teme s prefiksom *mysql-* (npr. *mysql-offers*) uvezejo domenski podatki o ponudbah, uporabnikih, restavracijah, dostavljalcih, ki so shranjeni v bazi MySQL. Naslednji korak je združena obdelava domenskih podatkov in podatkovnih tokov ter prikaz rezultatov. Za prvo varianto obdelave smo implementirali zasebno Kafka Streams aplikacijo, pri drugi pa je za ta namen izvedeno nekaj povpraševanj nad ksqlDB bazo s pomočjo sintakse KSQL. Za prikaz rezultatov smo razvili preprostega Kafka potrošnika.



Slika 3: Arhitektura praktičnega primera za realno-časovno obdelavo naročil v restavracijah.
Vir: lasten.

4.2 Primer uporabe Kafka Streams

Kot rečeno v predhodnem poglavju je za realno-časovno obdelavo s pomočjo Kafka Streams potrebno implementirati aplikacijo, ki jo zaženemo v sistemu. Znotraj aplikacije implementirati Streams procesorja, ki bere podatkovne toke iz ustreznih Kafka tem, jih shrani kot objekte KStream/KTable odvisno od potrebnih analiz, procesira podatke in rezultate ponovno shrani v ciljno Kafka temo. Pogoj za uspešno delovanje je, da so Kafka teme, ki se uporabljajo, predhodno ustvarjene na posrednikih. V našem primeru Streams procesor izvaja topologijo, ki je sestavljena iz dveh podtopologij znotraj katerih se izvajajo določeni koraki obdelave. Izsek programske kode za vsako podtopologijo je prikazan na Sliki 4. Prva podtopologija prebere podatkovne zapise iz teme *kafka-orders*, jih shrani kot objekt KStream (ker želimo zgodovinsko beležiti vsa prispela naročila), pri čemer bo ključ tega objekta identifikator uporabnika (tipa *Integer*), zabeležena vrednost pa predstavlja številko naročila (tipa *Order*). V naslednjem koraku topologije procesorsko vozlišče grupira posamezna naročila po uporabnikih, jih potem prešteje in v izhodno temo *output-orders* shrani število oddanih naročil za posamezne uporabnike. V drugi podtopologiji pa se seštevek količine naročenih izdelkov v vsakem naročilu shrani v temo *order-quantities*.

```
KStream<Integer, Order> inputStream = streamsBuilder.stream(props.getProperty("input.topic.name"), Consumed.with(Serdes.Integer(), serdes));

inputStream.map((k,v)->new KeyValue<>(v.getUser_id().intValue(),v.getOrder_no()))
        .groupByKey(Grouped.with(Serdes.Integer(), Serdes.String())).count().toStream().mapValues(value -> value.toString())
        .to( s: "output-orders", Produced.with(Serdes.Integer(), Serdes.String()));
inputStream
    .map((k,v)->new KeyValue<>(v.getOrder_no(),v.getItems())) KStream<String, Map<String, Integer>>
    .flatMapValues(value -> Arrays.asList(value.get("quantity"))) KStream<String, Integer>
    .groupByKey(Grouped.with(Serdes.String(),Serdes.Integer())) KGroupedStream<String, Integer>
    .reduce(Integer::sum) KTable<String, Integer>
    .toStream() KStream<String, Integer>
    .mapValues(v -> v.toString()) KStream<String, String>
    .to( s: "order-quantities", Produced.with(Serdes.String(), Serdes.String()));
```

Slika 4: Izsek programske kode Kafka Streams aplikacije.
Vir: lasten.

4.3 Primer uporabe KSQL/ksqlDB

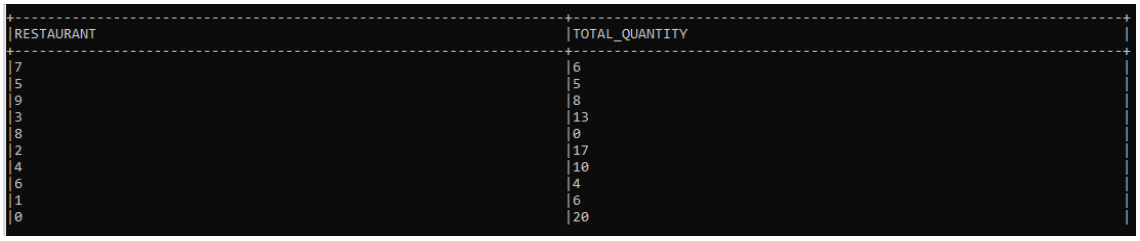
Pri KSQL/ksqlDB načinu obdelave ne potrebujemo programsko implementirati aplikacije, ampak se samo povežemo na ksqlDB strežnik. Podobno kot pri klasičnih podatkovnih bazah je tukaj prvi korak pripraviti objekte v katere bomo brali zapise iz podatkovnih tokov. Ustvarimo lahko podatkovni tok (z ukazom CREATE STREAM) ali tabelo (z ukazom CREATE TABLE). Obadva objekta sta analogija objektom KStream oz. KTable pri uporabi

Kafks Streams in za vsakega obstaja nabor primerov, ko je njegova uporaba najbolj smiselna. Sintaksa KSQL je zelo podobna sintaksi SQL-a, pri čemer je edino pomembno upoštevati podatkovno strukturo zapisov v izhodni Kafka temi, da bi jih pravilno prebrali. Kot primer lahko ustvarimo podatkovni tok za branje vseh naročil iz teme *kafka-orders* s pomočjo ukaza kjer vir podatkov opredelimo s ključno besedo *WITH*:

create stream kafka_sorders (order_no varchar, date bigint, rest_id integer, user_id integer, courier_id integer, item_id integer, quantity integer) with (kafka_topic='kafka-orders', key_format='kafka', value_format='avro');

Ko smo ustvarili podatkovni tok *kafka_sorders*, iz njega lahko izpišemo trenutne zapise s pomočjo ukaza SELECT, vendar se bo tisti izpis neprekinjeno posodabljal z vsako novo vrstico oz. objektom, ki ga preberemo iz Kafka teme. Kot naslednji korak lahko podatke shranjene v toku *kafka_sorders* ponovno združimo z domenskimi podatki o restavracijah (za katere smo predhodno tudi ustvarili podatkovni tok *kafka_srestaurants*, ki bere zapise iz teme *mysql-restaurants*) in v realnem času računamo seštevek količine naročenih izdelkov za posamezno restavracijo.

Za ta namen ustvarimo ksqlDB tabelo, kam bomo zabeležili edino zadnje stanje naročenih količin za posamezno restavracijo, pri čemer bo posamezna vrstica v tabeli rezultat združevanja tokov *kafka_sorders* in *kafka_srestaurants* po stolpcu *rest_id* v določenem časovnem okvirju (zadnja ura, mesec, ipd.). Kot rezultat obdelave se potem izpiše trenutni seštevek naročenih količin v posamezni restavraciji (Slika 5), vendar se tudi ta vpogled spreminja v realnem času, glede na to prispela naročila z novimi količinami izdelkov.



RESTAURANT	TOTAL_QUANTITY
7	6
5	5
9	8
3	13
8	0
2	17
4	10
6	4
1	6
0	20

Slika 5: Prikaz obdelave naročenih količin v posameznih restavracijah s pomočjo KSQL/ksqlDB. Vir: lasten.

5 Diskusija

V primerjavi s klasičnimi sporočilnimi sistemi, kot je RabbitMQ, je Kafka veliko bolj učinkovita pri uporabi sistemskih resursov. Kafka temelji na t. i. *povleci* (angl. pull) modelu sporočanja, pri katerem posredniki pošljejo potrošnikom nova sporočila šele na njihovo zahtevo in je zaradi tega možen scenarij, da določeni potrošnik v sistemu ne razpolaga z najnovejšimi podatki. Z uporabo *povleci* modela in razvrščanja sporočil (angl. message ordering) Kafka lahko doseže boljšo prepustnost v sistemu, saj je manjša verjetnost pojave zastojev pri prejemanju sporočil. V razliko od *povleci* modela pri Kafki, RabbitMQ in večina podobnih sporočilnih sistemov temeljijo na t. i. *potisni* (angl. push) modelu sporočanja, pri katerem posredniki obvestijo naročene potrošnike o novem sporočilu, takoj ko ga proizvajalec pošlje v sistem, s čimer se lahko doseže večja hitrost pri pošiljanju sporočil. Kafka je zaradi podpore za atomarnost pošiljanja in branja vsakega sporočila priporočena rešitev za podjetja, kjer se že uporabljajo relacijske podatkovne baze in je pomembno zagotavljati konsistentnost podatkov.

S pomočjo komponente Kafka Streams, platforma Kafka omogoča razvoj mikrostoritev, s čimer se poveča prilagodljivost sistema za različne scenarije in analize podatkov, ki jih uporabniki potrebujejo. Namen te komponente je kombinirati možnost realno-časovne obdelave podatkovnih tokov v rangu milisekund čez izvajanje klasičnih Java/Scala aplikacij znotraj Kafka gruče. Kot največji izziv uporabe Kafka Streams, uporabniki poudarjajo potrebo po dobrem (celo naprednem) poznavanju programiranja zaradi razvoja Streams aplikacije. Uporaba knjižnice Kafka Streams za programerje pomeni, da lahko obdelavo podatkovnih tokov implementirajo v obliki programske kode, ki se izvaja znotraj že obstoječe aplikacije in se izvaja neposredno znotraj Kafka gruče, kar

poenostavlja interakcijo v sistemu. Kar se tiče ponujenih možnosti pri obdelavi pa Kafka Streams podpira nestanovitno in stanovitno obdelavo podatkovnih tokov, kar ustvarja možnost za napredne analize in agregacije nad podatki. Omogoča tudi izvedbo časovne obdelave podatkovnih tokov, kar je danes zelo pomembno za množična podjetja, pri katerih je časovna komponenta dogodkov, ki jih beležijo, podlaga za poslovne odločitve (npr. v domeni Industrije 4.0/5.0).

KSQL oz. ksqlDB sta nastali z namenom izboljšanja uporabniške izkušnje pri obdelavi podatkovnih tokov, ki pridejo v sistem. Sintaksa KSQL/ksqlDB temelji na klasični sintaksi jezika SQL z nekaj prilagoditev, ki so potrebne zaradi pretočne obdelave podatkov. Čeprav je veliko manj zahtevna za zagon v Kafka gruči, komponenta KSQL/ksqlDB zahteva določanje sheme podatkovnega toka preden začne brati zapise iz toka. Slednja lastnost spominja na zmanjšano prilagodljivost pri spremembah v shemi podatkovnih zapisov, ki že vrsto let predstavlja eno izmed glavnih pomanjkljivosti relacijskih podatkovnih baz. Namreč, v današnjih poslovnih okoljih lahko prihaja do spremembe v strukturi podatkov, ki jih je potrebno obdelati in shraniti v sistemu. Vsaka takšna sprememba v shemi podatkovnega toka zahteva ponovno ustvarjanje novih podatkovnih struktur v ksqlDB, saj drugače lahko pride do neujemanja starih in novih zapisov ter posledično napak pri izvajanju povpraševanj.

Platforma Kafka je danes postala sopomenka za pretakanje podatkov in realno-časovno obdelavo podatkovnih tokov, vendar ne gre za univerzalno tehnološko rešitev, ki je najbolj primerna za uporabo v vseh scenarijih, saj tudi Kafka ima svoje slabosti in ne podpira vsega, kar je danes podjetjem potrebno. Na trgu tehnologij za obdelavo podatkovnih tokov se uporabljajo tudi rešitve, kot so Flink (podpira tudi paketno obdelavo), Storm (izjemno hiter za preproste scenarije) ter Spark Streaming (podpira lambda arhitekturo in zagotavja odpornost na izpade). V primerjavi z naštetimi rešitvami sta komponenti Kafka Streams in KSQL/ksqlDB nekaj novejše, bolj vezane za uporabo Kafke v sistemu in bolj primerne za enostavnejše scenarije, vendar kažejo prednosti Kafke v obdelavi podatkovnih tokov in omogočajo razvoj mikrostoritev za obdelavo podatkovnih tokov v obstoječih IKT sistemih.

6 Zaključek

V prispevku smo predstavili razliko med pretočno in paketno obdelavo podatkov, ki danes postaja vedno bolj očitna in pomembna pri izbiri ustreznih tehnoloških rešitev. Predstavili smo platformo Kafka za pretočno obdelavo podatkov ter osnovne komponente arhitekture platforme. Osredotočili smo se na proces obdelave podatkov v realnem času, ki ga ta platforma zagotavlja čez več načinov. Za namen preverjanja uporabnosti v realnem okolju smo izbrali obdelavo podatkov s pomočjo dveh rešitev, in sicer knjižnice Kafka Streams ter KSQL oz. ksqlDB. Razliko v pristopu pri obdelavi ter kompleksnost obdelave s posamezno rešitvijo smo prikazali tudi na praktičnem primeru realno-časovne obdelave naročil v restavracijah. Prikazali smo, da, čeprav obedve rešitvi uporabljata podobne podatkovne strukture pri obdelavi podatkov (podatkovni toki ali tabele), možnost obdelave z uporabo sintakse, ki temelji na poizvedovalnem jeziku SQL, pri rešitvi KSQL/ksqlDB bistveno olajša celoten proces za končnega uporabnika, vendar je nabor možnih analiz bolj omejen kot pri Kafka Streams. V prihodnosti načrtujemo izvedbo celovite primerjave učinkovitosti obeh rešitev za realno-časovno obdelavo ter vzpostavljanje ustreznih meril za primerjavo različnih rešitev na trgu, ki se trenutno uporabljajo za ta namen.

Literatura

[1] AKIDAU, Tyler, CHERNYAK, Slava, LAX, Reuven, Streaming systems: the what, where, when, and how of large-scale data processing, O'Reilly Media, Inc., 2018.

[2] FRIEDMAN, Ellen, TZOUMAS, Kostas, Introduction to Apache Flink: stream processing for real time and beyond, O'Reilly Media, Inc., 2016.

[3] What is the difference between real-time and streaming analytics?, https://www.bizdata.com.au/blogpost.php?p=real-time-vs-streaming-analytics, obiskano 1.8.2022.

[4] Kafka Clusters Architecture 101: A Comprehensive Guide, https://hevodata.com/learn/kafka-clusters/, obiskano 27. 7. 2022.

[5] ESTRADA, Raul, Apache Kafka 1.0 Cookbook: Over 100 practical recipes on using distributed enterprise messaging to handle real-time data, Packt Publishing Ltd., 2017.

[6] KSQL and Kafka Streams, https://docs.confluent.io/5.4.2/ksql/docs/concepts/ksql-and-kafka-streams.html, obiskano 27. 7. 2022.

[7] Kafka Streams – Core concepts, https://kafka.apache.org/32/documentation/streams/core-concepts, obiskano 2. 8. 2022.

[8] Introducing KSQL: Streaming SQL for Apache Kafka, https://www.confluent.io/blog/ksql-streaming-sql-for-apache-kafka/, obiskano 2. 8. 2022.

Razvoj spletnih rešitev na osnovi ogrodja Angular z uporabo mikro čelnih zaledij

Alen Granda[1], Matic Strajnšak[2]

[1] Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
alen.granda@student.um.si
[2] Alcad d.o.o., Zgornja Bistrica, Slovenija
matic.strajnsak@alcad.si

Sinopsis V današnjih časih so moderne spletne rešitve vedno bolj kompleksne. Posledično jih monolitno usmerjena arhitektura ne more več obvladovati, razvoj postane težaven, vzdrževanje pa nevzdržno. Prispevek predstavlja arhitekturni pristop mikro čelnih zaledij, ki rešuje opisano težavo. Pristop prinaša obetavne ugodnosti napram monolitni arhitekturi. Kljub temu se moramo vprašati, ali je naša aplikacija primerna za izbiro mikro čelnega zaledja. Ob nepravilni izbiri lahko koncept pripelje več negativnih posledic kot pozitivnih. Zato je v prispevku izpostavljeno, v katerih primerih je uporaba omenjene arhitekture primerna in opišemo njene pozitivne ter negativne lastnosti. Nato so opisane implementacijske podrobnosti v spletnem ogrodju Angular ter je predstavljen primer uporabe v praksi. Dodani so še koristni napotki ter možne izboljšave predstavljenega demonstracijskega primera v spletnem ogrodju Angular. Nazadnje smo izpostavili funkcionalnost samostojnih komponent spletnega ogrodja Angular, katerih namen je nadomestiti datoteke nalaganja modulov. Opisali smo razlike napram trenutni uporabi datotek nalaganja modulov ter prikazali primer izvedbe v praksi.

Ključne besede:
mikro čelna zaledja
Angular
Webpack 5
samostojna komponenta

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.11

1 Uvod

Razvoj spletnih aplikacij je skozi zgodovino prehodil mnogo različnih arhitekturnih konceptov in je dandanes eden izmed bolj priljubljenih pristopov za namen komunikacije s končnim uporabnikom. Tradicionalne rešitve iz preteklosti ponujajo monoliten razvoj spletnih rešitev zaradi njihove enostavnosti, nizkih realnočasovnih zahtev ter majhnega števila uporabnikov [11]. Kljub temu je uporaba omenjenega pristopa v današnjih časih, za katerega so značilni kompleksni sistemi z velikimi količinami podatkov in odjemalcev, neuporabna. Uporabnikov je vedno več, so vedno bolj zahtevni in uspešni v iskanju napak v aplikacijah. Zaradi tega so spletne aplikacije z vsakim popravkom ali dodano funkcionalnostjo večje. Posledično je vzdrževanje takšnih aplikacij zahtevno in izpostavljeno akumuliranju nepravilnosti v programski kodi. V ta namen se je že konec leta 2016 začelo govoriti o mikro čelnih zaledjih (angl. microfrontends) [1], ki razširijo koncept mikro storitev v svet čelnih zaledij spletnih aplikacij. Arhitektura razdeli bogato monolitno spletno aplikacijo na več manjših, obvladljivih aplikacij, katere lahko več razvojnih ekip neodvisno druga od druge razvijajo. Pristop mikro čelnih zaledij je postal uveljavljen način razvoja spletnih aplikacij med več podjetji, kot so: DAZN, Ikea, New Relic, SAP, Springer, Starbucks, Zalando ter mnogo ostalih [10].

V nadaljevanju bolj podrobno opišemo princip mikro čelnih zaledij, njihovih prednosti in slabosti ter predstavimo implementacijo arhitekture v spletnem ogrodju Angular. Za namen demonstracije smo ustvarili projekt, ki je sestavljen iz več mikro aplikacij ter lupine, ki je zadolžena za nalaganje mikro aplikacij na zahtevo. Mikro aplikacije smo namenoma razvili z različnimi verzijami, da smo lahko demonstrirali, kako lupina pouporabi enake verzije oziroma jih naloži ob prvem zahtevku. Obrazložili smo postopek konfiguracije celotnega projekta, njegove strukture ter zagona. Podali smo še nekaj ključnih informacij ob razvoju ter možnosti za izboljšave.

Prav tako smo predstavili vidik samostojnih komponent v spletnem ogrodju Angular. Samostojna komponenta služi nadomestitvi razredom nalaganja modulov, ki so okrašeni z dekoratorjem @NgModule. Ti razredi so namreč namenjeni le prevajalniku, ki z njihovo pomočjo poskrbi za uvoz, izvoz, deklaracijo in souporabo delov aplikacije. Samostojne komponente smo integrirali v demonstracijski projekt mikro čelnih zaledij, kjer se nam je to zdelo smiselno. Prav tako smo podali lastno mnenje glede integracije samostojnih komponent v obstoječe aplikacije čelnih zaledij.

2 Mikro čelna zaledja

2.1 Korelacija z mikro storitvami

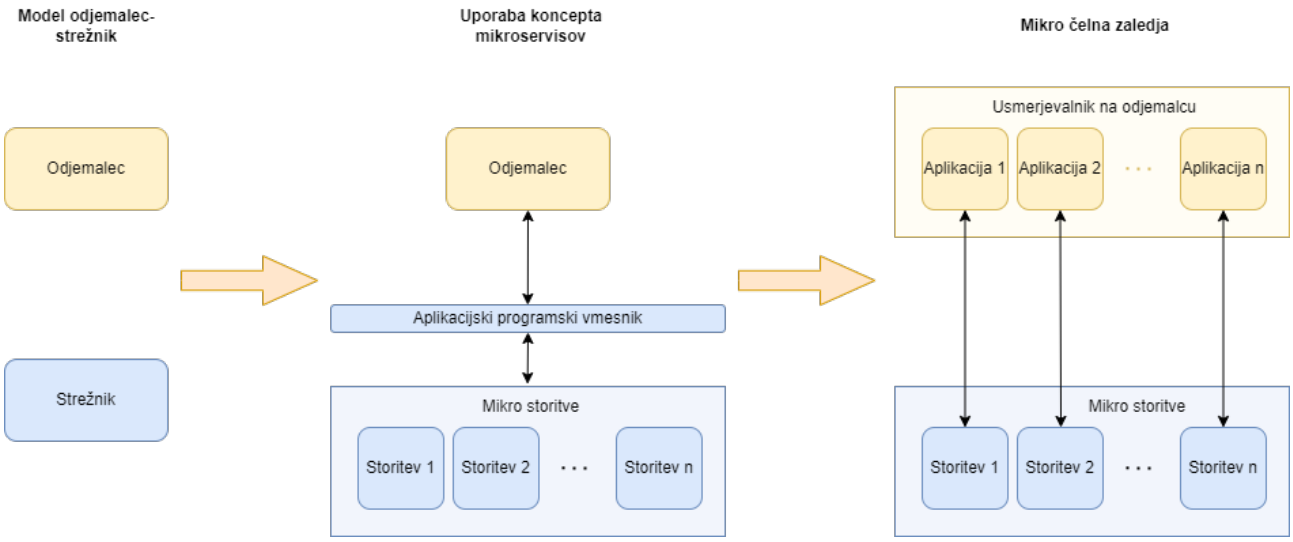
Pričetek arhitekture spletnih rešitev sega vse do modela odjemalec-strežnik [12]. Strežnik je namenjen ponujanju virov, ki jih odjemalec občasno zahteva čez internet. Koncept je zelo razširjen in je bil povod v pristop načrtovanja programske opreme, imenovan model-pogled-krmilnik (angl. Model-View-Controller - MVC) [12]. MVC razdeli aplikacijo na tri dele:

- Model: namenjen opisu podatkov, njihovemu procesiranju in implementaciji poslovne logike.
- Pogled: uporabniški vmesnik aplikacije.
- Krmilnik: skrbi za interakcijo med uporabnikom in izgledom ter posreduje akcije modelu.

Zaradi več uporabnih vrednosti se je MVC izkazal kot izredno uporaben arhitekturni načrtovalski vzorec ter je posledično še danes eden izmed bolj popularnih načinov razvoja spletnih aplikacij. Kljub temu je arhitektura zastavljena monolitno, kar pomeni, da je celotna aplikacija oblikovana kot skupek modulov, ki imajo medsebojne odvisnosti. S povečevanjem zahtev se moduli širijo in množijo. Posledično je meja med njimi vedno težje definirana, vzdrževanje zahtevno in možnost napak v programski kodi naenkrat začne rasti.

Na področju zalednih sistemov so omenjen zaplet pričeli reševati s pristopom mikro storitev okoli leta 2012 [12]. Mikro storitve temeljijo na storitveno usmerjenem arhitekturnem vzorcu (angl. Service-Oriented Architecture) in so predstavljene kot zbirka ohlapno povezanih storitev [11]. Razbijejo kompleksne zaledne aplikacije v manjše, obvladljive storitve, ki komunicirajo skozi zbirko jezikovno neodvisnih aplikacijskih programskih vmesnikov. Vsako storitev je mogoče neodvisno razvijati, testirati in namestiti, kar omogoča uporabo načela enotne odgovornosti (angl. Single-responsibility principle). Posledično je zaradi majhnosti razvoj lažji, razširljiv ter omogoča koncept neprekinjene dobave in namestitve (angl. Continuous delivery and deployment), saj s spremembo ene mikro storitve ne vplivamo na druge.

Prednosti uporabe mikro storitev na zalednem sistemu so očitne. Zato je pred kratkim v veljavo pristopil podoben koncept za čelni del spletnih rešitev, imenovan mikro čelna zaledja. Mikro čelna zaledja rešujejo težave monolitnih aplikacij tako, da razdelijo aplikacijo na več manjših, neodvisnih mikro aplikacij. Ponujajo torej prikaz spletne aplikacije kot kombinacijo funkcionalnosti, ki so pod nadzorom več različnih razvojnih ekip [11]. Glavna aplikacija čelnega zaledja deluje kot usmerjevalnik, ki lahko dinamično nalaga mikro čelne aplikacije. Slika 1 vizualno prikazuje razvoj arhitekture spletnih aplikacij skozi čas ter idejo mikro čelnih zaledij spletnih aplikacij.



Slika 1: Razvoj arhitektur spletnih aplikacij.
Vir: lasten.

Ključna ideja in doprinosi mikro čelnih zaledij spletnih aplikacij so zajeti v sledečih pogledih [10]:

- Neodvisnost ekip ob izbiri tehnologije za razvoj,
- Avtonomne in večnamenske ekipe. Vsaka ekipa ima namreč možnost razvoja dela spletne aplikacije, od mikro čelnega aplikacije do mikro storitve.
- Odpornost mikro čelnih aplikacij na izpade ostalih mikro čelnih aplikacij.
- Razširljivost. Majhne aplikacije je namreč lažje dodelati in razširiti.
- Takojšnje nameščanje novih verzij mikro čelnih aplikacij zaradi njihove majhnosti in samostojnosti.
- Enostavno testiranje in integracija novih spletnih ogrodij v arhitekturo mikro čelnih zaledij.

Kljub temu arhitekturni stil s seboj pripelje razne pomisleke:

- Zvišana količina podatkov, potrebnih za prenos. Čez omrežje moramo namreč prenesti več različnih spletnih ogrodij, kar lahko vpliva na čas nalaganja in končno uporabniško izkušnjo.
- Podvajanje programske kode zaradi neodvisnega razvoja.
- Upravljanje odvisnosti med programskimi knjižnicami mikro čelnih aplikacij ter zagotavljanje enkratnega nalaganja programske knjižnice, ki je zahtevana s strani več mikro čelnih aplikacij.

- Upravljanje med različnimi verzijami naloženih programskih knjižnic.
- Konsistentnost v izgledu aplikacije.
- Razhroščevanje in upravljanje aplikacije.
- Dodana kompleksnost.
- Časovna potratnost zaradi morebitnega podvajanja iskanja rešitve za problem, ki ga je v preteklosti že razrešila druga ekipa.
- Nevarnost objave napak v produkcijsko aplikacijo zaradi takojšnjega razvoja in namestitve.
- Orkestracija in obravnava nalaganja mikro čelnih aplikacij, kadar so zahtevane.
- Usmerjanje mikro čelnih aplikacij.
- Komunikacija med mikro čelnimi aplikacijami.

Z omenjenimi pozitivnimi in negativnimi vplivi uporabe pristopa mikro čelnih zaledij v mislih podajmo nekaj primerov, kdaj arhitekturni načrtovalski vzorec ponuja več neugodnih učinkov kot spodbudnih:

- Aplikacija čelnega zaledja vsebuje veliko komponent, ki so med seboj povezane. V tem primeru moramo poskrbeti za dodaten sistem komunikacije med komponentami mikro čelnih zaledij spletne rešitve, kar dodatno poveča zapletenost in oteži razhroščevanje celotne aplikacije. Zato je arhitekturni načrtovalski vzorec primeren zlasti za aplikacije z zelo malo povezanimi ali z nepovezanimi komponentami.

- Spletna programska rešitev je namenjena izvajanju manjšega števila opravil. Delitev programske rešitve na mikro čelna zaledja bi po nepotrebnem pripeljala kompleksnost ter posledično povišano možnost nabiranja nepravilnosti v programski kodi. Dodatno se podaljša čas nalaganja aplikacije in naraste zahteva po količini potrebnega pomnilnika v primeru nekompatibilnih verzij programskih knjižnic ali več različnih spletnih ogrodij.

- Potreba po avtorizaciji in avtentikaciji uporabnikov oziroma uporaba koncepta enotne prijave (angl. Single sign-on). Gre za poseben primer relacije med komponentami aplikacije, za katerega moramo pomisliti, kako zagotoviti zavedanje mikro čelnih zaledij spletnih aplikacij o uporabnikih, brez da se mora uporabnik prijaviti ob vsakem nalaganju mikro čelne aplikacije. Skupnost predlaga uporabo t.i. strategije mono repozitorijev (angl. monorepo) [6], ki s seboj prinesejo že omenjene posledice v kombinaciji z mikro čelnimi zaledji.

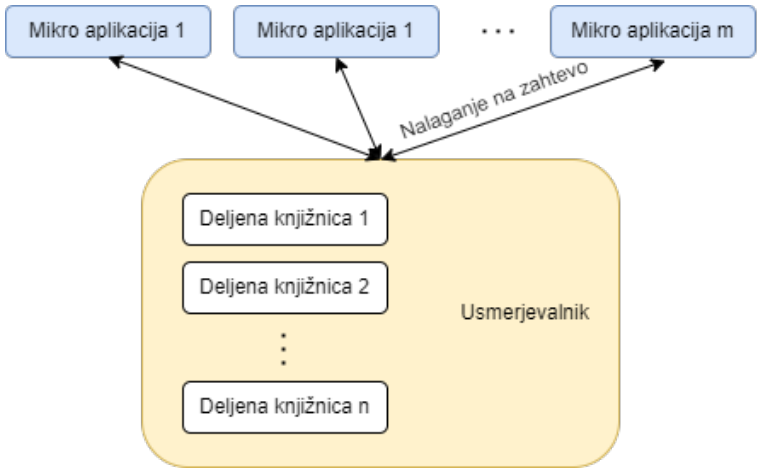
V nadaljevanju predstavimo predloge, kako zaobiti določenim slabostim mikro čelnih zaledij v spletnem ogrodju Angular. Pred tem poglejmo, kaj je implementacijska rešitev uporabe koncepta mikro čelnih zaledij v spletnem okolju ogrodij JavaScript.

2.2 Webpack 5

Opazili smo, da je pristop mikro čelnih zaledij spletnih rešitev močno povezan s konceptom mikro storitev. Mikro storitve imajo v svetu zalednih sistemov relativno naravno implementacijsko rešitev. Kljub temu je bilo v svetu čelnih zaledij spletnih aplikacij v preteklosti težavno implementirati arhitekturo mikro čelnih zaledij. Težava je bila v popularni tehnologiji prevajanja in povezovanja modulov JavaScript, imenovani Webpack [9]. Tehnologija Webpack je predvidevala, da je celotna programska koda na voljo med prevajanjem [3]

Leta 2020 nam je razvoj tovrstne arhitekture olajšal prihod vtičnika za povezovanje modulov (angl. Module federation) v tehnologiji Webpack 5 [2]. Vtičnik omogoča aplikaciji, ki deluje na osnovi programskega jezika JavaScript, dinamično nalaganje ločeno prevedenih delov programske kode. Posledično lahko dostopamo do delov programske kode, ki še niso znani v času prevajanja. Prav tako je omogočeno deljenje skupnih programskih knjižnic, kar onemogoči podvajanje. Na ta način lahko arhitekturo aplikacije zastavimo tako, da jo razdelimo na več ločenih aplikacij (Slika 2):

- Usmerjevalnik: služi dinamičnemu nalaganju mikro aplikacij, upravljanju deljenih programskih knjižnic, obravnavi nedostopnosti mikro aplikacij in podobno.
- Mikro aplikacija: del originalne aplikacije, delitev določena po smislu.



Slika 2: Shema mikro arhitekture čelnega zaledja spletne rešitve.
Vir: lasten.

Od prihoda tehnologije Webpack 5 je razvoj mikro čelnih zaledij v spletnih rešitvah močno pridobil v popularnosti. Koncept smo implementirali tudi mi, saj nas je zanimalo, ali se nam obrestuje. V nadaljevanju predstavimo implementacijske podrobnosti v spletnem ogrodju Angular.

3 Angular

3.1 Mikro čelna zaledja v spletnem ogrodju Angular

Angular je razvojno spletno ogrodje, zgrajeno na podlagi programskega jezika TypeScript. Omogoča nam:

- Multi platformnost: izdelava spletnih strani, mobilnih aplikacij z uporabo strategij iz ogrodja Ionic, aplikacij za namizne računalnike z uporabo ogrodja Electron.
- Učinkovitost: optimizacija prevajanja programske kode v virtualno napravo programskega jezika JavaScript.
- Produktivnost: uporaba predlog, komponent, orodij ukazne vrstice, podpora v integriranih razvojnih okoljih, ponuja bogat nabor knjižnic.
- Enostavna implementacija internacionalnih aplikacij.

Mi se osredotočimo predvsem na uporabo spletnega ogrodja Angular za namen implementacije mikro čelnih zaledij v spletnih rešitvah. V ta namen smo uporabili Angular verzije 14.0, ki vključuje podporo tehnologiji Webpack 5. Posledično se lahko poslužimo uporabe vtičnika povezovanja modulov. Implementacijo mikro čelnih zaledij predstavimo na enostavnem primeru aplikacije, sestavljene iz štirih strani.

3.2 Primer uporabe

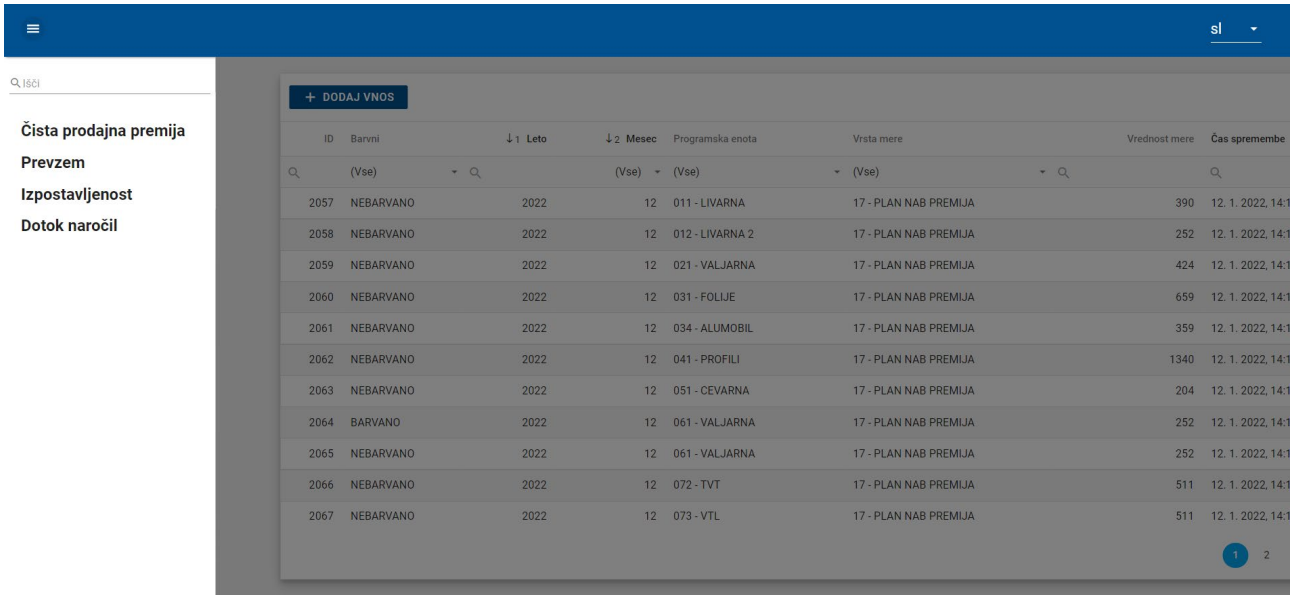
Za namen demonstracije smo se odločili preoblikovati aplikacijo, ki je sestavljena iz štirih različnih strani (Slika 3). Odločili smo se, da je mikro čelno zaledje aplikacije sestavljeno iz štirih mikro aplikacij (za vsak vnos v meniju ena mikro aplikacija) ter iz usmerjevalnika, ki dinamično kliče mikro aplikacije (Slika 4).

Najprej smo morali spremeniti strukturo aplikacije (Slika 5). Za namen demonstracije smo prestavili mikro aplikacijo Izpostavljenost izven strukture projekta, da lahko v nadaljevanju pokažemo, kako vtičnik povezovanja modulov razreši različne verzije ogrodja Angular. Po spremembi strukture smo ustrezno posodobili datoteko angular.json z dodanimi projekti. Nato je sledilo nalaganje knjižnice vtičnika za povezovanje modulov v spletnem ogrodju Angular [8], ki nam ponuja avtomatsko generiranje datotek, potrebnih za uporabo omenjenega vtičnika. V kolikor smo knjižnico naložili za aplikacijo Usmerjevalnik, smo v ukazni vrstici zagnali sledeč ukaz:

```
ng add @angular-architects/module-federation --project <ime_projekta_usmerjevalnik> --
type host --port 4200
```

Sicer pa smo zagnali sledeč ukaz:

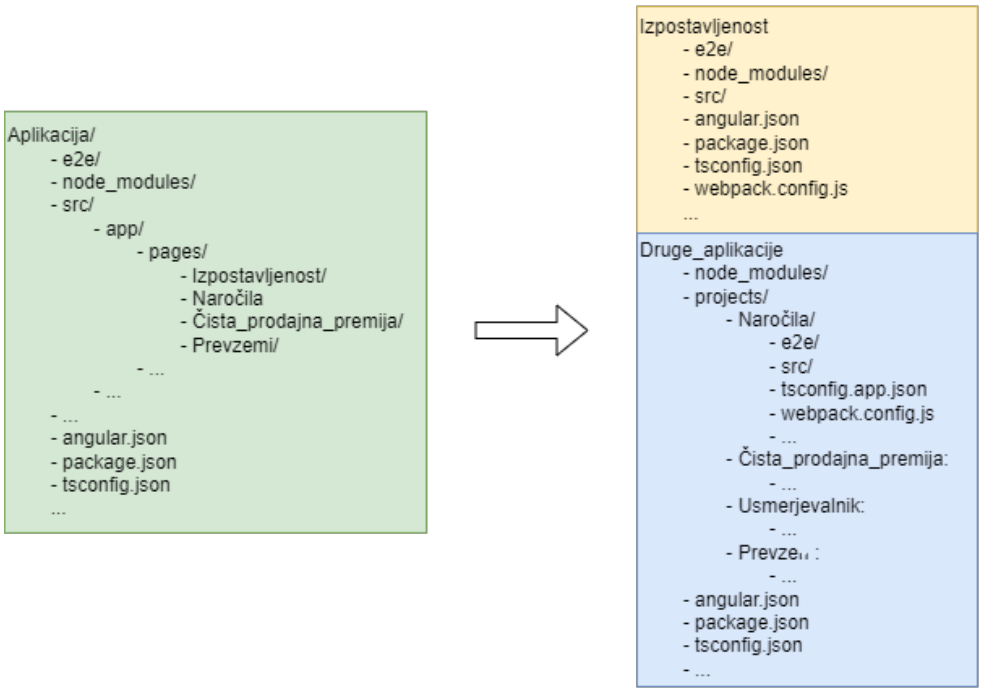
```
ng add @angular-architects/module-federation --project <ime_mikro_aplikacije> --type
remote --port 4201
```



Slika 3: Nespremenjena demonstracijska aplikacija.
Vir: lasten.

Slika 4: Delitev demonstracijske aplikacije na mikro čelne aplikacije.
Vir: lasten.

Opazimo, da je razlika le v tipu projekta ter v nastavljenih vratih, na katerih je aplikacija dostopna. Knjižnica nam generira datoteko webpack.config.js, ki služi konfiguraciji vtičnika povezovanja modulov, ter posodobi usmerjevalno datoteko app.routes.ts v aplikaciji Usmerjevalnik. Datoteko webpack.config.js moramo še ročno posodobiti tako, da definiramo, katere module mikro aplikacija izvaža in katere odvisnosti si deli z ostalimi mikro aplikacijami. Slika 6 nakazuje primer enostavnih nastavitev. Za mikro aplikacijo Usmerjevalnik je nastavitvena datoteka podobna, le da moramo določiti naslove oddaljenih mikro aplikacij, katere bo Usmerjevalnik klical (Slika 7).



Slika 5: Sprememba strukture demonstracijskega projekta.
Vir: lasten.

```
const { shareAll, withModuleFederationPlugin } = require('@angular-architects/module-federation/webpack');

module.exports = withModuleFederationPlugin({

  name: 'exposure',
  filename: 'remoteEntry.js',
  exposes: {
    './pages.module': './projects/exposure/src/app/pages/pages.module.ts',
  },

  shared: {
    ...shareAll({ singleton: true, strictVersion: true, requiredVersion: 'auto' }),
  },
});
```

Slika 6: Primer nastavitev datoteke webpack.config.js posamezne mikro aplikacije.
Vir: lasten.

```
const { shareAll, withModuleFederationPlugin } = require('@angular-architects/module-federation/webpack');

module.exports = withModuleFederationPlugin({

  remotes: {
    "exposure": "http://localhost:4201/remoteEntry.js"
  },

  shared: {
    ...shareAll({ singleton: true, strictVersion: true, requiredVersion: 'auto' }),
  },

});
```

Slika 7: Primer nastavitev datoteke webpack.config.js mikro aplikacije Usmerjevalnik.
Vir: lasten.

V obeh primerih smo za enostavnost demonstracije uporabili funkcijo programske knjižnice [8] shareAll za deljenje vseh odvisnosti, ki so določene v datoteki package.json. V kolikor imata mikro aplikaciji enake odvisnosti s kompatibilnimi verzijami, naša zahteva po deljenju odvisnosti povzroči enkratno nalaganje programskih knjižnic. S tem smo preprečili možnost večkratnega nameščanja odvisnosti.

Za prikaz delovanja nalaganja moramo le še prilagoditi usmerjanje v mikro aplikaciji Usmerjevalnik. V ta namen smo posodobili datoteko app.routes.ts (Slika 8). Ob tem omenimo, da pot 'exposure/pages.module' ne obstaja v aplikaciji Usmerjevalnik. Je le navidezna pot, ki nakazuje na oddaljeno mikro aplikacijo. Tega se prevajalnik ne zaveda, zato moramo ustvariti deklaracijsko datoteko decl.d.ts, v kateri deklariramo modul 'exposure/pages.module'.

V kolikor zaženemo obe aplikaciji v ločenih terminalih na vratih 4200 in 4201, opazimo, da Usmerjevalnik uspešno naloži oddaljeno mikro aplikacijo Izpostavljenost. Podoben postopek smo storili še za ostale mikro aplikacije in uspešno pretvorili začetno monolitno aplikacijo čelnega zaledja v več neodvisnih mikro aplikacij čelnega zaledja.

```
import { loadRemoteModule } from '@angular-architects/module-federation';
import { Routes } from '@angular/router';

export const APP_ROUTES: Routes = [
    {
      path: '',
      redirectTo: 'exposure',
      pathMatch: 'full'
    },

    {
      path: 'exposure',
      loadChildren: () => import('exposure/pages.module').then(m => m.PagesModule)
    },
];
```

Slika 8: Oddaljeno nalaganje mikro aplikacije.
Vir: lasten.

3.3 Koristni napotki in izboljšave

V kolikor pogledamo nastavitvene datoteke webpack.config.js, ki smo jih predstavili v primeru uporabe, vidimo, da smo za namen enostavnosti uporabili deljenje vseh paketov, ki jih mikro aplikacija koristi. Včasih to ni scenarij, ki bi si ga želeli. Še posebej, če mikro aplikacija vsebuje veliko takšnih modulov. V okolju soodvisnosti se namreč ne moremo poslužiti prednosti odstranitve tako imenovane mrtve kode (angl. Dead code), kar lahko povzroči večje količine nalaganja ločenih paketov programske kode. Kot optimizacijo nam programska knjižnica [8] omogoča uporabo funkcije share kot nadomestilo funkciji shareAll, v kateri ročno določimo, katere pakete delimo z ostalimi mikro aplikacijami.

Demonstracijski projekt je zastavljen tako, da se vse mikro aplikacije naložijo ob zagonu Usmerjevalnika. V primeru večjega števila mikro aplikacij je takšna implementacija neželena zaradi daljšega nalaganja. Prav tako nam aplikacija Usmerjevalnik prikaže prazno stran, v kolikor katera izmed mikro aplikacij ni dosegljiva, kar je v nasprotju s principi mikro čelnih zaledij spletnih rešitev. Z malo sprememb lahko omogočimo dinamično nalaganje oddaljenih mikro aplikacij na zahtevo, ki reši predstavljeni težavi. Vse kar moramo storiti je spremeniti 3 datoteke v projektu Usmerjevalnik (Slika 9):

- Potrebno je ustvariti datoteko /src/assets/mf.manifest.js, v kateri definiramo vse oddaljene mikro aplikacije v obliki formata JSON.
- Prilagoditi moramo datoteko main.ts tako, da najprej pokličemo funkcijo loadManifest, ki jo ponuja programska knjižnica [8]. S tem naložimo predhodno ustvarjeno datoteko, ki vsebuje vse naslove oddaljenih mikro aplikacij.
- Nadomestimo vsa mesta v datoteki app.routes.ts, kjer smo vključili vnaprej deklariran modul oddaljene mikro aplikacije, z metodo loadRemoteModule programske knjižnice [8]. Le-ta poskrbi za dinamično klicanje oddaljene mikro aplikacije ob zahtevi.

Kot smo že omenili, smo za namen demonstracije podpore različnih verzij spletnega ogrodja Angular mikro aplikacijo Izpostavljenost izolirali (Slika 5). Verzijo spletnega ogrodja Angular smo tej aplikaciji nastavili na 14.0.0 ter ustrezno posodobili pripadajočo datoteko webpack.config.js. Verzijo preostalih mikro aplikacij smo prepustili trenutno najnovejši verziji 14.0.3. Slika 7 prikazuje, da aplikacija Usmerjevalnik zahteva enkratno nalaganje deljenih knjižnic. V kolikor želimo naložiti mikro aplikacijo Izpostavljenost v Usmerjevalnik, nas Angular obvesti z napako: »Error: inject() must be called from an injection context«. S tem nam pove, da želimo naložiti več različnih verzij enake knjižnice, čeprav smo zahtevali enkratno nalaganje. Posledično se obe verziji naložita, prikaže pa se nam nič ne na zaslon. Ena izmed rešitev je sprememba datoteke webpack.config.js na strani mikro aplikacije Usmerjevalnik tako, da kot parameter deljene knjižnice podamo le možnost »requiredVersion: 'auto'«. Na ta način smo omogočili nalaganje več različnih verzij ene knjižnice, v kolikor med seboj niso kompatibilne. Kljub temu bi omenili, da nam

ta način prinese podaljšan čas nalaganja, kar je lahko v primeru več knjižnic uporabniku neprijazno. Prav tako je upravljanje nalaganja knjižnic, katerih verzije so ekstremno različne, težavno. Ob nepravilnih nastavitvah se lahko aplikacija zruši ali pa ne prikaže nič na zaslon. Zato priporočamo uporabo kompatibilnih verzij uporabljenih paketov. Z uporabo parametra »strictVersion« v nastavitvah deljenih knjižnic lahko priporočen način enostavno dosežemo, saj natančno zahtevamo verzijo deljene knjižnice. V kolikor dana verzija ni na voljo, lahko situacijo poljubno razrešimo s kakšnim obdelovalcem napak po meri.

1.) Dodajanje datoteke mf.manifest.js

```
// mf.manifest.json

{
  "sales-premium": "http://localhost:4201/remoteEntry.js",
  "exposure": "http://localhost:4202/remoteEntry.js",
  "takeover": "http://localhost:4203/remoteEntry.js",
  "orders": "http://localhost:4204/remoteEntry.js"
}
```

2.) Posodobitev datoteke main.ts

```
import('./bootstrap')
  .catch(err => console.error(err));
```

⇨

```
import { loadManifest } from '@angular-architects/module-federation';

loadManifest("/assets/mf.manifest.json")
  .catch(err => console.error(err))
  .then(_ => import('./bootstrap'))
  .catch(err => console.error(err));
```

3.) Dopolnitev datoteke app.routes.ts

```
import { loadRemoteModule } from '@angular-architects/module-federation';
import { Routes } from '@angular/router';

export const APP_ROUTES: Routes = [
    {
      path: '',
      redirectTo: 'exposure',
      pathMatch: 'full'
    },

    {
      path: 'exposure',
      loadChildren: () => import('exposure/pages.module').then(m => m.PagesModule)
    },
];
```

⇨

```
import { loadRemoteModule } from '@angular-architects/module-federation';
import { Routes } from '@angular/router';

export const APP_ROUTES: Routes = [
    {
      path: '',
      redirectTo: 'exposure',
      pathMatch: 'full'
    },

    {
      path: 'exposure',
      loadChildren: () => loadRemoteModule({
        type: 'manifest',
        remoteName: 'exposure',
        exposedModule: './pages.module'
      }).then(m => m.PagesModule)
    },
]
```

Slika 9: Potrebne posodobitve za dinamično nalaganje oddaljenih mikro aplikacij.
Vir: lasten.

Dodatno bi želeli izpostaviti, da moramo v datoteki app.module.ts aplikacije Usmerjevalnik vključiti globalne storitve (npr. storitev za omrežje - HttpClientModule), v kolikor te storitve oddaljena mikro čelna aplikacija pričakuje oziroma koristi. V kolikor tega ne storimo, oddaljena mikro čelna aplikacija ne bo vedela, kje naj dano globalno storitev poišče. Poleg tega omenimo, da oddaljene mikro čelne aplikacije ne bi smele izpostavljati svojih modulov aplikacije (angl. App module). Lahko se zgodi, da to niti ne bo mogoče zaradi napak v prevajanja modula usmerjanja (angl. Router module), ki zahteva enkratno klicanje metode forRoot. V kolikor te metode v oddaljeni mikro aplikaciji ne uporabimo, se nam lahko aplikacija celo zažene. Kljub temu je to slaba praksa, saj se s tem načinom podvojijo vse deljene storitve, ki so registrirane v obsegu mikro aplikacije [4].

4 Samostojna komponenta v spletnem ogrodju Angular

4.1 Uporabna vrednost

Pred kratkim je s prihodom verzije 14 spletnega ogrodja Angular prispela nova funkcionalnost, ki odpravi potrebo po stvaritvi nastavitvenih datotek nalaganja modulov (razredov z dekoratorjem @NgModule), ki so namenjene le prevajalniku. Funkcionalnost je poimenovana samostojna komponenta (angl. Standalone component), saj se potrebne informacije o vključevanju ostalih knjižnic prestavijo v metapodatke komponent. Na ta način zmanjšamo količino nepotrebne programske kode, olajšamo razvijalcu delo in natančno definiramo, katere programske

knjižnice potrebuje določena komponenta. Koncept samostojnih komponent ne velja le za komponente, ampak je razširjen tudi na cevi, direktive in storitve [5]. Opozoriti moramo, da je koncept samostojnih komponent zaenkrat le v predogled razvijalcem in se še lahko spremeni. Kljub temu pristop obljublja veliko pozitivnih lastnosti, zato smo želeli izkusiti, kako se obnese v praksi. V nadaljevanju opišemo postopek naše implementacije na predhodno demonstriranem primeru ter naše mnenje o doprinosu funkcionalnosti.

4.2 Integracija v demonstracijski primer uporabe mikro čelnih zaledij

Samostojne komponente si v bistvu lahko predstavljamo kot komponente z lastno nastavitveno datoteko nalaganja modulov. Integracija v demonstracijski primer je zelo enostavna. Najprej moramo v dekorator komponente dodati parameter »standalone« in mu nastaviti resnično vrednost. S tem povemo, da je komponenta samostojna. Nato v parametru »imports« določimo vse odvisnosti, ki jih komponenta potrebuje za delovanje. Kot odvisnost lahko vključimo programske knjižnice ali pa celo ostale samostojne komponente. Slika 10 prikazuje primer implementacije opisanega postopka. V kolikor po opisanih navodilih definiramo vse komponente v aplikaciji, lahko iz aplikacije brez skrbi odstranimo vse datoteke nalaganja modulov.

```
@Component({
  standalone: true,
  imports: [
    CommonModule,
    RouterModule,

    DxButtonModule,
    DxDataGridModule,
    DxDrawerModule,
    DxListModule,
    DxScrollViewModule,
    DxSelectBoxModule,
    DxToolbarModule,
    DxTreeViewModule,

    HttpClientModule,
    ToastrModule,
    BiManagementLibModule
  ],
  selector: 'app-exposure',
  templateUrl: './exposure.component.html',
  styleUrls: ['./exposure.component.scss']
})
export class ExposureComponent implements OnInit {
```

Slika 10: Primer nastavitve metapodatkov samostojne komponente.
Vir: lasten.

Preostane nam le še nastavitev začetnega nalaganja komponent in s tem posledično celotne aplikacije Angular. Do zdaj so za to poskrbele datoteke nalaganja modulov, ki smo jih izbrisali. Zato moramo na drug način razložiti spletnemu ogrodju Angular, kako naj naloži in zažene celotno aplikacijo. V ta namen nam verzija 14 spletnega ogrodja Angular ponuja funkcijo bootstrapApplication, ki smo jo uporabili v datoteki bootstrap.ts (Slika 11). Funkcija kot prvi parameter prejme glavno komponento aplikacije, kot drugi pa globalne ponudnike storitev. S tem smo opravili vse potrebne spremembe in lahko brez strahu na novo zaženemo našo aplikacijo.

Omeniti moramo, da je potrebno za delovanje posodobljene mikro čelne aplikacije posodobiti še nastavitveno datoteko webpack.config.js. Ker nimamo več razredov, ki so namenjeni nalaganju modulov, lahko oddaljenim mikro čelnim aplikacijam podamo na voljo datoteko poti (npr. app.routes.ts). Načeloma bi lahko izpostavili tudi samostojno komponento, kar pa ni dobra praksa mikro čelnih zaledij spletnih aplikacij [5]. Nato v aplikaciji Usmerjevalnik ustrezno spremenimo datoteko app.routes.ts, da naloži novo izpostavljeno usmerjevalno datoteko oddaljene mikro čelne aplikacije. Tako smo z uporabo principa samostojnih komponent uspešno nadomestili vse datoteke nalaganja modulov, ki so bile namenjene le prevajalniku.

```
if (environment.production) {
  enableProdMode();
}

bootstrapApplication(AppComponent, {
providers: [
  importProvidersFrom(RouterModule.forRoot(PAGES_ROUTES)),
  importProvidersFrom(HttpClientModule),
  importProvidersFrom(ToastrModule.forRoot({
    positionClass: 'toast-bottom-center',
    enableHtml: true
    })),
  importProvidersFrom(BiManagementLibModule.forRoot(environment))
  ]
});
```

Slika 11: Nalaganje spletne aplikacije Angular, sestavljene iz samostojnih komponent.
Vir: lasten.

5 Zaključek

Koncept mikro čelnih zaledij spletnih rešitev rešuje probleme, ki so posledica monolitnih aplikacij čelnega zaledja. Pristop izhaja iz arhitekturne rešitve mikro storitev na zalednem sistemu in prinaša mnogo pozitivnih vidikov. V spletnem ogrodju Angular se je s prihodom tehnologije Webpack 5 razvoj dane arhitekture razbesnel zaradi dodanega vtičnika povezovanja modulov in posledično enostavne implementacije v praksi. Pristop smo preizkusili in analizirali njegovo uporabnost ter dodano vrednost. Moramo se zavedati, ali je dana arhitektura primerna za naš problem. Z napačno izbranim projektom lahko z opisanim konceptom prinesemo več že opisanih negativnih posledic kot pozitivnih. Menimo, da je izbira mikro čelnih zaledij primerna predvsem za velike, dolgotrajne in kompleksne spletne rešitve z večjim številom razvijalcev.

V prispevku smo prav tako predstavili obetavno razširitev samostojnih komponent, ki jo je kot predogled prinesla verzija 14 spletnega ogrodja Angular. Funkcionalnost je implementacijsko zelo enostavna in prinaša mnogo pozitivnih pogledov, tako za razvijalca kot za arhitekturni pogled čelnih aplikacij spletnih rešitev. Koncept je preprost in enostaven za razumevanje, zato smo ga razširili v celoten ekosistem demonstracijske aplikacije. Verjamemo, da je dana razširitev eden izmed pomembnejših sprememb, ki jih ponuja verzija 14 spletnega ogrodja Angular. Menimo, da bo v prihodnosti vidik samostojnih komponent zaradi svoje praktičnosti zamenjal trenutno razširjen pristop uporabe datotek nalaganja modulov.

Literatura

[1] https://micro-frontends.org/, Micro Frontends, obiskano 22. 06. 2022.

[2] https://webpack.js.org/concepts/module-federation, Module federation, obiskano 23. 06. 2022.

[3] https://www.angulararchitects.io/en/aktuelles/the-microfrontend-revolution-module-federation-in-webpack-5/, STEYER, Manfred. The Microfrontend Revolution: Module Federation in Webpack 5, obiskano 23. 06. 2022.

[4] https://www.angulararchitects.io/en/aktuelles/pitfalls-with-module-federation-and-angular/, STEYER, Manfred. Pitfalls with Module Federation and Angular, obiskano 24. 06. 2022.

[5] https://www.angulararchitects.io/aktuelles/angulars-future-without-ngmodules-lightweight-solutions-on-top-of-standalone-components/, STEYER, Manfred. Module Federation with Angular's Standalone Components, obiskano 24. 06. 2022.

[6] https://www.angulararchitects.io/en/aktuelles/using-module-federation-with-monorepos-and-angular/, STEYER, Manfred. Using Module Federation with Nx Monorepos and Angular, obiskano 08. 07. 2022.

[7] https://www.angulararchitects.io/aktuelles/module-federation-with-angulars-standalone-components/, STEYER, Manfred. Angular's Future Without NgModules – Part 1: Lightweight Solutions Using Standalone Components, obiskano 27. 06. 2022.

[8] https://www.npmjs.com/package/@angular-architects/module-federation, @angular-architects/module-federation, obiskano 24. 06. 2022.

[9] https://webpack.js.org/, Webpack, obiskano 08. 07. 2022.

[10] PELTONEN, Severi; MEZZALIRA, Luca; TAIBI, Davide. Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review. *Information and Software Technology*, 2021, 136: 106571.

[11] YANG, Caifang; LIU, Chuanchang; SU, Zhiyuan. Research and application of micro frontends. In: *IOP conference series: materials science and engineering*. IOP Publishing, 2019. p. 062082.

[12] PAVLENKO, Andrey, et al. Micro-frontends: application of microservices to web front-ends. *J. Internet Serv. Inf. Secur.*, 2020, 10.2: 49-66.

Dodajanje poljubnih funkcionalnosti digitalni kripto denarnici MetaMask

Vid Keršič, Andraž Vrečko, Urban Vidovič, Martin Domajnko, Muhamed Turkanović

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
vid.kersic@um.si, andraz.vrecko@student.um.si, urban.vidovic2@um.si,
martin.domajnko@student.um.si, muhamed.turkanovic@um.si

Sinopsis Kripto denarnice omogočajo interakcijo z verigami blokov (angl. blockchain) in decentraliziranimi aplikacijami (angl. decentralized application, dApp), ter posledično upravljanje z zamenljivi ali nezamenljivimi žetoni (angl. NFT), interakcijo z navideznimi svetovi (angl. metaverse) itn. Danes je vsekakor najpriljubljenejša denarnica MetaMask. Ker se funkcionalnosti in možnosti uporabe tehnologije veriženja blokov zelo hitro povečujejo, so razvijalci denarnice MetaMask leta 2022 izdali razvojni sistem imenovan Snaps. Ta omogoča razvoj in integracijo poljubnih funkcionalnosti v obliki vtičnikov za kripto denarnico MetaMask.

V prispevku bomo predstavili kako lahko na osnovi sistema Snaps snujemo poljubno funkcionalnost h kripto denarnici MetaMask ter le to temu primerno razvijemo in integriramo. Za namen demonstracije bomo predstavili implementacijo razširitev, ki MetaMask denarnici doda podporo decentraliziranim identitetam in identifikatorjem (angl. decentralized identifier, DID) ter preverljivim poverilnicam (angl. verifiable credential, VC), ki so do sedaj za uporabo potrebovali ločeno rešitev.

Ključne besede:
kripto denarnice
digitalna identiteta
tehnologija veriženja blokov
MetaMask
samo-upravljana identiteta

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.12

1 Uvod

Tehnologija veriženja blokov (angl. blockchain) se je v zadnjih letih razširila na razna področja, na primer finančne trge, dobavne verige, zabavno industrijo, igričarsko industrijo in virtualne svetove [29]. Razlog za njeno uspešno vpeljavo je bila odprtost, nevtralnost, decentralizacija omrežij brez enega samega lastnika in interoperabilnost tehnologije s številnimi standardi, ki so omogočili razvoj kripto valut, decentraliziranih digitalnih identitet, nezamenljivih žetonov (angl. non-fungible token, NFT) kot sredstev za digitalno lastništvo itd. Kljub številnim prednostim je lahko uporaba tehnologije novim uporabnikom zelo zahtevna, saj je v primerjavi s sorodnimi tehnologijami bistveno kompleksnejša in orodja so ljudem na prvi pogled zapletena, podobno kot so bili spletni brskalniki v 90. letih. Za enostavnejšo uporabo tehnologije veriženja blokov so se razvile digitalne kripto denarnice, ki uporabnikom omogočajo interakcijo z verigami blokov in decentraliziranimi aplikacijami (angl. decentralized application, dApp) ter posledično upravljanje z vsemi produkti na verigi blokov, na primer kripto valutami in NFT-ji. Toda hitri razvoj funkcionalnosti in tudi novih verig blokov je povzročil veliko število digitalnih kripto denarnic, kjer vsaka pokriva le določene funkcionalnosti.

Danes prevladujoča kripto denarnica je MetaMask, ki podpira vsa omrežja, ki temeljijo na navideznem stroju Ethereum (angl. Ethereum Virtual Machine, EVM) [17, 3]. MetaMask omogoča upravljanje s kriptografskimi ključi, ki so ena izmed najpomembnejših komponent tehnologije veriženja blokov, pošiljanje kripto valut in NFT-jev, konstrukcijo in pošiljanje transakcij v omrežje, pregled stanja kripto valut in NFT-jev, zamenjavo kripto valut itd. Kljub mnogim funkcionalnostim se MetaMask ne more uporabiti za čisto vse operacije, ki se pogosto dogajajo v svetu verige blokov, na primer ne podpira uporabe omrežja Bitcoin, ali shranjevanja datotek na distribuirano omrežje IPFS. Ker razvijalci ne moremo sami podpreti vseh možnih funkcionalnosti in slediti vsem standardom, ki nastajajo zelo hitro, so leta 2022 izdali razvojni sistem Snaps. Ta omogoča razvoj in integracijo poljubnih funkcionalnosti v obliki vtičnikov za kripto denarnico MetaMask, pri čemer se osnovne funkcionalnosti lahko nadgradijo s poljubnimi, hkrati pa se ohrani varnost operacij, saj kriptografski ključi nikoli ne zapustijo varne hrambe. Tako lahko uporabniki uporabljajo znano rešitev za več namenov in ne potrebujejo novih orodij oz. aplikacij za vsako dodatno funkcionalnost.

Evropska unija razvija specifikacijo za digitalne identitete, ki je usmerjena v končne uporabnike, kjer imajo ti lastništvo nad svojimi podatki, so jim cel čas na voljo, sledi načelom GDPR in omogoča možnosti identifikacije z internetno povezavo ali brez. Eden izmed možnih načinov implementacije takšne digitalne identitete je s samoupravljano identiteto (angl. self-sovereign identity, SSI), ki temelji na podobni kriptografiji kot tehnologija veriženja blokov [21]. Kljub temu danes na trgu izključno prevladujejo kripto denarnice, ki so namenjene le SSI, kar zahteva od uporabnikov še eno dodatno orodje za digitalno življenje. Zato se poraja vprašanje: ali se lahko upravljanje z digitalno identiteto temelječo na SSI integrira v že obstoječo kripto denarnico in razvije enoten produkt za verige blokov ter digitalne identitete?

V prispevku bomo predstavili, kako lahko na osnovi sistema Snaps snujemo poljubno funkcionalnost h kripto denarnici MetaMask, jo implementiramo in integriramo v MetaMask ter posledično v dApp. Čeprav bomo pokazali, da nam sistemi Snaps omogočajo praktično neomejeno možnosti, bomo v prispevku, kot že omenjeno, za namen demonstracije predstavili implementacijo razširitve za podporo digitalni identiteti temelječi na principih in komponentah SSI. Prispevek zaključimo z analizo prednosti in slabosti, ki jih ima predstavljen pristop, in pregledom trenutnega stanja novih funkcionalnosti kripto denarnice MetaMask.

2 Osnovni koncepti

2.1 Tehnologija veriženja blokov

Tehnologija veriženja blokov predstavlja porazdeljene digitalne knjige transakcij (angl. distributed ledgers, DLT), ki nimajo centralne avtoritete in so odporne na poneverbo podatkov (angl. tamper-proof). Leta 2008 je bila ideja

veriženja blokov prvič združena z nekaterimi drugimi tehnologijami in računalniškimi koncepti, ko je Satoshi Nakamoto ustvaril prvo kriptovaluto imenovano Bitcoin [30], ki še danes ostaja na prvem mestu po tržnem kapitalu (angl. market capital). Druga največja po tržnem kapitalu[1] in največja, ko govorimo o ekosistemu javnih tehnologij omrežij blokov, je veriga blokov imenovana Ethereum[2], ki je bila vzpostavljena leta 2015 in predstavila podporo pametnim pogodbam. Omrežje verige blokov v primeru javnega tipa v grobem sestoji iz vozlišč (angl. nodes) porazdeljenih povsod po svetu, pridruži pa se jim lahko teoretično kdorkoli, kar v idealnem primeru pomeni decentraliziran sistem pri katerem zaupanje med uporabniki in vozlišči ni potrebno. Celotno upravljanje je v primeru javnih tipov verig blokov prav tako popolnoma decentralizirano oz. odvisno od 51 % vozlišč. Poznamo pa tudi t. i. zaprte oz. zasebne verige blokov, kjer pa so vozlišča vnaprej definirana s strani centraliziranih upraviteljev, med tem ko je sam dostop do verige blokov lahko javen ali zaprt [31].

Vsako vozlišče ima praviloma pri sebi shranjeno celotno verigo blokov, ki se posodablja in sinhronizira z vsakim novim dodanim blokom. V blokih verige oz. natančneje logičnih skupkih transakcij so zabeležene nespremenljive transakcije med uporabniki, pametne pogodbe in klici funkcij pametnih pogodb, ki spremenijo stanje računov ali prožijo dogodek (angl. event). Vsaka transakcija se pri vključitvi v blok overi, blok pa se pred dodajanjem v verigo potrdi z določenmim algoritmom konsenza.

Bloki se med seboj povezujejo s pomočjo zgoščenih vrednosti (angl. hash) na način, da ima vsak naslednji blok shranjeno zgoščeno vrednost prejšnjega bloka. V primeru, da napadalci poskušajo v verigo blokov vključiti nov zlonameren blok, ga vozlišča zavrnejo, saj se zgoščene vrednosti več ne ujemajo. Poznamo več tipov konsenza, kot so na primer dokazilo o delu (angl. proof of work, PoW), dokazilo o deležu (angl. proof of stake, PoS) in dokazilo o avtoriteti (angl. proof of authority, PoA) [32, 33]. Na začetku je bila večina omrežij blokov zavarovana s PoW, vendar v zadnjem času prevladuje PoS v javnih oz. PoA zasebnih omrežjih, saj je varnost PoW zagotovljena z rudarjenjem blokov, ki je zelo potraten in računsko zahteven računalniški algoritem.

Kot rečeno, je Ethereum kot nadgradnja Bitcoina uvedel pametne pogodbe (angl. smart contracts). To so deli programske kode, ki so zapisani v bloku, koda pa se izvaja ob klicih funkcij implementiranih v pametnih pogodbah [34]. Pri tradicionalnih pogodbah pogodbeniki potrebujejo zaupanja vredno avtoriteto, ki potrdi pogoje sklenjene pogodbe, medtem, ko se pri pametnih pogodbah izognemo potrebi po omenjeni tretji avtoriteti, saj je koda pogodbe nespremenljiva in vidna vsem, izvrši pa se le, ko so izpolnjeni vsi pogoji.

Vsak uporabnik ima svoj unikaten naslov, ki se enosmerno deterministično izpelje iz privatnega ključa, kar pomeni, da naslov iz privatnega ključa lahko izpeljemo, privatni ključ iz naslova pa ne. Naslovi na nek način predstavljajo identiteto uporabnika ali pametne pogodbe in so vidni v vsakem dogodku na omrežju. Tako lahko točno vidimo, kdo je komu kaj poslal, katero funkcijo pametne pogodbe je izvršil itd.

Na omrežju Ethereum že obstaja veliko t.i. decentraliziranih aplikacij [35]. Te aplikacije se pišejo po določenih standardih z željo, da so interoperabilne med vsemi platformami. Ene izmed najpopularnejših aplikacij so nezamenljivi žetoni. V grobem so to "žetoni", kjer je vsak unikaten in jih ne gre deliti na več delov. V verigi blokov je zapisano, kdo je trenutni lastnik katerega žetona, lastniki jih lahko pošljejo drugim uporabnikom in z njimi trgujejo, kar se prav tako zabeleži v verigo blokov. NFT-ji ponavadi referencirajo določeno medijsko predstavnost - sliko, zvok ali video. Pri tem ima vsak posamezen žeton svojo identifikacijsko številko, kar ga naredi unikatnega in nezamenljivega. Uporabljajo se tudi za druge namene, kot osnova za platforme, ki zagotavljajo globalno zaposljivost ali upravljanje z digitalnimi certifikati [36]. Najpopularnejša standarda, s katerima lahko implementiramo NFT-je sta ERC721 in ERC1155. Na omrežju Ethereum lahko sicer implementiramo tudi žetone po standardu ERC20. Za razliko od NFT-jev so to žetoni, ki so zamenljivi, kar pomeni, da so vsi žetoni med seboj popolnoma enaki. Primer aplikacije ERC20 žetonov je odklepanje novih funckionalnosti na določeni platformi (npr. platforma brezplačno ponuja poslušanje glasbe v nizki kakovosti, vsi uporabniki, ki si lastijo N določenih žetonov pa lahko na platformi glasbo poslušajo v višji kakovosti). Vse pogosteje slišan izraz, ki se prav tako povezuje s tehnologijo veriženja blokov, je Metaverse, ki opisuje splet kot združen digitalni svet s katerim smo v

[1] https://coinmarketcap.com/
[2] https://defillama.com/

stiku s pomočjo virtualne (angl. virtual reality, VR) in obogatene resničnosti (angl. augmented reality, AR). Vsak uporabnik upravlja svojega avatarja, ki je del Metaverse-a in raziskuje virtualni svet. ERC20 žetoni in NFT-ji predstavljajo dobrine v Metaverse-u, ERC20 npr. žetoni oz. valuta za kupovanje vozil, oblačil ipd., NFT-ji - npr. dejanska oblačila, ki jih lahko oblečemo našemu avatarju, vozilo, ki ga s svojim avatarjem vozimo. Tehnologija veriženja blokov se danes vključuje v tehnologije nove generacije svetovnega spleta, imenovane web3 [28].

2.2 Kriptografija

Kriptografija igra pomembno vlogo pri tehnologiji veriženja blokov, saj je poglavitna komponenta, ki zagotavlja, da veriga blokov ostane varna in da ponuja določeno stopnjo zasebnosti. Uporablja se pri šifriranju podatkov oz. za digitalno podpisovanje sleherne transakcije. V grobem zagotavlja, da se transakcije, ki jih uporabniki prožijo na osnovi t. i. blockchain računov, lahko verificirajo, kar je ključnega pomena za platformo, ki omogoča upravljanje s t. i. digitalnimi vrednostmi. Pri veriženju blokov se uporablja predvsem asimetrična kriptografija, ki temelji na šifriranju s parom ključev (javni in zasebni). Vsak akter ima svoj par javnega in zasebnega ključa, ki se generira s pomočjo različnih algoritmov, pri čemer je treba poskrbeti, da zasebni ključ ves čas ostane v rokah lastnika, saj lahko v nasprotnem primeru napadalec lastniku odtuji njegovo digitalno vrednost. V veliki večini platform verig blokov se tako uporablja eden izmed algoritmov eliptičnih krivulij (angl. elliptic curve digital signature algorithm, ECDSA). Javni ključ in s tem povezan tudi t. i. blockchain naslov je izpeljan iz zasebnega ključa enosmerno, kar pomeni, da iz zasebnega ključa izpeljemo javni ključ, iz javnega ključa pa naslov, v nasprotno smer pa izpeljevanje ni možno. Pri tehnologiji veriženja blokov je vsaka transakcija zabeležena na porazdeljeni knjigi in digitalno podpisana s privatnim ključem. Tako lahko z imetjem na določenem naslovu razpolaga le lastnik zasebnega ključa, medtem ko lahko veljavnost digitalnega podpisa preverijo vsi uporabniki omrežja, saj validacija poteka z javnim ključem, ki se lahko pridobi iz digitalno podpisane transakcije. Asimetrična kriptografija tako predstavlja višjo raven varnosti, saj si je s prejemnikom potrebno izmenjati le javni del kriptografskega para. Prav ta princip digitalnih podpisov se uporablja tudi pri upravljanju preverljivih poverilnic, šifriranje podatkov pa se v našem primeru uporablja za varno shranjevanje decentraliziranih identitet [1].

2.3 Kripto denarnice

Ena ključnih komponent za interakcijo s tehnologijo veriženja blokov so kripto denarnice. Primarna funkcionalnost kripto denarnic je varna hramba kriptografskih ključev , v večini primerov pa omogočajo tudi uporabo le teh za namene podpisovanja transakcij in v nekaterih primerih tudi šifriranje podatkov. Glavna razlika v primerjavi z digitalnimi denarnicami je ta, da kripto denarnice ne hranijo dejanskih sredstev. Ta so namreč shranjena na verigi blokov in do njih dostopamo s pomočjo naših zasebnih ključev s katerimi dokažemo lastništvo. Prav tako se kripto denarnice razlikujejo od trenutnih sistemov, npr. bančnih, pri katerih z uporabniškim imenom in geslom pridobimo dostop do centralizirane platforme, na kateri lahko izvedemo različne operacije. V kripto denarnicah se vse operacije z našimi ključi podpišejo digitalno lokalno, nato pa se pošljejo v omrežje verige blokov, kjer jih izvedejo vozlišča omrežja. Ta razlika krepko pripomore k varnosti teh sistemov, hkrati pa je verjetnost, da dve denarnici ustvarita enake ključe med 2^{128} in 2^{256}, kar dodatno podkrepi omenjeno trditev. Kripto denarnice lahko omogočajo tudi generiranje večih parov ključev, med katerimi ni povezave, in med katerimi lahko preprosto preklapljamo, kot med računi v aplikaciji.

Kripto denarnice generirajo zasebne ključe deterministično glede na naključno semensko frazo (angl. seed phrase). Ta je sestavljena iz 12 do 24 naključnih besed in služi tudi za obnovitev denarnice v primeru izgub. Ideja uporabe semenskih fraz za deterministično generiranje zasebnih ključev izhaja iz motivacije, da so besede preprostejše za človeško uporabo kot števila v binarni ali šestnajstiški obliki, in je bila predstavljena v 39. predlogu za izboljšanje Bitcoina (angl. Bitcoin improvement proposal, BIP) [20].

Kripto denarnice delimo na več načinov. Najpogostejša delitev razdeli kripto denarnice v tri kategorije, tj., (1) kripto denarnice v obliki programske opreme, (2) kripto denarnice v obliki strojne opreme in (3) kripto denarnice v papirni obliki. Slednja (angl. paper wallet) predstavlja list papirja, ki hrani natisnjene zasebne in javne ključe, ponavadi v obliki QR kod. Ta oblika denarnice sodi tudi pod kategorijo hladne hrambe (angl. cold storage), kar pomeni, da so ključi hranjeni na mediju, ki ni povezan z internetom. Kripto denarnice v obliki strojne opreme prav tako spadajo v kategorijo hladne hrambe, saj so kriptografski ključi hranjeni na ločenih fizičnih napravah, ki so ponavadi v obliki podobni USB ključem z dodatnimi mikrokrmilniki. Ta metoda hrambe se danes uveljavlja kot najvarnejši način hrambe kriptografskih ključev. Dve najbolj znani napravi iz te kategorije sta Ledger [18] in Trezor [23]. V kategorijo kripto denarnic v obliki programske opreme sodijo (1) mobilne kripto denarnice, (2) kripto denarnice kot aplikacije naložene na računalniku in (3) spletne kripto denarnice. Glavna razlika spletnih kripto denarnic od ostalih dveh kategorij je ta, da se kriptografski ključi ne hranijo pri uporabniku, ampak so pod okriljem tretje osebe, ponavadi so to kripto menjalnice (angl. crypto exchange), ki upravljajo s temi ključi v imenu uporabnika. Med najbolj znane kripto menjalnice, ki ponujajo spletne denarnice, sodijo Binance, FTX in Coinbase.

Kripto denarnice se razlikujejo tudi v tem, katera omrežja podpirajo. Nekatere kripto denarnice podpirajo le eno vrsto omrežja, primer sta kripto denarnici Electrum, ki podpira zgolj omrežje Bitcoin, in MetaMask, ki podpira le omrežja, ki temeljijo na navideznem stroju Ethereum. Kripto denarnice, kot so Trust Wallet, Coinbase Wallet in ZenGo, ter večina spletnih kripto denarnic, pa hkrati podpirajo hrambo kriptografskih ključev za interakcijo z različnimi vrstami omrežij.

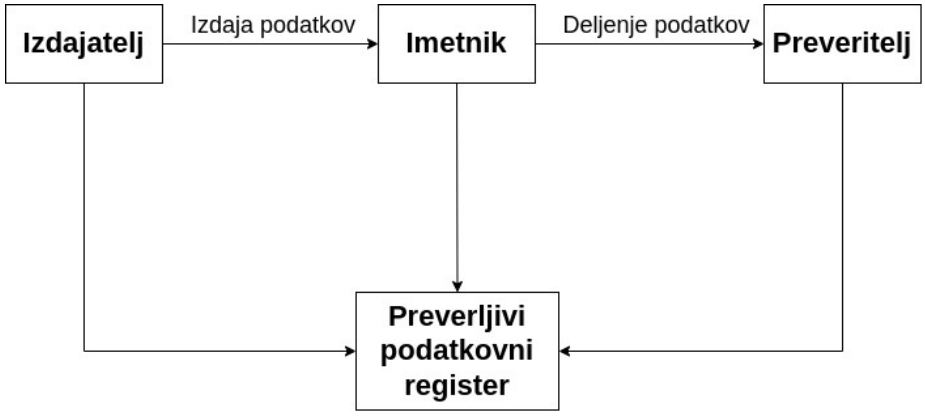
2.3.1 MetaMask

Kripto denarnica MetaMask sodi v kategorijo kripto denarnic v obliki programske opreme. Namesti se lahko kot razširitev v brskalniku ali kot samostojna mobilna aplikacija. Uporabniku zraven osnovnih funkcionalnosti za hrambo in upravljanje s ključi omogoča tudi povezavo s kripto denarnicami v obliki strojne opreme, nakup podprtih kripto valut s fiat denarjem (angl. fiat money), zamenjavo kripto valut, interakcijo s pametnimi pogodbami in digitalno podpisovanje. Nakup kripto valut je omogočen s pomočjo zunanjih storitev Transak, MoonPay in Wyre. Pri zamenjavi kripto valut pa kripto denarnica preišče izbrane decentralizirane menjalnice (angl. decentralized exchange, DEX) in izbere najboljše pretvorbeno razmerje z najnižjimi stroški menjave. Samo interakcijo s pametnimi pogodbami podpira na osnovi podpisovanja transakcij, hkrati pa s tem omogoča tudi funkcionalnost za prikaz stanj ERC20 žetonov v sami denarnici. Implementacija digitalnega podpisovanja temelji na standardih EIP712 [9] in EIP191 [7]. Mobilna različica denarnice podpira tudi protokol WalletConnect [26] in ponuja vgrajen brskalnik, s katerim lahko dostopamo do decentraliziranih aplikacij. MetaMask podpira tudi že nov standard EIP4361 [8], ki definira uporabo Ethereum računov za overjanje in vzpostavljanje sej brez uporabe klasičnih centraliziranih ponudnikov identitet (angl. identity providers, IdPs).

2.4 Samo-upravljanje identitete

Samo-upravljane identitete je nov princip implementacije digitalne identitete v računalništvo, ki sledi principom decentralizacije in usmerjenosti v uporabnika [21, 2]. Prvotni namen SSI je sama digitalna identiteta, torej digitalni identifikatorji in dokumenti oz. certifikati, vendar se lahko uporabi tudi za omejevanje dostopa v računalniških sistemih. Večina standardov nastaja pod Svetovnim spletnim konzorcijem (angl. World Wide Web Consortium, W3C) in Fundacijo za decentralizirano identiteto (angl. Decentralized Identity Foundation, DIF) [27, 6]. Sistem zaupanja v SSI temelji na treh akterjih in sicer izdajatelju (angl. issuer), imetniku (angl. holder) in preveritelj (angl. verifier). Njihova interakacija in prenos podatkov sta prikazana na sliki 1. Naloga izdajatelja je izdaja podatkov v kriptografski preverljivi obliki, kar pomeni, da so digitalno podpisani. Vsi podatki, ki so potrebni za delovanje sistema, na primer javni ključi za verificiranje digitalnih podpisov in register, ki hrani status digitalnih dokumentov (v obliki ZgoščenaVrednost(dokument) → veljaven ali neveljaven), so shranjeni v zaupanja vrednem preverljivem podatkovnem registru (angl. verifiable data registry, VDR), ki ga navadno predstavlja decentralizirano omrežje

tehnologije veriženja blokov, s katerim je mogoče implementirati sistem brez centralne točke. Izdajatelji izdajo podatke imetniku, ki jih navadno hrani pri sebi v lokalni hrambi ali šifrirano na spletnem strežniku oz. oblačni storitvi. S takšnim upravljanjem in hrambo podatkov ima le imetnik dostop do svojih podatkov. Imetnik lahko nato sam deli podatke s preveriteljem, pri čemer lahko slednji preveri verodostojnost podatkov brez neposredne interakcije z izdajateljem, saj lahko pridobi informacije o izdajatelju, tj. njegove javne ključe oz. digitalni certifikat X.509, iz skupnega podatkovnega registra VDR. Pri deljenju podatkov s preveriteljem, lahko imetnik tudi izbere le njihovo podmnožico, kar se imenuje izbirno razkritje (angl. selective disclosure), ki v kombinaciji z Zero Knowledge Proof-i (ZKP) omogoča razkritje in dokaz o lastništvu podatkov ter dokazovanje predikatov, na primer starejši od 18 let, brez razkritja njihove dejanske vrednosti in digitalnega podpisa izdajatelja [15, 37]. Osnovni komponenti, ki sestavljata SSI, in skrbita za standardno strukturo oz. obliko podatkov so decentralizirani identifikatorji (angl. decentralized identifier, DID) in preverljive overilnice (angl. verifiable credential, VC).



Slika 1: Model zaupanja pri digitalni identiteti SSI.
Vir: lasten

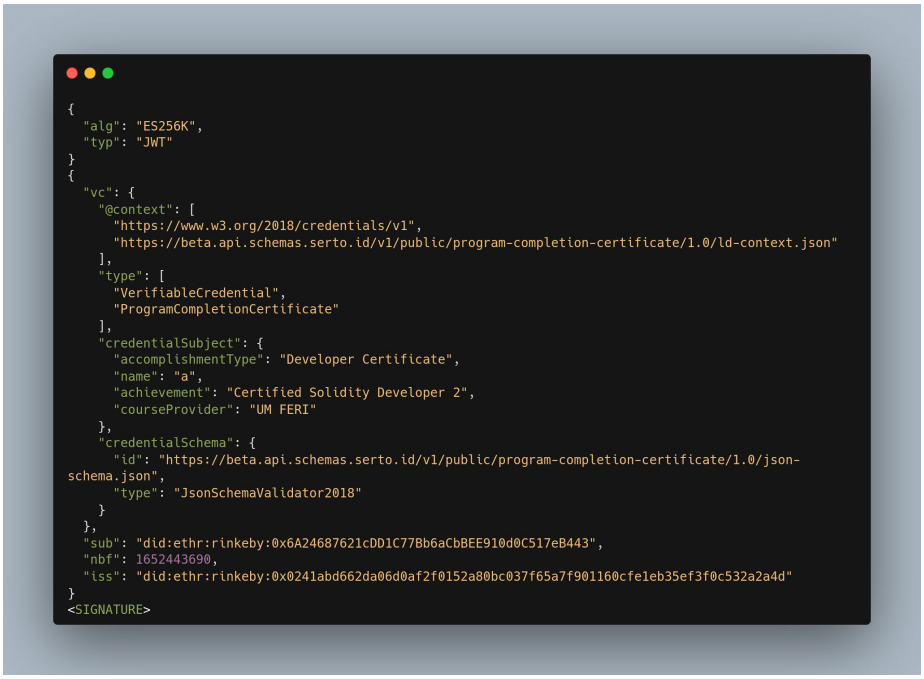
2.4.1 DID

DID je nova oblika identifikatorja, ki temelji na Uniform Resource Identifier (URI) in enolično identificirajo vsako entiteto v SSI sistemu [4]. Primer DID je did:ebsi:zvHWX359A3CvfJnCYaAiAde, kjer prvi del did definira, da URI prestavlja decentraliziran identifikator, drugi del, imenovan DID metoda (angl. DID method), pove kateri VDR se uporablja za hrambo podatkov, v tem primeru omrežje European Blockchain Services Infrastructure (EBSI), in tretji del predstavlja unikaten identifikator v tem omrežju [16]. Entitete pri ustvarjanju digitalne identitete registrirajo svoj identifikator v omrežje z DID dokumentom (angl. DID document). DID sam po sebi navadno ne hrani podatkov, na primer javnega ključa entitete, ampak ga je potrebno razrešiti (angl. DID resolution) s klicem na VDR. Ta klic vrne DID dokument, navadno v obliki JavaScript Object Notation Linked Data (JSON-LD), ki vsebuje vse potrebne informacije o entiteti za verificiranje njenih digitalnih podpisov. JSON-LD je podatkovni format, ki poleg podatkov v obliki JSON vsebuje tudi polje @context, s katerim se sklicuje na opis semantičnih definicij posameznih polj JSON-a. To omogoča interoperabilnost digitalnih dokumentov različnih podatkovnih shem.

Samo entiteta, ki ima komplementarni ključ, tj. zasebni ključ, od tistega v DID dokumentu, se lahko predstavi kot lastnik DID identifikatorja. DID dokument lahko entiteta tudi spremeni, kar se pogosto uporablja za rotacijo ključev (angl. key rotation). Za različne namene je bilo ustvarjenih več DID metod, na primer did:ebsi, did:ethr, did:key in did:peer, pri čemer vsaka služi svojemu namenu in vse sledijo W3C specifikaciji [4].

2.4.2 VC

Podatkovni model VC definira skupen splošen standard za vse digitalne dokumente [25]. Specifikacijo prav tako razvija W3C [27]. VC model sledi temeljem SSI. Njegova prednost je strojno berljiva oblika podatkov, saj se v specifikaciji priporoča format JSON, kateremu lahko natančno določimo strukturo z JSON shemo. Medtem ko DID predstavlja identifikator osebe, VC predstavlja dejanske podatke in informacije o entiteti. VC navadno vsebuje naslednje podatke: identifikator lastnika podatkov, identifikator izdajatelja, tip digitalnega dokumenta, jedrne informacije za ta tip dokumenta in digitalni podpis izdajatelja. VC-ji so najpogosteje shranjeni v digitalnih denarnicah uporabnika, kjer so uporabniku in le njemu cel čas na voljo ter samo on jih lahko deli z drugimi entitetami. Uporabnik pri delitvi VC-ja preveritelju generira preverljivo predstavitev (angl. verifiable presentation, VP), ki lahko vsebuje vse ali le podmnožico podatkov v VC. Preveritelj lahko verificira podatke z resolucijo DID identifikatorja izdajatelja, s čimer pridobi njegov javni ključ, in preverjanjem digitalnega podpisa. Jedrni podatki VC-jev so navadno v obliki JSON-LD, tako kot DID dokumenti. Toda digitalni podpis dokumenta se lahko doda na različne načine, pri čemer sta najpogostejši obliki Json Web Token (JWT) in JSON Linked Data Proofs (JSON-LD Proofs). Slika 2 prikazuje primer VC-ja v obliki JWT.



Slika 2: Primer preverljive overilnice v formatu JWT.
Vir: lasten.

Za celotno delovanje SSI ekosistema je nujno potrebnih še veliko ostalih standardov. Za pošiljanje in izmenjavo VC ter VP med vsemi akterji se uporabljajo različni protokoli, na primer DIDComm ali Self-Issued OpenID Connect Provider (SIOP OIDC), ki definirajo zaporedje in strukturo sporočil potrebnih za varno izmenjavo dokumentov [5, 22]. Ena izmed najpomembnejših komponent so digitalne denarnice, pogosto imenovani tudi agenti, ki hranijo zasebne kriptografske ključe in digitalne dokumente uporabnikov. Primeri takšnih denarnic so uPort, Gataca in Lissi [38]. Denarnice so navadno samostojne (mobilne) aplikacije, ki implementirajo vse potrebne operacije in protokole za SSI. Poleg denarnic se razvija tudi več ogrodij za enostavnejše razvijanje aplikacij temelječih na SSI, pri čemer ima vsako določene prednosti, na primer boljša podpora za integracijo v mobilne

aplikacije in razvoj strežniške aplikacije. Ena izmed najbolj uporabljenih orodij so Veramo[3], Hyperledger Aries[4], DID JWT VC[5] itd.

3 Razširitve MetaMask

Denarnica MetaMask [19] ponuja veliko število funkcionalnosti in načinov uporabe, vendar razvoj novih funkcionalnost poteka počasneje od razvoja ekosistema, v katerem se uporablja. Ta težava se lahko reši z razvojem in implementacijo lastne denarnice, kar pa je precej zahteven in dolgotrajen proces. MetaMask v te namene nudi popolno virtualizirano izvajalno okolje, v katerem se lahko izvajajo razširitve, ki razširijo funkcionalnost osnovne kripto denarnice. Izvajalno okolje, ki ga uporabljajo, je podnabor programskega jezika Javascript imenovan Secure EcmaScript (SES)[6], ki ga razvija podjetje Agoric[7].

3.1 Snaps

Najpopularnejša kripto denarnica, MetaMask, je s sistemom Snaps omogočila varen razvoj razširitev. Ta sistem omogoča razvoj poljubne razširitve, od preprostih aplikacij, ki omogočajo shranjevanje podatkov v stanje denarnice MetaMask, do kompleksnejših aplikacij, ki omogočajo podporo za druge verige blokov. Nekaj takšnih Snap-ov je bilo že implementiranih, med njimi Btcsnap [12], Solsnap [13], ter Filsnap [14]. Kot je razvidno iz imen, ti Snap-i omogočajo interakcijo z omrežji Bitcoin, Solana in Filecoin. Seznam vseh implementiranih Snap-ov se nahaja v aplikaciji Awesome Snaps [10].

MetaMask Snaps je JavaScript program, ki se izvaja v izoliranem okolju znotraj denarnice MetaMask. Ob obstoječih MetaMask metodah za oddaljen klic postopka (angl. remote procedure call, RPC) lahko s Snap-i ustvarimo nove RPC metode, katere lahko uporabijo dApp-i za nove funkcionalnosti. Trenutno je razvoj Snap-ov podprt samo v denarnici MetaMask Flask, namenjeni razvijalcem, s ciljem, da se v bližnji prihodnosti integrirajo v glavno denarnico MetaMask.

4 Dodajanje funkcionalnosti elektronskega podpisovanja poverilnic

4.1 Zasnova

Uporaba tehnologij VC omogoča številne inovativne rešitve za obstoječe probleme. Preprost primer takšnega problema je preverljiv dokaz polnoletnosti za namen vhoda v diskoteko. Ce želi Anita danes dokazati polnoletnost, bo morala pokazati svoj osebni dokument. Iz več razlogov to ni idealna rešitev. Anita lahko pokaže izposojen ali celo ponarejen osebni dokument, kar ni nujno, da varnostnik pred diskoteko tudi opazi. Prav tako pridobi varnostnik iz Anitinega osebnega dokumenta več podatkov, kot je dejansko potrebnih. Pri dokazu polnoletnosti ni potreben EMŠO, naslov bivališča ali točen datum rojstva. Potrebno je samo dejstvo, da je Anita starejša od 18 let ali ne.

Ta banalen problem lahko rešimo z uporabo tehnologije SSI. Z uporabo VC-ja in svojega DID-a, lahko Anita dokaže, ne samo da je ona lastnica tega VC-ja, temveč tudi da je polnoletna, brez da varnostniku izda več osebnih podatkov. Pristnost in veljavnost tega VC-ja lahko prodajalec preveri kadarkoli in kjerkoli. S trenutnimi tehnologijami takšne rešitve ne bi mogli implementirati. Entiteta, ki je izdala VC, bi morala biti dostopna pri

[3] https://veramo.io/
[4] https://www.hyperledger.org/use/aries
[5] https://github.com/decentralized-identity/did-jwt-vc
[6] https://github.com/endojs/endo/tree/master/packages/ses
[7] https://agoric.com/

verifikaciji podatkov, kar pri uporabi tehnologije SSI ni potrebno. Prav tako veliko prednost tehnologije SSI predstavlja zasebnost in varnost podatkov. Izdajalec VC-jev ne vidi, kje so podatki kasneje uporabljeni. Uporabnik ima popoln nadzor nad tem, s kom deli VC, in katere podatke iz samega VC-ja bo delil.

Žal je uporaba tehnologije SSI trenutno zelo zahtevna. Če želi uporabnik uporabiti SSI koncepte, mora namestiti dodatno aplikacijo. Zaupati mora, da aplikacija zagotavlja varnost podatkov in kriptografskih ključev. Trenutno še ne obstaja web3 aplikacija, ki bi omogočala preprosto in varno uporabo DID-ov ter VC-jev in hkrati služila kot kripto denarnica. Sicer obstaja ogromno denarnic za kripto valute, vendar nobena ne omogoča uporabe tehnologije SSI. Z željo, da bi omogočili uporabo tehnologije SSI milijonom obstoječih MetaMask uporabnikov, smo se lotili razvoja t. i. SSI Snap-a. Pri snovanju SSI Snap-a smo si zamislili preprost primer poteka delovanja; VC ponudnik izda VC izbranemu računu v denarnici MetaMask. Ta VC je nato s pomočjo RPC metod SSI Snapa shranjen v stanje MetaMask denarnice. Po potrebi lahko uporabnik s pomočjo RPC metod z VC-jem generira VP in ga posreduje aplikaciji, ki ga zahteva.

S prej omenjeno tehnologijo MetaMask Snaps smo se lotili razvoja aplikacije SSI Snap. Vendar smo pred samim začetkom implementacije morali razrešiti dilemo, katero DID metodo uporabiti. Vsak uporabnik mora biti namreč v sistemu unikatno identificiran in imeti popoln nadzor nad svojo identiteto. V SSI svetu se ta identiteta imenuje DID. Kot smo prej omenili, DID temelji na DID metodi, ki pove, kje in kako so DID dokumenti ustvarjeni, razrešeni, posodobljeni ter deaktivirani.

Na trgu je trenutno že veliko DID metod. Ena najpopularnejših se imenuje ”did:ethr” [11]. Ta metoda uporabi obstoječe Ethereum naslove (angl. Ethereum address) kot samostojne DID-e. Z drugimi besedami, vsak Ethereum račun je DID, kjer se ta začne s prepono ”did:ethr:<omrežje>:”, ter nadaljuje s samim Ethereum naslovom. V praksi to pomeni, da že imajo vsi uporabniki denarnic MetaMask svoje DID-e, katerim manjka le funkcionalnost za pravilno uporabo in izkoriščanje njihovega potenciala. Spremembe DID dokumentov se objavijo in hranijo na verigi blokov Ethereum. Za to verigo blokov smo se odločili zaradi več razlogov:

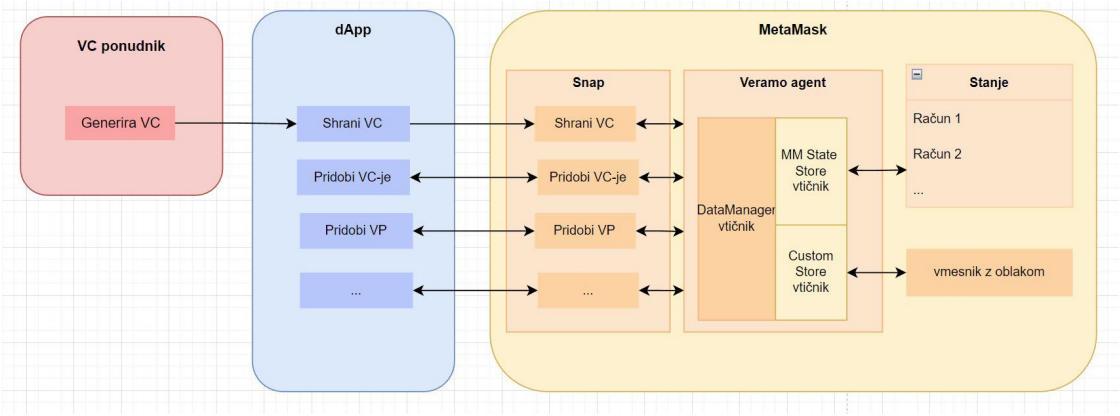
- ob Bitcoin-u je Ethereum najbolj decentralizirana veriga blokov,
- je najpopularnejša in najbolj uporabljena veriga blokov,
- ima veliko skupnost razvijalcev in veliko preizkušenih orodij, med katere spadajo tudi številna SSI orodja ter DID metoda did:ethr,
- v praksi se DID dokumenti spreminjajo zelo redko, zato pristojbine (angl. fees) za njihove spremembe na verigi blokov Ethereum niso visoke,
- denarnica MetaMask ima najboljšo podporo za Ethereum.

Metodo did:ethr bi sicer lahko zamenjali s katerokoli drugo metodo, npr. z did:ebsi, ki se uporablja za razvoj digitalne identitete na Evropskem trgu.

4.2 Arhitektura nadgrajene kripto denarnice

Jedro SSI Snap-a sestavlja ogrodje (angl. framework) Veramo [24], ki je zmogljiv in modularen API za preverljive podatke (angl. verifiable data) in SSI. Z ogrodjem razvijamo agenta, ki omogoča ustvarjanje in uporabljanje DID-ov, VC-jev in VP-jev. Razvijalci ogrodja Veramo so implementirali tudi več vtičnikov (angl. plugins) imenovane DIDManager, KeyManager, ter KeyPairManager, ki zagotavljajo istoimenske funkcionalnosti. Agent Veramo je prav tako zmožen validirati VC-je glede na podano shemo in ustvariti VP-je iz podanih VC-jev. Ker za ogrodje Veramo ne obstaja vtičnik, ki bi omogočal shranjevanje VC-jev, smo ta vtičnik implementirali sami. Implementirali smo razširljiv vtičnik imenovan Veramo VC Manager, ki omogoča upravljanje z VC-ji na podoben način kot delujejo vtičniki DIDManager, KeyManager ter KeyPairManager. Jedro našega vtičnika predstavlja abstrakten razred s funkcijami *import*, *get*, *delete* in *list*. Razvili smo tudi vtičnik imenovan SnapDataStore, ki implementira prej

omenjen abstraktni razred z zmožnostjo shranjevanja podatkov v stanje denarnice MetaMask. Razširljivost teh vtičnikov predstavlja veliko prednost za nadaljnje implementacije, kot je npr. shranjevanje podatkov v oblaku. Končna arhitektura SSI Snap-a je prikazana na sliki 3.



Slika 3: SSI Snap arhitektura.
Vir: lasten

SSI Snap shranjuje DID-e, VC-je in pare ključev (angl. keypairs) v stanje denarnice MetaMask. Pri prvi spremembi stanja se ustvari objekt imenovan SSISnapState. V ta objekt so dodani novi objekti, poimenovani po naslovu računa MetaMask, ki jih je ta naslov ustvaril. Znotraj teh objektov se hranijo DID-i, VC-ji, in pari ključev, kot je prikazano na sliki 4.

```
{
  ...
  objectCreatedByOtherSnaps,
  SSISnapState:
    {
      0xBea807A8...e59D:
        {
          snapKeyStore: Record<string, IKey>,
          snapPrivateKeyStore: Record<string, IKey>,
          identifiers: Record<string, IIdentifier>,
          vcs: VerifiableCredential[]
        },
      0x8Db2a08D...caD7:
        {
          snapKeyStore: Record<string, IKey>,
          snapPrivateKeyStore: Record<string, IKey>,
          identifiers: Record<string, IIdentifier>,
          vcs: VerifiableCredential[]
        },
       ...,
    },
}
```

Slika 4: Arhitektura stanja denarnice MetaMask.
Vir: lasten

Pri implementaciji SSI Snap-a se je pojavilo vprašanje, kako izvesti podpisovanje VP. Edini način, da bi VP podpisali z računom MetaMask, bi zahteval izvoz zasebnega ključa, saj MetaMask ne ponuja potrebnih metod za

podpisovanje VP-jev. Zaradi varnostnih pomislekov z omenjenim pristopom smo se odločili, da namesto izvoza zasebnega ključa ustvarimo nov DID. Vloga tega DID-a bo izključno podpisovanje VP-jev. Toda tukaj se pojavijo dodatni izzivi. DID lahko uporabi samo VC-je, ki si jih lasti, in ker so v večini primerov VC-ji izdani DID-u računa MetaMask, je podpis novega DID-a neveljaven. Ta problem rešimo tako, da novemu DID-u dodelimo pravico podpisovanja DID-ov od računa MetaMask. To storimo tako, da DID-u od računa MetaMask s pomočjo knjižnice ethr-did ustvarimo nov delegat. Vse omenjene akcije se zgodijo znotraj SSI Snap-a.

4.3 Validacija

S prototipno platformo smo želeli prikazati delovanje SSI Snap-a. Ustvarili smo platformo, na katero se lahko uporabnik poveže s svojo denarnico MetaMask, namesti oziroma posodobi SSI Snap, opravi test imenovan Solidity Developer Course in po uspešni opravitvi tega testa prejme VC. Ta VC vsebuje podatke, da je lastnik povezane denarnice MetaMask uspešno opravil test. Na tej platformi lahko uporabnik tudi izpiše seznam vseh shranjenih VC-jev. Implementirali smo tudi "skrivno sobo", v katero lahko uporabnik dostopa samo z veljavnim VP-jem. Demo se lahko brezplačno preizkusi[8]. Za namen testiranja se mora uporabiti MetaMask Flask (različica >10.18.0) in imeti nekaj ETH na testnem omrežju Rinkeby. Platforma je sestavljena iz obličja (angl. frontend), ki komunicira z zalednim delom (angl. backend), ki skrbi za VC-je in VP-je.

Uporabniški vmesnik, ki je prikazan na sliki 5, je implementiran s pomočjo ogrodja React, pri čemer je izgled strani implementiran s pomočjo ogrodja Tailwind CSS. Zaledje je implementirano s pomočjo ogrodja Express.js. Jedro zaledja predstavlja agent Veramo, ki je uporabljen za generiranje VC-jev in preverjanje VP-jev.

Slika 5: Uporabniški vmesnik platforme.
Vir: lasten

Za dostop do vsebine platforme je potrebna povezava z denarnico MetaMask. Ob prvotni povezavi aplikacija preveri, ali ima MetaMask že nameščen SSI Snap. V primeru, da ga nima, zaprosi uporabnika za njegovo namestitev. V primeru, da ima MetaMask nameščeno starejšo različico SSI Snap-a, bo posodobljena na najnovejšo verzijo. Po uspešni povezavi lahko uporabnik začne uporabljati platformo. V zavihku "Profile" lahko izpiše vse VC-je shranjene v stanju svoje denarnice MetaMask. V primeru, da nima še nobenega, lahko v zavihku "Course" opravi preprost Solidity test, in si pridobi VC, ki se generira na zaledju, kjer so tudi na varen način shranjeni kriptografski ključi izdajatelja. Ta VC vsebuje podatke, da je povezana denarnica MetaMask uspešno opravila test. Uporabnikom, ki so uspešno opravili test se pojavi nov zavihek "Secret Room", do katerega lahko dostopajo samo, če priložijo VP, ki vsebuje VC za opravljen Solidity test. Če je preveritev VP-ja na zaledju uspešna, bo uporabniku odobren dostop do skrivne sobe.

[8] https://blockchain-labum.github.io/course-dapp/

Za generiranje VC-jev sta na obličju potrebna ime uporabnika in DID povezanega račun. Ti podatki so nato posredovani zaledju, kjer se DID-u uporabnika izda VC, ki ga nato zaledje vrne obličju, da se shrani v denarnico MetaMask. Del vsebine izdanega VC-ja je prikazana na sliki 6. Pri preverjanju VP-jev mora obličje posredovati VP in Ethereum naslov povezanega računa. Pri preverjanju VP-ja zaledje preveri pravilnost JWT podpisov VP-ja in prisotnost vseh VCjev. Preveri se tudi, ali VP sploh vsebuje pravilen VC, in če je lastnik VC-ja res povezan račun MetaMask. Na koncu se še preveri, če je VP podpisal DID, kateremu je bil izdan VC. V primeru, da to ne drži, se preveri, ali je DID, ki je podpisal VP, prisoten v DID dokumentu od prejemnika VC-ja kot delegat. Če so izpolnjeni vsi pogoji, zaledje vrne odgovor, da je bil VP uspešno preverjen.

Podrobnejši opis prototipa je na voljo v repozitoriju GitHub[9].


```
"verifiableCredential": [
    {
        "credentialSubject": {
            "accomplishmentType": "Developer Certificate",
            "learnerName": "a",
            "achievement": "Certified Solidity Developer 2",
            "courseProvider": "https://blockchain-lab.um.si/",
            "id": "did:ethr:rinkeby:0x6A24687621cDD1C77Bb6aCbBEE910d0C517eB443"
        },
        "issuer": {
            "id": "did:ethr:rinkeby:0x0241abd662da06d0af2f0152a80bc037f65a7f901160c
        },
        "type": [
            "VerifiableCredential",
            "ProgramCompletionCertificate"
        ],
        "credentialSchema": {
            "id": "https://beta.api.schemas.serto.id/v1/public/program-completion-c
            "type": "JsonSchemaValidator2018"
        },
        "@context": [
            "https://www.w3.org/2018/credentials/v1",
            "https://beta.api.schemas.serto.id/v1/public/program-completion-certifi
        ],
        "issuanceDate": "2022-05-13T12:08:10.000Z",
        "proof": {
            "type": "JwtProof2020",
```


Slika 6: Del vsebine izdanega VC-ja.
Vir: lasten

5 Zaključek

V prispevku smo podrobno predstavili številne nove koncepte povezane s tehnologijo veriženja blokov oz. s konceptom decentralizacije. Izhajali smo iz trenutne kompleksnosti področja povezanega z digitalnimi (kripto) denarnicami, kjer so tako v raziskavah kot na trgu številne različice in tipi, pri čemer pa vsaka rešuje zgolj en del celote oz. zagotavlja zgolj specifične funkcionalnosti. Posledično smo ugotavljali, kako je kripto denarnica MetaMask de-facto standardna na programski opremi temelječa kripto denarnica, ki pa kot rečeno ponuja zgolj omejen nabor funkcionalnosti, povezane z verigami blokov oz. zamenljivimi in nezamenljivimi žetoni itn. Prav tako smo predstavili novo področje upravljanja digitalnih identitet, ki prav tako izhaja iz tehnologije veriženja blokov in temelji na decentralizaciji, tj. samo-upravljane identitete oz. SSI. Tako smo identificirali izziv, kako zagotoviti hitrejše sprejemanje novih tehnologij, kot je SSI, ki prav tako zahteva določeno vrsto digitalnih denarnic. Podrobneje smo izziv razvili na način, da smo iskali rešitev za nadgradnjo priljubljene kripto denarnice MetaMask s funkcionalnostmi, ki bi podprle zahteve, ki jih prinaša tehnologija SSI. Izziv smo rešili na način, da smo uspešno načrtovali in tudi razvili rešitev, ki s pomočjo t. i. Snap-ov razširi funkcionalnosti kripto denarnice MetaMask. Na

[9] https://github.com/blockchainlab-um/ssi-snap

ta način smo s kompleksnim primerom tudi demonstrirali, kako bi se lahko na osnovi sistema Snap MetaMask poljubno nadgradil.

Predstavljeni in validirani koncepti odpirajo pot do novih razširitev MetaMask, kakor tudi do reševanja drugih izzivov, kot je npr. glasovanje v decentralizirano avtonomnih organizacijah (angl. decentralized autonomous organisation, DAO) zgolj na osnovi količine žetonov, ki si jo posameznik lasti, kar samo po sebi predstavlja t.i. plutokratičen problem. V prihodnosti tako načrtujemo nadgraditi SSI Snap ter platformo Snapshot na način, da bosta omogočali uporabnikom, da z eno kripto denarnico upravljajo tudi DAO glasovanje, pri čemer pa se bo le to izvedlo zgolj v primeru, ko uporabnik na osnovi primerih VC/VP-jev pokaže svojo kompetentnost na področju tematike glasovanja.

Literatura

[1] BUTERIN Vitalik »Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform«.

[2] ČUČKO Špela, TURKANOVIĆ Muhamed »Decentralized and self-sovereign identity: Systematic mapping study«, IEEE Access 9, 2021, pp. 139009-139027.

[3] DANNEN Chris »Introducing Ethereum and solidity«, Springer, številka 1, 2017,

[4] https://www.w3.org/TR/did-core, Decentralized Identifiers (DIDs) v1.0, obiskano 1. 7. 2022.

[5] https://identity.foundation/didcomm-messaging/spec, DIDComm Messaging Specification, obiskano 1. 7. 2022.

[6] https://identity.foundation, DIF – Decentralized Identity Foundation, obiskano 1. 7. 2022.

[7] https://eips.ethereum.org/EIPS/eip-191, EIP-191: Signed Data Standard, obiskano 7. 7. 2022.

[8] https://eips.ethereum.org/EIPS/eip-4361, EIP-4361: Sign-In with Ethereum, obiskano 7. 7. 2022.

[9] https://eips.ethereum.org/EIPS/eip-712, EIP-712: Ethereum typed strucuted data hashing and signing, obiskano 7. 7. 2022.

[10] https://github.com/piotrroslaniec/awesome-metamask-snaps, Github page for Awesome Snaps, obiskano 7. 7. 2022.

[11] https://github.com/uport-project/ethr-did, Github page of did:ethr, obiskano 7. 7. 2022.

[12] https://github.com/KeystoneHQ/btcsnap, Github page of btcsnap, obiskano 7. 7. 2022.

[13] https://github.com/cavanmflynn/solsnap, Github page of solsnap, obiskano 7. 7. 2022.

[14] https://github.com/ChainSafe/filsnap, Github page of filsnap, obiskano 7. 7. 2022.

[15] GOLDREICH Oded, OREN Yair »Definitions and properties of zero-knowledge proof systems«, Journal of Cryptology 7.1, 1994, pp. 1-32.

[16] https://ec.europa.eu/digital-buildingblocks/wikis/display/ebsi, Home – EBSI, obiskano 1. 7. 2022.

[17] https://docs.metamask.io/guide, Introduction – MetaMask Docs, obiskano 1. 7. 2022.

[18] https://ledger.com, Ledger hardware wallet, obiskano 7. 7. 2022.

[19] https://metamask.io, MetaMask, obiskano 7. 7. 2022.

[20] https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki, Mnemonic code for generating deterministic keys, obiskano 7. 7. 2022.

[21] PREUKSCHAT Alex, REED Drummond »Self-sovereign identity«, Manning Publications, 2021.

[22] https://openid.net/developers/specs, Specifications – OpenID, obiskano 1. 7. 2022.

[23] https://trezor.io, Trezor hardware wallet, obiskano 7. 7. 2022.

[24] https://veramo.io, Veramo – A JavaScript Framework, obiskano 7. 7. 2022.

[25] https://www.w3.org/TR/vc-data-model, Verifiable Credentials Data Model v1.1, obiskano 1. 7. 2022.

[26] https://walletconnect.com, WalletConnect web3 standard, obiskano 7. 7. 2022.

[27] https://www.w3.org, World Wide Web Consortium (W3C), obiskano 1. 7. 2022.

[28] YAGA Dylan, MELL Peter M., ROBY, Nick, SCARFONE Karen, »Blockchain Technology Overview«, NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD, 2018, https://doi.org/10.6028/NIST.IR.8202.

[29] KASPARS Zīle, STRAZDINA Renāte »Blockchain Use Cases and Their Feasibility«, Applied Computer Systems 23.1, 2018, pp. 12-20, https://doi.org/10.2478/acss2018-0002.

[30] NAKAMOTO Satoshi »A Peer-to-Peer Electronic Cash System, 2008.

[31] PODGORELEC Blaž, REK Patrik, STREHAR Miha, TURKANOVIĆ, Muhamed, HERIČKO Marjan (ur.), KOUS, Katja (ur.) »Vzpostavitev konzorcijskega omrežja Ethereum«, Sodobne informacijske tehnologije in storitve: OTS 2019: zbornik štiriindvajsete konference, Maribor, 18. in 19. junij 2019. 1. izd. Maribor: Univerzitetna založba Univerze, 2019. Str. 69-79. ISBN 978-961-286-283-1.

[32] SRIMAN B., KUMAR Ganesh S., PRABAKARAN Shamili »Blockchain Technology: Consensus Protocol Proof of Work and Proof of Stake«, Intelligent Computing and Applications, Proceedings of ICICA 20ver (pp.395-406), 2020, https://doi.org/10.1007/978-981-15-5566-4_34.

[33] https://apla.readthedocs.io/en/latest/concepts/consensus.html, Proof of Authority conesnsus, obiskano 7. 7. 2022.

[34] PODGORELEC Blaž, TURKANOVIĆ Muhamed, HERIČKO Marjan (ur.), KOUS Katja (ur.) »Implementacija nadgradljivosti in zamenljivosti pametnih pogodb na platformi Ethereum«, Sodobne informacijske tehnologije in storitve: OTS 2018: zbornik triindvajsete konference, Maribor, 19. in 20. junij 2018. 1. izd. Maribor: Univerzitetna založba Univerze: Fakulteta za elektrotehniko, računalništvo in informatiko, 2018. Str. 8-19. ISBN 978-961-286-163-6, ISBN 978-961-286-162-9.

[35] REK Patrik, TURKANOVIĆ Muhamed, HERIČKO Marjan (ur.), KOUS Katja (ur.) »Orodja za podporo celovitemu razvoju decentraliziranih aplikacij«, Sodobne informacijske tehnologije in storitve: OTS 2019: zbornik štiriindvajsete konference, Maribor, 18. in 19. junij 2019. 1. izd. Maribor: Univerzitetna založba Univerze, 2019. Str. 173-180. ISBN 978-961-286-283-1.

[36] KERŠIČ Vid, ŠTUKELJ Primož, KAMIŠALIĆ Aida, KARAKATIČ Sašo, TURKANOVIĆ Muhamed, PRIETO Javier (ur.) »A blockchain- and ai-based platform for global employability«, Blockchain and applications: international congress [on Blockchain and applications, BLOCKCHAIN 2019, held in Avila, Spain, from 26th to 28th June, 2019]. Cham [etc.]: Springer, cop. 2020. Vol. 1010, str. 161-168. Advances in intelligent systems and computing (Print), 1010. ISBN 978-3-030-23812-4, ISBN 978-3-030-23813-1. ISSN 2194-5357.

[37] PODGORELEC Blaž, TURKANOVIĆ Muhamed, HERIČKO Marjan (ur.), KOUS Katja (ur.) »ZKP (zero-knowledge proof) pod drobnogledom«, Sodobne informacijske tehnologije in storitve: OTS 2019: zbornik štiriindvajsete konference, Maribor, 18. in 19. junij 2019. 1. izd. Maribor: Univerzitetna založba Univerze, 2019. Str. 109-118. ISBN 978-961-286-283-1.

[38] ČUČKO Špela, ŠUMAK Boštjan, TURKANOVIĆ Muhamed, HÖLBL Marko (ur.) »Načrtovalski vzorci uporabniškega vmesnika samoupravljanih identitet«, "Digitalno desetletje: varno, zeleno in odporno": zbornik: 29. konferenca Dnevi slovenske informatike: Portorož, 11. in 12. maj 2022. 1. izd. Ljubljana: Slovensko društvo Informatika, 2022. 8 str., ilustr. ISBN 978-961-6165-59-4.

Prihodnost dobrodelnih organizacij – DECA (Decentralized Children's Art)

Katerina Petrevska, Petar Stojkovski, Blagoj Soklevski

Nevladna organizacija Decentralized Children's Art, Graz, Avstrija
katerina.petrevska@decanfts.org, petar.stojkovski@decanfts.org,
blagoj.soklevski@decanfts.org

Sinopsis Načela delovanja dobrodelnih organizacij so skoraj povsod po svetu podobna in ustaljena. Potrebna je reforma, ki bo doprinesla h kakovostnejšemu izvajanju dobrodelnih projektov ter učinkovitejšemu spremljanju porabe sredstev. V okviru organizacija "DECA – Decentralized Children's Art" smo realizirali model ideje, ki uporablja in prilagaja najnovejše tehnologije veriženja blokov za namen spremembe načina delovanja dobrodelnih organizacij. Glavni cilj je doseči transparentnost organizacij, kar ne zajema zgolj zbiranja donacij in sredstev, temveč tudi njihovo upravljanje. Vsak izmed izvedenih projektov bo sledil več podobnim korakom za zbiranje donacij, in sicer: organiziranje delavnic v ustanovah oziroma centrih za delo z otroki s posebnimi potrebami, podpora pri ustvarjanju umetniških del, digitalizacija njihovih umetnin in digitalizacija umetnin kot (2D/3D) NFT-ji. S takšnim pristopom želimo povečati udeležbo in integracijo otrok v proces zbiranja donacij.

Ključne besede:
veriga blokov
NFTs
dobrodelne organizacije
nevladne organizacije
preglednost
otroci v stisk
3D in 2D
modeliranje

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.13

1 Kako in zakaj

Prispevek pokriva in predstavlja trenutno stanje v nevladnih organizacijah in predviden napredek v prihodnost s pomočjo decentralizacije in večje preglednosti nad nevladnimi organizacijami, ki jih lahko zagotovi samo veriga blokov. Predvsem bo govora o trenutnem stanju nevladnih organizacij ter o njihovih pomanjkljivostih in manjkajoči transparentnosti. Povzeta bo tudi trenutna analogna izvedba našega predloga uporabe verige blokov in NFT-jev. Poleg tega bodo predstavljeni kratki osnutki iz »white paper«-ja, ki pojasnjujejo decentralizirano rešitev za prihodnost nevladnih organizacij.

Napačno in neprimerno ravnanje nevladnih organizacij ter z njimi povezanimi sredstvi je pogost problem sodobnega razvoja dobrodelnih organizacij. Zajčja luknja je pregloboka, da bi se vanjo lahko potopili, glavne razloge za to pa predstavljajo pomanjkanje preglednosti in jasnosti porabe sredstev. Poleg tega načrtovanje projekta, razvoj, dejanski stroški in izvedba niso zmeraj javno dostopni, zaradi česar so dobrodelne organizacije lahki cilj za pranje denarja in zlorabo sredstev. Tehnologija veriženja blokov omogoča decentralizacijo omenjenih organizacij, pri čemer lahko le-to uporabimo za vpeljavo še kako pomembnega nivoja preglednosti. Cilj projekta DECA je izkoristiti prednosti decentraliziranih tehnologij, izboljšati delovanje dobrodelnih organizacij in, najpomembneje, ponuditi pomoč otrokom v stiski.

Vsak dobrodelni projekt se izvede in razvije na sledeč način, s slednjem nekaj standardnim načelom [1]:

- Iskanje ciljnega območja (lokacija, sektor, upravičenci)[1].
- Določitev časovni načrt.
- Določanje jasnih končnih izdelkov (rezultat in učinek).
- Izdelava podrobnega načrta aktivnosti (odgovorne osebe, urnik aktivnosti)[2].
- Dogovarjanje o fiksnem proračunu.

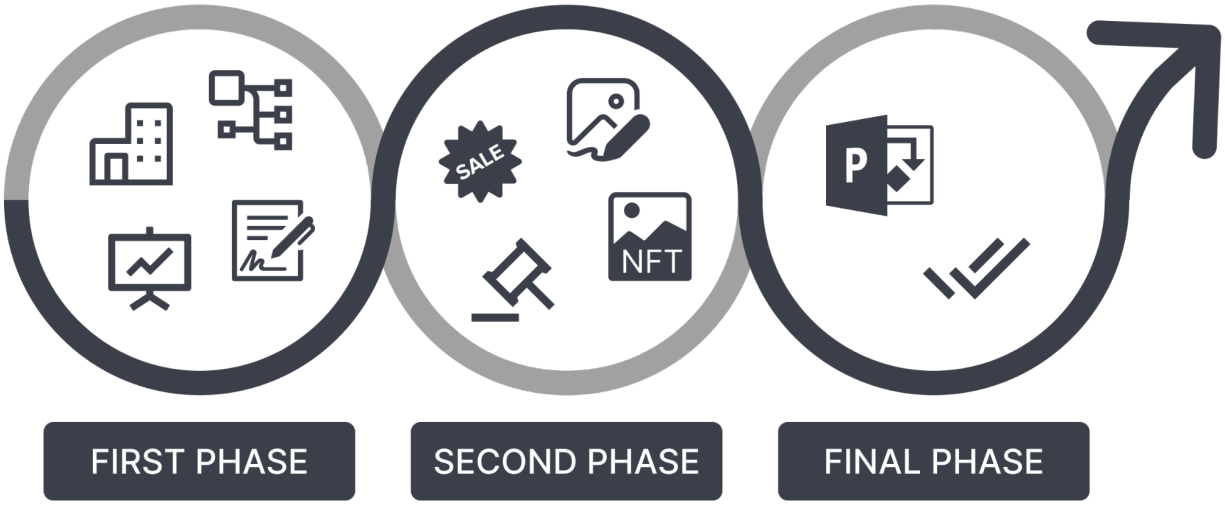
Načrtovanje in razvoj projekta lahko sicer odstopa od teh načel, vendar je pomebno, da se zmeraj zasleduje cilj. Odstopanje je lahko posledica različnih izzivov v okviru projektov, zato je prepoznavanje ključno in zelo pomembno težave projekta. Takšni podatki se navadno ne delijo z javnostjo in včasih le delno z deležniki[3] [4], zato mora biti preglednost uporabe in porabe sredstev nevladni organizaciji javno dostopna.

Nevladna organizacija DECA ima jasen cilj – ponuditi pomoč prikrajšanim in otrokom v stiski, s predpostavko da je cilj projekta jasen, poraba sredstev pa je popolnoma pregledno. Zagotovljena bo različna podpora otrokom, od šolskih in potrebščin za izobraževanje, do obnove in opremljanja ustanov za delo z otroki v stiski. Poleg tega bo majhen del ekipe poskušal pomagati družinam, ki potrebujejo finančno podporo za zdravljenje in okrevanje svojih otrok ter za stvari, ki niso pokrite z osnovnim zdravstvenim zavarovanjem.

2 Življenjski cikel projekta

Vsak projekt pod okriljem organizacije DECA se vodi in razvija na drugačen način, vendar pa vsaka sodelujoča organizacija opravi nekaj sekvenčnih korakov, ki vodijo k opredelitvi projekta. Vodenje projektov v DECA je razdeljeno v tri faze, kot prikazano na sliki 1.

[1] Npr. iskanje institucije za sodelovanje.
[2] Npr. Organizacija delavnic, iskanje orodij za vodenje, sodelavcev.
[3] Deležniki so osebe, skupine ali ustanove, ki imajo interese v projektu ali programu. Primarni deležniki so tisti, ki so končno prizadeti, bodisi pozitivno (upravičenci) ali negativno (na primer tisti, ki so bili neprostovoljno preseljeni). Sekundarni deležniki so posredniki v procesu dostave pomoči. Ta definicija deležnikov vključuje zmagovalce in poražence ter tiste, ki so vključeni ali izključeni iz procesov odločanja. Ključni deležniki so tisti, ki lahko vplivajo ali so pomembni za uspeh projekta [5].



Slika 1: Življenjski cikel projekta.
Vir: lasten.

2.1 Načrtovanje projekta

Kot omenjeno, je prvo in najpomembnejše načelo pri implementaciji projekta iskanje specifičnega ciljnega področja, tj. institucije. Po iskanju primerne ustanove, ki deluje na področju dela z otroki v stiski in/ali prikrajšanimi otroki se opravi temeljita raziskava o potrebah otrok in ustanove ter začrta projekt, ki vključuje tako obseg kot pričakovanja projekta. Oblikovan časovni načrt in projektne napovedi morajo biti del predloga projekta, z namenom oblikovanja jasne vizije in ciljev projekta. V primeru, da sta obe strani zadovoljni s projektnim predlogom, se podpiše pogodba. V tej fazi je treba opredeliti točke, kot so načrtovanje delavnice, priprava obrazcev za soglasja za starše/skrbnike, izbira med analogno in digitalno umetnostjo ter, če so potrebna obnovitvena/gradbena dela ali opremljanje, kontaktiranje potencialnih sodelavcev in poslana povpraševaja.

2.2 Razvoj projekta

Eden izmed ciljev naše nevladne organizacije je tudi spodbujanje otroške umetnosti ter motivacija za razvoj njihovih ustvarjalnih sposobnosti. Za zbiranje sredstev je torej ključnega pomena organiziranje delavnic, kjer se otroci lahko razigrajo s svojo domišljijo. Da se otroci lahko organiziranih delavnic udeležijo, morajo starši ali skrbniki pred vsako delavnico podpisati soglasje za sodelovanje[4].Najnovejši projekt DECA je analognen, kar pomeni, da je bila umetnost izdelana ročno brez uporabe digitalnih tehnologij ali orodij.

Ko so umetnine otrok enkrat končane, naslednji korak predstavlja njihova digitalizacija. Le to prevzamejo strokovnjaki za digitalizacijo, ki izdelajo 2D in 3D modele[5].

Se pa pri načrtovanju prihodnost za DECA razvija nova digitalno rojena umetniška kampanja, kjer bo umetnost ustvarjena s pomočjo digitalnih orodij, kot so tablice in iPadi. Ko bodo modeli pripravljeni, bodo otroške umetnine kot NFT ponujene na NFT tržnice. NFT-ji bodo ponujeni s fiksno ceno ali prodani s pomočjo dražb. Ko je cilj projektnega sklada dosežen, se lahko začne zadnja faza projekta.

[4] To je še posebej pomembno pri delu z otroki s posebnimi potrebami, saj potrebujejo posebno nego in podporo.
[5] Uporabljajo se orodja: Blender, Agisoft, Meshlab, Meshmixer itd.

Slika 2: Handcrafted vs. 3D model.
Vir: lasten.

2.3 Izvedba projekta

Ko se zbiranje sredstev zaključi, bodo izbrane najboljše ponudbe za porabo sredstev. Skupnost bo izbrala podjetje oziroma ponudbo za izvedbo projekta. Projekt se bo štel za zaključenega, ko bodo vsi cilji uspešno zajeti in bo celotna projektna dokumentacija, vključno s transakcijami in kontakti, objavljena na spletni strani.

3 Uporaba tehnologij

Veriga blokov[6] je sistem beleženja informacij na način, ki oteži, oziroma je (skoraj) nemogoče spremeniti, vdreti ali manipulirati s podatki. Blockchain tehnologija je obetavna, predvsem zavoljo njene decentralizirane narave izmenjave informacij med neodvisnimi vozlišči. Ponuja trajno sledenje in zgodovino transakcij, ki so še posebej pomembne za uspeh tega vsakega projekta.

Spreminjanje podatkov in vnosov v verigi blokov ni 100% nemogoče. Ko so podatki zabeleženi v verigi blokov, jih je izjemno težko spremeniti ali odstraniti, vendar ne nemogoče. Da bi spremenili informacije mora to spremembo podpreti velik del deležnikov [2]. Ker transakcije kot vnosa v verigi blokov ni mogoče spremeniti ali manipulirati, to nakazuje, da je implementacija verig blokov v nevladnih organizacijah zelo primerna za sledenje porabi sredstev, vodenju projekta, izsledkih projekta, pa tudi pri lažjem sledenju odgovornosti in analizi deležnikov. Vsaka donacija za projekte DECA ali katero koli nevladne organizacije se prenese v »multi-signature wallet«[7], kjer mora najmanj 60% članov nevladne organizacije odobriti vsako odhodno transakcijo.

Naj pojasnimo še način zbiranja donacij in celotno idejo nevladne organizacije. Kot je opisano v poglavju »Razvoj projekta«, se sredstva zbirajo s prodajo NFT-jev, bodisi s prodajo po fiksni ceni ali dražbo. NFT ali Non-Fungible Token so kriptografska sredstva v verigi blokov z edinstvenimi identifikacijskimi kodami in metapodatki, po katerih se razlikujejo drug od drugega[6]. Lahko predstavljajo predmete iz resničnega sveta, kot so umetnine, in so unikatne entitete, ki obstajajo v verigi blokov. Kupiti in prodati jih je mogoče prek spleta s kriptovalutami na NFT tržnicah, kot so npr. OpenSea, Rarible, LooksRare, Singular ali podobni.

---

6 "Blockchain je P2P sistem brez osrednjega organa, ki bi upravljal pretok podatkov. Eden ključnih načinov za odstranitev centralnega nadzora ob ohranjanju celovitosti podatkov je imeti veliko porazdeljeno omrežje neodvisnih uporabnikov. To pomeni, da so računalniki, ki sestavljajo omrežje, na več kot eno lokacijo." [2].

7 Denarnica z več podpisi je v primerjavi s posamezno denarnico z enim zasebnim ključem in podpisom denarnica za digitalna sredstva, ki potrebuje pooblastilo za transakcijo več imetnikov zasebnih ključev.

Ob začetku ukvarjanja z analogno realnostjo, kjer se večina dela opravi ročno, na primer digitalizacija in kovanje NFT-jev, se tudi upravljanje, vodenje in načrtovanje projektov in sredstev izvajata tudi ročno. Da bi izboljšali hitrost izvajanje projektov in dnevno rutino, je na obzorju nov interni projekt, ki bo razvil DApp in vstopil v svet Web3 z aplikacijo, ki bo digitalizirala celoten proces. Naslednji razdelek bo razdeljen na dva dela, ki zajemata funkcionalni obseg aplikacije DApp in tehnični sklad, ki bo uporabljen[8].

3.1 Funkcionalni obseg

Blockchain kot tehnologija je še vedno v zgodnji fazi sprejemanja. V zadnjem času je postala zanimiva za institucionalne vlagatelje, vendar je še zmeraj, že osnovna operacija, kot npr. ustvarjanje kripto denarnic in delo s kriptovalutami med ljudmi težko dosegljivo. Glede na to, da bo naša nova platforma osredotočena na uporabnika bo celotno ročno delo izvedeno v DApp. Poseben primer je zagotavljanje funkcionalnosti ustvarjanja kripto denarnice znotraj DAppa in odprava potrebe po povezovanju z zunanjo denarnico.

Naslednje alineje predstavljajo celoten funkcionalni obseg platforme:

- Vodenje projektov.

Podpora vodenju projektov za nevladne in dobrodelne organizacije, ki bo zajemala načrtovanje človeških virov, načrtovanje materialnih virov, načrtovanje delavnic in obvladovanje tveganj/napovedi. Nadaljnje funkcionalnosti upravljanja projektov so v razvoju in bodo o njih poročali že v prvem osnutku DApp.

- Kripto denarnice NVO.

Uporabnikom bo omogočena izdelava denarnic ali možnost povezovanja z zunanjimi denarnicami. Poleg tega bo implementacija denarnic z več podpisi uporabniku prijaznejša in dostopnejša.

- Orodje za poročanje.

Ena najpomembnejših nalog v procesu upravljanja nevladne organizacije je odgovornost do finančnih deležnikov, še pomembneje pa do združbo. Vsaka nevladna organizacija mora spremljati vsak vstop, vsako odhodno in dohodno transakcijo, vsak rok in tudi dokončanje ciljev. Da bi bili pripravljeni na vsako notranjo in zunanjo revizijo, je zagotovljeno orodje za poročanje.

- NFT orodje.

Če nevladna organizacija želi uporabljati NFTjev kot način za zbiranje donacij, bo na voljo avtomatizirana funkcija za nalaganje celih galerij s samo enim klikom. Najprej je treba izpolniti obrazec s standardom NFT ID, naslovom zbirke, besedami teme itd. Ko izpolnite vsa obvezna polja, lahko ustvarite celotno galerijo v trenutku.

- Upravljanje presežnih sredstev.

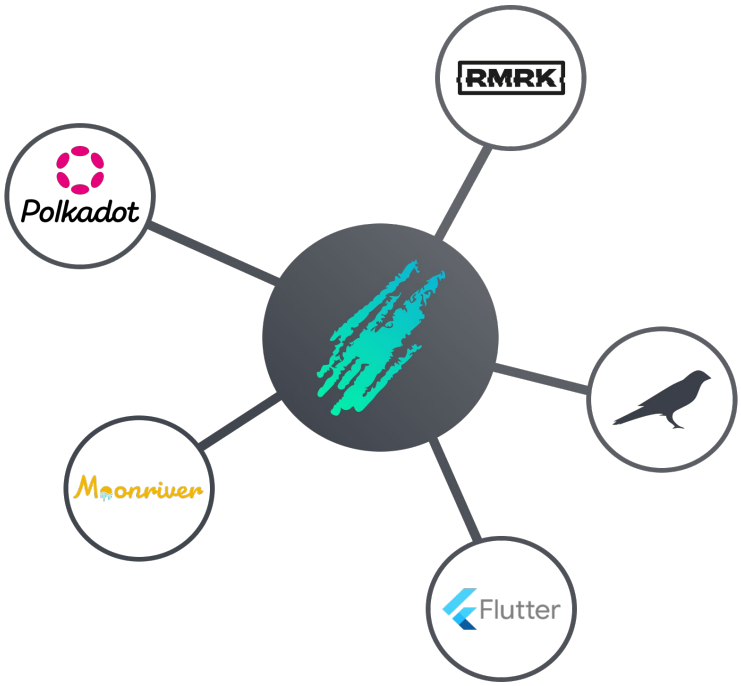
Če ostanejo sredstva, obstaja možnost upravljanja teh sredstev in njihove uporabe za pomoč drugemu cilju ali za upravljanje z orodji DeFi-ja. To je decentralizirana finančna razširitev in bo implementirana in dostopna uporabnikom decentralizirane aplikacije.

3.2 Tehnični sklad

Z izdajo nove različice Flutterja 3 ni bilo več nobenih dvomov o tem, katero ogrodje bo uporabljeno za našo decentralizirano Web3 aplikacijo. Ne samo, da Flutter pokriva mobilni (iOS in Android) in spletni svet, sedaj podpira tudi vse tri namizne operacijske sisteme macOS, Linux in Windows ter še nekatere vgrajene sisteme. Veliko prednost Flutterja predstavlja odlična skupnost za Web3, ki razvija in ponuja podporo za knjižnice, ki pokrivajo

[8] Pomembno je da se upošteva, da to izhaja iz trenutnega osnutka whipe paper.

širok spekter verig, kot so Polkadot, Kusama, Ethereum in številne druge. Izbrali smo Dotsama in Polkadot ekosistem ([7], [8]). Polkadot združuje in ščiti rastoči ekosistem specializiranih verig blokov, imenovanih paraverige, in omogoča razširljivost tako, da omogoča, da specializirane verige blokov komunicirajo med seboj v varnem okolju z zaupanja. Paraveriga je aplikacijsko specifična podatkovna struktura, katera je globalno skladna in validirana z Relay Chain, delijo varnost celotnega omrežja in lahko komunicirajo z drugimi paraverigami prek XCM formata.



Slika 2: Tehnični sklad.
Vir: lasten.

Kusama [7] je zgrajena kot testno omrežje, za prevzemanje tveganja za glavno verigo blokov, Polkadot. To je živa platforma, zgrajena za sprememb agentov, da prevzamejo nazaj nadzor, sprožijo inovacije in porušijo status quo. Za okolje ki je v celoti kompatibilno z Ethereum na Kusama raymišljamo za uporabo paraveriga Moonriver ki je več kot samo ENV za implementacija, je visoko specializirana Lazer 1 veriga, ki zrcali Ethereum-ov Web3 RPC, račune, ključe, naročnine, logov in več. Platforma Moonriver razširja osnovni nabor funkcij Ethereum z dodatnimi funkcijami, kot so upravljanje v verigi, staking in integracije med verigami. Zadnji kos tehnologije v naši tehnološki sklad je RMRK, ki je nabor lego kock NFT, ki NFT-jem dajejo neskončno razširljivost, ki gostuje v blokovni verigi Kusama, canary omrežju Polkadot, brez potrebe po paraverigah ali pametnih pogodbah. Upamo, ta zadnji kos da bo nam omogočil vključitev prikrajšanih otrok v ustvarjanje izpeljanih NFT-jev.

4 Zaključek

Nevladne organizacije so način izražanja volje ljudi, da bi naredili svet boljši. Izkoriščanje priložnosti za uporabo najnovejše tehnologije, ki obljublja ne le preglednost in enakost, ampak tudi ustvarjalnost, bo dobrodelne organizacije dvignilo na naslednjo raven in zgradilo boljši in bolj javno dostopen jutri za vse. V upanju, da bo spremenila dolgo zakoreninjeno negativno nevednost in mnenje ljudi o blockchainu in zlorabi dobrodelnih organizacij, bo ta nevladna organizacija predstavila najboljše strani obeh svetov.

Literatura

[1] https://www2.fundsforngos.org/cat/project-planning-and-development, Funds for NGO, "Project Planning and Development.", obiskano 26.07.2022.

[2] LAURENCE, Tiana "Blockchain For Dummies", Wiley, 2019.

[3] MANGIN, Marion "Stakeholder analysis in NGO evaluations", HAL open science, 2016.

[4] NEUHOLD, Christine and CHRISTIANSEN Thomas "International Handbook on Informal Governance", Edward Elga, 2012, str. 492.

[5] Overseas Development Administration (ODA) "Guidance Note On How To Do Stakeholder Analysis Of Aid Projects And Programmes.", Social Development Department, 1995.

[6] https://www.investopedia.com/non-fungible-tokens-nft-5115211, SHARMA, Rakesh "Non-Fungible Token (NFT) Definition." Investopedia, obiskano 27.07.2022.

[7] www.kusama.network, Kusama, obiskano 26.07.2022

[8] www.polkadot.network, Polkadot, obiskano 25.07.2022

Učinkovita in prilagojena podpora poslovnim procesom s pomočjo odprtokodnih rešitev

Miroslav Beranič

Bintegra d.o.o., Maribor, Slovenija
miroslav.beranic@bintegra.com

Sinopsis V prispevku bomo predstavili celostno infrastrukturno in arhitekturno rešitev za podjetja za povečanje konkurenčne prednosti in oplemenitenje dodane vrednosti obstoječim poslovnim procesom ter celovito pot razvoja sprememb v poslovnih procesih s pomočjo rešitev Red Hat OpenShift, Liferay DXP in Quarkus. Red Hat OpenShift je odprtokodna rešitev, ki nudi poenoteno delovanje in upravljanje izbranih aplikacij, zajema celovit nabor aplikacij in procesov za avtomatizirano delo z izbranimi rešitvami ter nadzor nad izvajanjem in delovanjem. Liferay DXP je odprtokodna rešitev, ki omogoča izgradnjo uporabnikom prijazno in prilagojeno spletno rešitev. Razvijalci lahko pri razvoju sledijo in sodelujejo z odločevalci poslovnih procesov, ki se ne razumejo v poglobljene razvojne procese. Quarkus je ogrodje, ki omogoča optimizirano izvajanje Java aplikacij ter odpira vrata v mikrostoritveno arhitekturo rešitve.

S pomočjo prikazanih rešitev je podjetjem omogočeno hitro prilagajanje pravilom in ažurna komunikacija s strankami na vseh kanalih, na katerih je podjetje prisotno, bodisi javna spletna stran, samopostrežni terminali, email ali socialni mediji.

Ključne besede:
poslovni procesi
optimizacija
Red Hat OpenShift
Liferay DXP
Quarkus

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.14

1 Uvod

Informacijska podpora v podjetjih se razlikuje glede na področje v katerim podjetje deluje, starostjo podjetja in tudi ciljnim bazenom končnih kupcev oz. strank. V tem kontekstu, bi lahko v grobem podjetja delili v tista, ki informacijske rešitev proizvajajo oz. jih nadgrajujejo in nudijo ostalim, podjetja ki uporabljajo IT kot jederno gonilno silo in prilagodijo poslovne procese uporabljenim IT rešitvam ter tistim, ki IT uporabljajo občasno oz. za delno avtomatizacijo mesečnih in letnih zahtev zakonodaje.

Razvoj IT rešitev ter sama vpeljava – namestitev in vzdrževanje se je skozi čas vršil bolj ali manj enako – po enakem postopku. Razvoj se je izvajal v IT podjetjih. Namestitev so izvajali informatiki s strani podjetja, ki je IT rešitev razvilo ali drugo specializirano IT podjetje, ki je imelo znanje namestiti programsko opremo na strojno opremo ali interni IT strokovnjaki. Strojna oprema je bila ali v lasti končnega podjetja ali ponudnika IT rešitev.

Vse te aktivnosti so imele cikel – od razvoja do vzpostavitve in nato vzdrževanja, ki je zajemalo tudi nadgradnje. Pogosto vidimo, tudi pri lokalnih ponudnikih bančnih storitev, obvestilo o ne-delovanju sistema zaradi nadgradnje. Takšne posodobitve se izvajajo v času, ko ni velikega števila uporabnikov, to so predvsem nočni časi.

Podjetje, ki posluje globalno pa časovnega okna, ko ni veliko uporabnikov nima. Takšno podjetje, si ne mora privoščiti časovnega okna, v katerem bi izvajalo posodobitve. V realnosti, se to dogaja in se izvaja na tak način, da se je potrebno odločiti, v katerem časovnem območju se bo to izvajalo – da bo izpad in vpliv čim manjši. To dejstvo je posledica dosedanjega načina že samega razvoja in predaje v uporabo programske rešitve. Pri takšnem načinu, določena ciljna skupina uporabnik, v času posodobitve ne mora uporabljati ali pa je uporaba rešitve omejena.

Predlagane in opisane rešitve so tako že dlje časa v uporabi s strani podjetij, ki imajo globalne trge in so bila primorana rešitve najti med prvimi. Skozi številne iteracije in različne industrije se je oblikoval konzorcij rešitev in predlaganih načinov razvoja, ki lahko pomagajo tudi manjšim podjetjem, ki nimajo primerljivih problemov, a hkrati se je zgodilo, da so uporabniki postali razvajeni z uporabo IT rešitev na način – da preprosto delujejo, ko jih potrebujejo, za namene, ki jih potrebujejo in niso vpleteni v »trenutno ne poslujemo, ker posodabljamo« programsko opremo. Primer tega so številne rešitve podjetji kot so Microsoft, Google, Amazon in Facebook.

V nadaljevanju bom poskušal opisati predlagano rešitev, ki temelji na odprtokodni in javno dostopni rešitvi imenovani Kubernetes. V prispevku se bom osredotočil na eno od številnih specializiranih rešitev, ki temelji na rešitvi Kubernetes. Predstavil bom rešitev podjetja Red Hat, imenovano OpenShift. V povezavi z rešitvijo OpenShift bo predstavljen tudi programski paket podjetja Liferay Inc imenovan Liferay DXP. Za dodajanje lastnih vmesnikov ter integracijo z zunanjimi viri bo predstavljeno odprtokodno ogrodje Quarkus.

2 Predstavitev predlaganih IT Rešitev

2.1. Liferay DXP

Liferay DXP je programska rešitev, ki omogoča podjetju, da izpostavi svojo digitalno prisotnost na spletu. Hkrati pa poenoti svoje notranje poslovne procese in jih digitalizira. Liferay DXP vsebuje gradnike, s katerimi je mogoče:

- izdelati javno spletno stran
- vprašalnike in zbiranje odziva uporabnikov,
- gradnja poslovnih samopostrežnih terminalov, ki pomagajo razbremeniti podporo
- združevanje komunikacijskih kanalov na družbenih omrežjih
- promocija produktov in rešitev

- objava poslovnih novic
- izdelava spletne trgovine povezane z ostalimi notranjimi podpornimi sistemi
- dostop poslovnim partnerjem do dokumentacije in podpori v poprodajnih kanalih

Liferay DXP ima vgrajeno podporo za delovanje na globalnih trgih, kjer je mogoče za vsako interesno ali geografsko področje prilagoditi vsebino in dostope.

2.2. OpenShift

Red Hat OpenShift je t.i. distribucija odprtokodnega projekta Kubernetes. Kot večina ostalih IT rešitev, ki jih nudi podjetje Red Hat je tudi OpenShift komercialni produkt drugače odprtokodnega t.i »upstream« projekta imenovanega OKD. Danes imam nadet Red Hat klobuk.

Red Hat OpenShift je izvajalno (ang. »runtime«) okolje za IT rešitve. Lahko si ga predstavljamo kot virtualno strojno opremo oz. operacijski sistem, ki omogoča in nudi prostor, kamor se namesti aplikacija, ki je nato dostopna preko enega od možnih spletnih dostopov, to je najpogosteje »običajen« spletni dostop – spletne aplikacije.

Red Hat OpenShift rešuje številne oz. vse izzive glede izvajanja in vzdrževanja t.i. mikrostoritvenih rešitev, predstavlja celostno vitualizirano infrastrukturno okolje.

2.3. Quarkus

Quarkus je projekt oz. ogrodje, ki omogoča optimiziranje razvojne poti projekta in izboljšanje metrik v času izvajanja programov. Primarno je namenjen za Java projekte, ki zajemajo tudi ogrodja in programske jezike, ki temeljijo na programskem jeziku Java – kot so Kotlin.

Quarkus je orientiran in primarno namenjen za razvoj t.i. mikrostoritev.

3 Virtualizacija infrastrukture

Virtualizacija infrastrukture omogoča hitrejše prilagajanje na spremembe v potrebah po procesorski zmogljivosti, dostopnosti za programerje in uporabnike ter preprostejšo in celovitejšo vzdrževanje za vzdrževalce.

Red Hat OpenShift Container Platform je temelj za razvoj in zagon kontejneriziranih aplikacij. Zasnovan je na način, da omogoča, da se začne z manjšim osnovnim naborom procesorskih virov in raste z rastjo potreb po dodatnih zmogljivostih. Omogoča upravljanje in zagon od nekaj kontejnerjev to nekaj tisoč in nudenje storitev miljonom uporabnikov.

OpenShift temelji na Kubernetes, ki ga nadgradi z dodatnimi orodji in izvedbenimi procesi, ki standardizirajo in poenotijo tok od analize, razvoja, testiranja, predaje v produkcijsko okolje in nadaljno vzdrževanje in upravljanje.

3.1. Kubernetes

Kubernetes je orhestracijsko orodje in ogrodje, ki omogoča upravljanje z večjim številom kontejneriziranih aplikacij in določitev odvisnosti med njimi. Orhestracija, ki jo opravlja, nudi avtomatsko posodabljanje verzij aplikacij in prilagoditev velikosti glede na potrebe ter nudi vpogled v delovanje vsake posamezne aplikacije. Samo prenos aplikacije v kontejnerizirano obliko ima za razvijalca in vzdrževalca veliko več dodatnega dela, ki ga je mogoče avtomatizirati in to avtomatizirano vlogo opravlja Kubernetes.

Zraven orhestracije samih kontejnerjev, pa Kubernetes nudi tudi upravljanje s podatkovnimi viri – upravlja z zahtevami po procesorski moči, potrebnim delovnim spominom in velikostjo dolgotrajne hrambe podatkov. Vse to je opisano v opisni obliki, Kubernetes se nato sam odloči (glede na svoje vedenje postavitve infrastrukture), kje, kako dolgo in na kak način bo izvedel posamezen kontejner. Analogno kot kontejnerji v pristanišču, Kubernetes, premika kontejnerje na način, da optimizira podatkovne in procesne tokove.

3.2. Hramba podatkov

Kontejnerji sami ne hranijo podatkov, ob zagonu nimajo vedenja o predhodnih zagonih, pomeni – vsak zagon je prvi zagon. Kontejnerji so t.i. »immutable«, pomeni, da nimajo spreminjajočega stanja. To pomeni, da si ničesar ne zapomnijo, pomeni da nimajo vedenja predhodnih zagonov. Na prvi pogled čudno delovanje, je jedro uspeha in ena glavnih lastnosti preboja tehnologije v ospredje in v širšo uporabo.

To vedno ni želeno stanje, recimo podatkovna baza – PostgreSQL – si želimo kontejnerizirati storitev »SQL podatnovne baze«, a hkrati želimo tudi obdržati podatke. Kontejnerje je potrebno povezati v zunanji sistem preko t.i. »volume« - izmenljivih podatkovnih vmesnikih. Podatkovni vmesniki omogočajo, da se pri opisu posameznega kontejnerja določi, katero mesto (direktorij) znotraj kontejnerja je povezan na katero mesto izven kontejnerja in na tak način omogoči, da kontejner ob ponovnem zagonu ne izgubi podatkov.

Pri tem je potrebno seveda paziti, na kak način se to izvede, da se podatki med kontejnerji ne prepisujejo ali pokvarijo.

3.3. Mrežna povezljivost

Kontejnerji privzeto ne vidijo zunanjega sveta – izven kontejnerja, kar je z varnosti zelo ugodno, iz uporabnega stališča pa ne tako zelo. Podobno kot pri izmenjavi podatkov, je potrebno za vsak kontejner določiti preko katerega porta se bo izvedla komunikacija. Določi se port na strani kontejnerja in na strani infrastrukure – kar omogoča, da zunanji sistemi in kontejnerji med seboj lahko komunicirajo.

3.4. Namestitev in posodobitev

Virtualizacija infrastrukure omogoča, da se neglede na jederno infrastrukturo, proti uporabniku uporabljena infrastruktura obnaša enako. OpenShift omogoča namestitev na lokalno fizično strojno opremo, na lokalno virtualizirano okolje ali strežniško okolje v oblaku ali pa vse hkrati – kot hibridno distribuirano okolje, ki je hkrati tudi priporočljiv način namestitve.

Virtualizacija omogoča, da se kompleksnost, ki jo številčni kontejnerji zahtevajo poenostavi s preprostim uporabniškim vmesnikom, ki je prilagojen za vlogo posameznika in daje popolno osredotočenost na določeno nalogo, ki jo dana vloga opravlja; tako je recimo uporabniški vmesnik za vlogo razvojnika drugačna od uporabniškega vmesnika za administratorja sistema.

Posodobitev virtualiziranega okolja omogoča hitre spremembe na vseh nivojih in ker so virtualizirani vsi sestavni deli infrastrukture, je mogoče ustvarjanje virtualnih omrežij in kompleksnih varnostnih pravil enostavno in avtomatizirano, hkrati pa se tudi samodejno prilagaja na spremembe v omrežju.

3.5. Razširljivost in odpornost

Virtualizirana infrastruktura omogoča, da se prilagaja potrebam bolje in hitreje, kot običajna fizična infrastruktura, predvsem iz stališča avtomatizacije nalog in »programskega« preklapljanja drugače fizičnih povezav – mrežni kabli, trdi diski, celotni strežniki, vse to je programska koda in virtualizirana za hiter prenos med podatkovnimi skladišči.

4 Postavitev okolja

Začeli bomo s postavitvijo okolja, ki bo predstavljajo osrednjo IT infrastrukturo. V članku bo predstavljen prosto dostopen način za vzpostavitev OpenShift okolja. Pripravljena instanca je brez večjih omejitev funkcionalnosti in je v polnem obsegu; omejena ja na vrhnjo domeno ».testing«, kar med drugimi omejitvami onemogoča, da se uporablja v produkcijskem okolju, prav tako niso omogočeni nekateri vtičniki, ki omogočajo izvajanje v produkcijskem obsegu.

Red Hat ponuja tudi portal, ki omogoča kreiranje lastne OpenShift instance in se nahaja na naslovu [1]. Trenutna verzija Red Hat OpenShift-a, ki je dostopna preko portala je 4.10, kar je praktično zadnja verzija. Omogoča vse, kar si bomo pogledali danes, z razliko – da se izvaja v oddaljenem računalniškem oblaku z omejenimi viri – CPU in RAM, kar terja daljši čas izvajanja operacij.

Dokumentacija je na voljo na [2], kjer so za vsako verzijo »Release notes«. Dokumentacija zajema veliko vsebine in je zelo uporabna.

Pogledali bomo Red Hat OpenShift Local oz. do pred kratkim imenoval CodeReady Containers [3]. Za namestitev zadošča srednje zmogljiv prenosnik računalnik, povprečno z 8 CPU in 16GB RAM ter 60 GB prostora na disku.

Uporabljena Red Hat OpenShift Local distribucija je verzije 4.10.

4.1. Namestitev

Predstavljena namestitev je opisana za operacijski sistem Linux. Celotna navodila, za ostale operacijske sisteme in v celoti so objavljene na [4].

Po namestitvi terminalnega vmesnika – CLI imenovanega crc [5], se preveri verzija:

```
$ crc version
CRC version: 2.6.0+d606e64
OpenShift version: 4.10.22
Podman version: 4.1.0
```

Določitev osnovnih začetnih parametrov in privolitev v zbiranje telemetričnih podatkov.

```
$ crc config set preset openshift
$ crc config set consent-telemetry yes
$ crc config set cpus 16
$ crc config set memory 24576
```

Sledi glavni namestitveni proces, ki se izvede z ukazom:

```
$ crc setup
INFO Using bundle path /home/miroslav/.crc/cache/crc_libvirt_4.10.22_amd64.crcbundle
INFO Checking if running as non-root
INFO Checking if running inside WSL2
INFO Checking if crc-admin-helper executable is cached
INFO Checking for obsolete admin-helper executable
INFO Checking if running on a supported CPU architecture
INFO Checking minimum RAM requirements
INFO Checking if crc executable symlink exists
INFO Checking if Virtualization is enabled
```

```
INFO Checking if KVM is enabled
INFO Checking if libvirt is installed
INFO Checking if user is part of libvirt group
INFO Checking if active user/process is currently part of the libvirt group
INFO Checking if libvirt daemon is running
INFO Checking if a supported libvirt version is installed
INFO Checking if crc-driver-libvirt is installed
INFO Checking crc daemon systemd service
INFO Checking crc daemon systemd socket units
INFO Checking if AppArmor is configured
INFO Checking if systemd-networkd is running
INFO Checking if NetworkManager is installed
INFO Checking if NetworkManager service is running
INFO Checking if dnsmasq configurations file exist for NetworkManager
INFO Checking if the systemd-resolved service is running
INFO Checking if /etc/NetworkManager/dispatcher.d/99-crc.sh exists
INFO Checking if libvirt 'crc' network is available
INFO Checking if libvirt 'crc' network is active
INFO Checking if CRC bundle is extracted in '$HOME/.crc'
INFO Checking if /home/miroslav/.crc/cache/crc_libvirt_4.10.22_amd64.crcbundle exists
INFO Getting bundle for the CRC executable
INFO Downloading crc_libvirt_4.10.22_amd64.crcbundle
3.16 GiB / 3.16 GiB [-------------------------------------------------------------------------------] 100.00% 9.00 MiB p/s
INFO Uncompressing /home/miroslav/.crc/cache/crc_libvirt_4.10.22_amd64.crcbundle
crc.qcow2: 12.62 GiB / 12.62 GiB [-----------------------------------------------------------------] 100.00%
oc: 117.14 MiB / 117.14 MiB [---------------------------------------------------------------------] 100.00%
Your system is correctly setup for using CRC. Use 'crc start' to start the instance
```

Po preveritvi in izpolnitvi vseh predpogojev, se namesti delujoč sistem. Za namestitvijo se lahko požene prva OpenShift Local Cluster, to se izvede preko ukaza:

```
$ crc start
INFO Checking if running as non-root
INFO Checking if running inside WSL2
INFO Checking if crc-admin-helper executable is cached
INFO Checking for obsolete admin-helper executable
INFO Checking if running on a supported CPU architecture
INFO Checking minimum RAM requirements
INFO Checking if crc executable symlink exists
INFO Checking if Virtualization is enabled
INFO Checking if KVM is enabled
INFO Checking if libvirt is installed
INFO Checking if user is part of libvirt group
INFO Checking if active user/process is currently part of the libvirt group
INFO Checking if libvirt daemon is running
INFO Checking if a supported libvirt version is installed
INFO Checking if crc-driver-libvirt is installed
INFO Checking crc daemon systemd socket units
INFO Checking if AppArmor is configured
INFO Checking if systemd-networkd is running
INFO Checking if NetworkManager is installed
INFO Checking if NetworkManager service is running
```

```
INFO Checking if dnsmasq configurations file exist for NetworkManager
INFO Checking if the systemd-resolved service is running
INFO Checking if /etc/NetworkManager/dispatcher.d/99-crc.sh exists
INFO Checking if libvirt 'crc' network is available
INFO Checking if libvirt 'crc' network is active
INFO Loading bundle: crc_libvirt_4.10.22_amd64...
INFO Starting CRC VM for OpenShift 4.10.22...
INFO CRC instance is running with IP 192.168.130.11
INFO CRC VM is running
INFO Check internal and public DNS query...
INFO Check DNS query from host...
WARN Wildcard DNS resolution for apps-crc.testing does not appear to be working
INFO Verifying validity of the kubelet certificates...
INFO Starting OpenShift kubelet service
INFO Waiting for kube-apiserver availability... [takes around 2min]
INFO Waiting for user's pull secret part of instance disk...
INFO Starting OpenShift cluster... [waiting for the cluster to stabilize]
INFO 2 operators are progressing: network, operator-lifecycle-manager-packageserver
INFO 3 operators are progressing: dns, network, operator-lifecycle-manager-packageserver
INFO Operator network is progressing
INFO Operator network is progressing
INFO Operator node-tuning is not yet available
INFO All operators are available. Ensuring stability...
INFO Operators are stable (2/3)...
INFO Operators are stable (3/3)...
INFO Adding crc-admin and crc-developer contexts to kubeconfig...
Started the OpenShift cluster.

The server is accessible via web console at:
https://console-openshift-console.apps-crc.test
Log in as administrator:
Username: kubeadmin
Password: I8Ube-zhGBv-wKzG7-8iDSe

Log in as user:
Username: developer
Password: developer

Use the 'oc' command line interface:
$ eval $(crc oc-env)
$ oc login -u developer https://api.crc.testing:6443
```

Po nekaj minutah, se v terminal izpišejo dostopni podatki za dostop do spletnega vmesnika in privzetih uporabniških računov. Pripravljena sta dva računa developer – za razvijalca in kubeadmin – za administratorja. Primarno se uporablja uporabniški račun za razvijalca.

4.2. Dostop do postavljene infrastrukture

Z nameščanje infrastrukture smo s tem praktično končali. Sledi urejanje uporabniških računov, diskovnih polj, mrežnih komunikacijskih kanalov ter ostala administracijska opravila, ki jih danes ne bomo pogledali.

Do postavljene infrastrukture lahko dostopamo na dva načina:

- Preko terminalne / ukazne lupine z uporabo orodja »oc« ali
- Spletna aplikacija, dostopna na naslovu izpisanem v postopku namestitve.

```
$ oc login -u developer https://api.crc.testing:6443


$ oc login -u kubeadmin -p I8Ube-zhGBv-wKzG7-8iDSe https://api.crc.testing:6443
Login successful.


You have access to 58 projects, the list has been suppressed. You can list all projects
with ' projects'


Using project "default".


$ oc whoami
kube:admin


$ crc console
Opening the OpenShift Web Console in the default browser...
```

4.3. Odstranitev OpenShiftLocal

Po končanem testiranju se lahko Red Hat OpenShift Local v celoti odstrani. To se izvede preko terminala. V terminalu se izvede ukaz za zaustavitev:

```
$ crc stop
```

Počakamo, da se OpenShift zaustavi in nato se izvede ukaz za izbris celotnega OpenShift Local Clustra. To se izvede z ukazom:

```
$ crc delete
```

4.4. Namestitev v produkcijsko okolje

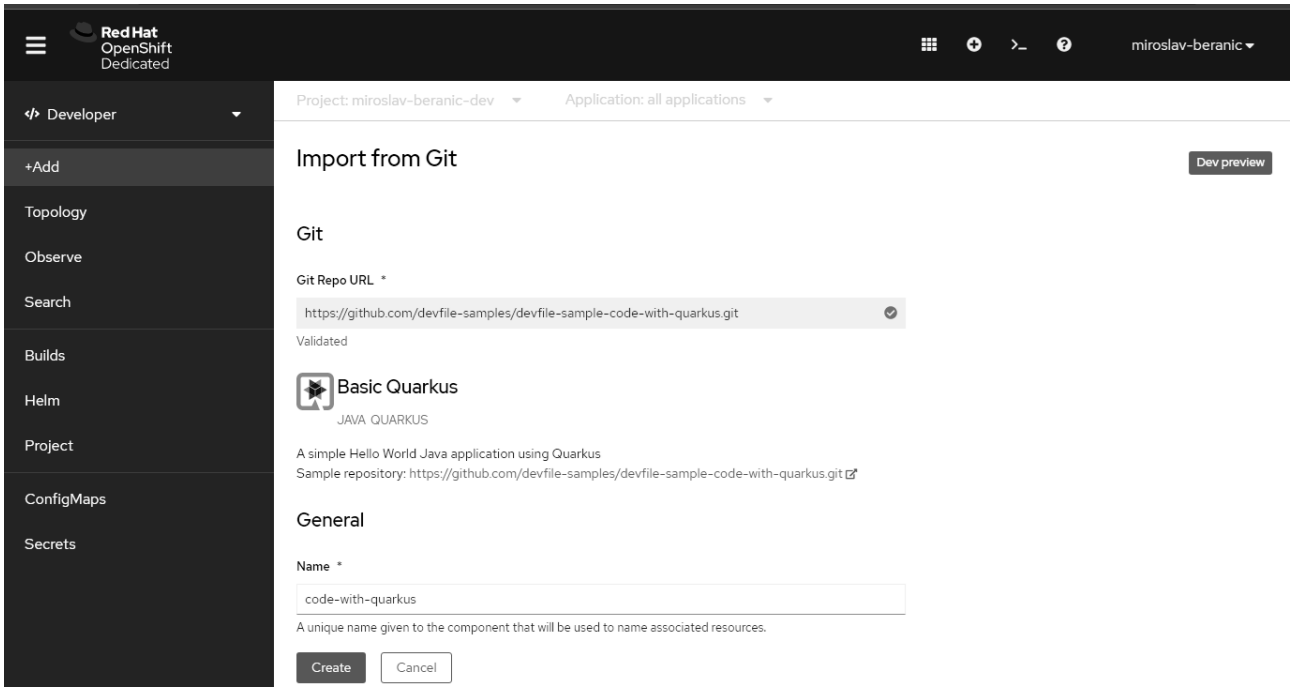
Namestitev v produkcijsko okolje je praviloma analogno prikazanemu, za razliko, da se uporablja večje število fizičnih ali virtualnih strojnih strežnikov. Praviloma se priporoča vsaj tri ločene strežnike za »worker node« in prilagojenim diskovnim poljem, za katerega je priporočljivo, da je lokalno in sestavljeno iz hitrih pogonskih enot.

5 Nameščanje lastnih aplikacijskih rešitev na postavljeni infrastrukturi

Ko imamo enkrat postavljeno infrastrukturo, lahko začnemo nameščati aplikacije. Danes bom pokazal kako se namesti večja IT rešitev in ločena mikrostoritev za uporabo Quarkus ogrodja. Na voljo so tudi številne druge pred-pripravljene rešitve, ki se jih lahko namesti iz vgrajenega kataloga dostopnih aplikacij. Začeli bomo s preprostim in potrebnim »Hello world« primerom.

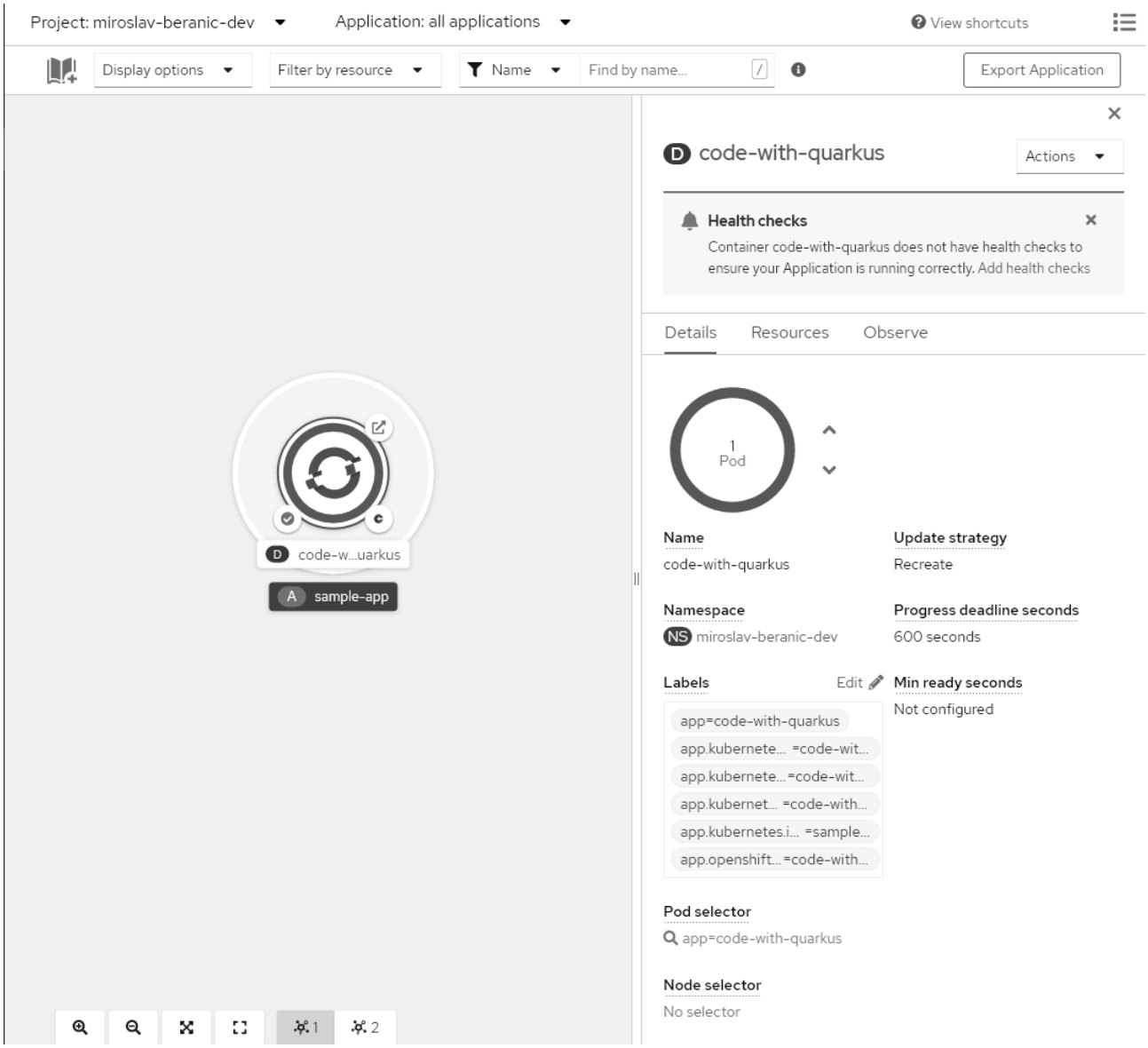
5.1. Hello world

Za začetek je vedno primerno narediti »Hello world« primer, da vidimo osnove, kako sistem deluje. Na desni strani izberemo »+Add«, nato pod »Getting started resources > Create applications using samples « izberemo »Basic Quarkus«. Prikaže se izborno okno, ki ga potrdilo z izborom »Create«.



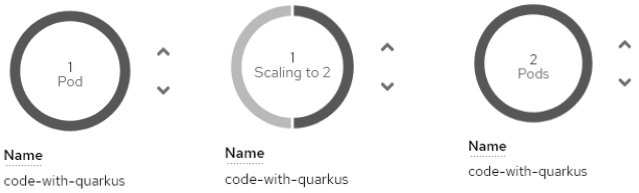
Slika 1: Red Hat dodajanje aplikacije preko Git povezave.
Vir: lasten.

V prikazanem oknu »Topology« se prikaže krogec, ki prikazuje ustvarjeno aplikacijo. Potrebno je počakati, da se kreirajo vsi viri, da se iz izvorne kode ustvari aplikacija in da se požene. Po parih minutah se prikaz posodobi in prikaže sledeče:



Slika 2: Red Hat Topology.
Vir: lasten.

Tukaj je mogoče začeti uporabljati sadeže naše postavljene infrastrukture. Lahko povečamo število instanc s katerimi delamo in s tem povečamo propustnost oz. omogočimo, da je storitev dostopna številčnejšim uporabnikom. Povečamo replikacije (mikro) storitve s tem, da povečamo število Pod-ov. To naredimo enostavno z izborom puščice gor, desno od kroga v katerem je izpisano trenutno aktivnih instanc storitve.



Slika 3: Red Hat OpenShift prehod med številom Pod-ov
Vir: lasten.

5.2. Zagon obstoječega kontejnerja aplikacije

Liferay DXP omogoča zagon aplikacije kot kontejnerizirana aplikacija. To je tudi v splošnem prvi korak, s katerim lahko obstoječe aplikacije prenesemo iz obstoječe infrastrukture in novo OpenShift infrastrukturo. To ni vedno optimalna rešitev, a omogoča prenos obstoječih aplikacij, brez sprememb ali z zelo malo spremembami. Na tak način omogočimo nadaljnji razvoj na aplikacijah in uporabo vseh funkcionalnosti, ki jih nudi nova infrastruktura.

Pri postavitvi Liferay aplikacije, si bomo pomagali z obstoječimi orodji in razširitvami, ki jih že omogoča Liferay za zagon oz. izvajanje v kontejnerski obliki. Za lažjo postavitev v OpenShift pa bomo dodali podporo za Helm chart-e, to so namenske opisne datoteke in infrastruktura, ki omogoča opisati aplikacijske zahteve kot podloga za nameščanje različnih instanc.

```
$ JAVA_HOME=/opt/jdk-8 ./gradlew clean deploy

$ JAVA_HOME=/opt/jdk-8 ./gradlew createDockerFile

$ podman build -t default-route-openshift-image-registry.apps-
crc.testing/ots2022/liferay-portal-7.4-ga35:1.0.0 .

STEP 1/10: FROM liferay/portal:7.4.3.35-ga35

STEP 2/10: ENV LIFERAY_WORKSPACE_ENVIRONMENT=local

--> Using cache 19f7fdcfdae63f14d277c5517ca796efa8c5d1784b77bb6191d52f8fa0de1e0c

--> 19f7fdcfdae

STEP 3/10: COPY --chown=liferay:liferay deploy /mnt/liferay/deploy

--> 361343206a3

STEP 4/10: COPY --chown=liferay:liferay patching /mnt/liferay/patching

--> a0bcfdf0e11

STEP 5/10: COPY --chown=liferay:liferay scripts /mnt/liferay/scripts

--> 632ece4bfd2

STEP 6/10: COPY --chown=liferay:liferay configs /home/liferay/configs

--> c3053bf3ce8

STEP 7/10: COPY --chown=liferay:liferay 100_liferay_image_setup.sh
/usr/local/liferay/scripts/pre-configure/100_liferay_image_setup.sh

--> a166be6a8b7

STEP 8/10: USER root

--> 62dbfc24f3e

STEP 9/10: RUN chgrp -R 0 /opt/liferay && chmod -R g=u /opt/liferay

--> 6c301d81b28

STEP 10/10: RUN chgrp -R 0 /mnt/liferay && chmod -R g=u /mnt/liferay

COMMIT default-route-openshift-image-registry.apps-crc.testing/ots2022/liferay-portal-
7.4-ga35:1.0.0

--> cd720f6c91f

Successfully tagged default-route-openshift-image-registry.apps-
crc.testing/ots2022/liferay-portal-7.4-ga35:1.0.0

cd720f6c91f5548da25992b1d45dccb3847c3e88f7d1d5318c81a3513eb0a093


$ podman push default-route-openshift-image-registry.apps-crc.testing/ots2022/liferay-
portal-7.4-ga35:1.0.0

Getting image source signatures

Copying blob 75e706e0ae82 done
```

```
Copying blob 7c2b733ed8ca done
Copying config cd720f6c91 done
Writing manifest to image destination
Storing signatures
```

Če bi želel vsako storitev nameščati ločeno, bi se to naredilo nas sledeč način:

```
$ oc apply -f database-deployment.yaml -n=ots2022
service/database created
persistentvolumeclaim/database-data created
deployment.apps/database created
```

Helm uporabljamo z namenom, da se naredi predloga, ki hrani vse odvisnosti in zahteve ter tako omogoča na enostaven način namestitev večih med seboj odvisnih storitev. Kot argument se poda datoteka z vrednostmi, ki zapolnijo prostornike v predlogi. To se naredi na sledeč način:

```
$ helm install liferay-chart . -f values.yaml
NAME: liferay-chart
LAST DEPLOYED: Mon Aug  1 18:26:29 2022
NAMESPACE: ots2022
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Spletna aplikacija je na voljo na spletnem naslovu:
http://liferay-dxp-ots2022.apps-crc.testing

6 Zaključek

V prispevku smo predstavili infrastrukturo IT sistema, ki omogoča hitro in učinkovito razširitev za nove IT projekte ter hkrati možnost uporabe obstoječih IT rešitev na novi infrastrukturi. Predstavljen je Red Hat OpenShift v povezavi z Liferay DXP in mikrostoritvami na osnovi ogrodja Quarkus.

Literatura

[1] https://developers.redhat.com/developer-sandbox, OpenShift sandbox, dostopano 10. 8. 2022.

[2] https://docs.openshift.com/container-platform, Welcome | About | OpenShift Container Platform 4.10, dostopano 10.08.2022

[3] https://developers.redhat.com/products/openshift-local/overview, Red Hat OpenShift Local Overview | Red Hat Developer, dostopano 10.08.2022

[4] https://access.redhat.com/documentation/en-us/red_hat_openshift_local/2.5, Product Documentation for Red Hat OpenShift Local 2.5 | Red Hat Customer Portal, dostopano 10.08.2022

[5] https://docs.openshift.com/container-platform/4.10/cli_reference/openshift_cli/getting-started-cli.html, Getting started with the OpenShift CLI - OpenShift CLI (oc) | CLI tools | OpenShift Container Platform 4.10, dostopano 10.08.2022

Uporaba ogrodja Scrum v neprogramerskih projektih

Štefan Masič

Univerza v Mariboru, Univerzitetna knjižnica Maribor, Slovenija
stefan.masic@um.si

Sinopsis Metodologija Scrum je sinonim agilnega pristopa projektov programerskega značaja, kjer je končni izdelek programska oprema. Ali je ogrodje oziroma metodologija uporabna tudi v okviru drugih, neprogramerskih, projektov? Odgovor na zadano vprašanje smo praktično preizkusili na dveh projektih, katerih končni produkt ni izključno programska oprema. V obeh primerih se je pristop izkazal kot primeren. Agilni metodološki pristop je zavaroval izvedbo sámo, predvsem pa končni produktni izdelek.

Ključne besede:
Scrum
agilno upravljanje
neprogramerski projekti
javni sektor
vodenje in upravljanje
monitoring

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.15

1 Uvod

Projektno vodenje in upravljanje s pomočjo agilnih pristopov je sinonim za programerske projekte. Ali je agilni pristop primeren tudi za projekte, katerih izdelek ni izključno programska oprema? Da bi potrdili ali zavrgli omenjeno, smo uporabili metodo Scrum pri dveh projektih, katerih narava ni bila izključno dostava programske opreme ali njenega dela.

Zastavljeno idejo smo želeli izvesti kar se da uspešno. Zato smo k izvedbi pristopili rezervirano in previdno. Preverili smo ponudbo orodij za vodenje in upravljanje projektov (angl. *Project Management Software*, PMS) in zbrali orodje primerno tudi za agilno delo. Po izbiri in nakupu licenc smo orodje predhodno preizkusili na več operativnih in razvojnih projektih. Vzporedno smo se podkovali v osnovnih znanjih metode Scrum in šele nato pristopili k izvedbi po agilnih principih. Ti koraki, predvsem pa izsledki prakse, bodo osrednja tema pričujočega članka.

2 Izbrana projekta

Prvi projekt je bil študentski, izveden v okviru krovnega projekta BlendEd [1]. Projektno skupino je tvorilo sedem članov, med katerimi so bili štirje »računalničarji«, medijski komunikator, grafična oblikovalka in strokovnjakinja marketinga. Projekt je tekel natanko štiri mesece. Obravnaval je specifično projektno tematiko, katere nosilec oziroma naročnik je bilo belgijsko podjetje dScribe. Izdelek oziroma dostava ni bil programski produkt temveč priprava koncepta uporabe orodja skozi igro (angl. *gamification*) v izbranem programskem produktu dScribe.

Drugi projekt se izvaja v okviru delovnega procesa organizacije UKM. Njegovo trajanje je eno leto. Skupina vključuje tri člane, enega z znanji računalništva in informatike, in dve sodelavki iz področja digitalnega knjižničarstva. Naloga projekta je oblikovati arhitekturno idejo nove spletne strani.

V obeh primerih nobeden od članov ekip ni pred tem praktično sodeloval pri agilnem projektu ali imel znanja iz agilnega vodenja in upravljanja. V obeh projektih je sodeloval član, ki je imel znanja, osnove in izkušnje iz področja klasičnega projektnega upravljanja.

3 Izbira projektnega orodja

Izbira primernega orodja je temeljila na naslednjih kriterijih, ki jih je moralo orodje obvezno izpolnjevati:

- orodje mora biti spletna aplikacija,
- orodje mora podpirati tako klasično kakor agilno projektno upravljanje,
- osnovna priprava agilnega projekta naj bo enostavna ali celo avtomatska (na osnovi predloge),
- orodje mora nuditi funkcionalnosti sodelovanja (angl. *Colaboration*).

Osnovne obvezne kriterije smo razširili z dodatnimi tehničnimi, vsebinskimi in finančnimi ter izvedli razširjeno analizo sedmih orodij, štirih plačljivih (Monday.com, Asana, Smartheet, Workfront) in treh odprto kodnih (Openproject, Focalboard, Restja). Tabela 1 prikazuje zgolj pregled štirih plačljivih programov, pregled vseh, tudi odprtokodnih, je podrobneje razviden v [2].

Tabela 1: Pregled plačljivih PMS orodij.

Funkcionalnost	Monday.com	Asana	Smartsheet	Workfront
Planiranje projektov	Da	Da	Da	Da
Pregled izvedbe	Da	Da	Da	Da
Pregled stanja	Da	Da	Da	Da
Upravljanje ponavljajočih nalog	Da		Da	Da
Upravljanje težav	Da	Da	Da	Da
Upravljanje zahteve	Da	Da	Da	Da
Upravljanje nalog	Da	Da	Da	Da
Kanban plošče	Da	Da	Da	Da
Pregled časovnice	Da	Da	Da	Da
Pregled časa in stroškov	Da	Da		Da
Upravljanje idej	Da	Da		Da
Upravljanje verzij	Da			Da
Audio in video konference	Da			
Diskusijske plošče	Da			Da
Upravljanje stikov	Da			Da
Upravljanje vsebine	Da			Da
Sporočilni sistem	Da			Da
Koledar skupine	Da			Da
Projektne predloge	Da	Da		
API	Da	Da	Da	Da
Jezik	angleški	angleški	angleški	angleški
Primerno za malo podjetje	Da	Da	Da	
Produkcija	S / W / C	S / W / C	S / W / C	S / W / C
Mobilna produkcija	A, M	A, iOS	A, W, M	A, iOS
Integracija platform	Da	Da	Da	Da
Podpora	p / t / ž / š / t	ž	ž	ž
Plačilni model	10 EUR/m/Std	10 EUR/m/Std	7 EUR/m/Pro	Custom/m/Pro

S = SaaS, W = Web, C = Cloud, W = Windows, M = Mac; p / t / ž / š / t = pošta, telefon, v živo, šolanje, ticketing

Na osnovi pregleda je bilo izbrano orodje Monday.com. Uporabljena je bila inačica Standard, v osnovi za pet članov, nato pa razširjena na deset licenc. Nakup takšne količine licenc ni bil pogojen zgolj s številom uporabnikov obeh projektnih skupin, temveč je bil namenjen tudi potrebam operativnih in razvojnih nalog ostalih projektov.

3.1 Monday.com

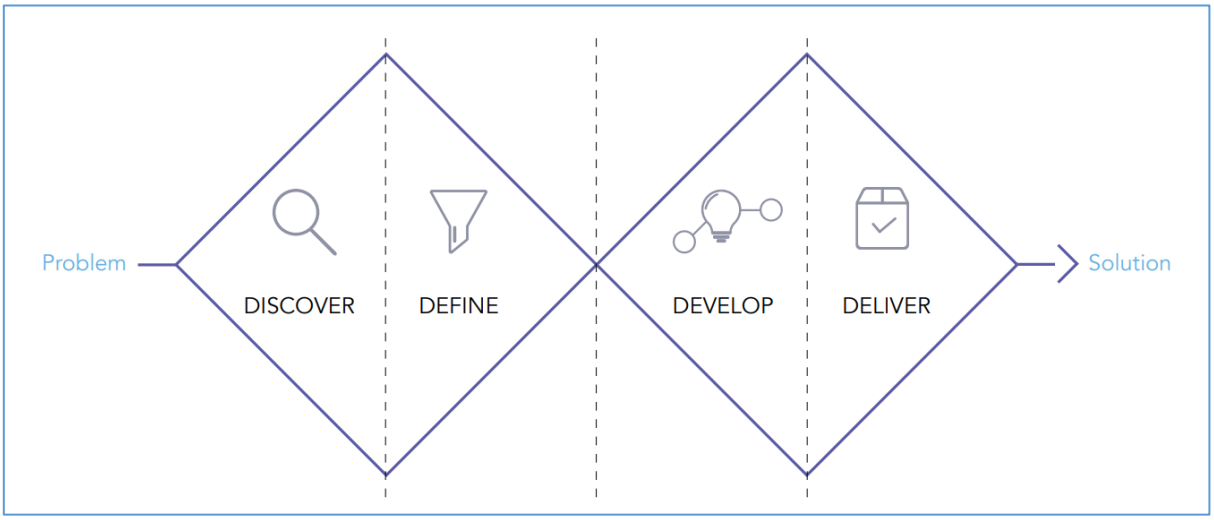
Orodje Monday je zadoščalo vsem obveznim kriterijem. Kljub manjšim »pomanjkljivostim« pri uporabi Gantt diagramov v različici Standard, se je orodje obneslo tudi pri uporabniškem modelu. Omogoča naslednje vrste računov: administratorski, članski, gost. V primeru prvega projekta s sedmimi člani smo lahko dodelili članske licence zgolj lastniku produkta (angl. *Product Owner*) in skrbniku procesa (angl. *Scrum Master*), vsi ostali člani pa so sodelovali kot gostje. Takšen način razporeditve licenc se je izkazal finančno sprejemljiv, obenem pa ni zavrl dela na administraciji projektnih podatkov. Bila je pravzaprav dobra odločitev, saj se je skrbništvo nad administracijo izvajalo dosledno in natančno s strani ene osebe, vsi ostali deležniki pa so bili seznanjeni in so lahko posredovali.

4 Projektna priprava

Zgodnja odločitev, da bomo projektno orodje Monday testirali na operativnih in razvojnih projektih, je bila pravilna. Tako smo dobro spoznali okolje in pridobili osnovne spretnosti. Sledila sta dva pomembna koraka: odločitev o strategiji izvedbe projekta in prilagoditev agilnega okolja lastnim potrebam glede na tematiko projekta.

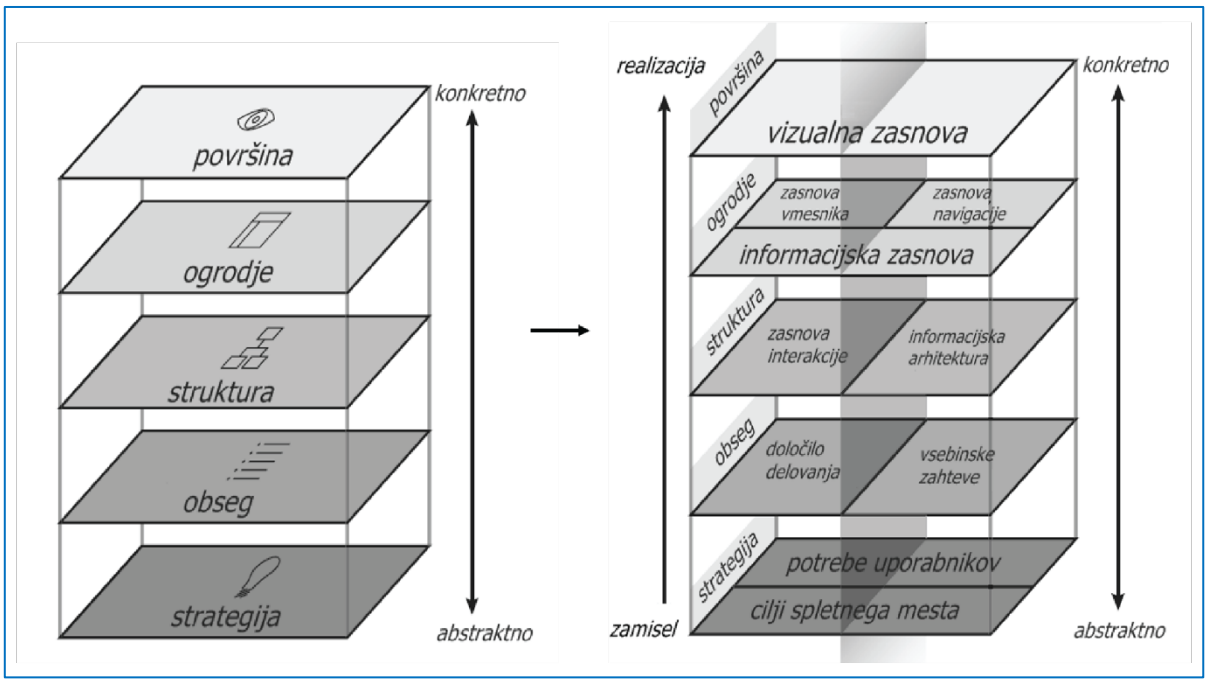
4.1 Strategija

V primeru prvega projekta smo izbrali strategijo dvojnega diamanta (angl. *Double Diamond*). Problematika projekta je nakazovala potrebo po dovolj poglobljeni analizi, ki mora pokazati pravo smer rešitve. Strategija se je ujemala z principom agilnega pristopa: **zgodnja napaka – hitro učenje** (angl. *fast fail – fast learn*). Izbrana strategija je bila primerna tudi zaradi v naprej znane časovne omejenosti projekta (štirje meseci). V prvem diamantu (koraka *discover* in *define*) smo zagotovili agilnost idej in načrtovanja, v drugem diamantu je sledila zgolj implementacija idej, kjer si agilnih sprememb nismo želeli oziroma smeli privoščiti.



Slika 1: Strategija prvega projekta – t. i. dvojni diamant.
Vir: lasten.

V primeru drugega projekta smo izbrali strategijo oziroma proces načrtovanja po avtorju Garret [3], ki je ena od možnih pristopov priprave nove spletne strani. Izbira je slonela na v naprej znanem dejstvu, da noben od deležnikov ekipe ni strokovnjak področja načrtovanja ali oblikovanja spletnih strani. Zato smo želeli znanja, ki jih premoremo – vsebinska, varno nadgraditi z obstoječim procesom načrtovanja nove spletne strani.



Slika 2: Proces načrtovanja nove spletne strani.
Vir: [3].

4.2 Planiranje obremenitev

Načrtovana in dejanska opravila (angl. *Task*) so bila planirana s pomočjo metrike točk (angl. *Story Point*, SP). Namenoma smo se pri obeh projektih odločili za omenjeno metriko, a smo si obenem pomagali z tabelo razmerij med delovnimi urami in točkami. Oblikovana pretvorbena tabela nam je omogočala dovolj veliko svobode. Privzeti moramo tudi dva dejstva: velikost posameznega delovnega opravila je ocenjena in ne izkustvena, realna ocena dela je prav tako posplošeno ocenjena in ni bila v nobenem primeru natančno znana. Iz tega sledi, da tudi razpon razmerja ur in točk (zgodbe) ni bistveno vplival na morebitna nesorazmerja pri oceni. Naša edina ideja merjenja je bila pridobiti in zagotoviti konsistenco izvajanja.

Tabela 2: Razmerje med točkami in urami.

Točke (SP)	Ure (h)
1	1-2
2	2 – 4
3	4 – 8
4	8 – 12
5	12 – 16
6	16 – 24
7	24 – 32
8	32 - 47

4.3 Prilagoditev agilnega dela

Agilno delo po metodi Scrum smo delno prilagodili članom ekipe oziroma njihovim drugim šolskim in/ali službenim obremenitvam. V primeru obeh projektov smo v naprej vedeli, da ne moremo doseči želene dnevne kontinuitete (npr. dnevnih sestankov), kaj šele celotne tedenske obremenitve (8 SP). Ne glede na to smo določili natančne delovne okvire, ki so veljali za oba projekta (op. v primeru razlike med projekti je okvir podan za vsak projekt posebej).

- Sprint traja natanko 14 dni.
- Ekipa prvega projekta se je sestajala na daljavo vsak drugi dan (torek, četrtek, sobota) brez izjeme. Ekipa drugega projekta se je sestajala dvakrat tedensko (torek, četrtek) z mnogo izjemami.
- Sestanki so trajali 15 minut, izjemoma do 30 minut (angl. *Team Meeting*).
- Vsaka druga sobota (prvi projekt) oziroma vsak drug četrtek (drugi projekt) je vključeval enourni sestanek za pregled tekočega sprinta in pripravo novega (angl. *Sprint Review*).
- Lastnik produkta izvaja sestanke s stranko ločeno od delovne skupine.
- Morebitni sestanki med članom ekipe ali stranko ali med člani ekipe ali drugim udeleženci izven projekta se obravnavajo in spremljajo ločeno, izven okvirja tekočega sprinta.

Iz omenjenega lahko povzamemo, da je možno tudi ne programerske projekte upravljati z agilno metodo. Seveda je vse odvisno od področja, v katero je projekt vpet. V našem primeru smo lahko ohranili smernice agilne teorija. Predvsem smo ohranili obe pomembni vlogi (lastnik produkta in skrbnik procesa). Na nekaterih drugih področjih (npr. finance, nadzorovanje) takšne pristop ni možen, saj ima lahko projekt več lastnikov produkta, kar nedvomno otežuje odločitven proces [4].

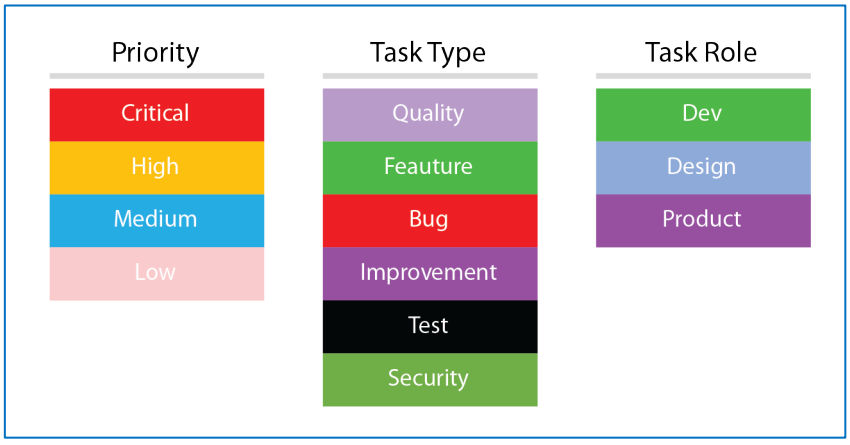
Da manjša odstopanja od agilnih smernic ne bi vplivala na potek in rezultat projekta smo ga zavarovali z nedvoumno administracijo oziroma natančnim upravljanjem.

4.4 Administrativna zasnova sprinta

V zgodnji fazi projekta je bilo sprejeto, da se bodo vse aktivnosti spremljale in beležile. Namen dodatnega dela je bil v pridobitvi podatkov poteka projekta in uporabi pridobljenih podatkov v smislu izkušnje za bodoče projekte. V ta namen smo pripravili osnovni model informacij, ki jih bomo spremljali. Model je naslednji:

- Naziv opravila (angl. *Task Name*).
- Člani, ki izvajajo opravilo (angl. *Team Members*).
- Vrsta opravila (angl. *Task Type*).
- Vloga opravila (angl. *Task Role*).
- Prioriteta opravila (angl. *Priority*).
- Planirane točke (angl. *Planned SP*).
- Dejanske točke (angl. *Actual SP*).
- Opis opravila, ki vsebuje naslednje elemente:
  - Opis (angl. *Description*).
  - Pričakovan rezultat (angl. *Result*).
  - Pričakovan izdelek (angl. *Delivery*).

Parametri vrste, vloge in prioritete delovnih nalog so ponujeni v Scrum predlogi orodja Monday. Za potrebe obeh projektov smo jih delno prilagodili z izločitvijo nekaterih vrst opravil in s spremembo vlog opravila.



Slika 3: Vrednosti parametrov vrste, vloge in prioritete delovnega opravila.
Vir: lasten.

Pomen obširnega opisa opravila (imenujemo jo lahko tudi specifikacija) se je izkazal kot zadetek v polno. Z natančnim opisom pričakovanega rezultata in natančno definicijo izdelka, ki ga moramo dostaviti stranki, smo zagotovili nedvoumnost izvedbe.

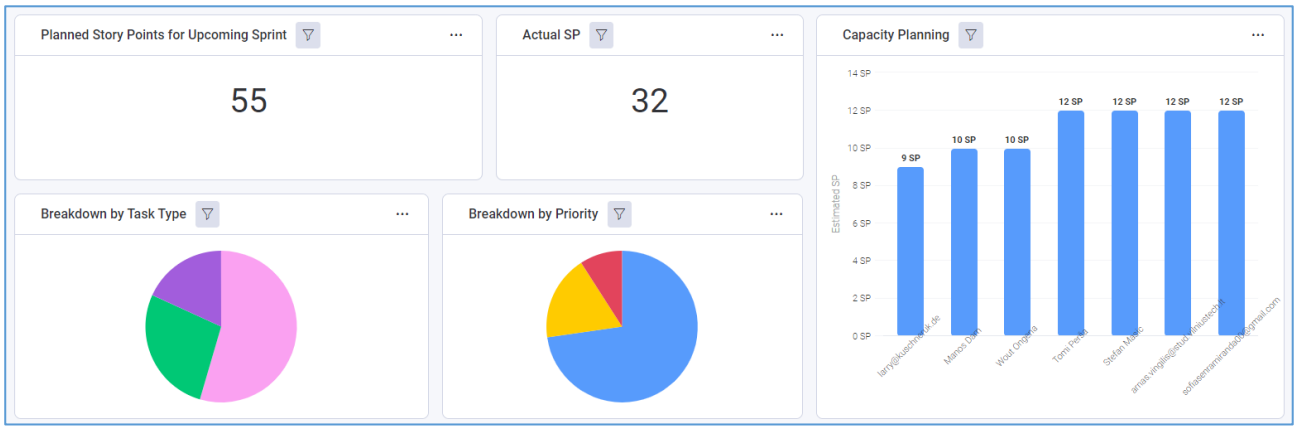
4.5 Analiza sprinta

Uporaba kategoriziranih parametrov je dobrodošla, saj so primerni tako za sprotno spremljavo, kakor za analitično obdelavo. Prav zaradi tega smo jih dosledno administrirali in spremljali. Da bi dobili celovito sliko, smo po vsakem intervalu pripravili kartico sprinta z osnovnimi in nekaj izvedenimi podatki. Tako smo dobili praktično vse informacije za zaključno analizo takoj po zaključku zadnjega sprinta.

Tabela 3: Kartica sprinta (primer za sprint 3, prvi projekt).

Parameter	Parameter	Vrednost (angl. Value)
Naziv	Name	Sprint 3
Trajanje	Duration	2.4.2022 – 16.4.2022
Število dni	Days	14
Planirane točke	Planned SP	44
Dejanske točke	Actual SP	38
Razlika	Difference	-6
Odstotek	Percent	86%

Analiza sprinta je seveda vidna in uporabna že v času izvajanja. Te informacije se lahko s pridom uporabljajo predvsem pri pripravi novega sprinta. Ena od informacij, ki smo jo upoštevali, je obremenjenost deležnikov (angl. *Capacity Planning*). Kljub temu, da člani ekipe zaradi svojih vlog in nalog niso bili skoraj nikoli enako obremenjeni, smo težili k usklajeni obremenjenosti.



Slika 4: Opazovanje prametrov izvedenega sprinta.
Vir: lasten.

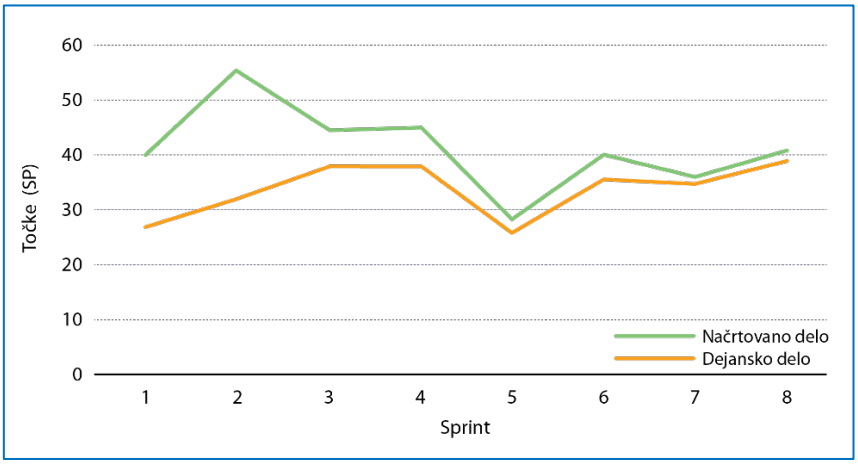
5 Projetna izvedba in agilnost vodenja

Prav posebej smo bili pozorni na realno zasedenost projektnih udeležencev z drugimi delovnimi opravili ali z nalogami na drugih projektih. To danost je potrebno posebej upoštevati v okoljih javnega sektorja (v našem primeru je bil to drugi projekt), ki ne goji projektno usmerjenega vodenja in upravljanja. Zato smo se pragmatično odločili, da bomo delovne aktivnosti planirali z maksimalno osemurno tedensko projektno obremenitvijo (2-3 SP). V praksi se je izkazalo, da je bila tudi tako nizka obremenitev preobširna. Rešitev problematike delovne razporeditve zato vidimo v postopnem prehodu na projektno vodenje operativnih in razvojnih nalog v katere so umeščeni deležniki agilne skupine.

Dva najbolj pomembna elementa agilne metode Scrum sta prav gotovo vlogi lastnika produkta in skrbnik procesa. Zato je odločitev o izbiri primernih oseb, ki zasedajo ti vlogi, zelo pomembna. Delovna skupina ali skupina, ki se predhodno ne pozna, seveda ne more vedeti ali bo izbira pravilna. V primeru prvega projekta se je zgodilo prav to. Ekipa je razpoznala, da lastnik produkta ne opravlja svoje vloge korektno, prav tako pa je skrbnik procesa izrazil željo po predaji vloge drugem članu. Poudariti je potrebno, da smo ob prvi retrospektivi izvedli zamenjave obeh najpomembnejših vlog brez kakršnih koli problemov. To nakazuje agilnost tudi na nivoju vodenja in upravljanja.

6 Analiza načrtovanega in dejanskega dela

Primer primerjave razmerja načrtovanega in dejanskega dela za prvi projekt prikazuje slika 5. Zelo nazorno je vidno, da se je tudi v tem projektu pojavil Dunning-Krugerjev učinek [5]. Prav zaradi odločitve o dosledni uporabi orodja in spremljanju kazalnika načrtovanega in dejanskega dela smo že zelo zgodaj izvedli korekcijo. Pri prvem projektu smo ugotovili, da planiramo sprint preobsežno oziroma drugače rečeno, naše želje so bile prevelike, znanje premajhno, časa pa vedno premalo. Člani ekipe so bili različno obremenjeni s študijskimi in službenimi nalogami. Zato smo maksimalne obremenitve prilagajali ob vsakem sprintu, od maksimalno začetnih 12 SP, do 6 SP proti koncu projekta.



Slika 5: Diagram načrtovanega in dejansko opravljenega dela (prvi projekt).
Vir: lasten.

Iz analize prvega projekta, ki je, kot smo že dejali, končan, je razvidno postopno prilagajanje načrtovanih in dejansko izvedenih del z vsakim naslednjim sprintom. Z izvedenimi korekcijami smo ohranili projektna pričakovanja v okvirih možnega in projekt zavarovali pred neizbežnim zavajanjem stranke, ki bi se gotovo pojavil, v kolikor bi nadaljevali z nerealnim načrtovanjem.

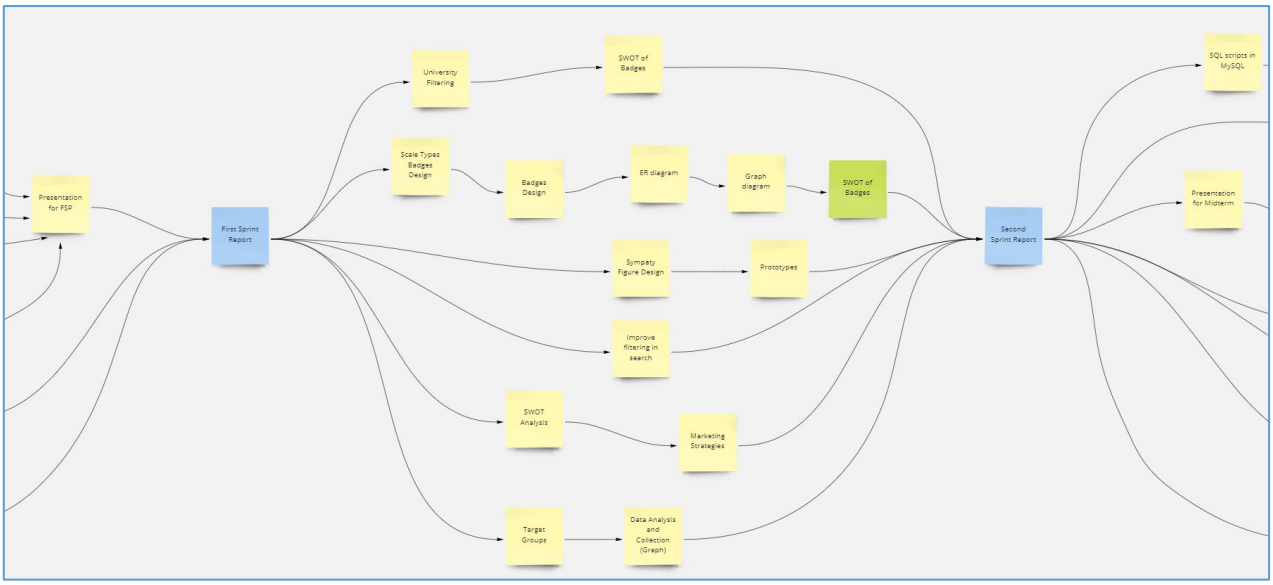
7 Druga projektna orodja in pristopi

Omeniti velja vsaj dva orodja, ki smo ju uporabljali pri obeh projektih. Prvo je orodje za skupinsko delo, Miro [6]. Za pripravo idej je nadvse dobrodošlo za ekipe, ki delajo na daljavo.

Pomembno je vedeti, da orodje za skupinsko delo nikakor ne nadomesti vloge formalne projektne dokumentacije. Vse pomembne izdelke, oblikovane v teh orodjih, npr. miselne zemljevide (angl. *Mind Map*), je primerno sproti prenašati v formalno obliko oziroma v okvir dokumentov, ki služijo kot del končnega izdelka.

Slika 6: Primer ideje, deljene v orodju Miro, izsek (drugi projekt).
Vir: lasten.

Sicer ne ključen, vendar pomemben pristop pri kontroli priprave projekta je bila tudi odločitev, da vzporedno modeliramo projekt z mrežnim opravilnim diagramom (angl. *Network Task Diagram*). Namen je bil dvojen: modeliranje in odkrivanje kar se da večjega števila projektnih opravil v zgodnji fazi projekta in izračun kritične projektne poti. S tem pristopom smo želeli zavarovati uspešnost izvedbe.



Slika 7: Primer mrežnega diagrama, izsek (prvi projekt).
Vir: lasten.

8 Česa smo se naučili?

Uporaba agilne metode kot je Scrum ni misija nemogoče za ekipe, ki omenjenih znanj nimajo in za področja, ki niso izključno programerska. Dobrodošlo je, da ima projektna ekipa vsaj enega člana, ki pozna osnove projektnega vodenja in uporabe orodij. Izbiri orodja je potrebno posvetiti dovolj časa, nato pa principe predhodno preizkusiti v praksi. S tem pridobimo prakso v uporabi orodja, spoznamo njegove omejitve in jih nadomestimo z drugimi prijemi ali pristopi. Zelo pomembno je tudi vzpodbujanje in ozaveščanje sodelavcev o možnostih novih pristopov

(projektna orodja, orodja za sodelovanje). To velja predvsem za okolja, ki niso naklonjena projektnemu pristopu oziroma principom projektnega vodenja in upravljanja.

Pomemben faktor uspešne izvedbe je gotovo monitoring dela. Omenjeno (opazovanje) je v naši moči ne glede na morebitno nerazpoznavnost pomena projektnega pristopa kot osnove za delo v organizaciji. Naj ekipi ne bo žal natančno beležiti vsak sestanek, prisotnost, opisati opravilo v standardni obliki, analizirati tekoče delo in izvajati hitre prilagoditve. Pridobljeni merjeni podatki nam bodo pokazali primerno uskladitev za naslednji projekt.

Izkustvene nasvete za naslednji ne programerski tip projekta lahko strnemo v naslednjih alinejah:

- V kolikor lahko ohranimo večino agilnih smernic to nakazuje, da je projekt primeren za agilen pristop.
- Izbirajmo med projekti, ki trajajo vsaj šest mesecev. Prva dva meseca izkoristimo za dosego kontinuiranosti dela (redni sestanki, realne ocene dela).
- Dobro vodena in urejena vzporedna dokumentacija je zagotovilo uspeha.
- Dokumentacija, nastala pod okriljem orodij za sodelovanje ni primerna za uradno projektno dokumentacijo in jo je potrebno sprotno prenašat v uradno dokumentacijo.
- Sprotno periodično poročanje stranki je zagotovilo za končni uspeh projekta.

9 Zaključek

Uporaba agilnega pristopa pri pripravi in upravljanju ne programerskih projektov se je v našem primeru izkazala kot primerna. Nedvomno sta k temu prispevali tudi primerni projektni temi, ki sta se vsaj delno povezovali z temami iz sveta računalništva in informatike. V primeru obeh projektov je presenetljivo dejstvo, da so agilno metodo privzeli tudi člani brez kakršnih koli izkušenj. V primeru drugega projekta se je pokazalo, da je agilen pristop k vodenju in upravljanju projektov bistveno bolj produktiven kot klasična organizacija dela, naslonjena na strukture kot so delovne ekipe ali delovne komisije, ki so stalnica v organizacijah javnega sektorja. Scrum sloni na preglednosti in možnosti prilagajanja [7]. Kot takšen se je izkazal tudi v našem primeru in nas je vzpodbudil, da bomo s to prakso vodenja in upravljanja nadaljevali tudi pri prihajajočih projektih.

Literatura

[1] www.blendedmobility.com, Projekt BendEd Mobility, obiskano 1.7.2022.

[2] Štefan Masič, Pregled PMS orodij, interni delovni dokument, 2021.

[3] Garrett, J. J., The elements of user experience, New Riders, 2010.

[4] Maryam Monde Njimamboue, Agile Scrum in Non-Software Development Projects: Case Study in Finance and Controlling, Hochschule Furtwangen, 2018.

[5] Kruger, Justin, Dunning, David, »Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments", Journal of Personality and Social Psychology, letnik 77, številka 6, str. 1121–34.

[6] www.miro.com, Miro, orodje za sodelovanje, obiskano 1.7.2022.

[7] Schwager, K., Sutherland, J., The definitive guide to Scrum: the rules of the game, https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf, obiskano 1.7.2022.

Izzivi in vzorci zasnove mikrostoritev v brezstrežniškem okolju

Tilen Hliš, Luka Pavlič

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

Sinopsis Mikrostoritvena arhitektura je preizkušen, in kot kažejo podatki, tudi vodilen arhitekturni stil informacijskih rešitev zadnjega desetletja. Kot je praksa na vseh inženirskih področjih, tudi pri vpeljavi mikrostoritvene arhitekture uporabljamo preizkušene dobre prakse. Načrtovalski vzorci ne predstavljajo konkretne implementacije, temveč praktično preizkušene ideje, kako rešiti določeno skupino izzivov. Vzorce lahko umestimo v kategorije ali skupine - vsaka izmed njih naslavlja določeno problematiko s katero se soočajo razvojne skupine. Brezstrežniško računalništvo z vedno hitrejšim razvojem oblačnega računalništva vedno bolj pridobiva na pomenu. Arhitekture brez strežnika predvidevajo zasnovo informacijskih rešitev, ki vključuje zaledje kot storitev (angl. Backend as a Service - BaaS) in kodo po meri (angl. Functions as a Service -FaaS), ki se izvaja v upravljanih, kratkotrajnih vsebnikih na izbrani platformi.

Ključne besede:
mikrostoritvena arhitektura
brezstrežniško računalništvo
načrtovalski vzorci

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.16

1 Uvod

Področje storitveno usmerjene arhitekture in računalništva v oblaku je v zadnjih 10 letih doživelo evolucijo pri razvoju, uvajanju in vzdrževanju informacijskih sistemov. Pojem storitveno usmerjene arhitekture, ki zajema spletne storitve opredeljuje, da so poslovna pravila aplikacij vgrajena v storitve same, komunicirajo pa preko standardnega protokola. Omenjen način omogoča enotnost pri zagotavljanju, iskanju, interakciji in uporabi storitev. Domenska področja podjetja s poslovnimi procesi lahko prenesemo na posamezno storitev. Osrednji namen uporabe storitveno usmerjene arhitekture je integracija nizko sklopljenih neodvisnih aplikacij, pri čemer posamezne gradnike aplikacij strukturiramo kot storitve. Na trgu se pojavlja vedno več spletnih storitev, ki ne podpirajo vseh načel storitveno usmerjene arhitekture (tj. pogodba o storitvi, nizka sklopljenost, ponovna uporaba, abstrakcija, delovanje brez stanja, široka združljivost, odkrivanje in interoperabilnost storitev) in s tem se je pojavil nov pojem, ki je v svetu IT znan kot mikrostoritvena arhitektura. [1, 2]

Mikrostoritvena arhitektura je pomembnejši načrtovalski slog v zadnjem desetletju. Mikrostoritve izhajajo iz problematike spletnih storitev, ki slabo ali sploh ne podpirajo vseh načel storitveno usmerjene arhitekture. Številne prednosti mikrostoritvene arhitekture vključujejo nizko sklopljenost, neodvisnost posameznih storitev in omogočeno razširljivost, ponujajo sisteme z višjimi zmogljivostmi, boljšim delovanjem, organizacijsko pa ponujajo možnost hkratnega neodvisnega razvoja posameznih gradnikov informacijskih rešitev. Arhitekturni stil se je razvijal vzporedno z razvojem programsko usmerjene industrije in tako je uporaba mikrostoritvene arhitekture v praksi v preteklosti prehitela akademsko sfero. Nastala vrzel se, zahvaljujoč številnim raziskavam, povezanimi z mikrostoritvami s področja novih metodologij, procesov in orodij, zmanjšuje [3]. Zaradi posebnosti in odstopanj od uveljavljenih dobrih praks je potrebno za vpeljavo mikrostoritvene arhitekture uporabiti številne inovativne pristope pri naslavljanju nefunkcionalnih zadev. Kljub vsemu lahko dobre prakse že najdemo v obliki zbirk in katalogov. Načrtovalski vzorci ne predstavljajo konkretne implementacije, temveč praktično preizkušene ideje, kako rešiti določeno skupino izzivov. Vzorce lahko umestimo v kategorije ali skupine - vsaka izmed njih naslavlja določeno problematiko s katero se soočajo razvijalci . Eno izmed področij, ki je povezano s svojo skupino načrtovalskih vzorcev mikrostoritvene arhitekture, je področje izvajanja mikrostoritev. Skupina zajema vzorce, ki so povezani z izvajanjem posameznih mikrostoritev in celotne informacijske rešitve, pri čemer izpostavljamo vzorec, ki temelji na uvedbi brez strežnika (angl. Serverless deployment) [4]. Vzorec je oblikovan okoli skrivanja osrednjih strežniških konceptov. Uporabljena infrastruktura sprejme kodo in jo požene. Vzorec je zanimiv, saj predstavlja povezavo med sicer samosvojima stiloma arhitekturne zasnove, mikrostoritveno in brezstrežniško.

Brezstrežniško računalništvo z hitrim razvojem oblačnega računalništva vedno bolj pridobiva na pomenu. Arhitekture brez strežnika predvidevajo zasnovo informacijskih rešitev, ki vključuje zaledje kot storitev (angl. *Backend as a Service - BaaS*) in kodo po meri (angl. *Functions as a Service -FaaS*), ki se izvaja v upravljanih, kratkotrajnih vsebnikih na izbrani platformi [5].

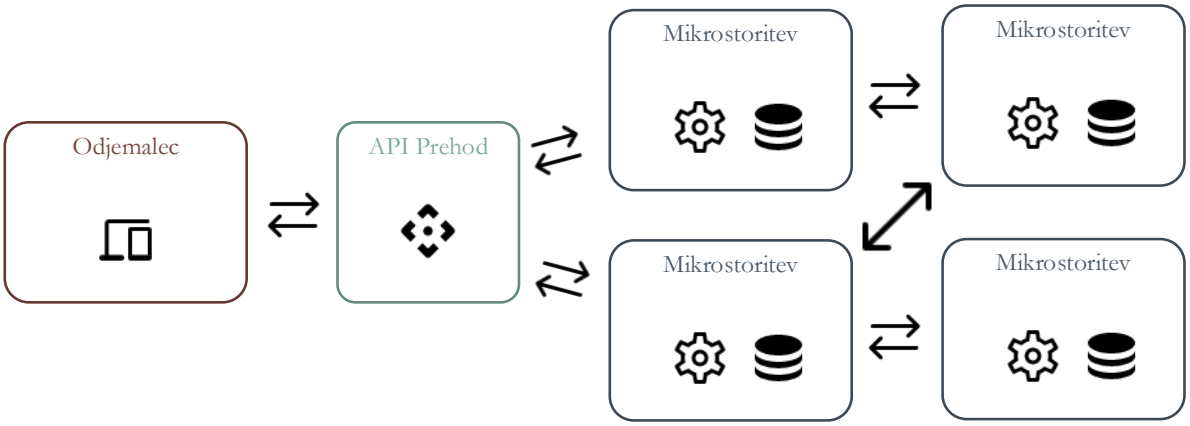
2 Mikrostoritvena arhitektura in brezstrežniško računalništvo

Brezstrežniško računalništvo ima več skupnih značilnosti z mikrostoritvenim stilom, a tudi nekaj pomembnih razlik. V nekaterih primerih je za delo bolj primerna uporaba sistema brez strežnika kot raba mikrostoritev. Da bi razumeli, katero izmed arhitekturnih zasnov uporabiti, morajo razvijalci upoštevati, kako se ta dva pristopa dopolnjujeta in predvsem razlikujeta.

2.1. Mikrostoritvena arhitektura

Trendi nakazujejo, da je uporaba mikrostoritvene arhitekture pogosta izbira pri snovanju sodobnih informacijskih rešitev [3]. Arhitekturni stil izhaja iz storitveno usmerjene arhitekture in tako omogoča skladno integracijo med seboj neodvisnih sistemov. Z nekaterimi prej znanimi in uporabljenimi arhitekturnimi zasnovami smo

informacijsko rešitev gradili v smislu t. i. monolita, ob upoštevanju smernic mikrostoritvene arhitekture pa gradimo množico neodvisnih mikrostoritev. Mikrostoritvena arhitektura sledi načelu, da morajo biti storitve majhne in lahke, pri čemer naj vsaka storitev opravlja samo eno poslovno nalogo. Vsaka storitev naj ima svojo bazo podatkov, kar še dodatno zmanjša povezanost med njimi. Celoten sistem je sestavljen iz več manjših, šibko sklopljenih delov, kar nam omogoča neodvisno spreminjanje vsakega dela posebej. Omogočena razširljivost ponuja sisteme z višjimi zmogljivostmi in boljšim delovanjem. Neodvisnost in nizka sklopljenost gresta pri mikrostoritveni arhitekturi preko meja izvorne kode in se nanašata na izvajalno okolje ter razvojno ekipo, ki skrbi za razvoj določene mikrostoritve. Slika 1 prikazuje shematski prikaz tipične zasnove mikrostoritvene rešitve.



Slika 1: Mikrostoritvena arhitektura.
Vir: lasten.

Čeprav so načela in smernice tega arhitekturnega stila znane že dalj časa, sta eno izmed najbolj pogosto uporabljenih definicij mikrostoritvene arhitekture šele leta 2014 zapisala Martin Fowler in James Lewis [2]. Definicijo lahko povzamemo kot: »Arhitekturni slog mikrostoritev je pristop k razvoju ene same aplikacije kot zbirke majhnih storitev, od katerih vsaka deluje v svojem procesu in komunicira s preostalimi storitvami s pomočjo določenih mehanizmov«. Storitve so zgrajene na podlagi poslovnih zmogljivosti in jih je mogoče neodvisno uvesti s popolnoma avtomatiziranimi stroji za uvajanje. Obstaja minimalno centralizirano upravljanje teh storitev, ki so lahko napisane v različnih programskih jezikih in uporabljajo različne tehnologije za shranjevanje podatkov. [1, 2, 3]

Posamezne mikrostoritve tipično ne vzdržujejo stanja. Zahteve in odgovori se med storitvami tipično prenašajo preko protokola HTTP, pri čemer ponudijo odgovore v formatu JSON (angl. JavaScript Object Notation) ali XML (angl. Extensible Markup Language). Končne točke izpostavljene s strani mikrostoritev ponujajo rabo metod protokola HTTP (za komunikacijo uporabljajo princip REST (angl. Representational state transfer)) in predstavljajo povezavo s poslovno logiko sistema. Asinhrono komunikacijo lahko dosežemo preko protokolov, kot je MQTT (angl. MQ Telemetry Transport); tako lahko pošiljamo in pridobivamo podatke iz različnih senzorjev. Uveljavljen sinhroni stil komunikacije RPC (oddaljen klic procedur, angl. Remote Procedure Call) je poleg uporabe HTTP vedno pogosteje implementiran tudi z ogrodji, kot je gRPC. Le-ti že privzeto ponujajo možnost asinhronega izvajanja klicev in za delovanje uporabljajo HTTP/2. Posamezne storitve v rešitvi, ki uporablja mikrostoritveni arhitekurni stil lahko med seboj na asinhorni način komunicirajo tudi z uporabo sporočilnih sistemov, kot so RabbitMQ, ActiveMQ in Apache Kafka. Zaradi arhitekturne zasnove mikrostoritve dopolnjujejo agilen razvoj in uporabo inženirskih praks DevOps. [1, 3]

Ena izmed prednosti mikrostoritvenih sistemov v primerjavi z uporabo monolitne zasnove sistema je odpornost. Kadar odpove eden izmed gradnikov sistema, to ne pomeni izpada celotnega sistema. Možnost lažje skalabilnosti rešuje morebitne izzive ob višji obremenitvi ene, več ali vseh storitev sistema. V izogib upočasnjenega delovanja in neodzivnosti lahko s pomočjo uporabe določenih tehnologij prilagodimo skalabilnost le storitvam, ki to dejansko potrebujejo. Po drugi strani pa predstavlja skalabilnost v sistemih z monolitno zasnovo precejšen izziv. Omeniti

velja, da je lahko kratkoročno gledano vpeljava mikrostoritev dražja, pri čemer je v zakup potrebno vzeti druge prednosti, ki nam olajšajo delo ter dolgoročno prihranijo čas in denar. [6, 7]

Na podlagi prej podane definicije in do sedaj zapisanega lahko izpeljemo sledeče lastnosti, ki bi jih naj imeli informacijski sistemi, ki uporabljajo mikrostoritveno arhitekturo [4]:

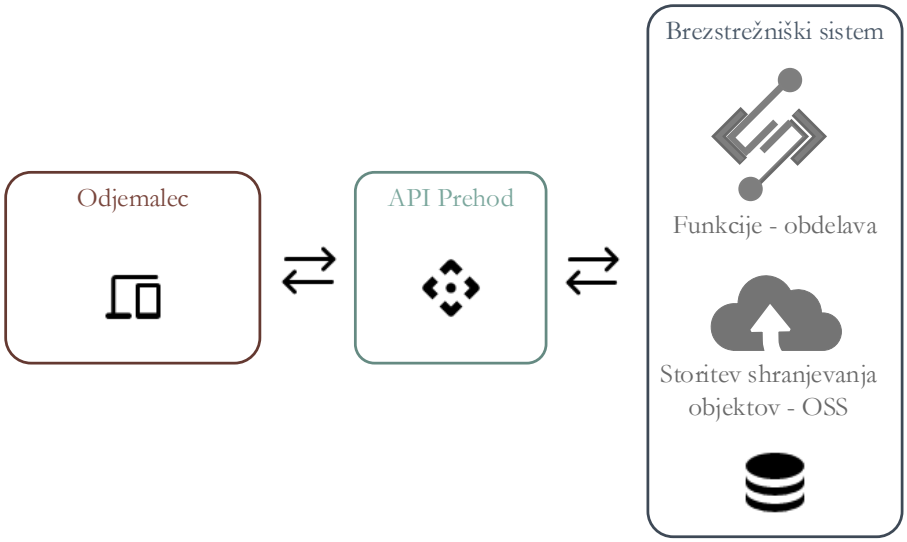
- mikrostoritvena arhitektura je stil, ki temelji na storitvah, ki za izpolnjevanje ciljev komunicirajo preko omrežja,
- komunikacija poteka preko sinhornih ali asinhornih protokolov,
- storitve so organizirane okrog organizacijskih zmogljivosti in omogočajo postopno implementacijo funkcionalnosti posameznega področja, neodvisno od drugih področij in storitev,
- vsaka mikrostoritev je avtonomna, zato obstaja možnost implementacije posameznih storitev v različnih programskih jezikih, platformah z različno strojno opremo ter različnih podatkovnih baz v ozadju,
- mikrostoritve so majhne, omogočajo pošiljanje sporočil, omejene so s kontekstom, v katerem delujejo,
- vsaka storitev deluje v svojem okolju, kar omogoča posodabljanje in namestitev novih različic z avtomatiziranimi procesi,
- vsaka storitev ima svoj načrt avtomatizacije (testiranje, posodobitve, prilagoditve, integriranje in vzdrževanje).

2.2. Brezstrežniško računalništvo

Računalništvo v oblaku je uporaba računalniške moči, pomnilnika, podatkovnih baz, storitev in preostalih IT virov na zahtevo. Praviloma se uporaba takšnih oblačnih storitev preko interneta zaračunava po modelu »plačevanje po uporabi« (angl. pay-as-you-go). Kot je splošno znano gre pri pojmu oblak za skupek računalnikov, ki se nahajajo na različnih lokacijah po svetu in so v lasti *ponudnikov oblačnih storitev*, kot so Amazon Web Service (AWS), Google Cloud in Microsoft Azure. Načelo računalništva v oblaku ostaja nespremenjeno a kljub temu doživlja določene spremembe. Ponudniki oblačnih storitev težijo k čim višji skalabilnosti, visoki odzivnosti storitev in agilnemu pristopu pri zagotavljanju virov, kar je zaradi vse hitrejšega razvoja na področju tehnologij za virtualizacijo tudi mogoče. V začetnih fazah razvoja oblačnega računalništva se je virtualizacija uporabljala kot sredstvo za konsolidacijo programske opreme in storitev, s katero je bila dosežena maksimalna izkoriščenost virov in enostavno upravljanje. V začetni fazi je prišlo do skupne souporabe strojne opreme. V naslednji fazi je bil na strežniku ustvarjen nabor virtualnih strojev (VM, angl. virtual machine), vsak VM je nosil kopijo operacijskega sistema. Razvoj se je nadaljeval z uporabo koncepta zabojnikov (angl. container). Zabojniki so platforma za shranjevanje virov, potrebnih za izvajanje določene aplikacije. V primerjavi z VM-ji, zabojniki omogočajo večjo abstrakcijo virov. Tako je zagotavljanje virov veliko hitrejše kot pri uporabi VM. Učinkovitost in hitro zagotavljanje virov s strani zabojnikov je omejena z osnovnimi infrastrukturnimi elementi, imenovanimi strežniki. Kot prikazuje slika 2 je brezstrežniško računalništvo model združevanja in uporabe virov, ki vključuje operacijski sistem, izvajalna okolja in strojno opremo. [5, 6, 8]

Pri brezstrežniškem računalništvu gre za uporabo dogodkovno vodenega razvoja informacijskih rešitev, ki torej temelji na dogodkih, kjer računalniške vire, potrebne za zagon kode, namesto nas upravlja ponudnik storitev v oblaku. Nanaša se na platforme, ki uporabnikom omogočajo izvajanje določenega dela kode na zahtevo. Imenuje se brezstrežniško (še vedno so v uporabi strežniki, gre za dodatno stopnjo abstrakcije), saj uporabnikom ni potrebno vzdrževati fizičnega ali virtualnega strežnika za izvajanje kode. Praviloma je potrebno v kateri koli organizaciji, kjer ne uporabljajo brestrežniških platform, vzdrževati in upravljati računalniške vire, ne glede na to, ali jih uporabljajo ali ne. Ob uporabi brezstrežniškega pristopa lahko svojo kodo gostimo in izvajamo v oddaljenem računalniškem viru, ki ga zagotovi ponudnik storitev, pri čemer moramo plačati za dejansko uporabo brez stroškov

vzdrževanja in upravljanja strežnika. Prav tako nismo sami odgovorni za morebitne nedejavnosti in izpade strežnikov. Brezstrežniško računalništvo ne odpravi uporabe strežnika, temveč le pomaga, da ga premaknemo v zaledje in s tem preložimo odgovornost na ponudnike teh storitev. Razvijalci se lahko osredotočijo na načrtovanje svoje informacijske rešitve. [6]



Slika 2: Brezstrežniška arhitektura.
Vir: lasten.

Ponudniki brezstrežniških tehnologij dinamično dodelijo strežnike, ki so na voljo uporabniku. Koda se ob klicu, ki ga sproži nek dogodek, izvaja v vsebnikih brez stanja. Izvajanje te kode lahko traja dalj časa in v več ponovitvah, za kar ni potreben ponoven klic. Poleg tega brezstrežniške rešitve vključujejo različne tehnologije, med katerimi sta dve ključnega pomena imenovani zaledje kot storitev (angl. Backend-as-a-service) ali BaaS in funkcije kot storitev ali krajše (angl. Functions-as-a-service) FaaS. *Baas* omogoča zamenjavo klasičnih strežniških komponent s standardnimi storitvami. Razvijalcem omogoča izkoriščanje zunanjih zmogljivosti (angl. outsourcing) v obliki zalednega dela, tako da se lahko razvijalci osredotočijo na pisanje in vzdrževanje logike uporabniških vmesnikov. Sistemi BaaS ponujajo storitve za avtentikacijo, upravljanje podatkovnih baz, shranjevanje v oblaku , gostovanje in podobno. FaaS je okolje, ki se uporablja za izvajanje programske opreme. Kot že omenjeno so brezstrežniške aplikacije dogodkovno vodeni sistemi, ki temeljijo na oblaku, kjer se razvoj aplikacij opira izključno na kombinacijo storitev tretjih oseb, logike na strani odjemalca in klicev oddaljenih postopkov, ki gostujejo v oblaku. FaaS razvijalcem omogoča uvedbo kode, ki se po sprožitvi izvede v izoliranem okolju. Vsaka funkcija običajno opisuje majhen del celotne aplikacije. Čas izvajanja funkcij je običajno omejen (npr. 15 min za AWS Lambda). Funkcije niso stalno aktivne, namesto tega jih platforme FaaS instancirajo po potrebi. Funkcije sprožijo dogodki, kot so zahteve odjemalcev, dogodki, ki jih ustvarijo zunanji sistemi, podatkovni tokovi ali drugi. Ponudnik FaaS sistemov so nato odgovorni za horizontalno skaliranje izvajanja funkcij v odvisnosti glede na število vhodnih dogodkov. [9]

Brezstrežniške aplikacije je mogoče razviti v več kontekstih, kljub temu so lahko na nekaterih področjih omejene. Na primer, dolgo delujoče funkcije, kot je izvajanje strojnega učenja ali dolgo delujoči algoritmi, imajo lahko težave s časovno omejitvijo. Takšne stalne delovne obremenitve lahko povzročijo višje stroške v primerjavi z neomejenim izvajanjem računalniških storitev na zahtevo, ki jih lahko najdemo pri postavitvi lastne mikrostoritvene arhitekture. [9, 10]

Omenjena pristopa BaaS in Faas sta primarna predstavnika implementacije brestrežniškega arhitekturnega stila. Predvsem model FaaS je po svojih karakteristikah (preprostosti in elastičnosti) popolnoma v skladu z načeli brestrežniškega računalništva. Omenjena modela lahko, kot je razvidno iz literature, pomotoma enačimo z modelom zabojnik kot storitev (angl. Container-as-a-Service) ali CaaS, ki je en izmed novejših pristopov v oblačnem računalništvu in spada med zbrane načrtovalske vzorce uvajanja storitev, pri uporabi mikrostoritvenega

arhitekturnega stila [4]. Glavni namen tega modela, je uporaba virtualizacije, ki temelji na zabojniških strukturah. V primerjavi z modelom FaaS, CaaS doda dodaten sloj v obliki zabojnikov, v katere zapakiramo bodisi uporabnikove aplikacije ali storitve. Zabojniki se namestijo na izbrano oblačno platformo, ki ob sprožitvenem dogodku zagotovi ali onemogoči posamezen zabojnik, odvisno od trenutnih zahtev. Dodaten nivo abstrakcije, v obliki pakiranja storitev v zabojnike, doda na kompleksnosti, a nam hkrati odpira dodatne možnosti na področju prilagajanja izvajalnega okolja in izvajanja aplikacij [11].

2.3. Razlike med arhitekturnima pristopoma

Kot smo že omenili ima brezstrežniško računalništvo več skupnih značilnosti z mikrostoritvami, vendar ima tudi nekaj razlik. V tabeli 1 prikazujemo primerjavo mikrostoritvene in brezstrežniške arhitekture po posameznih področjih, ki so pomemben faktor pri odločanju katero izmed arhitektur izbrati za razvoj informacijske rešitve.

Tabela 1: Prednosti in slabosti mikrostoritvene in brezstrežniške arhitekture.

	Mikrostoritvena arhitektura	Brezstrežniška arhitektura
Avtonomnost	Na nivoju posamezne mikrostoritve, ki deluje neodvisno od drugih in ima svoj življenjski cikel ter je zadolžena za določeno poslovno zmogljivost.	Na nivoju posamezne funkcije, ki bi morale biti oblikovane tako, da opravijo eno samo delo kot odgovor na dogodek.
Skalabilnost	Omogočeno je povečevanje ali zmanjšanje zmogljivosti storitev glede na potrebe. Lahko postane težavno, kadar za skaliranje skrbimo sami ali kršimo načela horizontalne skalabilnosti (npr. ohranjanje stanja).	Popolnoma samodejno skaliranje funkcij, ki se prilagajajo neodvisno od obremenitve. Zelo veliko zahtev se lahko obravnava kot zahteva enega odjemalca.
Odpornost	Vsaka mikrostoritev deluje samostojno v svojem okolju, tako izpad ene storitve ne pomeni izpad celotne aplikacije.	Ob izpadu posamezne funkcije ali ostalih delov smo odvisni od izbranega ponudnika, ki mora zagotoviti ustrezno politiko v primeru izpada.
Fleksibilnost	Vsako storitev lahko zgradimo z uporabo jezika ali ogrodja, ki ga potrebujemo, ne da bi to vplivalo na komunikacijo med mikrostoritvami.	Odvisni smo od izbranega ponudnika. Nekateri omogočajo hkratno podporo več jezikom in ogrodjem (npr. Google Cloud) – vsaka funkcija ima svoje izvajalno okolje – nekateri hkrati samo enemu jeziku (npr. Firebase).
Decentralizirano upravljanje podatkov	Mikrostoritve ponujajo možnost posameznega in neodvisnega dostopa do izvora podatkov, kar pomeni, da vsaka mikrostoritev upravlja s svojo podatkovno bazo.	Razvijalci lahko napišejo in uvedejo kodo, medtem ko ponudnik oblaka zagotovi strežnike za izvajanje njihovih aplikacij, podatkovne baze in sisteme za shranjevanje v poljubnem obsegu. Odvisni smo od izbranega ponudnika – praviloma izbrana podatkovna baza.

Vir: [4, 9, 11, 12]

Prednosti in slabosti posameznega arhitekturnega pristopa se pokažejo, ko želimo implementirati informacijsko rešitev, ki ponuja možnost uporabe tako mikrostoritvene, kot brezstrežniške arhitekture. V nadaljevanju bomo podali ključne iztočnice, ki vplivajo na izbiro arhitekturnega stila [11]:

– *Brezstrežniški sistemi lahko imajo ob začetni konfiguraciji podaljšanj čas prvega zagona.*

Kadarkoli je funkcija sprožena ali poklicana preko zahteve uporabnika, je razporejena v na novo ustvarjen vsebnik. Tako obstaja določeno kratko časovno obdobje, ko zahteva počaka, dokler ni vsebnik pripravljen za izvedbo. Ta pojav se v programerskem inženirstvu imenuje problem hladnega zagona.

– *Strategija uvajanja mikrostoritev ima težave z uravnoteženjem obremenitve in prerazporeditvijo prometa.*

Kljub morebiti daljšemu prvemu zagonu, so po začetnem obdobju brezstrežniški sistemi zelo stabilni. Nasprotno imajo lahko mikrostoritvene arhitekture težave ob povečani obremenitvi. Zaradi potrebe po prerazporeditvi prometa po dodatnih instancah storitev in hitrem skaliranju ob višji obremenitvi, se soočamo z zakasnitvijo pri podajanju odgovorov na zahteve. Zaradi narave brezstrežniških arhitektur so ti sistemi v takšnih situacijah praviloma hitrejši in ponujajo enakomerno časovno zakasnitev preko celotnega obdobja delovanja.

– *Strategija uvajanja mikrostoritev je uspešnejša pri pridobivanju majhnih in ponavljajočih se zahtev.*

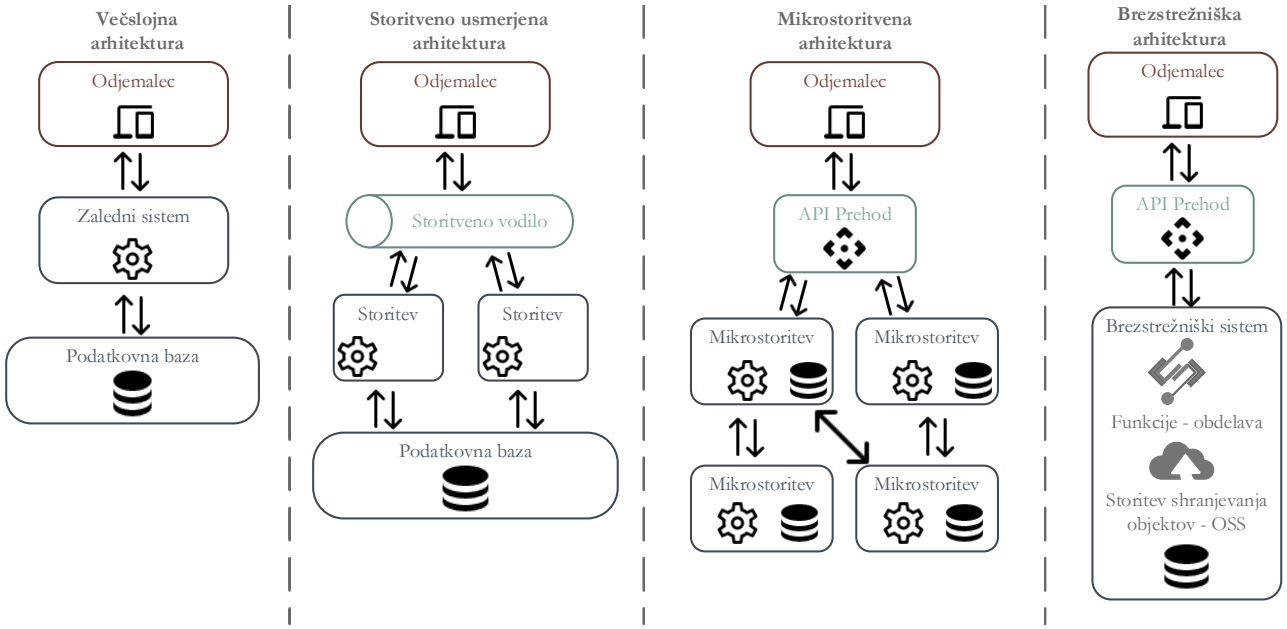
Kadar vemo, da bo imel naš informacijski sistem veliko število preprostih ponavljajočih zahtev z majhno velikostjo, je ustrezneje uporabiti mikrostoritveno arhitekturo. Izognemo se lahko velikemu strošku, ki bi nastal, kadar bi nam ponudnik brezstrežniške platforme zaračunaval za obdelavo vseh teh majhnih zahtev. Prav tako nimamo težav s hladnim zagonom, ki je najbolj opazen ob proženju majhnih zahtev.

– *Brezstrežniški sistemi so okretnejši in omogočajo lažjo konfiguriranje, ko govorimo o skalabilnosti.*

Ena izmed glavnih prednosti brezstrežniških sistemov, je odlična podpora samodejni skalabilnosti za katero skrbi infrastruktura izbranega ponudnika. Pri uporabi mikrostoritvene arhitekture je to pomanjkljivost mogoče odpraviti s konfiguracijo ustreznega mehanizma, po navadi z uporabo ustreznega orkestratorja (npr. Kubernetes).

3 Evolucija arhitektur programske opreme

Mikrostoritve izvirajo iz razvoja programske arhitekture z namenom zmanjšanja kompleksnosti monolitnih in storitveno usmerjenih sistemov. Pri mikrostoritveni arhitekturi gre za majhne in avtonomne storitve, ki imajo en sam jasno določen namen. Omogočajo navpično razgradnjo aplikacij v podmnožico poslovno usmerjenih neodvisnih storitev, povezanih z lahkimi komunikacijskimi protokoli ali vmesniki (API). Vsako storitev lahko neodvisno razvijejo, uvedejo in preizkusijo različne razvojne skupine z uporabo različnih tehnoloških skladov. Mikrostoritve je mogoče razviti v različnih programskih jezikih, lahko se prilagajajo neodvisno od drugih storitev. Namestiti jih je mogoče na strojno opremo, ki najbolj ustreza njihovim potrebam. Običajno so odzivne na njihov vmesnik, ki je primarni mehanizem za interakcijo z logiko. Slika 3 prikazuje razvoj arhitekturnih pristopov informacijskih sistemov.



Slika 3: Evolucija arhitektur.
Vir: lasten.

Nasprotno se brezstrežniške tehnologije odzivajo na dogodke, medtem ko so API-ji predvsem mehanizmi, ki so namenjeni za generiranje dogodkov. Sprožitev funkcije lahko izvedemo iz različnih virov dogodkov, kot so podatkovna baza ali sporočilni posrednik, pri čemer ne potrebujemo vmesnika API. Ključna razlika med mikrostoritvami in brezstrežniškimi sistemi je manjša razdrobljenost brezstrežniških funkcij in sprejetje paradigme, ki temelji na dogodkih. Kljub temu, da je mogoče razviti tudi mikrostoritve, ki so vodene na podlagi dogodkov, je za dogodkovno vodene sisteme zelo priporočljivo uporabiti brezstrežniški pristop. Ena izmed možnosti združevanja mikrostoritvene in brezstrežniške arhitekture je z uporabo različnih vzorcev**.** Kot primer lahko navedemo uporabo vzorca Saga, vratar (angl. Gatekeeper) in podobni. [9]

3.1. Razlikovanje mikrostoritvenega in brezstrežniškega arhitekturnega pristopa

Čeprav brezstrežniški arhitekturni pristop še morda ni dozorel, v primerjavi z mikrostoritvami ne izgublja v smislu svoje izvedljivosti, zanesljivosti in potenciala v prihodnosti. Brezstrežniška arhitektura ni neposredno povezana z mikrostoritvami, vendar obstajajo podobnosti med obema, kot so delitev poslovanja, možnost razvoja aplikacij brez stanja in agilne funkcionalnosti. [12]

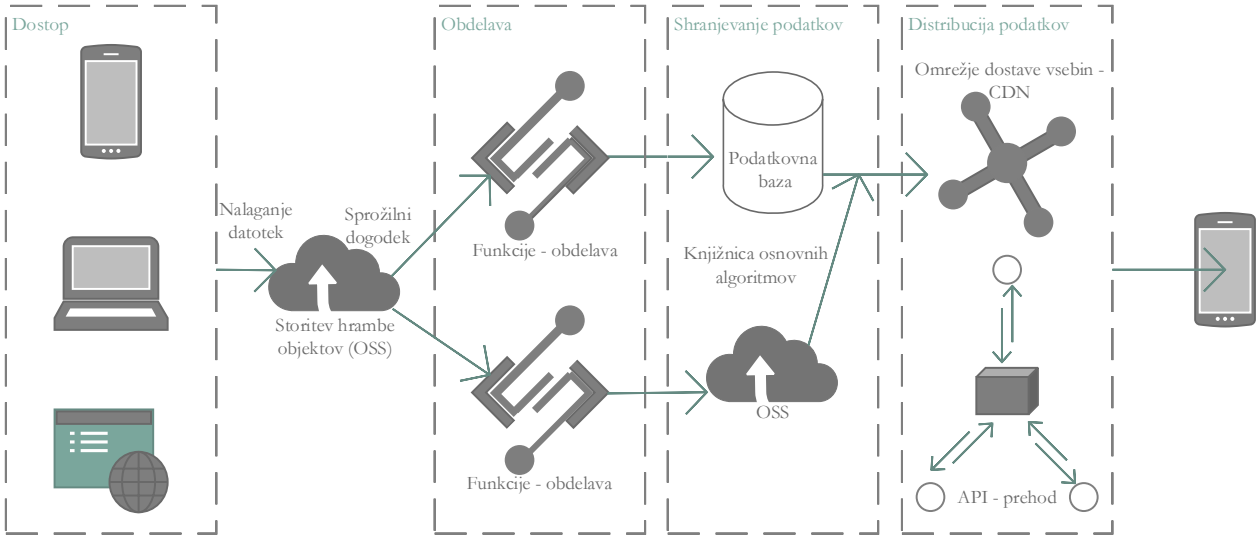
Pri razvoju novih sistemov v programskem inženirstvu, sledimo uporabi dokazano uspešnih gradnikov in izkušenj, kar je eden izmed temeljev vsake inženirske discipline. Na višjem nivoju abstrakcije se srečamo s ponovno uporabo s pomočjo vzorcev (angl. software patterns). Ena izmed splošno sprejetih skupin načrtovalskih vzorcev mikrostoritvene arhitekture, ki nudi povezovanje uporabe mikrostoritvene in brezstrežniške arhitekture, je skupina imenovana Izvajanje mikrostoritev (angl. Deployment patterns), v kateri najdemo vzorce, ki so povezani z izvajanjem posameznih mikrostoritev in celotne informacijske rešitve. Vzorec, ki mu bomo posvetili pozornost, je vzorec imenovan uvedba brez strežnika.

– *Uvedba brez strežnika* (angl. Serverless deployment).

  Vzorec govori o tem, da je najprej potrebno izbrati ustrezno uvedbeno infrastrukturo, ki ne razkriva kateri strežniški koncept je uporabljen – fizični ali virtualni gostitelj, zabojniki. Torej izberemo ponudnika brezstrežniških storitev, ki ustreza našim zahtevam. Ponudnik nam zaračunava na podlagi uporabe njihove uvedbene infrastrukture in je zadolžen za upravljanje infrastrukture na nizki ravni. Na uvedbeno

infrastrukturo namestimo informacijsko rešitev, ki je za to primerna [4]. Opisan vzorec je prikazan na sliki 4.

Čeprav gre za vzorec, ki je naveden znotraj katalogov dobrih praks razvoja mikrostoritev, ne smemo pozabiti, da se pri uporabi tega vzorca oddaljujemo od uporabe mikrostoritvene arhitekture (in uporabimo brezstrežniki arhitekturni pristop). Vzorec uvedba brez strežnika nam omogoča komplementarno dopolnjevanje in možnost, da razmislimo kateri izmed arhitekturnih pristopov je ustreznejši za našo informacijsko rešitev.



Slika 4: Načrtovalski vzorec uvedba brez strežnika.
Vir: [4]

Ko so na trg prišle prve platforme brez strežnikov, so strokovnjaki pogosto začeli napačno graditi mikrostoritve z uporabo funkcij brez strežnikov. Na primer, informacijsko rešitev lahko zgradimo kot mikrostoritev s prehodom (API) - vmesnik storitve, za njim pa uporabimo brezstrežniške funkcije in logiko, ki deluje kot usmerjevalnik. Ta zelo preprost pristop omogoča aplikacijam neodvisno prilagajanje velikosti, vendar še vedno ne izkorišča pravilno prednosti brezstrežniškega računalništva. [9]

Mikrostoritve so najbolj primerne za dolgotrajne, kompleksne aplikacije, ki imajo velike zahteve glede virov in upravljanja. Funkcije brez strežnika vodijo dogodki in se zato izvajajo le, ko je to potrebno. Ko je izvedba končana, se računalniška instanca, ki izvaja funkcijo, prekine. Brezstrežniške tehnologije so bolj primerne za obdelavo dogodkov in naloge, ki ne zahtevajo veliko virov. Brezstrežniški sistemi se prilagajajo samodejno, ne da bi za to potrebovali posebno infrastrukturo in so zato primernejši, ko razvijalci potrebujejo samodejno prilagajanje in nižje stroške izvajanja. Na drugi strani nam mikrostoritve ponujano večjo prilagodljivost, saj nismo odvisni od tretjih ponudnikov [12]. Povzetek osnovnih značilnosti posameznega arhitekturnega pristopa je zbran v tabeli 2.

Tabela 2: Karakteristike mikrostoritvene in brezstrežniške arhitekture.

	Mikrostoritvena arhitektura	Brezstrežniška arhitektura
Poslovni razcep	Vezano na storitev	Vezano na funkcijo
Zahteva strežnika	Deljenje stanja pomnilnika med klici	Popolnoma brez stanja
Tretje odvisnosti	Prosta izbira tujih odvisnosti	Znašanje na tuje odvisnosti - zagotavlja BaaS
Nivo abstrakcije	Razvojni model	Računalniška platforma

Vir: [12]

4 Zaključek

Na podlagi pregleda in opisa mikrostoritvene in brezstrežniške arhitekture ugotavljamo, da nobena posamezna vrsta arhitekturnega stila ne more ustrezati vsem vrstam aplikacij. Iz razprave v tem delu in po analizi brezstrežniških sistemov ter mikrostoritev ugotavljamo, da ima vsaka tehnologija svoje prednosti in slabosti. Vsak izmed obeh pregledanih arhitekturnih stilov, rešuje izzive na svoj način. Tako moramo biti pazljivi, da ju ustrezno ločujemo, saj se razlikujeta v svojih osnovnih načelih. Odvisno od zahtev sistema, lahko gresta oba arhitekturna stila z roko v roki, pri čemer ne smemo kršiti pravil implementacije. Primer sočasne uporabe je najpreprosteje ponazoriti s primerom informacijske rešitve, ki ima uporabniški vmesnik zgrajen s pomočjo več mikrostoritev, od katerih se pričakuje, da se nenehno izvajajo. Modele brezstrežniškega arhitekturnega stila pa lahko uporabimo, kot dopolnitve k celotnemu sitemu, ki skrbijo za proženje občasnih časovnih in dogodkovnih funkcij ali za varnost celotnega sistema. Kljub temu, da gre pri brezstrežniškem arhitekurnem stilu za novejši pristop, ki vedno bolj pridobiva na veljavi, se izogibamo nesmiselnemu preoblikovanju sistemov, ki uporabljajo mikrostoritveni arhitekturni stil na način, da bi vsako posamezno mikrostoritev poskušali implementirati v obliki več funkcij, ki se izvajajo na brezstrežniškem sistemu. Omeniti je potrebno, da brezstrežniški arhitekurni stil predstavlja enega izmed arhitekturnih stilov in ni namenjen zamenjavi za dalj časa uveljavljen mikrostoritveni arhitekturni stil. Na podlagi primerov uporabe in poslovnih potreb lahko uporabniki skrbno prepoznajo, kateri arhitekturni stil bodo uporabili in ga ustrezno uvedejo glede na funkcionalne potrebe. Brezstrežniška arhitektura je primerna za dogodkovno vodene sisteme, kjer se mora izvesti dejanje, ko pride do sproženega dogodka. Večinoma gre za funkcijsko voden pristop, kjer je nadzor pri gostitelju in ne pri uporabniku. Osrednja modela, ki se pojavljata pri brezstrežniškem arhitekturnem stilu sta BaaS in FaaS. Posebej moramo opozoriti na model CaaS, ki ponazarja enega izmed načinov izvajanja mikrostoritev pri uporabi mikrostoritvenega arhitekturnega stila in se oddaljuje od načel in modelov, ki so značilni za sisteme, ki uporabljajo brezstrežniški arhitekturni stil. Mikrostoritve so modularne, domenske, samostojne skupine storitev, ki lahko delujejo neodvisno druga od druge. Oba arhitekturna stila nam pri gradnji sodobnih rešitev v oblaku in s pravilno izbiro pristopov omogočata, da zgradimo zelo robustne spletne aplikacije z minimalnimi stroški, ki so ob enem zelo prožne, razširljive ter varne.

Literatura

[1] V. Raj and S. Ravichandra, "Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services," in *3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2018, p. 1531–1536.

[2] J. Lewis in M. Fowler, „Microservices a definition of this new architectural term," [Elektronski]. Na voljo: https://martinfowler.com/articles/microservices.html [dosegljivo 7/2022].

[3] N. Alshuqayran, N. Ali and R. Evans, "Systematic Mapping Study in Microservice Architecture«," in *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2016, p. 44–51.

[4] T. Hliš and L. Pavlič, *Dobre prakse pri razvoju mikrostoritev : študijski program: Informatika in tehnologije komuniciranja,* 2021.

[5] E. Eyk, Iosup, &. Alexandru, J. Grohmann, Eismann, &. Simon, Bauer, &. André, L. Versluis, L. Toader, N. Schmitt, Herbst and C. Nikolas & Abad, "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms," *IEEE Internet Computing. PP,* p. 1–1, 2019.

[6] B. Jambunathan and K. Yoganathan, "Architecture Decision on using Microservices or Serverless Functions with Containers," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, 2018, p. 1–7.

[7] R. Collier, O'neill, &. Eoin, Lillis, &. O. David and G. Hare, *MAMS: Multi-Agent MicroServices,* 2019.

[8] R. Rajan, "Serverless Architecture - A Revolution in Cloud Computing," in *Tenth International Conference on Advanced Computing (ICoAC)*, 2018, p. 88–93.

[9] D. Taibi, J. Spillner and K. Wawruch, "Serverless Computing-Where Are We Now, and Where Are We Heading?," in *IEEE Software*, vol. 38, 2021, p. 25–31.

[10] A. Rajan, *A review on serverless architectures - function as a service (FaaS) in cloud computing. TELKOMNIKA (Telecommunication Computing Electronics and Control),* 2020.

[11] K. Burkat, M. Pawlik, B. Balis, M. Malawski, K. Vahi, M. Rynge, R. F. d. Silva in E. Deelman, „Serverless Containers – Rising Viable Approach to Scientific Workflows," v *2021 IEEE 17th International Conference on eScience (eScience)*, 2021.

[12] C.-F. &. Fan, Jindal and M. Gerndt, *Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application,* 2020.

[13] L. Jiang, Pei and J. Zhao, "Overview Of Serverless Architecture Research," in *Journal of Physics: Conference Series*, 2020.

Razvoj večplatformne aplikacije za prikaz naprav pametnega doma

Sebastjan Mevlja

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Ljubljana, Slovenija
sm9299@student.uni-lj.si

Sinopsis Danes aplikacije uporabljamo na različnih napravah in na različne načine. Praviloma so avtohtone aplikacije hitrejše, ponujajo več funkcionalnosti ter omogočajo boljšo uporabniško izkušnjo. Slabost teh aplikacij je, da jih je potrebno razviti za vsako platformo posebej, kar je pogosto drago in zahtevno. Telekom Slovenije, d. d., razvija avtohtone aplikacije za upravljanje NEO pametnega doma za vsako platformo posebej. Zanimalo nas je, ali bi lahko več aplikacij nadomestili z eno večplatformno aplikacijo. Naredili smo primerjavo večplatformnih ogrodij in za razvoj aplikacije izbrali ogrodje Flutter. Izdelali smo večplatformno aplikacijo za upravljanje NEO pametnega doma, ki omogoča pregled povezanih naprav. Aplikacija podpira Android in iOS mobilni platformi ter spletne brskalnike. Postavili smo tudi avtomatiziran proces testiranja in gradnje aplikacije z uporabo platforme Github Actions.

Ključne besede:
Flutter
Android
iOS
splet
večplatformne aplikacije
pametni dom
CI/CD

Opomba: Prispevek temelji na: Mevlja, S. (2022). Razvoj večplatformne aplikacije za NEO pametni dom: diplomsko delo, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Ljubljana: S. Mevlja.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.17

1 Uvod

Danes aplikacije uporabljamo na različnih napravah in na različne načine. Lahko jih namestimo na svoje mobilne telefone, tablice in računalnike, lahko jih uporabljamo preko spletnega brskalnika. Za doseganje priljubljenosti aplikacij med uporabniki morajo te ustrezati vedno višjim standardom – morajo biti privlačne, enostavne ter odzivne. Praviloma so avtohtone[1] aplikacije hitrejše, ponujajo več funkcionalnosti ter omogočajo boljšo uporabniško izkušnjo. Njihova slabost pa je, da jih je potrebno razviti za vsako platformo posebej, kar je pogosto drago in zahtevno. Podjetja morajo torej podobno aplikacijo razviti za več platform, za to pa potrebujejo razvijalce z različnimi znanji.

Po podatkih portala statista.com [1] je junija 2022 med vsemi operacijskimi sistemi na trgu prevladoval Android z več kot 44 %, sledi Windows z 19 % ter iOS s 17 %. Pogosto želimo z aplikacijami podpreti več trgov in platform, vendar je lahko razvoj takšnih aplikacij zelo drag, zato se podjetja vedno pogosteje odločijo za razvoj večplatformnih[2] aplikacij. Večplatformne aplikacije hkrati pokrivajo več trgov in podjetja lažje dosežejo več uporabnikov. S tem podjetja prihranijo tudi denar, ker opravijo manj dela za razvojni proces in vzdrževanje, olajša se zagotavljanje konsistence med platformami in izboljša se uporabniška izkušnja.

Telekom Slovenije, d. d., razvija aplikacijo NEO [2] za vsako platformo posebej, kar zahteva veliko časa in denarja. Obstoječa aplikacija podpira mobilne naprave Android in iOS ter spletne brskalnike. Trenutno ne obstaja večplatformna aplikacija za upravljanje NEO pametnega doma. Aplikacija poleg upravljanja pametnega doma omogoča tudi spremljanje televizijskih kanalov in videotek.. Osredotočili se bomo na del aplikacije za upravljanje pametnega doma, saj je kompleksnejši.

Glavni cilj je preveriti, ali je mogoča izdelava večplatformne aplikacije z uporabniškim vmesnikom v slovenščini za upravljanje NEO pametnega doma Telekoma Slovenije, d. d. Aplikacija bo omogočala prikaz naprav pametnega doma ter podpirala Android in iOS mobilne naprave ter spletne brskalnike. Za uporabo aplikacije bo potrebna prijava z NEO uporabniškim imenom in geslom. Najtežji del naloge bo predstavljala integracija obstoječega paketa za upravljanje pametnega doma OBLO [3], saj ta ne podpira večplatformnega razvoja in je razvit za vsako platformo posebej. Primerjali bomo najbolj priljubljena ogrodja za razvoj večplatformnih aplikacij in izbrali najustreznejše. Za preverjanje delovanja aplikacije bodo pripravljeni avtomatski testi. Postavili bomo tudi avtomatiziran proces testiranja in gradnje aplikacije.

2 Ogrodja za razvoj večplatformnih aplikacij

V današnjem konkurenčnem in hitro razvijajočem okolju razvijalci nenehno iščejo orodja za razvoj aplikacij, ki bi jim olajšala delo in zmanjšala stroške. Kot odgovor na to povpraševanje se je na trgu pojavilo veliko ogrodij za razvoj večplatformnih aplikacij. Primerjavo med avtohtonimi in večplatformnimi aplikacijami prikazuje tabela 1. Opisali bomo najpomembnejše prednosti večplatformnih aplikacij.

[1] avtohtona – izvorno "native"
[2] večplatformna – izvorno "cross-platform"

Tabela 1: Primerjava med avtohtonimi in večplatformnimi aplikacijami [9].

kategorija/vrsta aplikacije	avtohtona	večplatformna
arhitektura	odvisna od platforme	enotna
cena razvoja	višja	nižja
ponovna uporaba kode	nizka	visoka
dostop do strojne opreme	popoln	omejen
UI in UX	odvisno od platforme	konsistentno
zmogljivost	višja	nižja
ciljni trg	ena platforma	več platform
trajanje razvoja	daljše	krajše
število razvijalcev	več	manj

2.1 Ponovna uporaba programske kode

Največja prednost večplatformnih aplikacij je zagotovo ponovna uporaba kode. Razvijalci programsko kodo napišejo enkrat, čeprav aplikacija deluje na več platformah. To odpravlja ponavljanje in posledično prihrani čas ter stroške.

2.2 Cenejši in krajši razvoj

Ogrodja za razvoj aplikacij za več platform ponujajo dobro ravnovesje med kakovostjo in ceno, saj omogočajo, da z manj razvijalci, ki dobro obvladajo svoje področje, pokrijemo več platform. Manj časa, sredstev in truda, porabljenega za razvoj aplikacij, je neposredno sorazmerno z nižjimi stroški.

2.3 Enostavnejša namestitev in vzdrževanje

Ker se večina izvorne kode napiše samo enkrat, imajo razvijalci manj dela s pisanjem in vzdrževanjem kode.
To pomeni enostavno in hitro uvajanje sprememb, vzdrževanje, objavljanje posodobitev ter popravljanje napak.
Aplikacije na več platformah so posodobljene hkrati, kar je pomembno za konsistentnost.

2.4 Večji trg

Večplatformne aplikacije dosegajo širši tržni doseg, saj z njimi lažje dosežemo več uporabnikov, ki uporabljajo različne platforme.

2.5 Konsistenten uporabniški vmesnik

Večplatformne aplikacije nudijo enotni uporabniški vmesnik na različnih platformah, hkrati pa spoštujejo standarde, specifične za platformo. Če uporabnikom ponudimo konsistentno izkušnjo, bodo znali aplikacijo uporabljati na kateri koli platformi.

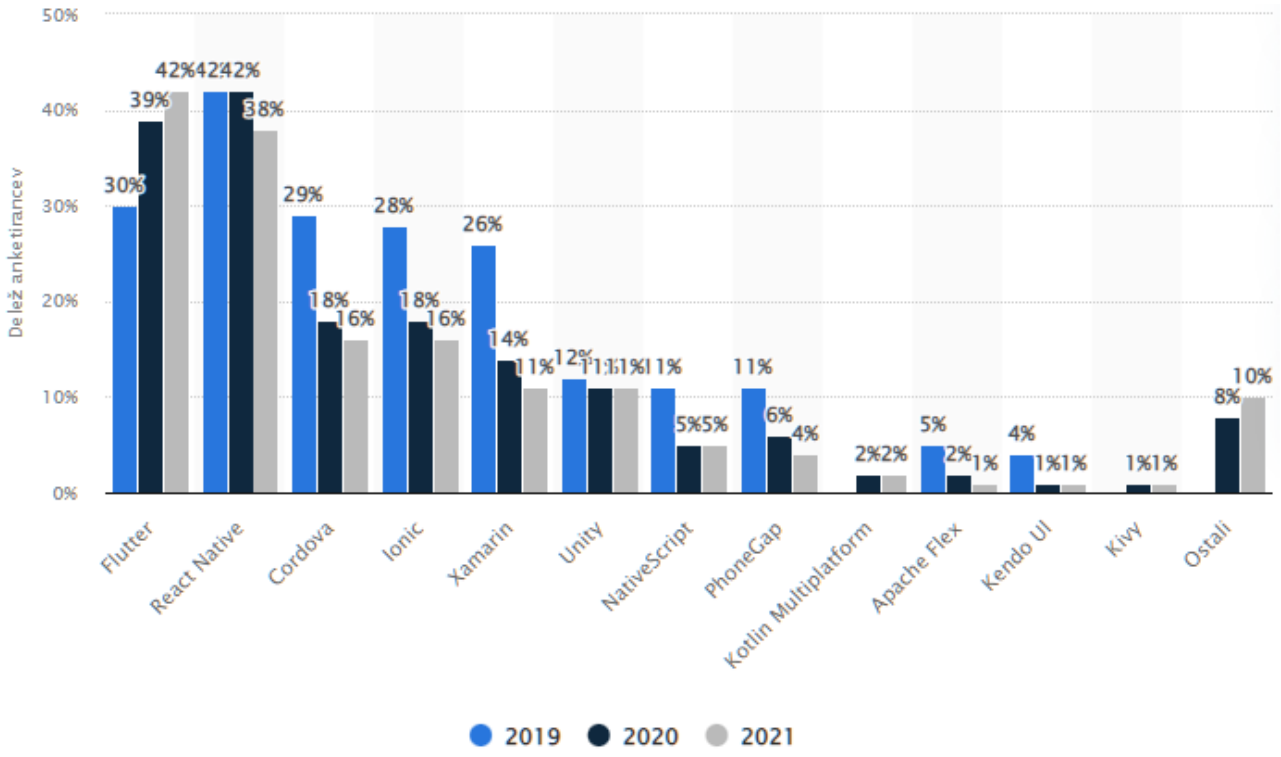
3 Primerjava večplatformnih ogrodij

Obstaja veliko ogrodij za razvoj aplikacij za več platform, ki imajo svoje prednosti in slabosti. Ker vsa ogrodja niso primerna za vse vrste aplikacij, smo pripravili primerjavo najbolj priljubljenih ogrodij, ki jo prikazuje tabela 2.

Tabela 2: Primerjava najbolj priljubljenih ogrodij [9].

kategorija/ogrodje	Flutter	React Native	Apache Cordova
Android podpora	da	da	da
iOS podpora	da	da	da
Windows podpora	da	da	da
Linux podpora	da	ne	ne
Mac podpora	da	da	ne
podpora za splet	da	da	ne
hitrost aplikacije	najvišja	srednja	najnižja
hiter ponovni zagon	da	da	ne
velikost aplikacije	srednja	velika	majhna
kompleksnost	srednja	srednja	nizka
sistemske funkcije	vse	večina	nekatere
vizualna konsistenca	da	ne	da
vtičniki in knjižnice	srednje	veliko	malo
programski jezik	Dart	JavaScript	JavaScript
plačljivo	ne	ne	ne
cena razvoja ($/h)	15–50	15–25	30–80
Priljubljenost	visoka	srednja	nizka
gradniki vmesnika	lastni	sistemski	lastni
poraba virov	nizka	srednja	visoka

Razna ogrodja za razvoj večplatformnih aplikacij so v zadnjih letih zelo napredovala in so vedno bolj priljubljena. Najbolj priljubljen je Flutter, ki je prehitel React Native. Uporablja ga 42 % razvijalcev večplatformnih aplikacij, medtem ko React Native 38 % [6]. To je tudi razvidno iz slike 1, ki prikazuje rezultate analize priljubljenosti večplatformnih ogrodij.



Slika 1: Priljubljenost ogrodij za razvoj večplatformnih aplikacij.
Vir: [4].

3.1 Izbira večplatformnega ogrodja

S svojo aplikacijo želimo podpreti Android in iOS mobilne naprave ter spletne brskalnike. Zaradi omejitev podpore različnih platform ogrodje Apache Cordova ne ustreza zahtevam. Zaradi večjega števila podprtih platform, hitrejših in manjših končnih aplikacij, dostopa do več sistemskih funkcij naprav, vizualne konsistence, manjše porabe virov [5] in višje priljubljenosti [4] smo za razvoj aplikacije izbrali ogrodje Flutter.

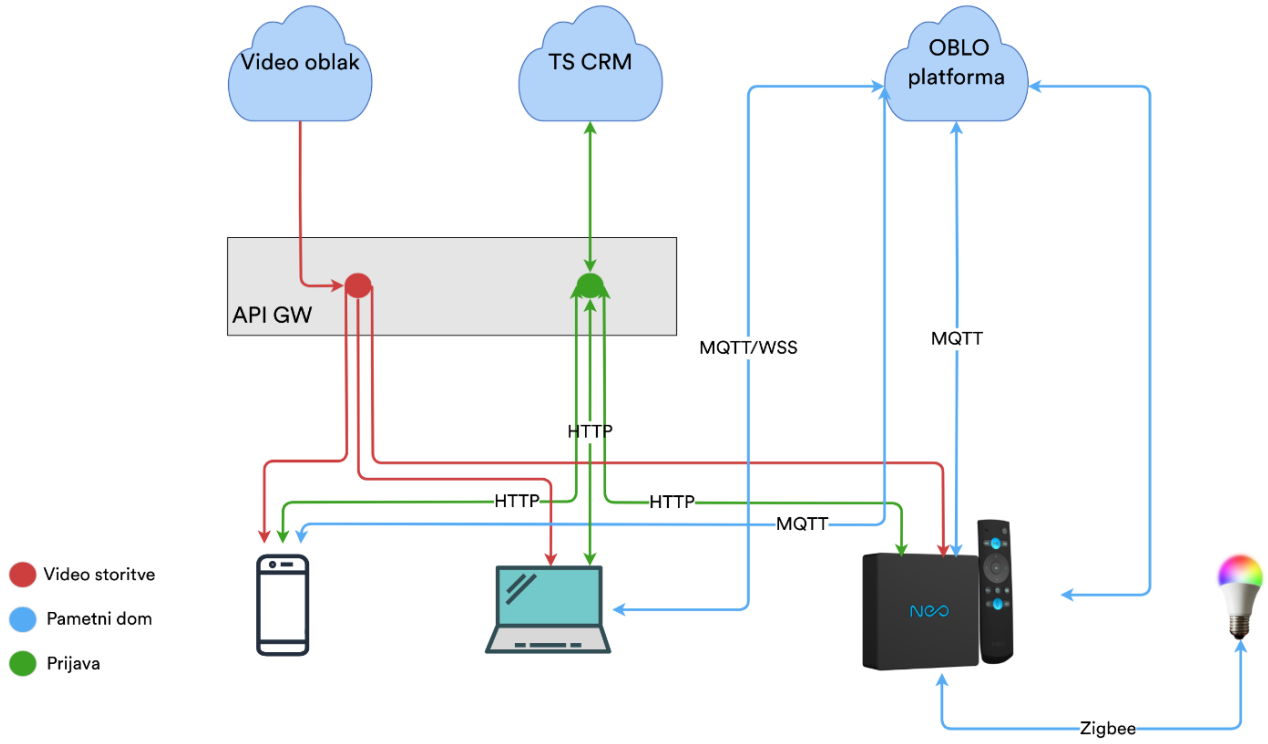
4 Orodja in tehnologije

4.1 Ogrodje Flutter

Flutter je brezplačno in odprtokodno ogrodje, namenjeno razvoju aplikacij za več platform. Omogoča hiter razvoj avtohtonih aplikacij za mobilne naprave, spletne brskalnike ter računalnike iz ene izvorne kode. Razvijalcem nudi vse, kar je potrebno za ustvarjanje aplikacij za več platform. Ponuja vse potrebne gradnike, izrisovalni pogon ter vmesnike za testiranje. Razvili so ga pri podjetju Google in ga tudi redno vzdržujejo. Njegova edinstvena lastnost je, da namesto uporabe sistemskih gradnikov, ki so na voljo v brskalniku oziroma na mobilni napravi, te upodobi sam s pomočjo lastnega 2D izrisovalnega pogona Skia. Za razvoj uporablja programski jezik Dart in omogoča integracijo z obstoječo avtohtono kodo (Kotlin in Swift). Na trgu obstaja že več kot pol milijona aplikacij, izdelanih v ogrodju Flutter [7].

4.2 Platforma NEO

NEO je platforma za pametno življenje Telekoma Slovenije, d. d.. Na enem mestu povezuje rešitve za dom in zabavo: spremljanje in napredno iskanje vsebin, upravljanje pametnih naprav, opravljanje nakupov preko televizijskega zaslona, glasovno upravljanje v slovenščini in igranje iger. Uporabniki lahko do platforme NEO dostopajo preko mobilne naprave, računalnika ali digitalnega sprejemnika v kombinaciji s televizijo. Vse naprave za spremljanje TV vsebin, avtorizacijo uporabnikov in upravljanje naprav pametnega doma pošiljajo sporočila na zaledni sistem, ki je sestavljen iz aplikacijskega vmesnika Stargate API Gateway in platforme za upravljanje pametnega doma OBLO. Digitalni sprejemnik NEO Smartbox je pogoj za uporabo pametnega doma, saj je med drugim tudi sprejemnik za Zigbee naprave. Za komuniciranje med napravami pametnega doma se uporablja protokol MQTT in platforma OBLO, ki ponuja vmesnike za pametni dom. Shema platforme NEO je prikazana na sliki 3.



Slika 3: Shema platforme NEO.
Vir: [9].

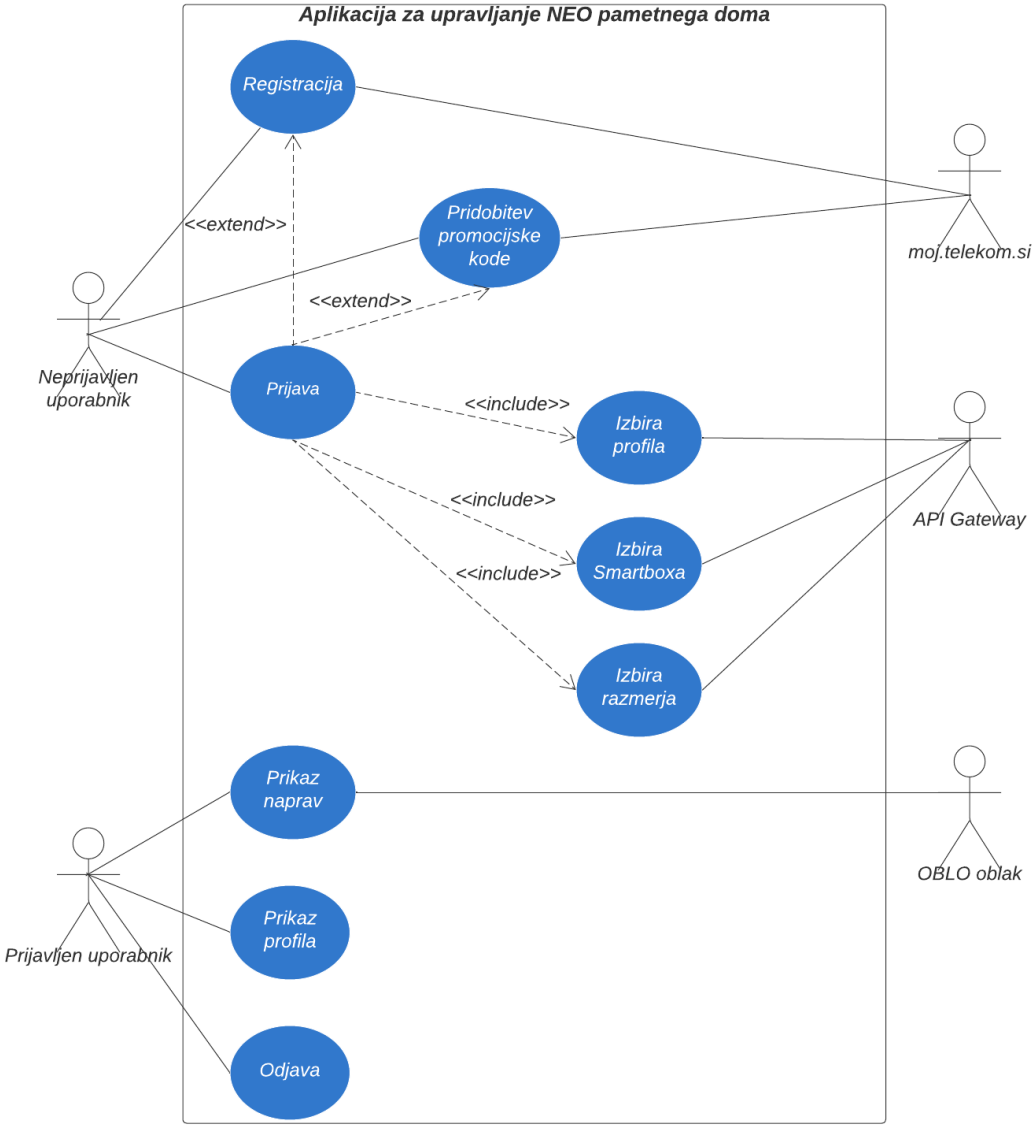
5 Razvoj aplikacije

Za razvoj smo izbrali ogrodje Flutter. Aplikacija podpira Android in iOS mobilne naprave ter spletne brskalnike. Med razvojem uporabniškega vmesnika smo sledili NEO oblikovalskim pravilom, zato je tema aplikacije temno modra z odtenkom vijoličaste. Namen razvoja aplikacije je bil preveriti, ali je možno razviti večplatformno aplikacijo za upravljanje NEO pametnega doma. Osrednji in bistveni namen aplikacije je prikaz naprav pametnega doma z uporabo protokola MQTT.

Med razvojem aplikacije je bil poudarek na uporabi dobrih praks in čitljivosti kode. Na začetku smo na podlagi uporabniških zgodb razvili statičen uporabniški vmesnik. Nadaljevali smo z implementacijo NEO prijave, s katero aplikacija pridobi sejo za dostop do zalednih sistemov. Sledil je najtežji del razvoja, to je priprava modula za povezavo s platformo OBLO in napravami pametnega doma. Zadnji del je bila povezava statičnega uporabniškega vmesnika s samo logiko aplikacije. Aplikacija je bila med razvojem temeljito testirana, napisanih je bilo tudi nekaj avtomatskih testov. Z uporabo CI/CD platforme Github Actions smo postavili delovni tok, ki skrbi za avtomatsko testiranje in grajenje aplikacije. Izvorna koda je bila na koncu formatirana in statično analizirana.

5.1 Funkcionalnosti aplikacije

Na sliki 4 so prikazani primeri uporabe aplikacije. Za pregled naprav pametnega doma je potrebna prijava z NEO uporabniškim računom. Če uporabnik računa nima, se lahko registrira ali zaprosi za pridobitev promocijske kode. Prijava je sestavljena iz več korakov. Uporabnik mora po vpisu uporabniškega imena in gesla izbrati uporabniško razmerje, profil in NEO Smartbox, na katerega se povezujejo naprave pametnega doma. Po uspešni prijavi je uporabniku omogočen pregled naprav pametnega doma, profila in odjava iz aplikacije.



Slika 4: Diagram funkcionalnosti aplikacije.
Vir: [9].

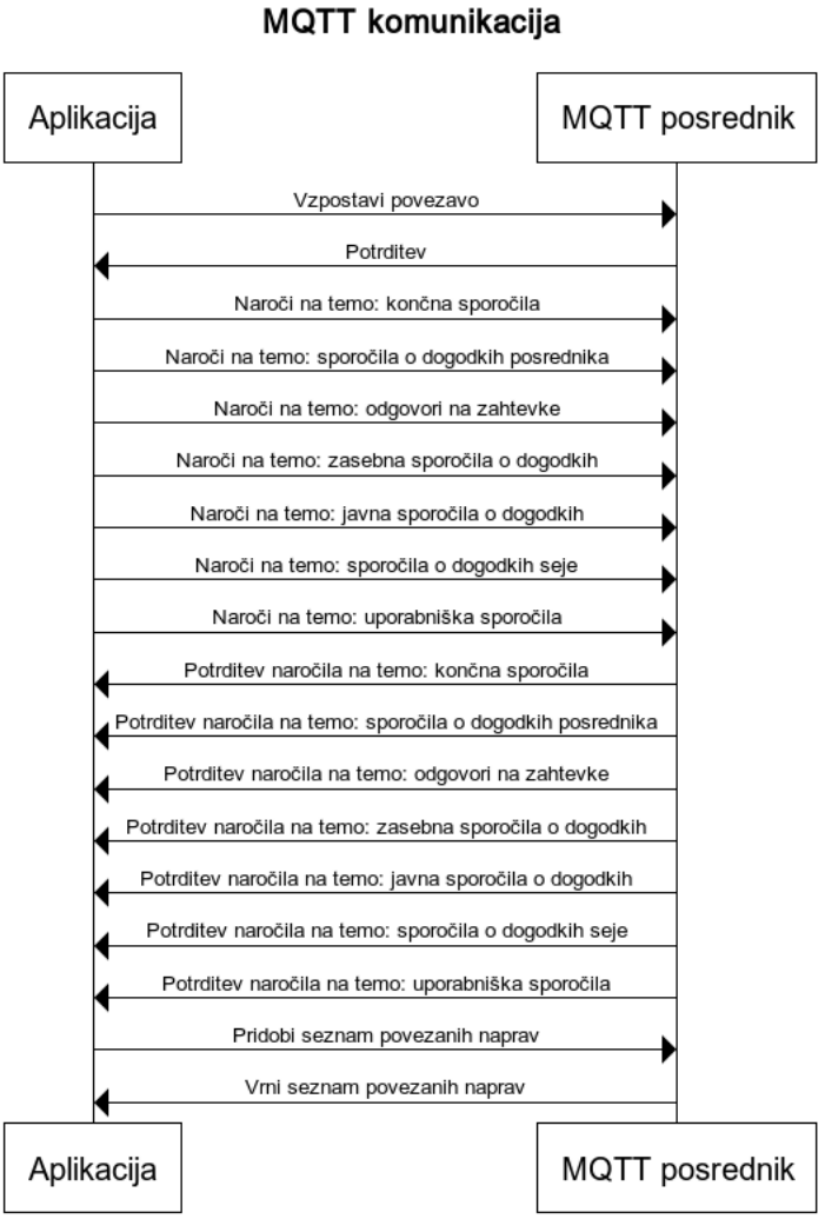
5.2 Integracije zalednih sistemov

Aplikacija uporablja obstoječi zaledni sistem Telekoma Slovenije, d. d. Za overjanje in avtorizacijo uporabnika se uporablja strežnik Stargate API Gateway. To je aplikacijski vmesnik, ki se uporablja tudi za komunikacijo z ostalimi zalednimi sistemi. Po uspešni prijavi uporabniku dodeli sejo za uporabo storitev pametnega doma.

Za delovanje pametnega doma se uporablja oblak OBLO, ki nudi vmesnike za pametni dom. Aplikacija se poveže na obstoječi posrednik MQTT, preko katerega pošilja in prejema sporočila. Naprave pametnega doma se z uporabo protokola Zigbee povežejo na digitalni sprejemnik NEO Smartbox, ki za komuniciranje z aplikacijo uporablja protokol MQTT.

5.3 Komunikacija MQTT

Komuniciranje z napravami pametnega doma poteka z uporabo protokola MQTT. Podatki se prenašajo v obliki JSON. Aplikacija vzpostavi povezavo s posrednikom MQTT in posreduje sejo, ki jo dobi ob prijavi. Način vzpostavitve povezave je odvisen od vrste naprave. Posebnost spletne različice je, da za komunikacijo uporablja spletne vtičnice[3]. Po vzpostavljeni povezavi se aplikacija naroči na vse potrebne teme in počaka na potrditev. Po prejeti potrditvi aplikacija zahteva seznam naprav pametnega doma, ki so trenutno povezane na NEO Smartbox. Posrednik aplikaciji pošlje seznam in naprave se prikažejo na uporabniškem vmesniku. Celoten postopek komunikacije prikazuje slika 5.



Slika 5: Diagram zaporedja poteka komunikacije MQTT.
Vir: [9].

[3] spletna vtičnica – izvorno "websocket"

5.4 Testiranje aplikacije

Pričakovano delovanje aplikacije smo potrdili z ročnim in avtomatskim testiranjem. Napisali smo avtomatske teste enot, gradnikov ter integracijske teste. Zaradi modularne zasnove aplikacije in odvisnosti od zunanjih storitev je bilo potrebno izdelati maketo[4] uporabljenih funkcij. Napisali smo 137 avtomatskih testov in dosegli 49,5-odstotno pokritost programske kode. Nismo dosegli visoke pokritosti programske kode, smo pa pokrili vse kritične dele aplikacije. Doseganje visoke pokritosti nam je tudi oteževalo problematično izdelovanje maket modulov ogrodja GetX [8] in knjižnice odjemalca MQTT. Delovanje aplikacije smo preverili tudi na različnih fizičnih napravah.

5.5 Delovni tok Github Actions CI/CD

Z uporabo Github Actions smo postavili delovni tok, ki se sproži ob vsaki objavi kode na Git veji main. Na začetku se požene virtualni računalnik Ubuntu 20.04 LTS in nastavi vse potrebne okoljske spremenljivke. Nanj se namestita Java in Flutter, ki ju bomo potrebovali za grajenje aplikacije. Iz Flutter repozitorija se naložijo vse knjižnice, ki jih aplikacija potrebuje. Ko je priprava okolja končana, se začne testiranje aplikacije. Na začetku se preveri format izvornih datotek in izvede statična analiza programske kode. Nadaljujemo z izvajanjem testov enot. Po prestanih testih se iz izvorne kode zgradita Android in spletna aplikacija. Namestitvene datoteke za Android aplikacijo se odložijo na Github, kjer so na voljo za prenos. Spletna aplikacija se postavi na Githubov spletni strežnik.

6 Ovrednotenje rezultatov

Razvili smo večplatformno aplikacijo za upravljanje NEO pametnega doma, ki omogoča pregled povezanih naprav. Mogoče jo je uporabljati na Android in iOS mobilnih platformah ter z uporabo spletnih brskalnikov.

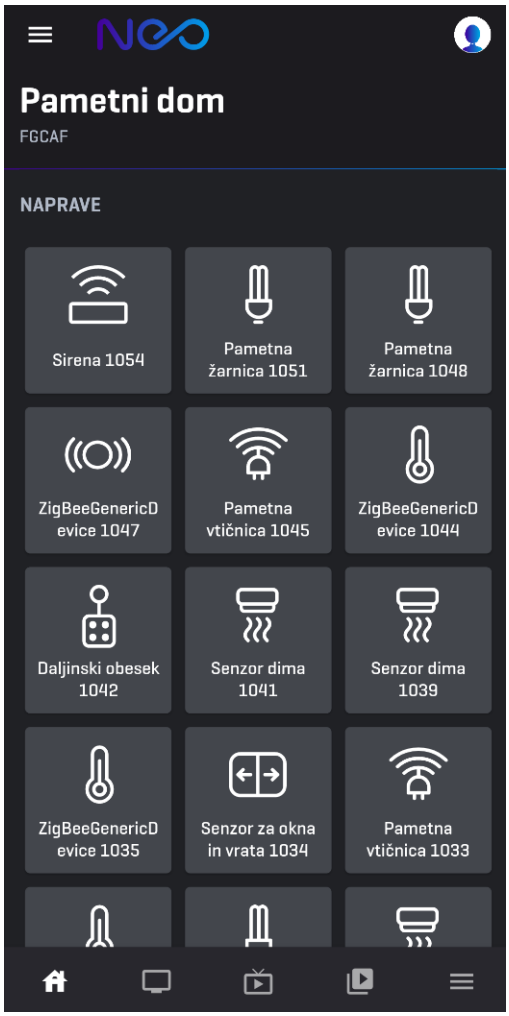
Ob zagonu aplikacije se prikaže začetni zaslon, ki je prikazan, dokler se v ozadju ne naloži vse potrebno. Po končanem nalaganju se prikaže zaslon za prijavo z obrazcem za uporabniške poverilnice. Prijava je sestavljena iz več obveznih zaporednih korakov. Prvi korak od uporabnika zahteva vpis uporabniškega imena in gesla. Če uporabnik nima uporabniškega računa, ga lahko ustvari s klikom na povezavo za registracijo. Uporabnik ima tudi možnost pridobiti tridesetdnevno promocijsko kodo, ki jo lahko uporabi za prijavo. Če uporabnik vpiše napačno uporabniško ime ali geslo, se prikaže rdeče obvestilo. Med čakanjem na odgovor zalednega sistema se prikaže okrogel kazalnik nalaganja.

Po opravljenem prvem koraku prijave uporabnik nadaljuje na naslednji korak. Drugi korak je izbira uporabniškega podračuna. Na zaslonu se prikaže seznam podračunov, ki so na voljo uporabniku. Večina uporabnikov ima samo en podračun. V tem primeru se izbira izvede samodejno, brez akcije uporabnika. Po izbiri uporabniškega podračuna uporabnik nadaljuje na naslednji korak. Tretji korak je izbira uporabniškega profila. Uporabnik izbere enega izmed ponujenih profilov in nadaljuje na naslednji korak. Zadnji korak prijave je izbira NEO Smartboxa, na katerega se povezujejo naprave pametnega doma. Če na tem koraku uporabnik izbere napačen NEO Smartbox, se naprave pametnega doma ne bodo pokazale.

Po uspešno opravljeni večstopenjski prijavi je uporabnik preusmerjen na glavno stran aplikacije. Prikaže se statična domača stran pametnega doma. Če ima uporabnik priključeno vsaj eno napravo, se statična stran zamenja z dinamičnim prikazom povezanih naprav, kot je vidno na sliki 6. Če je naprav več, kot jih lahko prikažemo na zaslonu, se prikaže drsnik, ki omogoča prikaz ostalih naprav. Postavitev spletne različice aplikacije se malenkost razlikuje, saj se zaradi drugačnega razmerja stranic zaslona lahko prikaže več naprav. Za vsako napravo se kreira

[4] maketa - izvorno "mockup"

nova kartica, na kateri sta ime naprave in ikona, ki se nastavi glede na vrsto naprave. Uporabnik lahko s klikom na sliko profila odpre stran za pregled profila. Tam lahko poleg pregleda izbranega podračuna, profila in NEO Smartboxa opravi tudi odjavo iz aplikacije.



Slika 6: Zaslon za prikaz naprav pametnega doma mobilne različice.
Vir: [9].

Razvita aplikacija deluje dobro na vseh ciljnih platformah. Uporabniški vmesnik je konsistenten in uporabniška izkušnja je dobra. Vse naprave NEO pametnega doma so podprte na vseh platformah. Aplikacije se razlikujejo v delovanju protokola MQTT, ker spletna različica za povezovanje uporablja spletne vtičnice. Razlikujejo se tudi v število prikazanih naprav, saj lahko spletna različica zaradi drugačnega razmerja stranic zaslona prikaže več naprav.

7 Zaključek

Med razvojem smo dobro spoznali orodje Flutter in njegove prednosti ter slabosti. Ugotovili smo, da je zelo dobro orodje za razvoj večplatformnih aplikacij, ki ima konstantno in urejeno sintakso ter ne zahteva pisanja odvečne kode. Flutter ponuja nesporne prednosti, kot sta hitrejši razvoj in prihranek stroškov, te prednosti pa lahko pomembno vplivajo na proces razvoja, ko gre za predvideno uporabo sredstev in hitrejše trženje izdelka. Hitrejši razvoj omogoča hitrejšo potrditev ideje, zgodnje testiranje uporabnikov in prihranek denarja. Flutter spreminja način razvoja aplikacij. Pod isto streho združuje mobilne, spletne in namizne aplikacije. Razvijalcem omogoča, da se osredotočajo na sam razvoj funkcionalnosti in ne zapravljajo časa za podpiranje vsake platforme posebej. Je

visoko zmogljivo in visoko produktivno orodje, ki skrajša razvojne cikle in omogoča enkratno pisanje izvorne kode za več platform. Hiter ponovni zagon pohitri razvoj in razhroščevanje aplikacij.

Zastavljene cilje smo uspešno izvedli. Dokazali smo, da je mogoče razviti večplatformno aplikacijo NEO. Razvili smo kakovostno in robustno aplikacijo, ki uporabniku omogoča pregled nad napravami NEO pametnega doma. Podprli smo veliko različnih naprav, saj aplikacija podpira vse naprave, ki so združljive z NEO pametnim domom.

Poseben izziv pri izdelavi aplikacije je bila implementacija komunikacije MQTT, saj platforma OBLO nima podpore za Flutter oziroma Dart. Uradni razvojni paketi obstajajo samo za Android, iOS in spletne aplikacije. Potrebno je bilo implementirati vmesnike MQTT, saj še niso obstajali. Pri tem se nismo mogli zanašati na obstoječo dokumentacijo, ker ta način uporabe ni bil podprt.

Možnosti za izboljšave je veliko. Aplikacija trenutno nima posebne uporabne vrednosti, saj omogoča le pregled naprav. Vnaprej smo sicer pripravili vmesnike tudi za ostala sporočila MQTT, vendar jih nismo uporabili. Na uporabniškem vmesniku pa bi bilo potrebno dodati upravljanje, dodajanje in brisanje naprav.

Literatura

[1] gs.statcounter.com/os-market-share, Operating System Market Share Worldwide - March 2022, obiskano 17. 7. 2022.

[2] neo.io/info, NEO, obiskano 16. 7. 2022.

[3] www.obloliving.com, OBLO, obiskano 16. 7. 2022.

[4] codenameone.com/blog/top-10-best-cross-platform-app-development-frameworks-in-2022.html, Top 10 Best Cross Platform App Development Frameworks in 2022, obiskano 16. 7. 2022.

[5] HUBER Stefan, DEMETZ Lukas Performance Analysis of Mobile Cross-platform Development Approaches based on Typical UI Interactions, julij 2019.

[6] statista.com/statistics/869224/worldwide-software-developer-working-hours/, Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021, obiskano 16. 7. 2022.

[7] medium.com/flutter/introducing-flutter-3-5eb69151622f, Introducing Flutter 3, obiskano 16. 7. 2022.

[8] pub.dev/packages/get, GetX, obiskano 16. 7. 2022.

[9] MEVLJA, S. 2022. Razvoj večplatformne aplikacije za NEO pametni dom. Univerza v Ljubljani, Fakulteta za računalništvo in informatiko.

Napredne IT rešitve za pospeševanje zelenega prehoda

Robert Meolic, Miha Lenko, Andrej Souvent

Operato, energetske rešitve, d.o.o., Maribor, Slovenija
robert.meolic@operato.eu, miha.lenko@operato.eu, andrej.souvent@operato.eu

Sinopsis Zeleni prehod energetike temelji na masovni integraciji obnovljivih in razpršenih virov energije. Zahteve investitorjev po vključevanju novih virov povzročajo težave, ker posamezni elementi omrežja zaradi termičnih omejitev ne dopuščajo dodatnih obremenitev. Zato je pomembno, da obstoječo infrastrukturo čim bolje izkoristimo. To nam omogoča tehnologija dinamičnega ocenjevanja prenosne zmogljivosti, ki temelji na izračunu termične obremenitve gradnikov omrežja. Ta tehnologija se lahko uporabi za daljnovode in za transformatorje. Še posebej zanimiva je indirektna metoda, pri kateri dobimo rezultate brez namestitve senzorjev, le z upoštevanjem vremenskih dejavnikov, ki vplivajo na segrevanje in hlajenje posameznih gradnikov. Slovenski operater prenosnega omrežja Eles je eden prvih operaterjev v Evropi, ki je indirektno metodo za dinamično ocenjevanje prenosnih zmogljivosti začel uporabljati v okviru obratovanja elektroenergetskega sistema. Skupaj s partnerji je razvil SUMO, modularen sistem IT z vsemi potrebnimi funkcionalnostmi. Dosedanje izkušnje kažejo, da sistem omogoča v povprečju od 15 do 20% večjo izkoriščenost daljnovodov od nominalne vrednosti. V članku je SUMO predstavljen z vidika funkcionalnosti in izvedbe. Opisana sta izvedba SUMO na Elesu in projekt uporabe SUMO za transformatorje v distribucijskih omrežjih. Podani so tudi trendi nadaljnjega razvoja, ki so povezani z varnostnimi izzivi ter načrtom, da SUMO postane odprtokodna platforma za povezovanje sistemov IT in OT.

Ključne besede:
SUMO
operativne tehnologije
vodenje elektroenergetskega sistema
dinamično ocenjevanje prenosne zmogljivosti

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.18

1 Uvod

Elektroenergetski sistemi spadajo med velike sisteme, tako glede količine podatkov kot tudi glede števila komponent sistema, ki delajo s podatki. Že dolgo časa ne gre več za lokalno omejene ter nepovezane naprave, danes se pričakuje, da bo podatek pridobljen na enem koncu elektroenergetskega omrežja porabljen na drugem koncu. Pri čemer si želimo, da so podprti in vključeni tako proizvodnja, prenos in distribucija energije kot tudi končni odjemalci.

Z uvajanjem zelenega prehoda, ki temelji na masovni integraciji obnovljivih in razpršenih virov energije, se izzivi obratovanja energetskega sistema le še povečujejo. Posamezni elementi omrežja zaradi termičnih omejitev pogosto ne dopuščajo dodatnih obremenitev. Nadgradnja ali celo postavitev novih transformatorjev in daljnovodov je draga in počasna rešitev, ki je odvisna od dolgotrajnega umeščanja energetske infrastrukture v prostor. Zato želimo čim večjo, a še vedno varno izkoriščenost obstoječih naprav, kar nam lahko omogoči učinkovit sistem vodenja podprt z naprednimi informacijskimi rešitvami.

Dinamično ocenjevanje prenosne zmogljivosti (DTR, angl. dynamic thermal rating oz. včasih tudi DLR, angl. dynamic line rating) je napredna tehnologija v prenosnih in distribucijskih električnih omrežjih. Temelji na izračunu termične obremenitve daljnovodov in transformatorjev. Še posebej zanimiva je indirektna metoda, pri kateri dobimo rezultate brez namestitve senzorjev, le z upoštevanjem vremenskih dejavnikov, ki vplivajo na segrevanje in hlajenje posameznih elementov. Ta pristop omogoča tudi planiranje prenosne zmogljivosti, če so poleg vremenskih podatkov za trenutne razmere (angl. nowcast) na voljo tudi tisti za prihodnost (angl. forecast). Slovenski operater prenosnega omrežja Eles je eden prvih operaterjev v Evropi, ki je indirektno metodo za dinamično ocenjevanje prenosnih zmogljivosti vpeljal v produkcijsko obratovanje. Njihova rešitev, ki so jo razvili skupaj s partnerji, se imenuje SUMO (Sistem za ugotavljanje meja obratovanja).

SUMO je modularen sistem z elementi informacijskih tehnologij (IT) in operativnih tehnologij (OT). V praksi se del omrežja, ki je povezan v internet in kjer prevladujejo sistemi IT, pogosto označuje kot poslovno omrežje, produkcijsko omrežje z integriranimi procesnimi nadzornimi sistemi, ki je ločeno in kjer prevladujejo sistemi OT, pa je označeno kot tehnično omrežje. SUMO povezuje oba segmenta, kar mu omogoča prav posebno mesto v produkcijskih okoljih in ga je zato mogoče opremiti tudi z drugimi funkcionalnostmi, ki niso neposredno povezane s tehnologijo DTR/DLR. Tipični elementi IT v sistemu SUMO so baza podatkov, integracijsko vodilo (angl. integration bus), različne aplikacije, ki so preko tega vodila integrirane, sistemi za sprotni nadzor in sistemi za vizualizacijo ter analizo podatkov. Med tipične gradnike OT pa spadajo komunikacija s senzorji in števci na terenu, komunikacija s sistemi SCADA (angl. Supervisory Control And Data Acquisition) in sistemi za namensko vizualizacijo podatkov.

V nadaljevanju članka je SUMO predstavljen z vidika funkcionalnosti in izvedbe. V drugem poglavju opišemo področje in izzive, ki so spodbudili razvoj sistema in dogajanje, ki vplivajo na njegovo uveljavitev in širitev v produkcijskih sistemih. V tretjem poglavju je podan opis SUMO, ki je zgrajen kot storitveno usmerjena arhitektura. Podprta sta protokol SOAP in tehnologija RESTful. Pomembna sestavna dela sta še Nagios XI za sprotni nadzor ter Grafana za vizualizacijo podatkov in podporo analitiki. V četrtem poglavju predstavimo nekaj primerov iz dosedanje uporabe. Peto poglavje je namenjeno izzivom, ki so bili deležni največje pozornosti pri razvoju SUMO 3. V zaključku strnemo dosedanje izkušnje in izpostavimo nekaj načrtov za prihodnost.

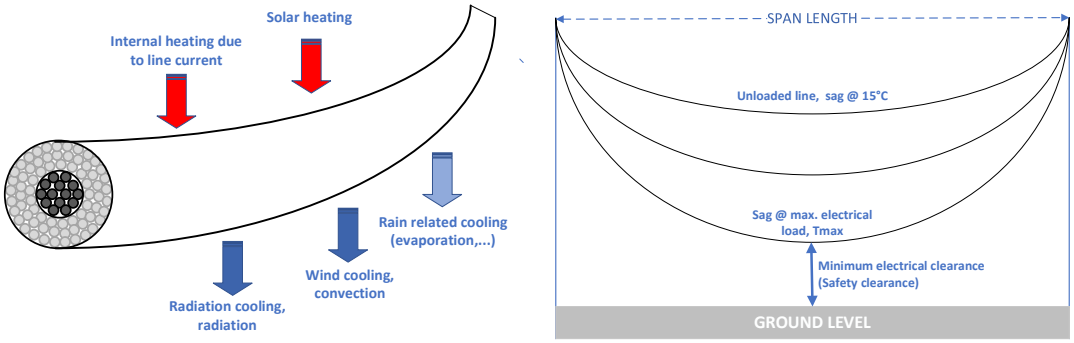
2 Zeleni prehod energetike

V skladu z evropskim zelenim dogovorom, cilja Evropska unija na 55 % zmanjšanje emisij toplogrednih plinov do 2030 in ogljično nevtralnost do 2050 [6]. Ti cilji so zelo ambiciozni in tako zahtevajo precejšnje spremembe tudi v evropskem elektroenergetskem sistemu. Množična integracija obnovljivih virov energije, ob sočasnem izklapljanju nekaterih vrst konvencionalnih virov in vzpostavljanjem evropskega notranjega energetskega trga, povzroča

številne težave povezane z nestanovitnostjo proizvodnje električne energije, stabilnosti sistema, inercijo sistema, sistemskimi rezervami, zamašitvami v omrežju, itd.

Trenutno je hitrost nastajanja težav večja od hitrosti uvajanja rešitev, kar predstavlja visoko tveganje za visoko kakovost oskrbe z električno energijo, ki smo je vajeni v EU. Hitro inoviranje in uvajanje naprednih rešitev je zato ključnega pomena pomen za pravočasno reševanje problemov. Ukrepanje je potrebno za vse deležnike, tudi za sistemske operaterje prenosih omrežij. Za njih je ključno vprašanje, kako obratovati sistem fleksibilno in hkrati varno, stabilno ter ekonomsko učinkovito. Posledično to tudi pomeni, da moramo obstoječo infrastrukturo čim bolje izkoristiti. Tu igrajo ključno vlogo sistemi za dinamično ocenjevanje zmogljivosti (DRS, angl. dynamic rating systems), ki rezultate izračuna dinamičnega ocenjevanja prenosnih zmogljivosti uvajajo v produkcijska okolja.

Osnovna ideja DRS je upoštevati vremenske razmere vzdolž vodnikov in ob transformatorjih tako, da omogočimo čim večji prenos energije ob še varnem obratovanju [1]. Varnost obratovanja je pri daljnovodu na primer pogojena s tem, koliko se zaradi segrevanja vodnika poveča njegov poves (slika 1) oziroma s tem, da se ne zgodi deformacija materiala vodnika. Z uporabo DRS povečamo prenosno zmogljivost v primerjavi s situacijo, ko za obratovanje ves čas uporabljamo statično mejo, ki je izračunana na osnovi za hlajenje vodnikov najbolj neugodnih pogojev, kot so zunanja temperatura 35 °C, hitrost vetra 0,6 m/s s pravokotnim vpadnim kotom in sončno obsevanje 900 W/m² [2]. Daljnovodi so projektirani tako, da tudi pri tako neugodnih vremenskih pogojih ne pride do kršenja varnostnih višin. Ker pa so v večini časa vremenski pogoji ugodnejši od navedenih, lahko ta dodaten pas izkoristimo za povečanje prenosne zmogljivosti, ob predpostavki seveda, da imamo dobre modele, ki nam pravilno izračunajo dinamično termično mejo, ki zagotavlja, da vodnik ne bo presegel kritične temperature oziroma povesa.



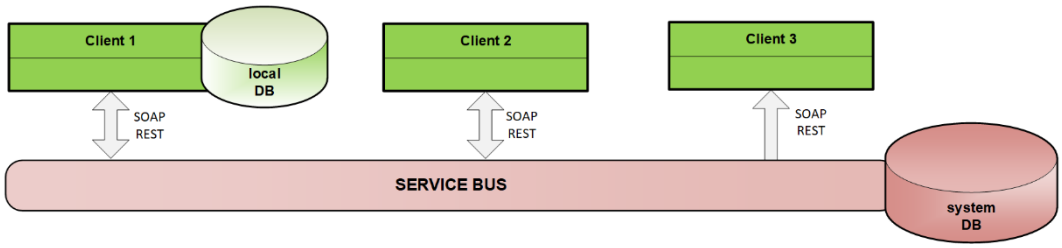
Slika 1: Tehnologija DTR omogoča obratovanje z dinamičnim prilagajanjem vremenskim razmeram.
Vir: lasten.

Nekoliko manj očitna, a pomembna komponenta zelenega preboja v energetiki je povečati odpornost elektroenergetskega sistema tako, da se je zmožen prilagajati ne samo na povečane obremenitve s strani uporabnikov, ampak tudi na povečane grožnje s strani ekstremnih vremenskih pojavov zaradi podnebnih sprememb. Vedno silovitejše vremensko dogajanje lahko prizadene posamezne odseke omrežja in takrat mora preostali del prevzeti večje obremenitve. Zelo poučen primer take situacije v Sloveniji se je zgodil med žledom leta 2014, ko je bilo poškodovanih okoli 52 km daljnovodov (slika 2) [3].

Slika 2: Posledice žleda leta 2014 v Sloveniji.
Vir: [3].

3 Sistem za ugotavljanje meja obratovanja

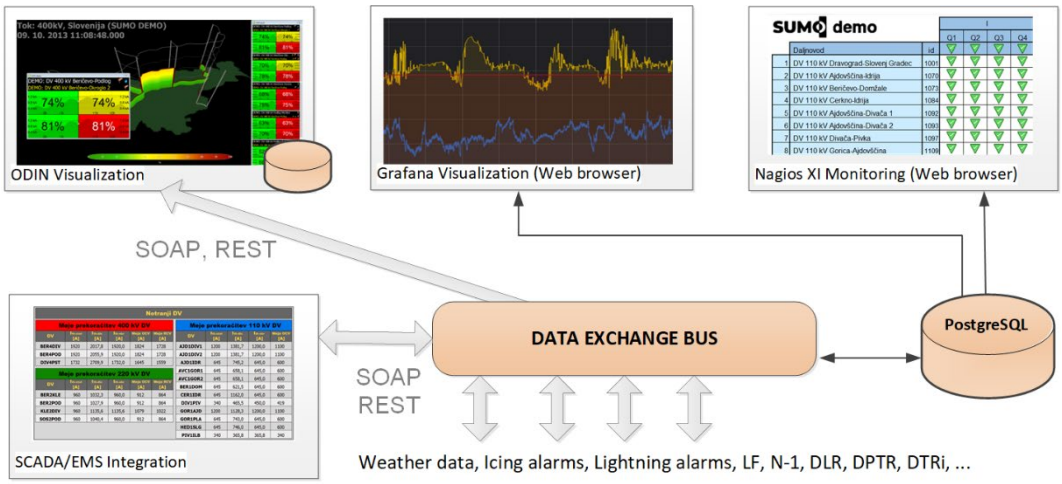
Uveljavljen pristop pri izgradnji velikih porazdeljenih sistemov je storitveno usmerjena arhitektura (angl. service-oriented architecture, SOA), kot je prikazana na sliki 3. Njen osrednji del je integracijsko vodilo, ki ponuja enovit in univerzalen vmesnik. Za izmenjavo podatkov lahko izberemo različne tehnologije, najpogosteje so to spletne storitve SOAP in REST.



Slika 3: Arhitektura s servisnim vodilom.
Vir: [5].

Sistem za **u**gotavljanje **m**eja **o**bratovanja SUMO je v osnovi storitveno usmerjena arhitektura, okoli katere so postavljene druge komponente [4][5]. Jedrni del sestavljata integracijsko vodilo zgrajeno s tehnologijama Apache Karaf in Apache CXF in baza podatkov PostgreSQL. Integracijsko vodilo uporabniki dojemajo kot skupek povezanih spletnih storitev. Vse ostale komponente so postavljene izven jedra in morajo za komunikacijo z vodilom ali bazo izpolnjevati predpisane zahteve, ki so povezane predvsem z zagotavljanjem varnosti. Izpostavimo lahko, da je tudi proces za izračun DTR obravnavan enako kot drugi uporabniki storitvenega vodila. Vizija SUMO je, da postane splošen gradnik za povezovanje IT in OT elementov v elektroenergetskih sistemih, izračun DTR je tako le ena od podprtih funkcionalnosti.

Na sliki 4 je dejanska arhitektura SUMO. Zaradi učinkovitosti določenim komponentam dovoljujemo branje podatkov neposredno iz baze, pisanje podatkov gre vedno preko integracijskega vodila.



Slika 4: Dejanska arhitektura SUMO.
Vir: lasten.

Najpomembnejše komponente, ki so v trenutni izvedbi zbrane okoli jedrnega dela, so:

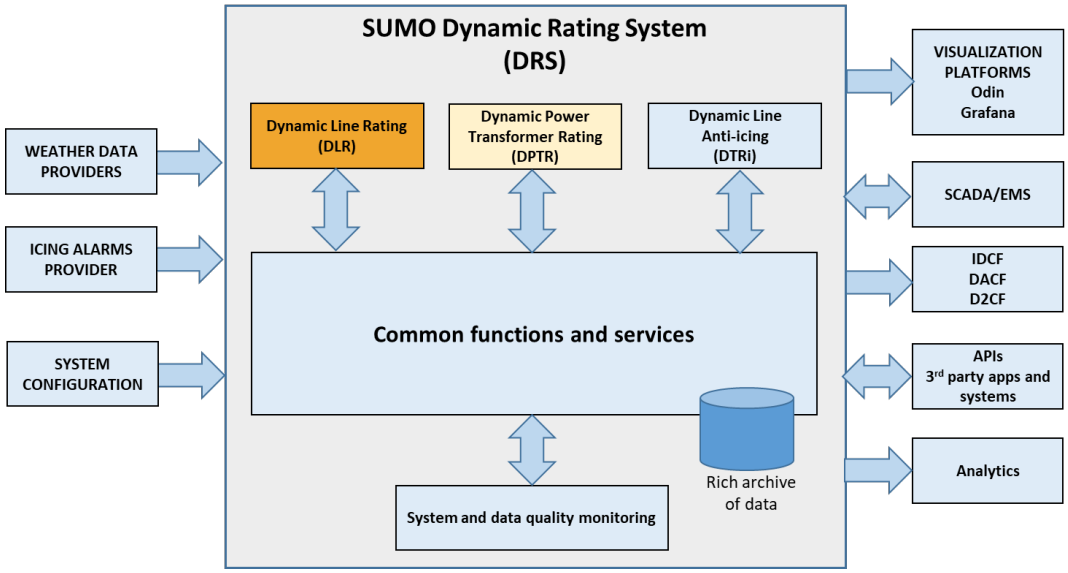
- SCADA/EMS integracija – Običajno je izvedena s posebnimi procesnimi protokoli kot so IEC 104 (standard IEC 60870-5-104), ICCP/TASE.2 (standard IEC 60870-6), OPC UA in drugimi. V ta namen uporabljamo posebne prehode, ki pretvarjajo med temi protokoli in SOAP/REST storitvami SUMO.
- Sistem za vizualizacijo in analitiko Grafana – Omogoča spremljanje podatkov v živo, je pa njegova osnovna vloga podpora analitiki. Poleg podatkov iz SUMO so vključeni tudi rezultati iz drugih neodvisnih

virov in podatki, ki so izračunani na osnovi podatkov iz SUMO v drugih sistemih. Najbolj zanimiv primer je izvoz podatkov iz baze SUMO v računalniški oblak, kjer se obdelajo z metodami strojnega učenja in umetne inteligence, rezultate pa potem prikažemo poleg izvornih podatkov. Kot celota predstavlja ta komponenta zaključen podsistem IT, pri katerem je potrebno poskrbeti predvsem za povezovanje z drugimi sistemi, upravljanje z množico uporabnikov in skrbno spremljati varnostno stanje. Zaradi učinkovitosti črpa Grafana podatke neposredno iz baze.

- Sistem za nadzor NagiosXI – Ima več ločenih funkcionalnosti. Njegovo bistvo je v zmožnosti obveščanja in alarmiranja uporabnikov preko elektronske pošte, preko kratkih sporočil SMS, preko kanalov na raznih platformah in podobno. V osnovi se nadzira delovanje strojne in sistemske programske opreme na vseh strežnikih ter dotok in kvaliteta podatkov, ki prihajajo v bazo SUMO. Dodatno pa lahko, na primer, ta podsistem skrbi za centralizirano zbiranje dnevnikov, obveščanje o dogajanju na sistemu in podobno. Tudi ta sistem ima omogočeno branje nekaterih podatkov neposredno iz baze.
- ODIN – Je namenska vizualizacija, ki je namenjena operaterjem v centru vodenja. Kot taka sledi smernicam sistemov OT in je osredotočena na prikaz najpomembnejših informacij v zelo nazorni obliki podprti s 3D OpenGL vizualizacijo.
- Pridobivanje vremenskih podatkov in raznih modelskih napovedi ter pošiljanje rezultatov – to so podatkovni tokovi, ki so na sliki prikazani pod storitvenim vodilom. Nekatere rezultate in modelske napovedi pripravijo strežniki, ki so postavljeni poleg jedrnega dela SUMO (na primer DLR in LF), nekateri od teh podatkov pa pridejo z interneta. Za pridobivanje podatkov iz interneta se uporabljajo strežniki v coni DMZ. Podatki prihajajo na zunanje strežnike z uporabo različnih protokolov, recimo preko HTTPS, FTPS, z uporabo komunikacije s sporočili MQTT in še kako drugače. Za branje podatkov iz SUMO (imajo le nekateri procesi) in za nalaganje podatkov na SUMO se vedno uporabljajo podprti spletni vmesniki.

SUMO je velik sistem, ki teče na skupku od 4 do 8 virtualiziranih računalnikov. Operacijski sistem na vseh računalnikih je Red Hat Linux (v razvojnih instancah ga včasih nadomestimo s CentOS Stream). Velikost SUMO lahko ilustriramo z naslednjimi podatki:

- Integracijsko vodilo ima okoli 30 spletnih vmesnikov in več kot 100 metod.
- Baza podatkov ima okoli 30 tabel, ki imajo skupaj približno 400 stolpcev.
- Za izvedbo posameznih komponent SUMO uporabljamo Javo, C++ in Python. Analitiko izvajamo z uporabo Python in R.
- SUMO ima modularno zgradbo tako, da je omogočena integracija in sočasna uporaba več algoritmov. Na primer, za delovanje učinkovitega DRS (slika 5) so potrebni izračuni trenutne prenosne zmogljivosti daljnovodov in trenutne obremenljivosti transformatorjev, kratkoročna in srednjeročna napoved prenosne zmogljivosti daljnovodov in obremenljivosti transformatorjev za potrebe vodenja sistema ter za potrebe planiranja kapacitet, izračun ukrepov za preprečevanje žledu in tudi drugi izračuni potrebni za učinkovito vizualizacijo in analitiko.
- V produkciji na Elesu ima nadzorni sistem Nagios XI definiranih več kot 1500 servisov, za katere se redno spremlja status.
- V produkciji na Elesu za vsak daljnovod, ki je vključen v izračun, vsako uro pridobimo od 20.000 do 40.000 podatkov o vremenu in tokovih (količina podatkov je odvisna od dolžine daljnovoda oz. števila stebrov) in izračunamo še precej več rezultatov.



Slika 5: Struktura sistema SUMO.
Vir: lasten.

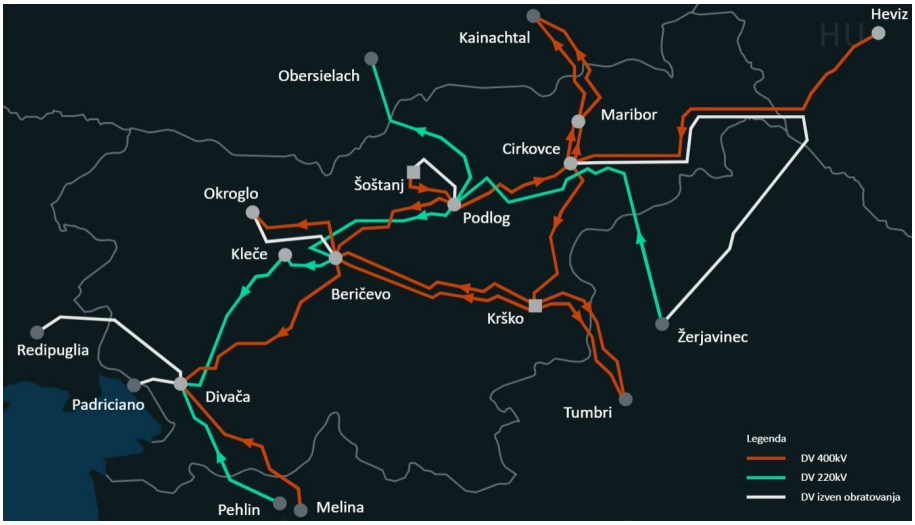
Če se poglobimo v tokove podatkov v SUMO, iz SUMO in znotraj SUMO, ki jih prikazuje slika 5, lahko opazimo tudi varnostno - tehnične izzive, ki niso povezane z velikostjo sistema. Omenimo le nekaj najbolj očitnih:

- sistem je hkrati IT (veliko uporabnikov, zunanji uporabniki) in OT (obdeluje procesne podatke in s svojimi rezultati vpliva na obratovanje elektroenergetskega sistema);
- sistem se hkrati povezuje z nekaterimi storitvami preko interneta (dobava vremenskih podatkov), kot tudi z varnostno kritičnimi procesnimi sistemi, kar zahteva skrbno načrtovanje, izvedbo in obratovanje tudi z vidika zahtev kibernetske varnosti;
- sistem se dinamično konfigurira in uporablja neodvisne vire vhodnih podatkov, hkrati pa mora biti robusten za vodenje procesnih sistemov;
- v sistem vstopajo aplikacije neodvisnih razvijalcev, hkrati pa ima sistem dostop do kritične infrastrukture.

4 Uporaba sistema SUMO

Najdlje trajajoča uporaba SUMO je na področju obratovanja slovenskega prenosnega omrežja. Prenosni operater Eles ima v sistem vključenih 29 daljnovodov in 11 transformatorjev. Med novejšimi uporabami SUMO pa izpostavimo še projekt TrafoFlex, katerega nosilec je SODO – Sistemski operater distribucijskega omrežja z električno energijo in ki se ukvarja z dinamičnim ocenjevanjem prenosnih zmogljivosti distribucijskih transformatorjev.

Eles upravlja prenosno omrežje, ki deluje na 400, 220 in 110 kV napetostnih nivojih. Slika 6 prikazuje 400 in 220 kV omrežja ter povezave teh omrežij z omrežji sosednjih držav. Pomembno je poudariti, da mora prenosni operater zaradi specifične lege Slovenije obvladovati tudi čezmejne pretoke energije. Prenos energije je skrbno planiran za do dva dni vnaprej, pri čemer so zelo pomembne napovedi razpoložljivih prenosnih kapacitet in s tem povezane sigurnostne analize. SUMO je bil na Eles vključen v produkcijo leta 2017. Razpoložljive prenosne kapacitete se računajo za topologijo N in N-1, slednja označuje situacijo, ko izpade eden (najbolj neugoden) element, sistem pa mora to vseeno preživeti. V produkciji je tudi modul za preprečevanje nastanka žleda, ki tehnologijo DLR uporablja v nasprotni smeri (namesto vzdrževanja temperature pod zgornjo mejo se osredotočimo na vzdrževanje temperature nad spodnjo mejo, v tem primeru nad lediščem), ki ga leta 2014 žal še ni bilo, so pa naknadne analize pokazale, da bi lahko z njim pri nekaterih daljnovodih preprečili škodo.

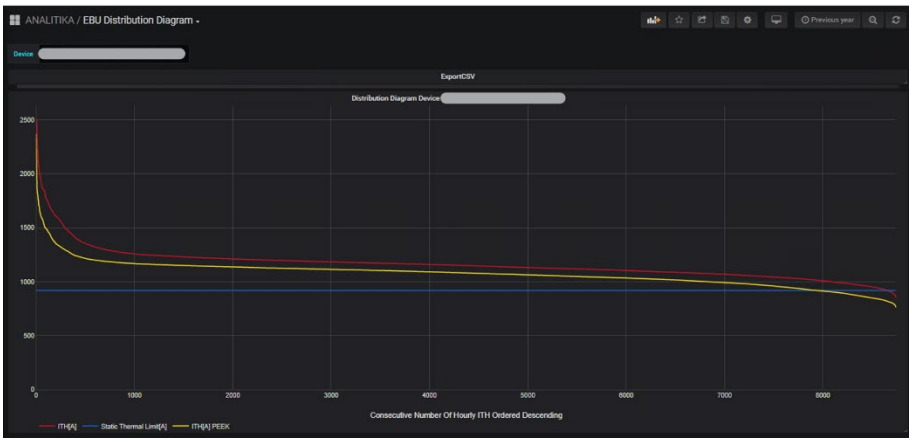


Slika 6: Omrežje Elesa z označenimi smermi toka energije za izbrani dan.
Vir: eles.si.

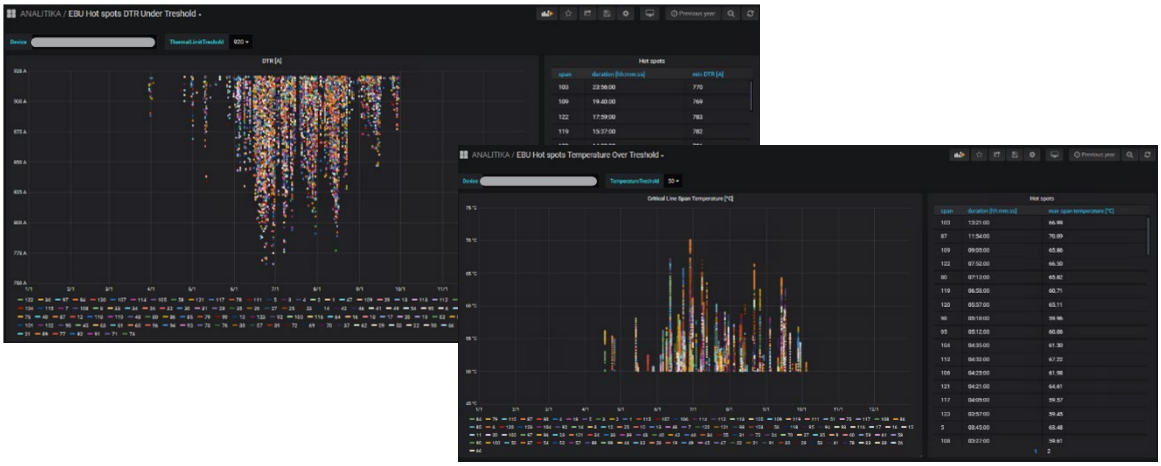
Eles s pomočjo SUMO med obratovanjem omrežja vsako leto v poprečju reši 50 primerov prekoračitev statične meje v topologiji N in več kot 500 primerov prekoračitev statične meje v topologiji N-1. To pomeni, da ko obremenitev prekorači statično mejo, izračun SUMO pa pokaže na dovolj veliko prenosno zmogljivost nad statično mejo, operaterjem ni potrebno odreagirati s predvidenimi mehanizmi odziva, ki so lahko tudi finančno obremenjujoči za družbo Eles.

Bogat arhiv podatkov, ki se je na Elesu skozi leta zbral v bazi SUMO, omogoča kompleksne vizualizacije in analize. Na slikah 7, 8 in 9 so nekateri zanimivi analitični prikazi v Grafani. Ti rezultati, ki nastajajo z uporabo naprednih IT rešitev, imajo poleg neposredne vključenosti v dnevno vodenje omrežja tudi veliko vrednost pri načrtovanju nadgradenj omrežja in pri planiranju obratovanja in zato neposredno pripomorejo k doseganju ciljev zelenega prehoda v energetiki.

Slika 7: Analiza razmer vzdolž vodnika.
Vir: lasten.



Slika 8: Distribucijski diagram
Vir: lasten.



Slika 9: Različne »hot-spots« analize.
Vir: lasten.

Cilj **projekta TrafoFlex** je dinamično določanje termične obremenljivosti transformatorjev za potrebe obratovanja, trga prožnosti in upravljanja s sredstvi. Želimo torej povečati zmožnost obremenjevanja transformatorjev na varen način, brez škodljivega vpliva na življenjsko dobo. Rešitev bo hkrati predstavljala nov in zelo učinkovit način ocenjevanja dejanske obremenjenosti transformatorjev, kar je pomembno z vidika spremljanja t.i. »zdravstvenega stanja« naprav za potrebe upravljanja s sredstvi.

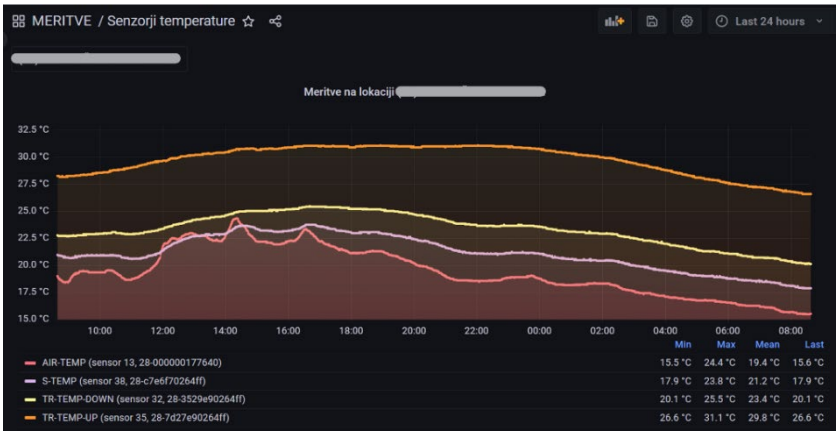
V projektu je bil razvit nov splošen algoritem za izračun dinamične ocene prenosne zmogljivosti distribucijskih transformatorjev. Distribucijski transformatorji se namreč precej razlikujejo med seboj, predvsem po načinu umestitve v transformatorske postaje. V projekt smo vključili vse najpomembnejše oblike izvedb v Sloveniji, ki smo jih označili kot: zidana izvedba, pločevinasta izvedba, izvedba v stavbi in jamborska izvedba. Glavna ideja je izdelati model, ki izračuna dopustni tok brez uporabe drage senzorske opreme in samo z uporabo živih vremenskih podatkov ter zgodovinskih podatkov o obnašanju transformatorja. Projekt traja od decembra 2021 do marca 2023, uporabljena pa je instanca SUMO, ki teče na infrastrukturi Operato.

V projekt je vključenih 19 transformatorjev iz območja Elektro Gorenjska in en transformator iz območja Elektro Celje. Vse te naprave smo opremili s senzorji, ki merijo temperaturo na ohišju transformatorja in temperaturo v okolici (oboje lahko v več točkah). Ti podatki služijo za verifikacijo modelov. Komunikacijski prehod je izveden z IoT napravo GW4100 nemškega podjetja IoTmaxx (https://www.iotmaxx.com/). GW4100 podpira različne komunikacijske protokole, med drugimi tudi 1-wire, ki se uporablja za priključitev senzorjev. Na njem teče različica Linux sistema primernega za vgrajene sisteme. Operato je izdelal programsko opremo, ki iz senzorjev vsako minuti odčita vrednosti ter jih pošlje preko MQTT protokola do SUMO MQTT posrednika. MQTT je zaščiten s TLS –

na vsaki napravi je nameščen certifikat, ki zagotavlja avtentikacijo naprave in šifriranje prometa. Za dostop do mobilnega omrežja uporabljamo multi-roaming SIM kartice podjetja 1NCE (https://www.1nce.com/). Vsi zajeti podatki se shranjujejo tudi na pomnilniško kartico, tako da ob morebitnem izpadu mobilnega omrežja podatkov ne izgubimo. Temperaturni senzorji so DS18B20 digitalni temperaturni senzorji vgrajeni v vodoodporno ohišje iz nerjavečega jekla. Temperaturno območje senzorja je od -55°C do +125°C, natančnost pa ±0,5°C v območju med -10°C to +85°C. Senzorji se priključijo na 1-wire vodilo. Vgrajeni so v 3D natiskana ohišja iz PLA HT filamenta, ki so jih sami oblikovali in izdelali. V ohišja smo dodali magnete za enostavno pritrditev na kovinsko podlago.

Na sliki 10 je nekaj fotografij s terena, slika 11 pa prikazuje vizualizacijo meritev v Grafani. Največji izziv s strani sistemskega vzdrževanja je zagotoviti zanesljiv in stalen zajem podatkov, obveščanje o morebitnih izpadih, varno hranjenje podatkov in njihovo razpoložljivost vsem sodelujočim. Projekt se sedaj bliža polovici in trenutno je v teku produkcijska izvedba algoritmov, ki jih kot prototipe pripravljajo sodelujoči raziskovalci iz IJS in EIMV.

Slika 10: Namestitev merilne tehnike ob transformatorjih.
Vir: lasten.



Slika 11: Vizualizacija meritev v projektu TrafoFlex..
Vir: lasten.

5 Razvoj SUMO 3

Trendi, ki so narekovali pripravo SUMO 3, so povezani predvsem z dvema ciljema:

- obvladovanje varnostnih izzivov, ker ima SUMO v produkciji značilnosti varnostno-kritičnega sistema, pri katerem napačno delovanje oz. nedelovanje lahko povzroči materialno škodo,
- razvoj SUMO v smeri povezovalne platforme, ki poleg DTR omogoča integracijo drugih podsistemov.

Kritična infrastruktura zaradi vedno večje povezanosti z drugimi omrežji postaja pogosta tarča kibernetskih napadov. Sploh proizvajalci in operaterji bolj popularnih SCADA in industrijskih krmilnih sistemov (angl. Industrial Automation Control System – IACS) poročajo o vedno več poskusov vdora, ker bolj kot je sistem razširjen, bolj je zanimiv zaradi možnosti uporabe enakih napadov na več tarčah.

Povezava poslovnega in tehničnega omrežja povzroči dovzetnost kritičnih sistemov za ranljivosti in njihovo izkoriščanje. Napad na tako omrežje lahko povzroči zaustavitev celotnih sistemov in ogroža varnost v okolici sistema. Industrijski krmilni sistemi se glede zagotavljanja njihove varnosti od tradicionalnih okolij IT bistveno razlikujejo v nekaj kritičnih točkah, kot je npr. daljše življenjsko obdobje sistemov, slabša ozaveščenost glede varnosti, slabše upravljanje z varnostnimi popravki, potrebna 24/7 razpoložljivost, redkejši varnostni testi in redkeje razpoložljiva testna okolja. Iz teh razlogov je zagotavljanje varnosti v tehničnih omrežjih velikokrat težavnejše in obenem izredno pomembno. Izpadi teh sistemov so namreč lahko posledica tako zlonamernih dejanj s strani zunanjih napadalcev, kot tudi posledica malomarnega obnašanja zaposlenih.

SUMO 2.0 je bil zgrajen na predpostavki značilni za starejše sisteme OT, da je varnost sistema zagotovljena z njegovo nepovezanostjo – računalniki, ki tvorijo SUMO 2.0 nimajo dostopa do interneta [4]. Vendar pa se med obratovanjem v produkciji izkaže, da popolna izolacija sistema sama po sebi ne preprečuje napačnih posegov niti realna predpostavka. Funkcionalnosti nadzornega sistema, sistemov za vizualizacijo in predvsem sistemov za izvoz in izmenjavo podatkov zahtevajo, da se do notranjega dela sistema dostopa od zunaj, torej iz drugih delov tehničnega omrežja ali pa celo iz poslovnega omrežja.

Industrija se je na problematiko zagotavljanja varnosti tehničnih omrežij odzvala s pripravo standardov za pomoč končnim uporabnikom in ponudnikom pri zaščiti teh sistemov. IEC 62443 sta razvila komiteja ISA99 in IEC z namenom izboljšanja varnosti, razpoložljivosti, integritete in zaupnosti komponent oz. sistemov v uporabi v industrijski avtomatizaciji in nadzoru. Družina standardov IEC 62443 je uporabna v vseh segmentih industrijskega nadzora in se razvija v vodilni standard v industriji.

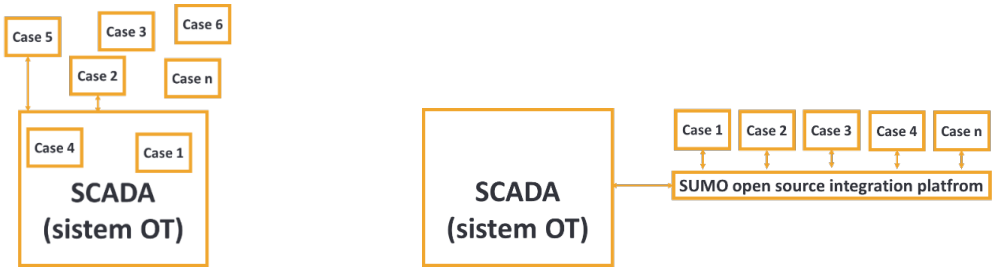
Pri razvoju SUMO 3 smo se osredotočili na pogoje za varnostni nivo SL3, kakor je opredeljen v standardu IEC 62443-3-3: Industrial communication networks - Network and system security - Part 3-3 (slika 12).

Varnostni nivo	Tarča	Namen	Viri
SL1	Naključna	Nenamerno	Posameznik
SL2	Žrtve kriminala	Preprost	Mali viri, izoliran posameznik
SL3	Teroristične tarče	Sofisticiran napad	Zmerni viri, skupina hekerjev
SL4	Državna varnost	Sofisticirana kampanja	Veliki viri, multidisciplinarne skupine

Slika 12: Tabela varnostnih nivojev po standardu IEC 62443.
Vir: lasten.

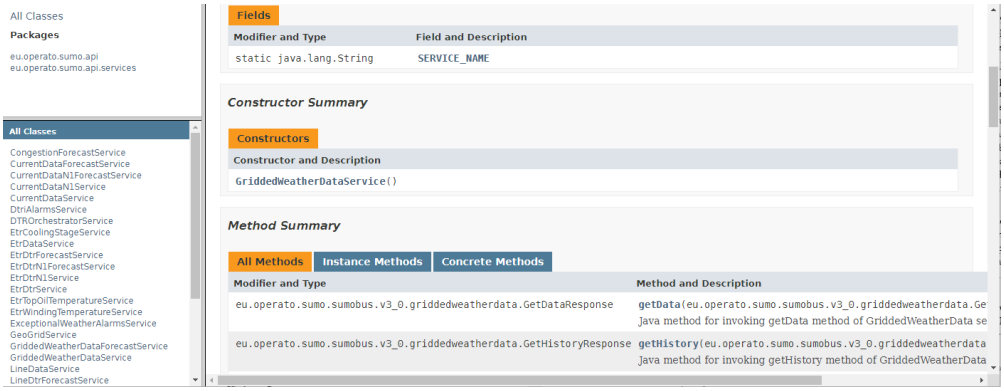
Varnostni nivo SL3 predpostavlja tudi izpolnjevanje vseh zahtev nižjih varnostnih nivojev. Ob pregledu zahtev (ki jih je za vsak varnostni nivo okoli 20) smo se osredotočili na najpomembnejše in ugotovili, da jih lahko vse enotno in ustrezno naslovimo z uvedbo podpisovanja prometa. Za podpisovanje se v SUMO 3 uporabljajo certifikati X.509 v navezi s knjižnico Apache CXF, pri vmesnikih REST je uporabljen protokol izmenjave sporočil JOSE.

Glede razvoja SUMO v smeri povezovalne platforme je bilo naše izhodišče trenutno stanje na področju OT v energetiki. V praksi se pojavlja veliko različnih rešitev, ki se vsaka po svoje povezuje s kritično infrastrukturo, to je s sistemi SCADA (slika 13, diagram na levi). Posledica je drago in zahtevno vzdrževanje ter varnostna tveganja. SUMO tukaj vstopa kot povezovalna platforma in spremeni pristop tako, kot je prikazano na sliki 13 na desnem diagramu. Da bi to omogočili, je treba API za SUMO 3 narediti kot odprtokodni izdelek in omogočiti nastanek ekosistema aplikacij okoli te integracijske platforme.



Slika 13: Vloga SUMO pri povezovanju s sistemi OT.
Vir: lasten.

SUMO 3 gradi v začrtano smer, trenutno sta na voljo odprtokodna API za Javo in C++. Slednji je narejen z uporabo knjižnice gSOAP (https://www.genivia.com/dev.html) in razširitve WS-Security za podpisovanje prometa (https://www.genivia.com/doc/wsse/html/wsse.html). Na sliki 14 sta prikazana dokumentacija za SUMO 3 Java API primer uporabe SUMO 3 C++ API.




```
const string endpoint = "http://bus.sumo3.operato.eu:7000/sumobus/v3.0/"
const string griddedWeatherDataUrl = endpoint + "griddedweatherdata";
Sumo::GriddedWeatherData griddedweatherdata(&griddedWeatherDataUrl[0]);
griddedweatherdata.setContext(partner,"OPERATO","PUTGWD-WEATHER");
griddedweatherdata.makeSigned(cert_default0,rsa_privk_default0);
string griddedWeatherDataVersion = griddedweatherdata.getVersionService();
if (griddedweatherdata.error_msg != "")
  cout << griddedweatherdata.error_msg << endl;
else
  cout << "GriddedWeatherData version " << griddedWeatherDataVersion;
```

Slika

14: Dokumentacija za Java API in primer uporabe C++ API za SUMO 3.
Vir: lasten.

6 Zaključek

Namen članka je širši javnosti predstaviti tehnične značilnosti SUMO in podati primere njegove uporabe. Gre za slovenski produkt, ki je nastal z investicijami družbe Eles ob sodelovanju raziskovalnih inštitucij EIMV, IJS in Fakultete za elektrotehniko Univerze v Ljubljani. Leta 2021 je bil predan v skrbništvo in nadaljnji razvoj podjetju

Operato. Osnovni sistem ima tudi ambicijo, da postane odprtokodni povezovalni gradnik za IT in OT v elektroenergetskih podjetjih. V projektu SUMO nastajajo tudi napredni algoritmi za dinamično ocenjevanje prenosne zmogljivosti in druge aplikacije, ki omogočajo uporabo sistema na področju obratovanja elektroenergetskega sistema in trga z energijo.

Uporaba tehnologije DTR v Sloveniji v povprečju omogoča, da daljnovodi v prenosnem omrežju od 92% do 96% časa delujejo nad postavljeno statično mejo z dobitkom kapacitete od 15% do 20%. Pilotni projekt glede uporabe SUMO v distribucijskih omrežjih se je začel konec leta 2021, prvi rezultati pa bodo na voljo konec leta 2022.

SUMO je tudi v svetovnem merilu eden od redkih sistemov za izračun dinamične obremenljivosti daljnovodov in transformatorjev z indirektno metodo, ki je razvit ter preizkušen do stopnje, da se ga lahko vpelje v produkcijsko okolje. Ideja indirektne metode je, da se izračun izvaja na podlagi modelskih vremenskih podatkov in ne z uporabo dragih senzorjev nameščenih na vodnike. Glavna prednost indirektne metode je, da modeli vremena lahko predvidijo tudi razmere v bližnji prihodnosti, česar senzorji ne zmorejo. Zato lahko SUMO izvaja tudi napovedi prenosnih kapacitet, do dva dni vnaprej. Globalni modeli za ocenjevanje in napovedovanje vremena se sicer ob podpori satelitov in vremenskih postaj neprestano izboljšujejo ter vključujejo vedno več mikroskalnih značilnosti. Nekateri ponudniki komercialnih vremenskih modelov se danes pohvalijo, da imajo njihovi podatki prostorsko ločljivost 100 metrov, kar je primerno za uporabo v SUMO, ker je večina stebrov daljnovodov od 100 do 300 metrov narazen. Vendar analize pokažejo, da kakovost teh podatkov ni na dovolj visokem nivoju, zato je za sedaj še nujno lastno mikroskalno procesiranje z uporaba vremenskih postaj vzdolž tras daljnovodov.

SUMO ima veliko izzivov tako na področju algoritmov s področja energetike in termodinamike kot tudi glede načrtovanja in vzdrževanja informacijskega sistema. Zaradi povezovanja IT in OT moramo zagotoviti namenske gradnike za varno izmenjavo podatkov. Zbirati, hraniti in analizirati je potrebno velike količine podatkov, na primer, za daljnovode Elesa, ki so vključeni v SUMO in imajo skupaj približno 2500 razpetin, vsak dan pridobimo približno 20 milijonov vremenskih podatkov, baza podatkov pa narašča s hitrostjo 2 TB na leto. Tudi oblikovanje odprtokodnega jedra ter usklajevanje vmesnikov API zahteva precej organizacijskih spretnosti.

SUMO je že v uporabi na Hrvaškem, postavljamo pa ga še v Litvi. V prihodnje se nadejamo še več uspešnih namestitev v tujini. Pričakujemo tudi dodatne primere uporabe, ki niso tesno povezani s tehnologijo DTR. V teku je že sodelovanj pri aplikativni nalogi Obvladovanje obratovalnih tveganj v realnem času na Elesu, pri kateri bo SUMO predvsem posrednik med aplikacijami, ki se nahajajo v IT in OT omrežjih.

Za dosego svojih ciljev v družbi Operato aktivno iščemo nove partnerje in tudi sodelovanje z raziskovalnimi in akademskimi inštitucijami.

Literatura

[1] KOSEC Gregor, MAKSIĆ Miloš, DJURICA Vladimir. "Dynamic Thermal Rating of Power Lines – Model and Measurements in Rainy Conditions". International Journal of Electrical Power & Energy Systems, 2017.

[2] MAKSIĆ Miloš, DJURICA Vladimir, SOUVENT Andrej, SLAK Jure, DEPOLLI Matjaž, KOSEC Gregor. "Cooling of Overhead Power Lines Due to the Natural Convection". Int. Journal of Electrical Power & Energy Systems, 2019.

[3] KOSMAČ Janko, KOSEC Gregor, VERTAČNIK Borut, ZIMA Nejc. »Resilience Improvement Attempts after Severe Icing Storm«, 2nd South East European Regional CIGRE Conference, Kyiv, 2018

[4] KOLEŽNIK Janko in drugi. "Nadgradnja SUMO BUS z optimizacijo podatkovnega modela in arhitekture vodila: študija št. 2438". Elektroinštitut Milan Vidmar, Ljubljana, 2019.

[5] MEOLIC Robert, KOLEŽNIK Janko, BERANIČ Miroslav. »Testiranje sprejemljivosti izvedbe SOAP spletnih servisov sistema SUMO«. PIES 2019.

[6] Position on incentivising smart investments to improve the efficient use of electricity transmission assets. ACER. 2021.

Arhitekturni izzivi razvoja rešitve Connected mHealth

Damjan Kovač, Sebastjan Juhart, Tina Maček, Matevž Klevže, Saša Saje Wang

Mikropis Holding d.o.o., Žalec, Slovenija
damjan.kovac@mikropis.si, sebastjan.juhart@mikropis.si, tina.macek@24alife.com, matevz.klevze@24alife.com, sasa.sajewang@mikropis.com

Sinopsis Izraz mHealth (m-zdravje) se nanaša na medicinske in javnozdravstvene prakse, ki se izvajajo in so podprte z mobilnimi napravami. Aplikacije mHealth vključujejo uporabo mobilnih naprav za zbiranje zdravstvenih podatkov, dostavo in izmenjavo zdravstvenih informacij za zdravnike, raziskovalce in paciente, spremljanje bolnika, neposredno zagotavljanje oskrbe (prek mobilne telemedicine), pa tudi usposabljanje in sodelovanje zdravstvenih delavcev. mHealth rešitve so prisotne že nekaj časa, vendar se je povpraševanje po njih enormno povečalo v času pandemije COVID-19. Connected mHealth, rešitev, ki smo jo razvili v podjetju Mikropis je ena izmed tovrstnih mHealth rešitev, ki se osredotoča predvsem na rehabilitacijo na daljavo. Namen rešitve je zagotoviti kvalitetno fizioterapevtsko obravnavo in obravnavo delovne terapije ter ju prenesti v izvajanje na pacientovem domu. V želji, da bi resnično ustvarili edinstveno rešitev na področju rehabilitacije, smo v razvoj produkta vključili strokovnjake iz različnih področij tehnologij, medicine in zdravja. Naš cilj je bil razviti enostavno in uporabno platformo, ki je na eni strani namenjena različnim strokovnjakom v zdravstvenih in rehabilitacijskih centrih, pri ustvarjanju in dodeljevanju individualiziranih rehabilitacijskih programov svojim pacientom, na drugi strani pa izboljšani uporabniški izkušnji, večji motivaciji in uspešnejši rehabilitaciji pacientov. Pri razvijanju tovrstne rešitve smo izbrali komponente, programski jezik in uporabniški vmesnik, ki omogoča varno, zanesljivo in enostavno uporabo tako za fizioterapevte, ki planirajo rehabilitacijske plane pacientom, kot tudi za paciente, ki planirane rehabilitacijske plane nato izvajajo preko mobilne aplikacije.

Ključne besede:
mHeatlh,
varnost podatkov
Keycloak SSO
IBM CDN

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.19

1 Uvod

Zdravstvene ustanove se srečujejo z vedno večjimi težavami pri zagotavljanju kakovostne in cenovno sprejemljive rehabilitacije, saj se število prebivalstva, ki se srečuje z eno od oblik zmanjšane zmožnosti gibanja ali celo invalidnosti na letni ravni povečuje. Zaradi povečanega povpraševanja in potrebi po rehabilitaciji so se začeli razvijati sistemi eHealth (dostop do informacij na spletnih portalih), TeleHealth (termin za vse zdravstvene storitve, ki se izvajajo s pomočjo informacijske in komunikacijske tehnologije), Telemedicina (raba sodobne informacijske tehnologije, z namenom zagotavljanja zdravstvenih storitev pacientom na daljavo), mHealth (kjer uporabnik prejme na pametni telefon personalizirano informacijo oz vodeno vadbo, preko katere je voden med zdravljenjem, lahko pa se uporabi tudi kot nadaljevanje zdravljenja, ko uporabnik zapusti rehabilitacijski center in nadaljuje rehabilitacijo doma). Telemedicina in mHealth storitve zajemajo zgodnjo oziroma podaljšano rehabilitacijsko obravnavo in terapevtsko vadbo, katero pacienti izvajajo v bližnji zdravstveni ustanovi ali kar doma. mHealth je zaradi široke dostopnosti in razširjenosti mobilnih naprav optimalna rešitev, ki omogoča oskrbo in storitve širokemu spektru prebivalstva. V podjetju Mikropis, skupaj s kliniko Mayo Clinic razvijamo rešitve na področju preventive. Zaradi vse večje potrebe po naprednih in mobilnih rešitvah na področju rehabilitacije pa smo se skupaj lotili razvoja mHealth rešitve Connected mHealth – rehabilitacija na daljavo. Pri implementaciji rešitve smo morali poiskati in rešiti številne vsebinske in programerske izzive, ki so podrobneje predstavljeni v nadaljevanju.

2 Izzivi razvoja rešitve Connected mHealth

2.1. Zasebnost in varnost podatkov

Cilj razvoja rešitve Connected mHealth je bil razviti enostavno in uporabno platformo, ki je na eni strani namenjena različnim strokovnjakom v zdravstvenih in rehabilitacijskih centrih, pri ustvarjanju in dodeljevanju individualiziranih rehabilitacijskih programov svojim pacientom, na drugi strani pa izboljšani uporabniški izkušnji, večji dostopnosti, motivaciji in uspešnejši rehabilitaciji pacientov.

Glede na cilj rešitve, Connected mHealth sodi med rešitve namenjene uporabi za zdravstvene oziroma rehabilitacijske namene, v obliki mobilne aplikacije in spletnega portala, ki morata biti razvita tako, da je izmenjava podatkov med pacienti (uporabniki) in strokovnjaki v zdravstvenih in rehabilitacijskih centrih popolnoma zanesljiva in varna. To je bila glavna stvar, ki smo jo imeli v mislih pri sami zasnovi in implementaciji produkta.

Prvi izziv s katerim smo se soočili je bil torej točnost, zasebnost in varnost podatkov ter implementacija dostopov za različne vloge, pri čemer smo upoštevali previdnostna načela, ki zagotavljajo, da ne bo škode tako za strokovnjaka, kot za pacienta – oba ključna uporabnika rešitve. Dostop do podatkov pacienta je tako omogočen le strokovnjakom in nikomur drugemu, niti npr. administrativnim osebam, ki uporabnike t.i. paciente naročajo na preglede ipd. Ti imajo le omejen dostop, ki ga potrebujejo za nemoteno delo pri kreiranju novih pacientov, deaktivaciji ipd.

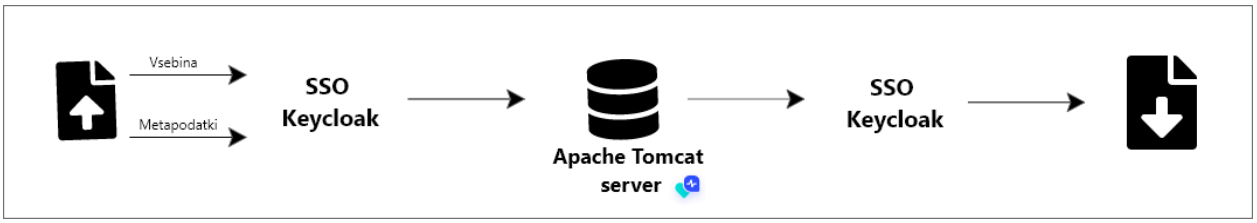
Znotraj rešitve Connected mHealth ima strokovnjak med drugim tudi možnost, da na svojega pacienta pripne razne priloge (dokumente) ter si tako ustvari "zdravstveni karton aktivnosti" pacienta.

Za zagotavljanje varnosti teh podatkov smo razvili lastnem sistem shranjevanja in hranjenja datotek (angl. File storage), ki je opremljen s pravicami nalaganja in dostopanja. Glede na različne vloge, ki so integrirane s SSO sistemom, se razlikujejo pravice in dostopi do naloženih podatkov in datotek. Zaenkrat imamo implementirane štiri različne vloge: "public", "company", "corporation" in "private". Kadar se gre za zdravstveno kartoteko uporabnika govorimo o "private", kadar se gre za shranjevanje novih vaj znotraj ene organizacije, ki jih kreirajo in naložijo strokovnjaki govorimo o "company" ali "corporation". Odvisno od velikosti naročnika in njihovih želja. Kadar je na voljo vsem uporabnikom rešitve pa govorimo o "public".

Vsi podatki oziroma dokumenti, ki jih strokovnjaki naložijo preko Connected mHealth rešitve so avtomatsko kriptirani in se servirajo dekriptirano, preko SSO tokena. Kriptirani so tako, da so zunaj rešitve neuporabni. V kolikor je SSO token potekel oziroma je neveljaven pomeni, da uporabnik do datotek ne more dostopati. Vsaka datoteka je ob kriptiranju sestavljena iz dveh delov – iz vsebine in iz meta podatkov, ki sta shranjena ločeno ter se ob vpogledu nato sestavita nazaj v eno datoteko.

V metapodatkih datoteke se beleži kompleten »audit log« datoteke v katerem so navedene vse spremembe in podatki o datoteki. Ti so:

- lastnik oziroma avtor datoteke,
- datum in čas nastanka,
- vrsta datoteke,
- pravice in vloge ki lahko dostopajo do datoteke,
- vsi ogledi datoteke (datum in čas) ter morebitne spremembe (datum in čas).



Slika 1: Nalaganje, enkriptiranje in dekriptiranje datoteke.
Vir: lasten.

2.2. »Audit log« oziroma revizijska sled

Zaradi podatkov, ki jih lahko strokovnjaki znotraj rešitve shranijo na profilu uporabnika (npr. zdravstveni izvid pacienta) in zaradi splošne zagotovitve ustrezne zasebnosti in varnosti osebnih podatkov smo znotraj rešitve Connected mHealth imlementirali tudi »audit log« oziroma revizijsko sled.

»Revizijska sled je dnevnik z zapisi o operacijah nad poslovnimi podatki. Je nespremenljiva sled oziroma niz podatkov, ki se je zgodila v informacijskem sistemu, z natančnim časovnim zapisom v obliki dnevniškega zapisa, ki omogoča natančni pregled vseh zapisov povezanih z vsemi poslovnimi dogodki in vsemi shranjenimi informacijami od nastanka podatka ali informacije dalje do trenutnega stanja. Omogoča ugotavljanje skladnosti in zakonitosti. Pomembno je, da beleži kdo, kdaj, do kakšnih podatkov je dostopal in kakšno operacijo je izvedel nad določenimi shranjenimi podatki. S tem je omogočen naknaden vpogled do vseh podatkov do katerih so dostopali uporabniki in vpogled v spremembe, ki so jih s svojimi aktivnostmi ustvarili. Revizijska sled mora biti nespremenjena, pregledna, sporočilna, dokumentirana in zaupna.«[8]

Implementirali smo torej sistem, ki zabeleži vsak vpogled strokovnjaka v podatke uporabnika (pacienta) in beleži vsako spremembo, ter tako zagotavlja večjo zasebnost in varnost pacienta oziroma njegovih osebnih podatkov. Za spremljanje vpogledov in delovanje »Audit loga« oziroma revizijske sledi je omogočeno filtriranje po datumu in vrsti dogodka ter vpogled v aktivnosti posameznega strokovnjaka.

Za vsak dogodek se zapišejo naslednji podatki:

- datum in čas,
- ime strokovnjaka,
- in opis dogodka.

Beležijo se naslednji dogodki:

- vpogled v podatke oziroma v profil uporabnika (pacienta),
- sprememba osebnih podatkov uporabnika (pacienta),
    - sprememba imena,
    - sprememba priimka,
    - sprememba kontaktnih podatkov,
- sprememba BIO podatkov,
    - sprememba spola,
    - sprememba datuma rojstva,
    - sprememba podatka teže,
    - sprememba podatka višine,
    - sprememba podatka max. heart rate,
    - sprememba podatka utripa v mirovanju,
    - sprememba naziva diagnoze,
    - sprememba opisa diagnoze,
    - dodana priloga,
    - odstranjena priloga,
    - vpogled v prilogo oziroma priloženo datoteko,
- sprememba plana,
    - sprememba naziva plana,
    - dodan nov plan,
    - odstranjen plan,
    - sprememba plana,
    - dodan nov protokol,
    - odstranjen protokol,
    - sprememba protokola,
- sprememba dodeljenega strokovnjaka,
    - dodeljen strokovnjak,
    - odstranjen strokovnjak,
- sprememba statusa skupin,
    - dodan v skupino,
    - odstranjen iz skupine.

3 Enotna prijava in deljenje podatkov z obstoječimi sistemi

Naslednji izziv nam je predstavljala enotna prijava in deljenje podatkov z obstoječimi sistemi. Naše obstoječe stranke, med katerimi je tudi klinika Mayo Clinic, s katero smo se tudi lotili razvoja rešitve, namreč že uporabljajo naše rešitve na področju preventive, ob tem pa tudi druge zunanje programe oziroma ponudnike, ki so si jih zaželeli v sklopu razvoja rešitve povezati in pacientom, ter zdravstvenemu osebju omogočiti enotno prijavo. Izgradnja novega sistema za overitev in pooblastitev, ki bi zadovoljila vsem zahtevam po povezovanju različnih produktov, sistemov in znanj bi nam predstavljala veliko časovno in stroškovno oviro, zato smo se lotili iskanja najprimernejše rešitve.

Po preučitvi kar nekaj rešitev smo se odločili za uporabo Keycloak SSO strežnika, ki omogoča enotno prijavo (IdP) z upravljanjem identitete in upravljanjem dostopa za sodobne aplikacije in storitve. Rešitev ponuja enkraten vpis v sistem, kar omogoča, da se uporabnik vpiše z enim ID-jem in geslom, ki velja v več medsebojno povezanih neodvisnih sistemih. Vsebuje nadzorno konzolo za upravljanje z domenami in za vsako domeno je mogoče nastaviti odjemalce, vloge, uporabnike in skupine. Keycloak SSO strežnik je napisan v Javi in privzeto podpira protokole federacije identitete SAML v2 in OpenID Connect (OIDC) / OAuth2. Je odprtokodni sistem, ki ima Licenco Apache.

Prednosti Keycloack SSO, ki so nas prepričale:

- enotna prijava (angl. Single-sign On),
- podpora za standardne protokole (OpenID Connect, OAuth 2.0 and SAML 2.0),
- centralizirano upravljanje,
- omogoča varne aplikacije in storitve,
- LDAP in Active Directory omogoča povezave z obstoječimi uporabniškimi imeni,
- možna "družabna prijava" (angl. Social Login),
- posredovanje identitete preko OpenID Connect or SAML 2.0 IdPs,
- visoka zmogljivost: lightweight, fast and scalable,
- preproste teme za izvedbo,
- močno preverjanje pristnosti z izvorno enkratno kodo (OTP) prek FreeOTP ali Google Authenticator,
- prilagajanje politike gesel,
- odprte možnosti razširljivosti: uporabniška baza, metode overjanja, protokoli.

Preko enotne prijave smo tako omogočili enotno prijavo in povezovanje naše rešitve Connected mHealth z obstoječo rešitvijo klinike Mayo Clinic, prav tako, so lahko pacienti klinike Mayo Clinic, ki so že uporabljali naše rešitve 24alife, sedaj lahko prijavljeni z enakim računom v novo rešitev. Podatki med rešitvami se v kolikor uporabnik to želi, prenesejo iz ene rešitve v drugo, s čimer terapevtom omogočamo, da imajo kompleten in celovit vpogled v pacientovo (uporabnikovo) aktivnost.

4 Strokovnjakova ali pacientova lokacija ne smeta imeti vpliva na prikaz slik in videoposnetkov

V sklopu rešitve Connected mHealth so strokovnjakom in uporabnikom na voljo razni materiali, večinoma v obliki videov in slik, kar nam je predstavljalo tretji večji izziv. Torej, kako zagotoviti hiter prenos slik in videoposnetkov vaj na portal strokovnjaka oziroma na mobilno aplikacijo uporabnika, ki se lahko nahajata širom sveta. Connected mHealth je namreč globalna rešitev, ki pa je locirana na IBM strežniku v ZDA. Geografska razdalja med fizičnim strežnikom, ki se torej nahaja v ZDA in uporabnikom, ki rešitev uporablja npr. v Sloveniji, Angliji, Dubaju ipd., je precejšna, kar pa lahko pomeni tudi do nekaj sekund zamika pri odpiranju vsebine strani.

Za rešitev tega izziva smo se odločili uporabiti rešitev IBM CDN. Kratica CDN izhaja iz angleške zloženke Content Delivery Network. Gre za mrežo proxy strežnikov v številnih podatkovnih centrih po vsem svetu. Osnovna naloga CDN sistema je učinkovito, varno in hitro serviranje vsebin z uporabniku najbližjih strežnikov. To pomeni, da CDN uporabniku vsebino prikaže s strežnika, ki je njemu najbližji. S tem se uporabniku vsebina prikaže hitreje, kar pozitivno vpliva na uporabniško izkušnjo in posledično tudi na večjo uporabo rešitve in zadovoljstvo uporabnikov. Z implementacijo IBM CDN smo tako uporabnikom naše rešitve Connected mHealth, ki so locirani zunaj ZDA zagotovili nemoteno uporabo in serviranje večjih podatkov s strežnikov iz najbližje lokacije.

IBM CDN pa poleg pospeševanja dostave vsebin ponujajo še številne druge prednosti. Vsebine, ki so uporabnikom prikazane prek IBM CDN porabijo manj kot polovico pasovne širine (ang. bandwidth) kot sicer, prav tako pa se nekje v podobnem obsegu zmanjšajo tudi zahtevki za prikaz. Strežnik, na katerem se nahajajo "originalne" vsebine, je torej precej manj obremenjen in se hitreje nalaga, kar omogoča tudi večji promet. Dodatna velika prednost IBM CDN je tudi varnost vsebine, IBM namreč zagotavlja napredne varnostne sisteme, ki preprečujejo možnost prestrezanja in zlorabe podatkov, kar je v današnjih časih zelo dobrodošlo.

Literatura

[1] https://www.keycloak.org/, Open Source Identity and Access Management, obiskano 9.6.2022

[2] https://eurohealthnet.eu/sl/publication/mhealth-provides-opportunities-but-risks-widening-inequalities/, mZdravje ponuja priložnosti, vendar tvega povečanje neenakosti, obiskano 9.6.2022

[3] https://ibmi.mf.uni-lj.si/rehabilitacija/vsebina/Rehabilitacija_2013_S1_p104-111.pdf, TELEREHABILITACIJA V CELOSTNI REHABILITACIJI PACIENTOV, obiskano 29.6.2022

[4] https://www.vortex.si/kaj-je-cdn, Kaj je CDN in kako z njegovo uporabo pohitrimo odzivnost spletne strani?, obiskano 29.6.2022

[5] https://www.optiweb.com/sl/blog/pohitritev-globalne-spletne-strani-s-pomocjo-cdn-storitev/, Pohitritev globalne spletne strani s pomočjo CDN storitev, obiskano 9.6.2022

[6] https://www.cakalnedobe.si/nasvet/telezdravje-telemedicina-kaj-je-kako-deluje/, Telezdravje in telemedicina: Kaj to sploh je in kako deluje?, obiskano 4.7.2022

[7] https://www.ibm.com/docs/sl/planning-analytics/2.0.0?topic=logs-audit-log, Revizijski dnevnik, obiskano 4.7.2022

[8] https://sl.wikipedia.org/wiki/Revizijska_sled, Revizijska sled, obiskano 4.7.2022

Vpeljava umetne inteligence in strojnega učenja v poslovni proces napovedovanja porabe električne energije

Vili Podgorelec[1], Sašo Karakatič[1], Grega Vrbančič[1], Špela Pečnik[1], Iztok Fister ml.[1], Lucija Brezočnik[1], Miro Rogina[2], Franci Klauzner[2]

[1] Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
saso.karakatic@um.si, spela.pecnik@um.si, grega.vrbancic@um.si, rok.kukovec@um.si, vili.podogrelec@um.si

[2] Informatika, Informacijske storitve in inženiring, d.o.o., Maribor, Slovenija
miro.rogina@informatika.si, franc.klauzner@informatika.si

Sinopsis Podjetja dandanes vneto tekmujejo v zagotavljanju najboljših možnih storitev svojim strankam, pri čemer podjetja na trgu električne energije niso izjema. Glede na negotove razmere na področju zagotavljanja energetskih virov, v katerih se je znašel svet, vse večje potrebe po električni energiji in trend strme rasti cen energije je postala optimizacija poslovanja eno ključnih vprašanj celotnega sektorja. Ker umetna inteligenca (AI) in strojno učenje (ML) veljata za temeljni večnamenski tehnologiji ter za inovacijsko sredstvo z največjim potencialom preboja, si podjetja prizadevajo sprejeti te tehnologije in jih integrirati v svoje poslovne procese. Za konkretno preizkušanje možnosti uvedbe in uporabe metod AI in ML v svojem poslovnem procesu in informacijskih sistemih je podjetje Informatika informacijske storitve in inženiring d.o.o., osrednji partner v slovenskem elektroenergetskem prostoru, vzpostavilo in izvedlo pilotni projekt s Fakulteto za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Razvitih je bilo več modelov strojnega učenja za inteligentno analizo in napovedovanje porabe električne energije skupaj z nekaterimi podpornimi komponentami. Dobljeni rezultati kažejo, da lahko pravilna uporaba metod AI in ML omogoči podjetju zagotavljanje novih in naprednih storitev za svoje partnerje in različne vrste strank..

Ključne besede:
strojno učenje
umetna inteligenca
inteligentne programske rešitve
napovedovanje porabe električne energije

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.20

1 Uvod

Učinkovita obdelava poslovnih podatkov je že od nekdaj v središču elektroenergetike in z njo povezanih dejavnosti, katerim omogoča zagotavljanje varnih, zanesljivih in kakovostnih storitev ter pregledno poslovanje [1]. V času nenehne rasti obsega in podrobnosti zajetih podatkov na eni strani ter želje in potrebe po njihovem čim boljšem izkoristku na drugi smo priča izjemno hitremu razvoju informacijske tehnologije, programskih rešitev in storitev. Hkrati z rastjo količine podatkov, ki se dnevno ustvarijo, se namreč veča tudi sposobnost računalniških sistemov in pristopov za obdelavo te množice podatkov in odkrivanje novih, bolj točnih spoznanj. Pri tem so sodobne metode umetne inteligence (AI) in strojnega učenja (ML) tiste, ki snovalcem informacijskih sistemov in programskim inženirjem omogočajo razvoj novih storitev, temelječih na poglobljeni samodejni obdelavi množice zajetih podatkov. Inteligentne programske rešitve, ki temeljijo na metodah AI in algoritmih ML, lahko tako ob ustrezni implementaciji ponudijo vrsto funkcionalnosti ter nove in izboljšane poslovne rešitve, ki zagotavljajo strankam novo raven uporabniške izkušnje, ponudniku pa konkurenčno prednost.

A napredne metode ML in teorija, ki stoji za vse sposobnejšimi inteligentnimi rešitvami, so vse zahtevnejše. Medtem ko prinašajo uporabnikom vedno nove funkcionalnosti, za informatike – snovalce in razvijalce novih sistemov in rešitev – tak razvoj predstavlja resen izziv, saj jim ob svojih temeljnih in operativnih zadolžitvah ne ostane dovolj časa in priložnosti, da bi razvoju ustrezno sledili. Dejansko so izzivi še toliko večji, ker se sočasno z razvojem tehnologij pogosto spreminjajo tudi koncepti uporabe le-teh. V podjetju Informatika informacijske storitve in inženiring d.o.o. in v Laboratoriju za inteligentne sisteme na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru smo vzpostavili pilotni projekt, da bi preizkusili možnosti za uvedbo in integracijo metod umetne inteligence in strojnega učenja v informacijski sistem in poslovne procese podjetja.

2 Napovedovanje porabe električne energije

Evropska komisija si je v Evropskem zelenem dogovoru [2] zastavila cilj, da Evropa do leta 2050 postane podnebno nevtralna. Za dosego tega cilja je potrebno resno zmanjšanje emisij toplogrednih plinov. Bistveno vlogo pri doseganju tega cilja ima raba energije, saj je npr. v letu 2016 skoraj tri četrtine svetovnih emisij (merjenih v ekvivalentih ogljikovega dioksida, CO2-eq) povzročila poraba energije [3]. K zmanjšanju emisij poleg okoljskih vidikov hkrati vodi še ekonomska motivacija za povečanje energetske učinkovitosti [4]. Tudi zato se povečuje izraba obnovljivih virov energije – v letu 2019 se je proizvodnja električne energije iz obnovljivih virov povečala za 6% na skupno skoraj 27% delež obnovljivih virov energije v svetovni proizvodnji električne energije [5]. A za obnovljive vire energije je značilna nestanovitna proizvodnja električne energije. Ta nestanovitnost in s tem zmanjšana predvidljivost v primerjavi s konvencionalno proizvodnjo električne energije vodi do novih priložnosti za prihranke pri nabavi električne energije [6], za kar pa je potrebno čim bolj natančno predvideti obseg predvidene porabe.

Tehnično gledano zaporedje meritev porabe električne energije predstavlja časovno vrsto, ki si jo lahko predstavljamo kot dvodimenzionalni problem – na x osi je čas (kronološko zaporedje), medtem ko y os predstavlja porabo, pri čemer želimo napovedati porabo (y_i) za poljubno časovno točko v prihodnosti (x_i). Sodobne metode napovedovanja časovnih vrst temeljijo predvsem na principu zgodovinskega napovedovanja prihodnosti. Posebnost pri napovedovanju porabe električne energije je prisotnost večsmernih trendov, sezonskih in cikličnih nihanj, strukturnih prelomov, kar predstavlja določene omejitve in določa zahteve za izbiro ustreznih metod in modelov [7]. Za namen napovedovanja porabe električne energije obstaja cela vrsta metod in pristopov, od klasičnih tehnik časovnih vrst (avtoregresije, modeli eksponentnega glajenja, dinamične regresije), preko tradicionalnih metod strojnega učenja pa vse do naprednih ansambelskih modelov in globokih nevronskih mrež, sposobnih obravnavati nestacionarnost, heteroskedastičnost in serijsko korelacijo nestabilnih kratkoročnih podatkov [7]. Primerna vpeljava metod AI in ML je tako nujna, če želimo pri napovedovanju porabe električne energije in povezanih dejavnosti doseči zadovoljive rezultate.

3 Vpeljava metod AI in ML v poslovni proces podjetja

AI je tehnologija, ki spreminja podjetja in naše vsakdanje življenje. Je široko uporabno orodje, ki ljudem omogoča, da na novo premislijo, kako povezujemo informacije, analiziramo podatke in uporabljamo pridobljena spoznanja za boljše odločanje. AI velja za inovacijsko entiteto z največjim potencialom za preboj [8] in temeljno večnamensko tehnologijo, zlasti v povezavi z ML [9].

Z ustrezno uvedbo inteligentnih programskih rešitev in storitev lahko podjetja izboljšajo funkcionalnost izdelkov in kakovost storitev, bolje komunicirajo s strankami, racionalizirajo poslovanje ter oblikujejo predvidljive in natančne poslovne strategije. Vendar pa dandanes ni veliko podjetij, ki bi imela znanja in zmogljivosti za ustrezno obravnavo in spopadanje s celotnim procesom vpeljave AI in ML. V ta namen potrebujejo pomoč kompetentnega ponudnika znanja. Takšna pomoč običajno zajema tri glavne faze:

- faza **identifikacije** z iskanjem poslovnih primerov, v katerih je najbolj smiselno uvesti inteligentne rešitve,
- faza **načrtovanja**, v kateri se izvede raziskava o tem, kako oblikovati in razviti inteligentne metode in rešitve ter kako takšne rešitve vključiti v poslovni model podjetja, in
- faza **izvajanja**, ki vključuje pilotno izvajanje in njegovo oceno ter prenos znanja od ponudnika znanja na podjetje.

Če se podjetje odloči, da bo v celoti uporabilo preizkušene tehnologije za izbrane in morda tudi druge poslovne primere, bo moralo vzpostaviti ustrezne zmogljivosti in spretnosti. Zato je bistvenega pomena, da vse te faze spremljata svetovanje in izobraževanje.

3.1 Digitalizacija storitev z napovedovanjem porabe električne energije

Informatika je izkušeno podjetje s preko 40 let izkušenj pri razvoju IT rešitev in nudenju storitev, ki opravlja informacijske storitve po meri za vsa slovenska elektrodistribucijska podjetja. Njihove storitve vključujejo:

- sistem obračunavanja za distribucijo električne energije (za okrog 900.000 odjemalcev),
- sistem obračunavanja za izbrane dobavitelje električne energije (z več kot 140.000 strankami),
- zagotavljanje platforme za izmenjavo podatkov o trgu električne energije (enotna vstopna točka), ki jo uporabljajo vsi slovenski distributerji električne energije in vsi dobavitelji električne energije (več kot 30),
- integracijsko platformo, ki temelji na storitvenem vodilu, prek katerega je že integriranih več kot 20 različnih informacijskih sistemov, in
- aplikacije za upravljanje vseh postopkov, povezanih s potrošniki, na strani distributerja (izdajanje mnenj in smernic, soglasij za projekte, soglasij za priključitev, sporazumov o uporabi sistema itd.).

Informatika je tako podjetje, ki je že ponotranjilo procese digitalne preobrazbe, želi pa svoje poslovne procese še izboljšati z vpeljavo rešitev AI. V ta namen je z Laboratorijem za inteligentne sisteme Univerze v Mariboru, kot ponudnikom znanja z več kot 20-letnimi izkušnjami z analizo, načrtovanjem, implementacijo, integracijo in evalvacijo sistemov AI, ML in podatkovne znanosti zasnovalo in izvedlo pilotni projekt vpeljave metod AI in ML. Osnovni namen vzpostavljenega sodelovanja je bila izvedba digitalne preobrazbe dela poslovnega modela podjetja z uporabo modelov ML, ki lahko pomagajo pri pridobivanju konkurenčne prednosti pri strankah v sedanjosti in prihodnosti, ter pridobivanje znanja na konkretnem projektu, da bi lahko ta znanja uporabili in vključevali v prihodnje projekte. Ker je osnovna dejavnost Informatike zagotavljanje storitev za podjetja za distribucijo električne energije, se je projekt osredotočil na analizo in napovedovanje porabe električne energije ter potencialne storitve, ki ju takšno napovedovanje omogoča. Cilj je bil tako še izgradnja pilotnega napovednega modela in operacionalizacija pilotnega modela v produkcijsko aplikacijo.

4 Vzpostavitev razvojnega okolja

Za potrebe analize podatkov, razvoj modelov strojnega učenja ter enostavnejše sodelovanje med soudeleženimi smo vzpostavili razvojno okolje, temelječe na obstoječih odprtokodnih rešitvah, ki nam omogočajo prilagoditve in integracije z drugimi sistemi oz. platformami. V grobem vzpostavljeno razvojno okolje sestoji iz treh osnovnih odprtokodnih komponent – platform jupyterhub, MLflow in Apache Airflow (slika 1).

Slika 1: Konceptualni diagram zasnove vzpostavljenega razvojnega okolja.
Vir: lasten.

Jupyterhub uporabnikom poenostavlja dostop do računskih virov brez potrebe po nameščanju razvojne programske opreme in njenim vzdrževanjem. Razvijalcem služi kot enotna vstopna točka do pred-pripravljenega razvojnega okolja z nameščeno programsko opremo in razvojnimi knjižnicami, sistemskemu administratorju pa kot orodje za nadzorovanje delovanja ter orodje za upravljanje z uporabniki in razpoložljivimi viri, ter omogoča ustrezno avtentikacijo uporabnikov kot tudi integracijo z zunanjimi avtentikacijskimi storitvami. Jupyterlab za vsakega posameznega uporabnika z uporabo t.i. spawner storitve zažene izoliran enouporabniški jupyterlab strežnik na podlagi vnaprej določenih dodeljenih sistemskih virov in vnaprej nameščene programske opreme in razvojnih knjižnic. Na tak način je omogočeno sočasno brezprekinitveno izvajanje različnih programskih kod neodvisno od ostalih uporabnikov. Zagon takšnih enouporabniških jupyterlab strežnikov je mogoč na istem gostiteljskem sistemu, na katerem je nameščen jupyterhub, ali pa na oddaljenih strežnikih. Razvojno okolje je smiselno vzpostaviti z uporabo zabojniške tehnologije Docker in orkestracijskega orodja Docker Swarm.

Odprtokodna platforma MLflow je namenjena upravljanju življenjskega cikla razvoja modelov strojnega učenja, vključno z eksperimenti in sledenjem poteku učenja, možnostjo ponovitve eksperimentov, namestitvijo napovednih modelov in njihovim centralnim registrom. Omogoča širok nabor funkcionalnosti, v našem primeru pa smo jo primarno izkoristili predvsem za sledenje napredka učenja naprednejših napovednih modelov, skupaj s sledljivostjo vrednosti različnih zagonskih hiper-parametrov. Platformo lahko uporabimo tudi kot centralni repozitorij zgrajenih napovednih modelov, katere lahko tudi namestimo v obliki spletnih storitev bodisi na lokalno ali na zunanjo infrastrukturo, kot na primer: Microsoft Azure ML, Amazon SageMaker ali Apache Spark UDF.

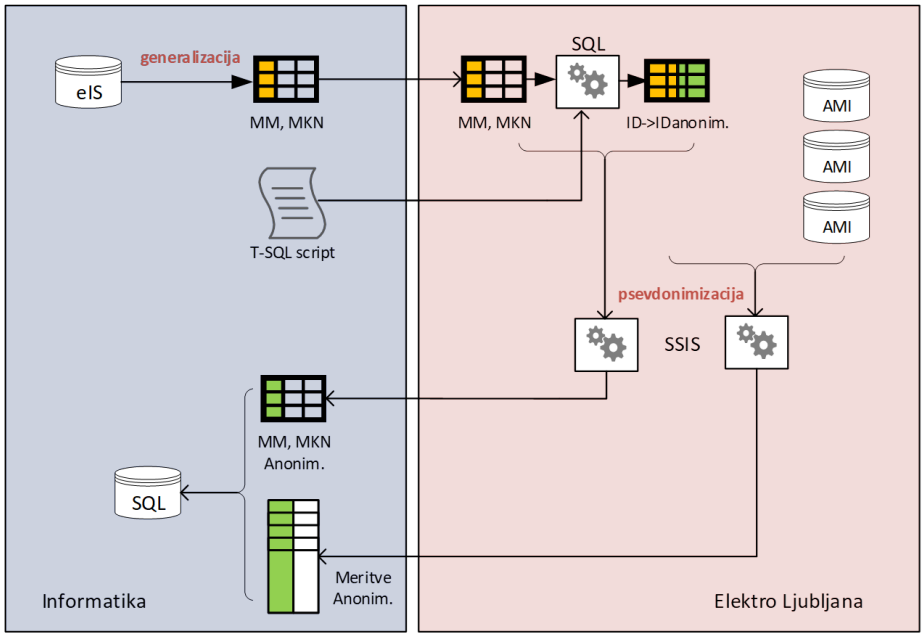
Za namen oddaljenega izvajanja dalj časa trajajočih opravil, kot je učenje kompleksnih modelov strojnega učenja, je smiselno uporabiti odprtokodno platformo Apache Airflow, ki omogoča programsko implementacijo, razporejanje in nadzorovanje delovnih tokov ter avtomatizacijo širokega nabora opravil z uporabo Python skript. Poleg skrbi za razporejanje in nadzorovanje delovnih tokov omogoča uporabnikom vpogled v izvajanje posameznih delovnih tokov z uporabo grafičnega vmesnika in integracijo z obstoječimi sistemi. Koristna funkcionalnost je še oddaljeno izvajanje jupyter zvezkov. Ker je učenje naprednejših modelov strojnega učenja časovno zahtevna naloga, lahko z uporabo Apache Airflow uporabnikom omogočimo, da želeno opravilo pošljejo v izvajanje na oddaljen sistem, sami pa nemoteno nadaljujejo z delom znotraj razvojnega okolja JupyterLab. Vpogled v napredek učenja pa je uporabnikom omogočen z uporabo prej omenjene platforme MLflow.

5 Pridobitev in pred-obdelava podatkov

Ključno vlogo pri razvoju inteligentnih rešitev z uporabo ML imajo kakovostni podatki, ki so pridobljeni na enak način in v enaki obliki, kot se pričakuje, da bodo na voljo tudi za kasnejši namen uporabe razvitih rešitev v produkcijskem okolju. V ta namen so v Informatiki poskrbeli za pridobitev podatkov o porabi električne energije na nivoju merilnih mest od Elektra Ljubljana. Definiran in vzpostavljen je bil protokol za zajem podatkov iz podatkovnih virov Elektra Ljubljane, njihovo anonimizacijo in pretvorbo v obliko, ustrezno informacijski infrastrukturi Informatike (slika 2). Na ta način je bilo pridobljeno skoraj 20 milijard meritev (19.339.546.565) za več kot pol milijona števcev.

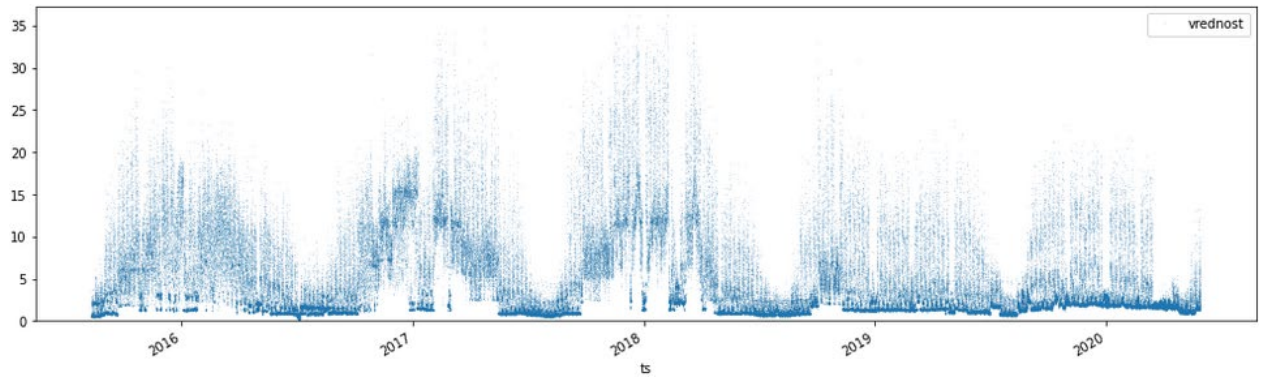
V dejansko zajetih meritvah seveda prihaja tudi do napak, odstopanj in anomalij, v podatkovni shrambi so merilna mesta, ki več niso aktivna, na določenih merilnih mestih se lahko (tudi večkrat) zamenjajo števci, pojavljajo se podvojene tovarniške številke (vir podatkov so tri različne baze in v njih je nekaj različnih števcev, različnih proizvajalcev in tipov, ki pa imajo enake tovarniške številke) ipd. V ta namen je bilo potrebno najprej opraviti pred-obdelavo in izločitev neustreznih podatkov. Na nivoju posameznega merilnega mesta smo izračunali različne statistične vrednosti in postavili kriterije, kateri podatki so za analizo neuporabni.

Izločili smo zapise za registre, ki predstavljajo jalovo energijo (R+, R-), tako da so ostali le podatki za prevzeto (A+, P+) in oddano (A-) delovno energijo. Izločili smo zapise za registre števcev s samimi ničelnimi vrednostmi. Za števce, ki so imeli zapisana stanja, smo vrednosti preračunali v količine energije. Izločili smo še zapise za števce s podvojenimi tovarniškimi številkami, števce s premajhnim številom zajetih meritev (npr. nedavno priključene, brez znane zgodovine porabe) ter števce, ki so imeli zapisane le vrednosti na vsako polno uro, ne pa na 15 minut.



Slika 2: Koncept pridobitve anonimiziranih podatkov o porabi.
Vir: lasten.

Nazadnje smo skušali identificirati (s primerjanjem srednje in maksimalne vrednosti ter variance) še števce, pri katerih se pojavljajo očitne napake – odbirki z nerealnimi (zelo velikimi) vrednostmi. Za namen razvoja rešitev ML je tako ostalo 12.033.896.137 meritev za 173.429 merilnih mest. Primer pred-obdelanih meritev za posamezno merilno mesto prikazuje slika 3.



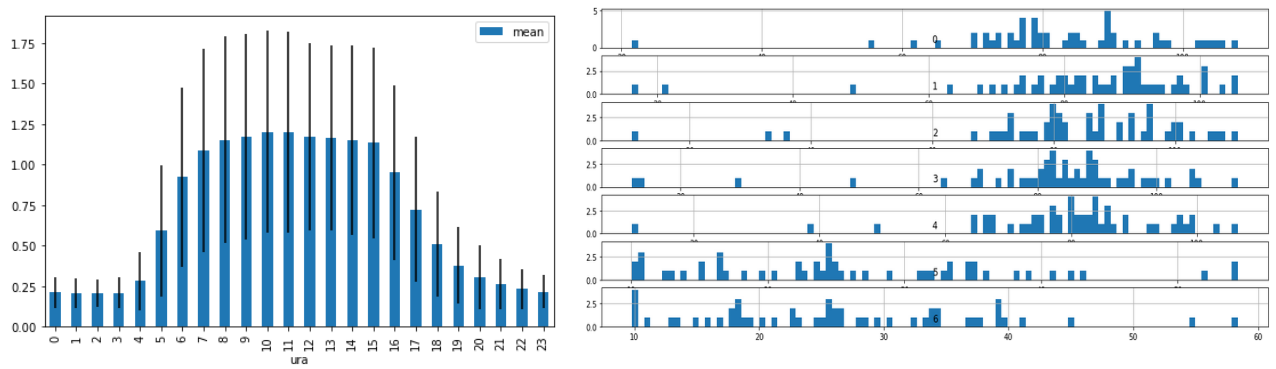
Slika 3: Primer pred-obdelanih podatkov o porabi električne energije za specifično merilno mesto, ki služi kot osnova za nadaljnjo obdelavo z metodami strojnega učenja.
Vir: lasten.

6 Razvoj napovednih modelov s strojnim učenjem

6.1 Inteligentna analiza podatkov – iskanje osnovnih značilnosti v meritvah porabe

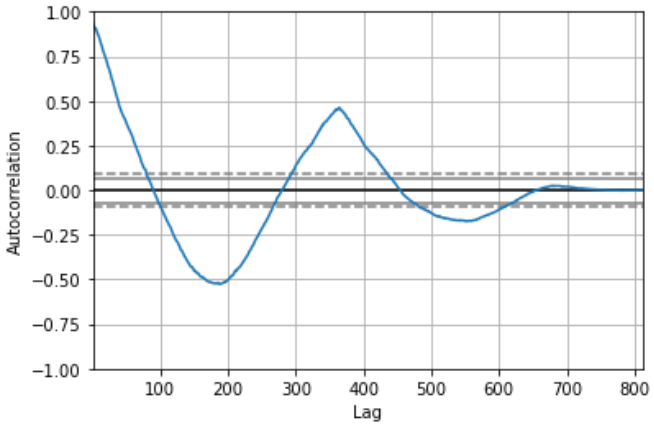
Surovi podatki o porabi električne energije, zajeti v meritvah, niso neposredno uporabni za namene strojnega učenja. Zaporedje meritev porabe za posamezno merilno mesto predstavlja časovno vrsto, ki si jo lahko predstavljamo kot dvodimenzionalni problem – na x osi je čas (kronološko zaporedje), medtem ko y os predstavlja porabo. Poraba je pri tem odvisna od časa – osnovna ideja je v tem, da bi znali za poljubno časovno točko x_i (nekje v prihodnosti) izračunati (oz. napovedati) predvideno porabo y_i, ki bi se nato od dejanske porabe v tisti časovni točki kar se le da malo razlikovala. Da bi lahko to dosegli, mora obstajati povezava (korelacija) med časom in porabo, in prav to povezavo želimo poiskati. Če npr. ugotovimo, da je poraba poleti manjša kot pozimi, se bomo pri iskanju predvidene porabe za zimsko obdobje ozirali po višjih vrednostih kot pri iskanju za poletno obdobje, pri napovedovanju pa upoštevali sezono. Podobno dragocene informacije nam nudi npr. ugotovitev, da je poraba ponoči manjša kot podnevi, da je poraba višja ob delavnikih in nižja čez vikend ipd. Da bi poiskali tovrstne značilnosti, moramo čim bolj podrobno raziskati karseda velik nabor podatkov, ki so nam na voljo.

Z analizo meritev hitro ugotovimo, da je poraba precej odvisna od različnih časovnih, sezonskih (vreme), družbenih (delovni dnevi, prazniki) in ostalih zunanjih dejavnikov, ki jih bomo zato pri razvoju in gradnji napovednih modelov morali ustrezno upoštevati. Prav tako ugotovimo, da se lahko splošni vzorci porabe med različnimi merilnimi mesti zelo razlikujejo, kar pomeni, da pri napovedovanju ne bomo mogli temeljiti na enem samem, univerzalnem napovednem modelu (slika 4 prikazuje primer profila urne porabe za gospodinjstvo, ki je npr. pri poslovnem odjemalcu povsem drugačen). Po drugi strani bi razvoj in gradnja lastnih napovednih modelov za vsako merilno mesto posebej (opravka imamo z več sto tisoč porabniki, kar lahko gre v produkcijski verziji v milijone) predstavljalo daleč preveliko časovno zahtevnost problema. Sklenemo lahko, da bomo vsekakor upoštevali odkrite odvisnosti, pri čemer bomo le-te najprej izračunali za množice med seboj dovolj podobnih merilnih mest (homogene gruče odjemalcev glede na vzorce obnašanja oz. porabe) in jih šele nato prilagodili vsakemu posameznemu merilnemu mestu posebej. Na ta način dosežemo zadovoljiv in nujen kompromis med uspešnostjo napovedi ter časovno učinkovitostjo napovednega modela.



Slika 4: Primer profila urne porabe za specifično merilno mesto. Na levi sliki vidimo porabo skozi ure v dnevu, medtem ko desna slika prikazuje časovno porabo skozi različne dneve v tednu, kjer je dobro razvidna višja popoldanska poraba ob delavnikih in bolj porazdeljena urna poraba med vikendi.
Vir: lasten.

Tako na čas, potreben za učenje napovednih modelov, kot tudi na samo uspešnost napovedi ključno vpliva obseg podatkov, ki jih uporabimo za učenje modelov. Če uporabimo več podatkov (zajete meritve porabe za daljše obdobje v preteklost), imamo na voljo več informacij za odkrivanje trendov porabe; po drugi strani pa večji obseg podatkov predstavlja večjo časovno zahtevnost za učenje napovednega modela. Zato je potrebno ugotoviti, kakšno časovno obdobje je pri razvoju napovednih modelov smiselno upoštevati. Odvisnostim, kjer nam o bodoči porabi lahko največ pove poznavanje pretekle porabe, ne glede na izhodiščni čas, rečemo avto-korelacije. Slika 5 prikazuje, kako močno je poraba v določeni časovni točki (v tem primeru gre za porabo v določenem dnevu) odvisna od porabe v prejšnjih časovnih točkah (v prejšnjih dneh). Lepo lahko vidimo, kako odvisnost s časovno oddaljenostjo pada – današnja poraba najbolj korelira z včerajšnjo, malo manj s predvčerajšnjo in tako naprej. Opazimo lahko, da avto-korelacije popolnoma opišejo sezonsko gibanje skozi letne čase, v katerih si je poraba med seboj najbolj podobna in od katerih je poraba značilno odvisna. V naslednjem letu se zadeva ponovi, a z nižjo stopnjo korelacije. V tretjem letu so zaznane stopnje korelacije že zelo nizke. Vse povedano pomeni, da je v prikazanem primeru za uspešne napovedne modele smiselno uporabiti za dve leti zgodovinskih meritev.



Slika 5: Prikaz (avto-)korelacijskih stopenj dnevne porabe električne energije, ki ponazarja, kako je vrednost porabe povezana z izmerjenimi vrednostmi v preteklosti.
Vir: lasten.

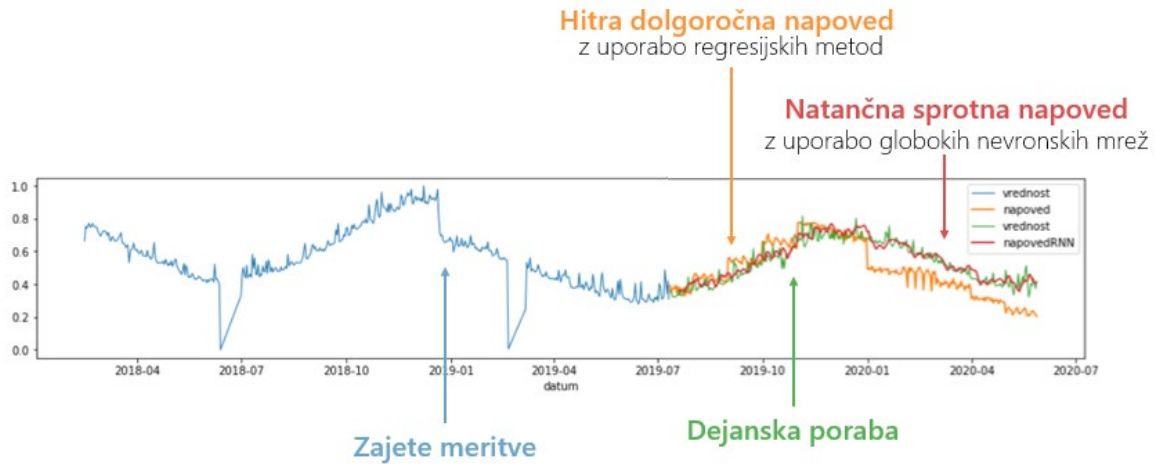
6.2 Zasnova in učenje napovednih modelov strojnega učenja

Za namen projekta smo razvili prototipno zaledno programsko rešitev za poglobljeno analizo podatkov o porabi električne energije in različne vidike napovedovanja. Osnovne razvite funkcionalnosti vključujejo:

- modele strojnega učenja za napovedovanje porabe električne energije za posameznega porabnika (dnevne, urne in 15-minutne napovedi),

- zaznavanje neobičajnih vzorcev porabe in ugotavljanje odstopanj v porabi, ter
- združevanje porabnikov električne energije na podlagi njihovih vzorcev porabe.

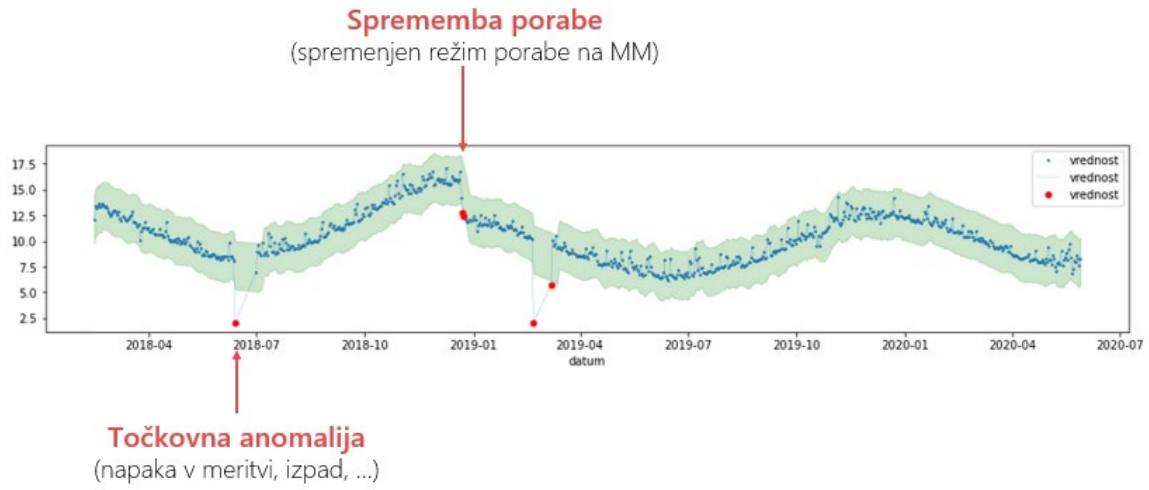
Temeljni del sistema je model za napovedovanje porabe električne energije. Napovedovanje porabe je bistvenega pomena za prestrukturirano okolje upravljanja energije na trgu električne energije [10]. Kljub napredku tehnologij pametnih omrežij in raziskav na področju varčevanja z energijo ostaja veliko izzivov za natančno napovedovanje proizvodnje in/ali porabe električne energije z uporabo velepodatkov ali obsežnih zbirk podatkov [11]. Za napovedovanje porabe električne energije so bili v zadnjem času uporabljeni različni pristopi, od različnih regresijskih metod [12], različnih tehnik verjetnostnega napovedovanja [13] pa vse do najnaprednejših metod ML, kot so umetne nevronske mreže [14] in zlasti metode globokega učenja [10]. Sami smo zasnovali hibridni algoritem za napovedovanje z uporabo polinomske regresije z regularizacijo za hitro posplošeno dolgoročno napovedovanje in globoko rekurentno nevronsko mrežo LSTM (Long Short-Term Memory) za natančno kratkoročno napovedovanje, za katerega se je izkazalo, da zagotavlja natančne napovedi (glej primer na sliki 6). Multivariatna nevronska mreža LSTM je vsebovala niz parov slojev LSTM in Dropout s padajočo stopnjo verjetnosti osipa (dropout probability rate), končno Dense plast in optimizacijski algoritem RMSprop. Omrežje se je napajalo z dnevnimi meritvami vsaj dveh let, da bi lahko zajeli sezonske učinke. Učenje mreže je običajno potekalo, dokler ni bila dosežena stabilnost stopnje izgube (loss rate). Napovedni model porabe električne energije na nivoju posameznega odjemalca (merilnega mesta) je hkrati osnova za vse ostale razvite napovedne modele.



Slika 6: Primer napovedovanja porabe električne energije z razvitim pilotnim napovednim modelom.
Vir: lasten.

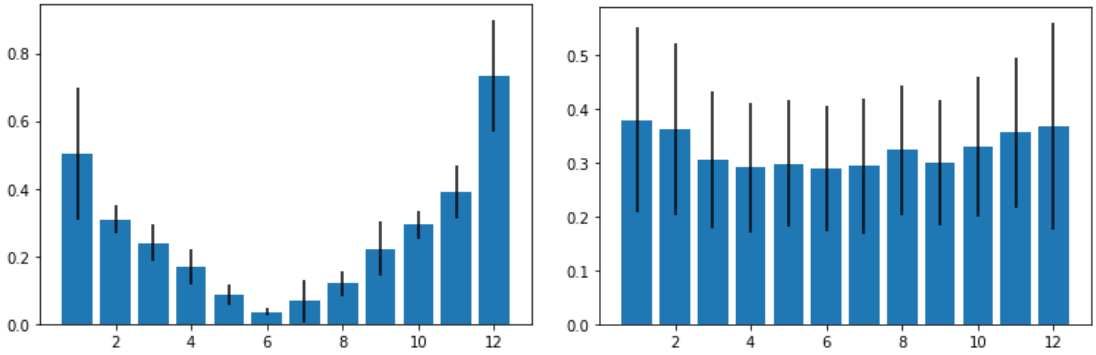
Drugi del sistema je komponenta za odkrivanje anomalij, ki je bila razvita za samodejno prepoznavanje nenavadne porabe električne energije in odkrivanje značilnih odstopanj od ocenjene porabe za posameznega porabnika oz. posamezno merilno mesto. V ta namen smo razvili metode in modele za zaznavanje tako točkovnih (npr. današnja poraba odstopa od vsakodnevne) kot kontekstnih nepravilnosti (npr. poraba se ni povečala/zmanjšala toliko, kot bi bilo glede na okoliščine pričakovati). Medtem ko lahko točkovne anomalije predstavljajo morebitne okvare, izpade itd., lahko kontekstne anomalije kažejo na spremenjen režim porabe električne energije določenega porabnika. Slika 7 prikazuje porabo za izbrano gospodinjstvo, kjer sta bili zaznani obe vrsti anomalij – prva ugotovljena anomalija z leve strani predstavlja točkovno anomalijo (morebitno napako pri merjenju ali izpad), medtem ko majhna skupina več točk anomalij predstavlja morebitni spremenjen režim delovanja (ker se skupna poraba od te točke zmanjša, a sledi praktično enakemu gibanju kot doslej, je lahko prišlo do prenehanja delovanja neke električne naprave ali za njeno zamenjavo z bolj varčno napravo). Odkrivanje anomalij je temeljilo na razvitem modelu napovedovanja porabe. Eksponentno zglajene napovedane vrednosti so bile uporabljene za določitev pričakovane "normalne" vrednosti, pri čemer je izračunana napaka (z upoštevano standardno deviacijo) na učni množici predstavlja prag za "območje normalnosti". Če je dejanska meritev presegla pričakovano območje normalnosti, je bila identificirana kot možna točkovna anomalija. Če je bilo več zaporednih meritev opredeljenih

kot anomalije, je bila skupina opredeljena kot možna kontekstna anomalija, sam model odkrivanja anomalij pa se je samodejno prilagodil novemu režimu porabe.



Slika 7: Primer odkrivanja anomalij z razvitim pilotnim napovednim modelom.
Vir: lasten.

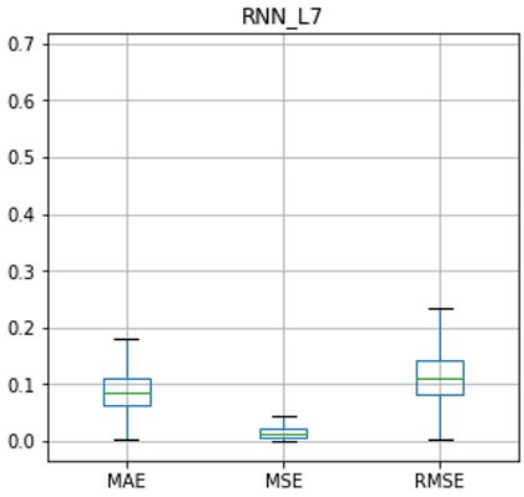
Tretji del sistema je komponenta za gručenje, namenjena analizi in segmentaciji odjemalcev v skupine podobnih porabnikov na podlagi vzorcev porabe električne energije (slika 8 prikazuje dva zelo različna profila, od katerih je vsak precej tipičen za skupino porabnikov s podobnim vedenjem). Takšno združevanje v gruče lahko omogoči npr. boljše upravljanje porabnikov in napredno statistiko porabe ali natančnejše napovedovanje, če omenimo le nekatere možnosti.



Slika 8: Primer dveh različnih tipičnih profilov porabe po mesecih, ki označujeta dve različni skupini porabnikov električne energije.
Vir: lasten.

6.3 Rezultati napovedovanja porabe električne energije

Razviti napovedni modeli so bili preizkušeni na množici meritev porabnikov električne energije. Rezultati kratkoročnega napovedovanja z uporabo rekurentnih nevronskih mrež (RNN) so se izkazali za precej dobre (glej sliko 9), saj so v povprečju dosegli manj kot 10% povprečne absolutne napake (MAE) in približno 12 % povprečne kvadratne napake (RMSE) – obe metriki sta izračunani na normaliziranih vrednostih meritev in tako izraženi v odstotkih. Ker je lahko poraba ob nekaterih priložnostih zelo razpršena, je standardni odklon napovedi žal precej velik (okoli ±10% za MAE in ±12% za RMSE).



Slika 9: Rezultati napovedovanja porabe električne energije z uporabo enega izmed razvitih napovednih modelov.
Vir: lasten.

7 Namestitev in uporaba programskih storitev

Napovedne modele smo razvili s programskim jezikom Python in jih po podrobni evalvaciji izpostavili v obliki spletnih storitev kot API. Za razvoj API smo kot osnovo vzeli spletni strežnik Tornado ter nad njim implementirali spletne storitve za namen pridobivanja ter pred-obdelave podatkov, identifikacijo anomalij v podatkih, gručenje merilnih mest in napoved porabe električne energije. Za dostop do podatkov v relacijski podatkovni bazi smo se poslužili knjižnic psycopg2 ter sqlalchemy, za delo s podatki knjižnici NumPy in pandas, za potrebe implementacije metod in algoritmov identifikacije anomalij, gručenja in strojnega učenja pa smo uporabili knjižnice scikit-learn, XGBoost, scipt ter TensorFlow. Za potrebe samodejnega generiranja interaktivne dokumentacije API smo se poslužili uporabe orodja Swagger, prilagojenega za uporabo s spletnim strežnikom Tornado. Dodatno smo za pomoč pri delu z časovnimi žigi uporabili podporno knjižnico pytz. Za izgradnjo namestitvene različice API smo uporabili tehnologijo Docker, ki nam omogoča izgradnjo prenosljivih, »stateless« slik za potrebe namestitve in izvajanja API.

Po namestitvi API je njegova uporaba s programskimi klici zelo preprosta (slika 10). Preko izpostavljenih spletnih storitev smo omogočili storitve za praktično celoten proces strojnega učenja in inteligentne analize podatkov: pred-procesiranje, učenje in testiranje različnih modelov, napovedovanje porabe, odkrivanje anomalij, gručenje odjemalcev. Sistem omogoča samodejno shranjevanje in pred-pomnjenje naučenih modelov, kar izboljša časovno učinkovitost in odzivnost storitev. Poleg sklicevanja na posamezna merilna mesta, omogoča sistem tudi gradnjo in uporabo modelov za neposredno podane meritve porabe.

```
# Napovedovanje z nevronsko mrežo za neposredno podane meritve

interval = "day"

json_data = meritve.to_dict(orient="records") # pretvorimo DataFrame v dictionary

mm_response = requests.post(BASE_URL + f"/prediction/network/{interval}",
                            json={"ida_mm": 0, "consumption": json_data})

if mm_response.status_code == 200:
    predictions_data = mm_response.json() # json string je pretvorjen v dictionary
    train_scores = predictions_data["train_scores"]
    predictions = predictions_data["predictions"]
```

Slika 10: Primer uporabe razvitega API – klic storitve iz programskega jezika Python za napoved predvidene porabe z uporabo RNN za neposredno podane meritve.
Vir: lasten.

8 Diskusija in zaključek

V članku smo na kratko predstavili primer vpeljave metod AI in ML v poslovni proces podjetja, namenjen razvoju novih naprednih inteligentnih rešitev in storitev, temelječih na metodah, algoritmih in orodjih umetne inteligence. Na kratko smo pojasnili, kako lahko ponudnik znanja pomaga podjetju, ki deluje na trgu električne energije, pri sprejemanju metod ML za svoje poslovne dejavnosti in storitve. Navedli smo primer takšnega pilotnega projekta, namenjenega zagotavljanju novih naprednih storitev na podlagi učinkovitega in natančnega modela napovedovanja porabe električne energije.

Opisan pilotni projekt vpeljave metod AI in ML v poslovni proces je podjetju Informatika pomagal prepoznati, zasnovati in implementirati (trenutno v pilotnem okolju) inteligentne rešitve, ki temeljijo na umetni inteligenci in strojnem učenju ter lahko omogočijo vrsto novih in naprednih storitev njihovim poslovnim strankam in končnim uporabnikom.

Z učinkovitimi in točnimi napovednimi modeli za porabo električne energije na nivoju posameznega odjemalca oz. merilnega mesta (skupaj z drugimi komponentami) lahko Informatika različnim strankam ponudi vrsto naprednih novih storitev:

- Za distributerje in dobavitelje: optimizacija količine električne energije, optimizacija alternativnih virov, načrtovanje razvoja elektrodistribucijskega omrežja in naložb v električno omrežje, ...
- Za zunanje deležnike (npr. regulatorje): vizualizacija in analiza porabe po regijah, občinah, ..., analiza vedenjskih navad pri porabi, določitev tarifnega sistema in novih pravil za obračunavanje, ...
- Za končne uporabnike (npr. porabnike): boljše spremljanje in načrtovanje porabe, odkrivanje napak na napravah, optimalnejša poraba električne energije, ...

Pri tem je potrebno poudariti, da je bil glavni namen projekta predvsem sistematično preizkusiti, na kak način lahko podjetje načrtuje razvoj in uporabo metod strojnega učenja za zagotavljanje boljših storitev svojim strankam, ne pa razviti popolnoma optimiziran napovedni model za porabo električne energije. Čeprav razviti modeli napovedovanja kažejo obetavne rezultate, zahtevna naloga zagotavljanja tehnologije za napovedovanje porabe električne energije, ki bo primerna za produkcijsko okolje, še zdaleč ni končana. Noben posamezen model strojnega učenja ne prekaša vseh drugih modelov za vsak problem napovedovanja. Zato je izbira najboljšega algoritma odvisna od specifičnih nalog in izzivov napovedovanja. Ob tem obstaja ogromno možnosti za fino prilagajanje parametrov in nastavitev izbranih algoritmov, ki lahko bistveno vplivajo na rezultate, tako z vidika napovedne uspešnosti kot tudi glede njihove računske zahtevnosti. Poleg tega razvoj natančnih in učinkovitih modelov ML sam po sebi niti približno ni dovolj za doseganje vseh prednosti, ki jih lahko ponudi ustrezna digitalna preobrazba poslovnega procesa. Je pa vsekakor zelo dober in obetaven začetek v tej smeri, ki ga bomo zagotovo nadaljevali tudi v prihodnje.

Literatura

[1] Zhang, Y., Huang, T., Bompard, E.F., Big data analytics in smart grids: a review, Energy informatics, 1(1): 1-24 (2018).

[2] European Commission. The European Green Deal: Communication from the Commission to the European Parliament, the European Council, the Council, the European Economic and Social Committee and the Committee of the Regions; European Commission: Brussels, Belgium, 2019.

[3] Walther, J., Weigold, M., A systematic review on predicting and forecasting the electrical energy consumption in the manufacturing industry, Energies, 14(4) (2021).

[4] Hesselbach, J.; Herrmann, C.; Detzer, R.; Martin, L.; Thiede, S.; Ludemann, B., Energy efficiency through optimised coordination of production and technical building services, In LCE 2008: 15th CIRP International Conference on Life Cycle Engineering: Conference Proceedings, The University of New South Wales: Sydney, Australia, p. 62 (2008).

[5] International Energy Agency (IEA), Tracking Power. Available online: https://www.iea.org/reports/tracking-power-2020

[6] Beier, J. Simulation approach towards energy flexible manufacturing systems, Sustainable Production, Life Cycle Engineering and Management; Springer International Publishing: Cham, Switzerland (2017).

[7] Kalimoldayev, M., Drozdenko, A., Koplyk, I., Marinich, T., Abdildayeva, A., Zhukabayeva, T., Analysis of modern approaches for the prediction of electric energy consumption, Open Engineering, 10(1): 350-361 (2020).

[8] Duan, Y., Edwards, J.S., Dwivedi, Y.K., Artificial intelligence for decision making in the era of Big Data–evolution, challenges and research agenda, International Journal of Information Management, 48: 63-71 (2019).

[9] Lichtenthaler, U., Building blocks of successful digital transformation: Complementing technology and market issues, International Journal of Innovation and Technology Management, 17(1): 2050004 (2020).

[10] Hafeez, G., et al.: A Novel Accurate and Fast Converging Deep Learning-Based Model for Electrical Energy Consumption Forecasting in a Smart Grid, Energies, 13(2244): 1-25 (2020). doi:10.3390/en1309224

[11] Almalaq, A., Zhang, J.J.: Deep Learning Application: Load Forecasting in Big Data of Smart Grids, in W. Pedrycz and S.-M. Chen (eds.), Deep Learning: Algorithms and Applications, vol. 865 (New York: Springer-Verlag) 103-128. (2020)

[12] Liu, H., Wang, Y., Wei, C., Li, J., Lin, Y., Two-Stage Short-Term Load Forecasting for Power Transformers Under Different Substation Operating Conditions, IEEE Access, 7: 161424-161436 (2019).

[13] Nespoli, L., Medici, V., Lopatichki, K., & Sossan, F., Hierarchical demand forecasting benchmark for the distribution grid, Electric Power Systems Research, 189: 106755 (2020).

[14] Veeramsetty, V., Deshmukh, R., Electric power load forecasting on a 33/11 kV substation using artificial neural networks, SN Applied Sciences, 2(5): 1-10 (2020).

Reševanje industrijskih problemov z uporabo računske inteligence: primer avtomatskega načrtovanja delovnega časa

Grega Vrbančič, Iztok Fister ml., Vili Podgorelec

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
grega.vrbancic@um.si, iztok.fister1@um.si, vili.podgorelec@um.si

Sinopsis Dostopnost velikih količin podatkov in relativno poceni in dostopne računske moči je v zadnjih nekaj letih pripomogla k enormnemu vzponu računske inteligence. Čeprav se mnoga podjetja in organizacije že dolga leta poslužujejo uporabe različnih tehnik matematične optimizacije, ki se najpogosteje uporabljajo za namen optimizacije poslovnih procesov, proizvodnih procesov, logistike in upravljanja distribucijskih omrežij, pa so novodobna gigantska podjetja kot so Uber, Netflix in Tesla vpeljala metode in tehnike računske inteligence tudi v bolj uporabniku usmerjene funkcionalnosti ter tako še dodatno povečale njihovo prepoznavnost. V industrijskih okoljih je tako sedaj, bolj kot kadarkoli, pomembno da je organizacija in izraba razpoložljivih virov kar se da učinkovita, saj lahko tako dosežejo pomembno konkurenčno prednost. Pri vpeljavi računske inteligence za reševanje industrijskih problemov pa se podjetja pogosto soočijo s številnimi izzivi, med katerimi pogosto izstopa predvsem zahtevna prilagoditev naprednih optimizacijskih algoritmov, predstavljenih v literaturi, na domensko specifični problem.

V prispevku predstavimo pristope, metode in tehnike, s katerimi lahko učinkovito naslavljamo najpogostejše industrijske probleme, obenem pa opozorimo na morebitne pasti, s katerimi se lahko soočimo pri vpeljavi ter podali smernice za njihovo naslovitev. V nadaljevanju podrobneje predstavimo primer uporabe optimizacijskih algoritmov, ki delujejo po vzorih iz narave, za reševanje problema avtomatske izdelave kompleksnih urnikov.

Ključne besede:
reševanje industrijskih problemov
računska inteligenca
diferencialna evolucija
optimizacija z rojem delcev
avtomatsko načrtovanje

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.21

1 Uvod

Dandanes je tekmovalnost na globalnem trgu prisotna že od začetka razvoja nekega izdelka. Uspeti z nekim novim produktom, storitvijo ali izdelkom na globalnem trgu ni več samoumevno, saj konkurenca, ki jo najdemo domala na vseh kontinentih, lahko prav tako razvija podoben produkt, ki je lahko celo boljši in cenejši. Zaradi teh razlogov podjetja svoje izdelke poskušajo venomer bolj razvijati in oblikovati na stroškovno učinkovitejši in trajnostni način, da lahko povečajo svoj dobiček in uspešnost ter seveda čim bolj zmanjšajo stroške in potencialne napake. To od inženirjev zahteva, da sledijo najnovejšim trendom v raziskavah ter da uporabljajo stroškovno najučinkovitejše tehnologije in moderne računalniške modele. V zadnjih letih postaja umetna inteligenca ena najbolj priljubljenih orodij v industriji, saj lahko pomaga pri različnih fazah razvoja nekega izdelka ali storitve. Z uporabo umetne inteligence lahko načrtujemo, oblikujemo in izdelujemo izdelke na hitrejši in učinkovitejši način, kjer prav tako že med razvojem poskušamo minimizirati prisotnost napak.

Celotno področje umetne inteligence je izjemno široko saj so raziskovalci že razvili ogromno različnih metod, ki jih lahko štejemo pod okrilje umetne inteligence. Posledično je zato dosti težje odkriti neko metodo, ki je primerna za reševanje določenega domensko specifičnega problema. V tem članku se osredotočamo na računsko inteligenco, ki je podveja umetne inteligence. Računska inteligenca se je v zadnjem desetletju izkazalo kot hitro rastoče področje. Poleg metod in tehnik kot so genetski algoritmi, evolucijsko računalništvo in strojno učenje, računska inteligenca vključuje tudi metode, ki delujejo po vzorih iz narave in poskušajo reševati težke probleme s posnemanjem naravnih sistemov. Računska inteligenca tako predstavlja skupek metod, tehnik in orodij za inteligentno obdelavo podatkov na katerih temelji odločanje in upravljanje v različnih organizacijah od proizvodnih podjetij do medicinskih in zdravstvenih ustanov.

Ne glede na uspešnost uporabe omenjenih metod za reševanje različnih domensko specifičnih problemov v raznovrstnih organizacijah, pa uporaba oz. vpeljava teh prinaša nemalo izzivov s katerimi se moramo soočiti v kolikor želimo prednosti posameznih metod in tehnik računske inteligence kar se da dobro izkoristiti. Nenazadnje lahko že sama izbira metode ali tehnike za naslovitev specifičnega problema predstavlja velik izziv, še posebej v današnjih časih, ko se novi pristopi in rešitve problemov razvijajo na skorajda dnevni ravni. Izbira primerne metode ali tehnike je ključna pri naslovitvi problema, saj lahko z neprimerno izbiro navidezno uspešno rešimo ciljni problem, dolgoročno pa se izkaže da takšna rešitev ne naslavlja problema na način kot smo si ga zadali oz. ne dosega željene uspešnosti. Z neprimerno izbiro metode ali tehnike želenih se lahko dogodi tudi, da ciljev sploh ne dosežemo in posledično opustimo idejo o vpeljavi računske inteligence za rešitev specifičnega domenskega problema ter tako potencialno ne pridobimo konkurenčne prednosti. Poleg same izbire metode ali tehnike pa ne smemo zanemariti tudi problematike same vpeljave te za naslovitev domensko specifičnega problema. Za uspešno vpeljavo moramo namreč dovolj dobro poznati sam problem, podatke, ki so nam na voljo ter določiti omejitve, da lahko oblikujemo model problema, ki zajema vse zahteve in omejitve ter v končni fazi uspešno rešuje zadan problem. Našteti izzivi pa nemalokrat povzročajo probleme tudi ekspertom s področja računske inteligence.

S ciljem približati in olajšati vpeljavo računske inteligence za reševanje realnih industrijskih problemov, v prispevku predstavimo metode in tehnike, s katerimi lahko učinkovito naslavljamo najpogostejše industrijske probleme, obenem pa tudi opozorimo na morebitne pasti, s katerimi se lahko soočimo pri vpeljavi ter podamo smernice za njihovo naslovitev. V nadaljevanju prispevka podrobneje predstavimo primer uporabe dveh metod iz skupine računske inteligence, tj. evolucijskim algoritmom in algoritmom inteligence rojev, za namen reševanja realnega problema avtomatskega načrtovanja delovnega časa. Obe izmed uporabljenih metod sta primerni za reševanje težkih optimizacijskih problemov v raznovrstnih industrijskih okoljih.

2 Uporaba računske inteligence

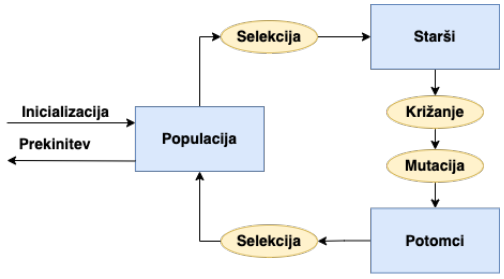
V tem poglavju na kratko opredelimo pojem računske inteligence in predstavimo glavne metode, ki spadajo pod to okrilje. Vsako metodo tudi na kratko opišemo. V nadaljevanju bralca bolj podrobno seznanimo z evolucijskimi algoritmi in algoritmi inteligence rojev, ki so pomembni za nadaljno razumevanje tematike. Poglavje zaključimo s kratkim opisom industrijskih aplikacij, ki temeljijo na uporabi evolucijskih algoritmov in algoritmov inteligence rojev.

2.1 Računska inteligenca

Računska inteligenca (angl. Computational Intelligence, krajše CI) obsega metode, ki delujejo po vzorih iz narave in poskušajo reševati težke probleme s posnemanjem principov naravnih sistemov [1]. V skupino metod računske inteligence štejemo: umetne nevronske mreže (angl. Artificial Neural Networks, krajše ANN) [4], umetne imunske sisteme (angl. Artificial Immune Systems, krajše AIS) [6], sisteme na osnovi mehke logike (angl. Fuzzy Systems, krajše FS) [3], evolucijske algoritme (angl. Evolutionary Algorithms, krajše EA) [5], ter algoritme inteligence rojev (angl. Swarm Intelligence, krajše SI) [2]. Umetne nevronske mreže delujejo po vzgledu delovanja človeških možganov in se precej uporabljajo za probleme, ki vključujejo klasifikacijo, razpoznavanje vzorcev, regresijske probleme kot tudi za gradnjo napovednih modelov. Umetni imunski sistemi temeljijo na principu delovanja naravnih imunskih sistemov in njihova glavna značilnost je sposobnost razpoznavanja vzorcev, zato veliko aplikacij uporabe umetnih imunskih sistemov najdemo na področju varnosti računalniških sistemov in iskanja anomalij. Prav tako so tudi odlično orodje za reševanje optimizacijskih problemov. Sistemi mehke logike uporabljajo aproksimativno sklepanje namesto eksaktnega sklepanja in se posledično zato dobro obnesejo pri problemih regulacije, nadzoru robotskih sistemov, ipd. Evolucijski algoritmi posnemajo Darwinovo evolucijsko teorijo in so zelo učinkovito orodje za reševanje težkih optimizacijskih problemov v računalništvu, fiziki, matematiki, medicini in tudi v industriji. Sorodna skupina evolucijskih algoritmov so algoritmi inteligence rojev, ki temeljijo na agentih, kjer je vsak agent sposoben izvajanja enostavnega opravila. V primeru povezave v skupnost so sposobni tudi kompleksnega obnašanja [1].

2.2 Evolucijski algoritmi in algoritmi inteligence rojev

Evolucijske algoritme in algoritme inteligence rojev štejemo tudi v skupino stohastičnih populacijskih algoritmov po vzorih iz narave (angl. Stochastic population-based nature-inspired algorithms). Njihovo delovanje je navdihnjeno s strani narave in različnih procesov, ki potekajo v naravi [7]. Prvi predstavniki, ki so se pojavili že v zgodnjih 60-tih letih prejšnjega stoletja so evolucijski algoritmi. Njihov glavni navdih je Darwinova evolucijska teorija, ki pravi, da imajo posamezniki, ki se najbolj prilagodijo razmeram v okolju več možnosti, da preživijo in se posledično reproducirajo. Slabše posameznike pa proces naravne selekcije izloči iz populacije. Osnovni evolucijski algoritem sestoji iz populacije posameznikov, ki je podvržena različnim variacijskim operatorjem, kot npr. mutacija, križanje in selekcija. Slika 1 prikazuje osnoven princip delovanja splošnega evolucijskega algoritma.



Slika 1: Osnoven princip delovanja evolucijskih algoritmov.
Vir: lasten.

Skozi obdobje razvoja evolucijskih algoritmov se je pojavilo več različnih variacij teh algoritmov. Dandanes družina evolucijskih algoritmov zajema: genetske algoritme, evolucijske strategije, genetsko programiranje in diferencialno evolucijo (DE). Glavne razlike med temi skupinami ležijo v predstavitvi posameznikov v populaciji, npr. pri genetskih algoritmih so posamezniki predstavljeni z binarnim zapisom, medtem ko so v diferencialni evoluciji posamezniki predstavljeni z vektorjem realnih števil. Poleg predstavitve posameznikov v populaciji so glavne razlike še v uporabi variacijskih operatorjev.

Druga skupina so algoritmi inteligence rojev, kateri so navdihnjeni na podlagi socializiranih žuželk (npr. mravelj, čebel, ipd.) in nekaterih vrst socializiranih živali (npr. ptic, rib, ipd.). Te živali so sposobne opravljanja kompleksnih opravil v skupini, čeprav jim je narava kot posameznikom namenila avtonomno izvajanje samo preprostih akcij. Definicija iz članka [8] pravi, da se inteligenca rojev nanaša na kolektivno nastajajoče vedenje več agentov, ki pri sobivanju upoštevajo nekaj preprostih pravil. Medtem ko lahko agenta kot posameznika obravnavamo kot omejenega, ker je sposoben izvajanja zgolj preprostih akcij, pa večagentni sistem, kjer ti delujejo kot celota, lahko kaže samoorganizacijsko vedenje in izkazujejo določeno vrsto kolektivne inteligence.

2.3 Reševanje industrijskih problemov

Področje evolucijskih algoritmov in algoritmov inteligence rojev je izredno dinamično, saj se skoraj vsak dan pojavijo kakšni novi oziroma prilagojeni algoritmi, ki so primerni za reševanje praktičnih problemov. Domala skoraj ni možno najti področja, kjer ni vsaj ene aplikacije, ki temelji na teh pristopih. Nekaj izbranih primerov je prikazanih v nadaljevanju.

Evolucijski algoritmi se uporabljajo za načrtovanje in optimizacijo različnih anten [10], avtomatski razvoj avtomobilskih motorjev [12], optimizacijo poti pri brezpilotnih vozilih [13]. Njihova prisotnost je utrjena tudi na področju energetike, kjer se uporabljajo za ocenjevanje parametrov fotovoltaičnih modelov [14], iskanja najboljših pozicij za postavitev vetrnih elektrarn [15] in napovedovanja proizvodnje električne energije [16, 17]. Algoritmi inteligence rojev se prav tako uporabljajo za reševanje podobnih problemov, vendar se veliko teh pristopov uporablja tudi za reševanje kombinatoričnih optimizacijskih problemov [21, 22]. Primeri uporabe obeh skupin pristopov se lahko najdejo tudi na ostalih področjih, kot so področje športa [18], ekonomije in bančništva [11] ter medicine [19]. Za več primerov uporabe vabimo bralca k branju naslednjega prispevka [20].

3 Vpeljava računske inteligence za naslovitev industrijskih problemov

V prispevku smo se omejili na vpeljavo algoritmov iz skupin evolucijskih algoritmov in algoritmov inteligence rojev. Kot smo predstavili v prejšnji sekciji so algoritmi iz teh skupin uspešno uporabljeni za naslovitev številnih industrijskih problemov pri čemer dosegajo visoko stopnjo uspešnosti. Za uspešno vpeljavo tovrstnih algoritmov je potrebno razumeti njihov osnoven princip delovanja. Razumevanje osnovnega principa delovanja je namreč ključno, da lahko ciljni problem v osnovi ustrezno opredelimo.

3.1 Kdaj uporabiti evolucijske algoritme ali algoritme inteligence rojev?

V praksi se velikokrat srečamo z novim problemov za katerega potrebujemo rešitev. Za veliko problemov obstaja kopica rešitev in jih lahko zelo hitro rešimo oziroma se približamo željeni rešitvi. Včasih pa za določen problem ni na voljo neke metode s pomočjo katere pridemo do rešitve. Včasih smo prav tako postavljeni pred nek problem za katerega nikakor ne moremo izpeljati neke metode, ki bi nas privedla do rešitve. Prav tako veliko ljudi v industrijskih okoljih ne zanima kako smo prišli do rešitve, samo da jo najdemo [30]. Če ima naš problem podobne karakteristike, potem je velika možnost, da se ga da rešiti z metodami računske inteligence oziroma bolj natančno z evolucijskimi algoritmi ali algoritmi inteligence rojev. A po drugi strani je zelo pomembno še omeniti, da so določeni problemi, ki jih rešujemo z eksaktnimi metodami zelo časovno in prostorsko zahtevni. V tem primeru

nam lahko uporaba metod računske inteligence eliminira veliko časovno zahtevnost, vendar nam ne zagotovi najbolj optimalne rešitve.

3.2 Opredelitev in modeliranje problema

Preden začnemo načrtovati evolucijski algoritem za določen problem je potrebno problem natančno opredeliti. Takoj na začetku je potrebno določiti kakšen izhod pričakujemo in če obstajajo kakšni vhodi. Opredeliti je potrebno tudi vse omejitve in predpostavke za določen problem. Zatem je potrebno določiti kaj nam bodo predstavljali posamezniki oziroma kandidatne rešitve v populaciji v evolucijskem ciklu. Če iščemo npr. najkrajšo pot od točke A do točke B, potem lahko posameznik predstavlja vozlišča med obema točkama. V primeru da iščemo optimalni načrt športnega treninga, potem nam lahko posamezniki predstavljajo obremenitev treninga po dnevih. Na podlagi tega je potem lažje izbrati tudi primeren algoritem. Če so posamezniki predstavljeni kot zaporedje binarnih števil, potem je smiselno izbrati genetski algoritem, če so posamezniki predstavljeni kot programi, potem se lahko zatečemo k genetskemu programiranju. Če so posamezniki predstavljeni kot vektorji realnih števil, potem lahko uporabimo na primer diferencialno evolucijo. Po tem koraku je potrebno definirati kriterijsko funkcijo. Naloga kriterijske funkcije je ocenitev kvalitete posameznika v populaciji. Na podlagi te ocenitve se potem boljši posamezniki prenesejo v naslednjo generacijo, medtem ko se slabši posamezniki ponavadi zavržejo oziroma je to čisto odvisno od algoritma. Po definiciji kriterijske funkcije pa je potrebno še potem tudi prilagoditi variacijske operatorje in/ali dodati kakšno lokalno iskanje, s katerimi lahko izboljšamo delovanje algoritma za nek specifičen nabor ciljnih problemov.

3.3 Izzivi

Trenutno obstaja izjemno veliko število različnih algoritmov, ki jih je predlagala akademska skupnost [31]. V vsej tej poplavi je zelo težko za začetnika izbrati najbolj primeren algoritem za določen problem. Po drugi strani je veliko algoritmov v bistvu kopija že prej obstoječih, vendar v drugi preobleki. Zaradi tega se priporoča, da uporabljamo algoritme, ki so najbolj razširjeni in obstajajo v literaturi že več kot desetletje in je možno preveriti njihovo ozadje. Zelo pomembno je omeniti NFL teorem [32], ki pravi, da ni univerzalnega iskalnega algoritma, ki bi uspešno deloval na vseh družinah problemov. Zato je potrebno vedno tudi raziskati kako so ostali raziskovalci rokovali z določenim problemov in kateri algoritmi so se najbolje obnesli na določenih problemih. Naslednji izziv je velikokrat nastavljanje nadzornih parametrov pri algoritmih. Skoraj vsak evolucijski algoritem ali algoritem inteligence rojev ima nekaj nadzornih parametrov, ki krmilijo njegovo delovanje tekom evolucijskega cikla. Včasih je potrebno vložiti veliko časa preden najdemo uspešno kombinacijo teh parametrov. Na srečo pa obstajajo tudi metode, kjer se parametri prilagajajo tekom evolucijskega cikla [33]. Dodatno je pri reševanju prostorsko zahtevnih problemov v splošnem težko ovrednotiti koliko dobra je trenutno najboljša najdena rešitev in ali sploh obstaja boljša. S tega stališča nam lahko izziv predstavlja tudi določitev zaustavitvenega pogoja, ki je navadno definiral z maksimalnim številom ovrednotenj kriterijske funkcije.

4 Primer avtomatskega načrtovanja delovnega časa

V nadaljevanju bomo predstavili praktični primer uporabe algoritmov računske inteligence za namen avtomatskega načrtovanja delovnih časov. V mnogih poklicih narava dela ki ga opravljajo, zahteva, da zaposlenih delajo v različnih izmenah, pri čemer je lahko tudi trajanje same izmene dinamično ali prilagojeno. Tipični poklici kjer je to izrazito vidno so zdravstvena oskrba, storitve varovanja, logistika in transport, proizvodnje in različni podporni centri. Poleg same narave dela določenega poklica je potrebno pri načrtovanju delovnega časa upoštevati tudi mnogo formalnih zahtev, da je lahko nek sestavljen urnik izvedljiv tudi v praksi ter pokriva vse poslovne kot tudi zakonske zahteve in omejitve. V splošnem delovni čas v posameznih organizacijah načrtujejo upravljalski oz. vodstveni kadri, ki dodobra poznajo zahteve iz vidika poslovanja kot tudi imajo bogate izkušnje pri načrtovanju

delovnega časa. Priprava načrta delovnega časa je v splošnem kompleksen problem, ki od načrtovalca zahteva visoko stopnjo napora. Nemalokrat se lahko tudi prepeti, da se nenadoma spremenijo poslovne zahteve ali pa se lahko zaradi različnih razlogov soočamo s pomanjkanjem človeških virov. V takih primerih morajo načrtovalci reagirati hitro in poskrbeti za čim učinkovitejšo naslovitev problema ter prilagoditi načrt delovnega časa, kar navadno sila stresno opravilo. Opisana problematika je več kot primeren kandidat za naslovitev z uporabo računalniških sistemov, saj nam razpoložljiva računska moč in napredni inteligentni algoritmi omogočajo reševanje takšnih problemov z manj potrebnega vloženega truda, s čimer lahko v različnih situacijah hitreje in učinkoviteje pridemo do ustrezne rešitve oz. načrta delovnega časa. V nadaljevanju bomo predstavili rešitev za splošno načrtovanje delovnega časa zaposlenih na delovna mesta glede na podane kriterije končnega uporabnika.

4.1 Opredelitev problema

Načrtovanje delovnega časa nemalokrat predstavlja veliko težavo saj je pri načrtovanju potrebno upoštevati specifike organizacije kot tudi razpoložljive človeške vire in nenazadnje zakonske omejitve. Načrtovanje in razvoj rešitve, ki omogoča avtomatsko načrtovanje delovnega časa predstavlja kompleksen problem, ki je lahko v primeru načrtovanja delovnega časa za večja podjetja težko rešiljiv ali celo nerešljiv. Glede na različne specifike podjetij in druge omejitve želimo razviti kar se da »splošno« rešitev, ki omogoča načrtovanje delovnega časa ob upoštevanju specifičnih želja in zahtev končnega uporabnika. Rešitev mora torej omogočati visoko stopnjo prilagodljivosti pri čemer mora upoštevati sledeče:

- Podjetje ima lahko več poslovnih enot, vsaka poslovna enota pa lahko ima več zaposlenih.
- Vsaka poslovna enota ima določeno za vsak dan koliko zaposlenih potrebuje za določeno izmeno in določeno znanje oz. delovno mesto.
- Vsak zaposleni ima eno ali več znanj oz. opravlja delo na različnih delovnih mestih (blagajnik, prodajnik, skladiščnik...). Vsak zaposleni ima naziv in seznam znanj pri čemer ima vsak posameznik eno ali več znanj, ki so primarne ter poljubno mnogo sekundarnih znanj s katerimi lahko opravlja delo na določenem delovnem mestu. Na primer zaposleni Jože Novak, je lahko blagajnik in prodajnik, izjemoma pa je lahko tudi skladiščnik. Rešitev ga torej mora primarno skušati dodeliti na delovni mesti blagajnika ali prodajnika, izjemoma pa ga lahko tudi razporedi na delovno mesto skladiščnika. Dodatno rešitev upošteva za vsakega zaposlenega tudi kdaj in koliko lahko dela oz. kdaj dela ne more opravljati (dopust, bolniška, maksimalno število ur na posamezno izmeno). Vsak zaposleni ima tudi stanje ur na začetku meseca, ter število ur, ki jih mora opraviti v mesecu, pri čemer ima lahko vsak delavec za opraviti različno število ur (dopusti, bolniške)
- Podjetje ima definiran šifrant izmen, na primer: dopoldan (8.00 -14.00), popoldan (14.00 – 22.00
- Podjetje ima definiran šifrant vrste dela (redno delo, nedeljsko delo, praznično delo...).

Tabela 1 prikazuje primer definicije zahtev glede števila zaposlenih na določenem delovnem mestu v posamezni poslovni enoti na določen dan.

Tabela 1: Primer vhodnih zahtev za avtomatsko načrtovanje delovnega časa

Poslovna enota	Datum	Vrsta dela	Izmena	Zaposleni
PE 1	1.10.2022	Redno delo	Dopoldan	2 blagajnika
		Redno delo	Dopoldan	3 prodajniki
		Redno delo	Popoldan	1 blagajnik
	2.10.2022	Nedeljsko delo	Dopoldan	2 blagajnika
PE 2	1.10.2022	Redno delo	Dopoldan	1 blagajnik

Glede na to, da želimo čim višjo prilagodljivost rešitve je smiselno tudi, da ta upošteva oz. omogoča delno zapolnitev delovnega urnika kar končnemu uporabniku omoča, da vnaprej določi kateri zaposleni bo na kateri dan

delal na nekem delovnem mestu in v izbrani poslovni enoti. Tako »zasedene« izmene razvita rešitev ne sme več spreminjati, seveda pa jih mora vključiti oz. upoštevati pri izračunu morebitnih prekoračitev omejitev.

Pogosto se dogodi, da imajo podjetja premalo na voljo premalo človeških virov glede na potrebe poslovanja. Posledično prihaja do nadur oz. do kršitev določenih omejitev. Rešitev mora tako omogočati, da lahko posamezno podjetje definira za določeno kršitev nekakšno kazen, cilj rešitve pa je seveda, da je seštevek morebitnih kazni čim manjši ob pogoju da je izdelan načrt dela še vedno izvedljiv tudi v praksi. Za lažjo predstavo si poglejmo primer. V primeru, da rešitev oblikuje načrt dela na način, da se nek zaposleni ponovno vrne na delo 10 ur po oddelani izmeni se v tem primeru krši minimalni dnevni počitek med dvema izmenama, ki znaša 12 ur. Če podjetje definira, da je 1 ura kršitve dnevnega počitka 50 točk kazni, pomeni da je rešitev pri načrtovanju prekršila dnevni počitek za 2 uri, kar znese 100 točk kazni. Pri tem je potrebno vse kršitve omejitev izračunavati na za vsakega zaposlenega posebej. Prav tako je pomembno, da so takšne morebitne kazni čimbolj enakomerno razporejene med zaposlene, saj s tem posledično prihaja do preobremenitev določenih zaposlenih in potencialno več bolniških odsotnosti ter dodatnega primankljaja zaposlenih. Neuravnotežene kršitve omejitev na ravni posameznega zaposlenega pa tudi lahko privedejo do splošnega nazadovoljstva zaposlenih. Rešitev mora končnemu uporabniku omogočati, da določi število kazenskih točk za posamezno kršitev glede na svoje želje oz. potrebe. Tabela 2 prikazuje seznam omejitev, ki jih razvita rešitev upošteva pri načrtovanju, pripadajoče vrednosti kršitev podanih omejitev pa uporabi za namen načrtovanja načrta dela, ki bo kršil omejitve v čim manjšem obsegu. Prijagajanje kazenskih točk za posamezno omejitev, končnemu uporabniku omogoča, da določi z njegovega vidika bolj pomembne (višje število kazenskih točk) in manj pomembne (nižje število kazenskih točk) omejitve.

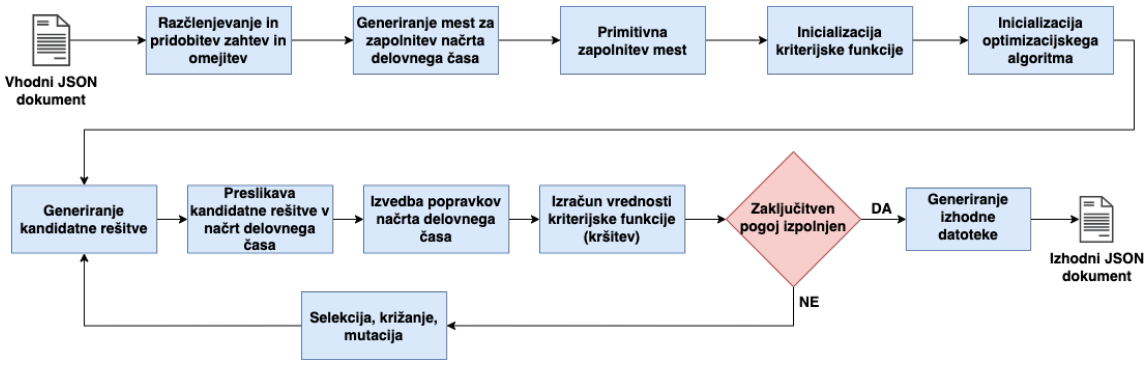
Tabela 2: Seznam kršitev podanih omejitev

Tip kršitve	Kazen
Kršitev za nepravilno vrsto dela	10 točk
Kršitev za prekoračitev delovne izmene	10 točk
Kršitev za nepravilno delovno mesto	5 točk
Kršitev za kršenje dnevnega počitka	20 točk
Kršitev za kršenje tedenskega počitka	20 točk
Kršitev za umestitev zaposlenega v drugo poslovno enoto	5 točk

Z definiranimi omejitvami lahko tako usmerjamo načrtovanje urnika v smeri zahtev podjetja, ne glede na kršitve pa gre v osnovi za mehke omejitve pri katerih bo končni rezultat rešitve izvedljiv načrt delovnega časa v kolikor seveda obstaja rešitev za dan problem. Poleg omenjenih omejitev mora razvita rešitev upoštevati tudi strožje omejitve katerih kršenje se odraža v obliki neizvedljivega načrta delovnega časa. V našem primeru smo si zadali dve takšni omejitvi in sicer razporeditev zaposlenega na dve delovni mesti v isti izmeni ter razporeditev zaposlenega na delovno mesto v terminu ko je ta odsoten (dopust, bolniška...).

4.2 Načrtovanje in implementacija rešitve

Skladno s predstavljenim problemom in zahtevami smo oblikovali rešitev temelječo na evolucijskem algoritmu oz. algoritmu inteligence rojev. Problem smo namreč zmodelirali na način, da nam omogoča uporabo praktično kateregakoli algoritma, ki operira z posamezniki definiranimi v obliki vektorja realnih števil. To nam posledično daje odlično izhodišče za eksperimentiranje z različnimi algoritmi brez, da bi pri tem morali posegati v samo zasnovo rešitve. Diagram (Slika 2) predstavlja konceptualno zasnovo naše rešitve za definiran problem avtomatskega načrtovanju delovnega časa.



Slika 2: Konceptualni diagram rešitve problema avtomatskega načrtovanja delovnega časa.
Vir: lasten.

Na vhodu prejemo JSON dokument z definiranimi zahtevami in omejitvami ter človeškimi viri, ki so nam na voljo na načrtovanje urnika dela. Podan dokument razčlenimo ter pridobimo vrednosti omejitev ter nabor zahtev. Sledi generiranje mest za zapolnitev načrta delovnega časa. V tem koraku vsako podano zahtevo preoblikujemo na način, da iz posamezne zahteve dobimo posamezna mesta za načrt delovnega časa. Vzemimo na primer zahtevo, da morata biti v ponedeljek 18.7.2022 v Poslovalnici 1, v jutranji izmeni, ki traja od 7.00 do 15.00 zaposlena dva blagajničarja. Takšno zahtevo preoblikujemo v dve mesti v načrtu delovnega časa kjer ima vsako mesto definirane enake zahteve kot so bile podane, le da lahko posamezno izmed tako oblikovanih mest možno zapolniti zgolj z enim zaposlenim. Po zaključku preoblikovanja zahtev in omejitev v obliko posameznih mest sledi zapolnitev morebitnih že v naprej določenih mest, ki so bila definirana že v vhodni datoteki. Takšnih mest rešitev ne sme premikati. Število mest, ki ostanejo nezapolnjena nam predstavlja dimenzijo našega problema – za toliko mest bo namreč naloga algoritma, da jih zapolni. Sledi inicializacija kriterijske funkcije. Ta skrbi za izračunavanje kazni v primeru morebitnih kršitev podanih omejitev načrtovanega urnika dela. V fazi inicializacije optimizacijskega algoritma, izbranemu algoritmu (v našem primeru smo preizkusili dva – diferencialno evolucijo in algoritem optimizacije z rojem delcev), nastavimo pripadajoče vrednosti parametrov. Število parametrov se od algoritma do algoritma razlikuje, kot tudi se razlikujejo njihove privzete vrednosti. Za primerno nastavitev vrednosti parametrov ni splošnega pravila kako jih nastaviti. Navadno je potrebno veliko eksperimentiranja, da dosežemo kar se da optimalne vrednosti parametrov, ki nam dajejo kar se da dobre rezultate. Poleg parametrov specifičnih za posamezen algoritem pa je potrebno nastaviti tudi velikost populacije ter maksimalno število ovrednotenj kriterijske funkcije. Na podlagi izkušenj je smiselno število populacije nastaviti na 15 v kolikor je dimenzija problema manjša od 30, na 30 v kolikor je dimenzija problema med 30 in 200 ter na 60 v kolikor je dimenzija problema večja od 200. Inicializiran algoritem zaženemo nam ciljnim problemomter pridobimo t.i. kandidatno rešitev v obliki vektorja realnih števil, katere na to preslikamo v obliko kjer posamezno mesto predstavlja delovno izmeno nekega zaposlenega. Sledi izvedba popravkov v tako pridobljenem načrtu delovnega časa kjer z zamenjavo posameznih zaposlenih na delovnih mesti dosežemo, da je ujemanje z omejitvami (delovno mesto oz. potrebna znanja)čim boljše. Nad tako »popravljenim« načrtom delovnega časa izračunamo morebitne kršitve vsakega posameznega zaposlenega za posamezno kršitev posebej. Vse kršitve istega tipa seštejemo ter nad njimi izračunamo uteženo vsoto kjer so kršitve definirane z večjim številom točk bolj obtežene kot tiste z nižjim številom točk. Sledi preverjanje zaključitvenega pogoja, ki je v našem primeru definiran z maksimalnim številom ovrednotenj kriterijske funkcije oz. z maksimalnim časom izvajanja. V kolikor pogoj ni izpolnjen se vrednost kriterjske funkcije pošlje algoritmu kjer sledi izvedba operacij selekcije, križanja in mutacije z namenom pridobitve nove kandidatne rešitve na podlagi ocene prejšnje kandidatne rešitve (vrednost kriterijske funkcije). V kolikor pa je zaključitven pogoj izpolnjen oblikuje rešitev načrta delovnega časa na podlagi do sedaj najboljše oblikovane rešitve, torej rešitve, ki je dosegla najnižje število kazenskih točk. Izhodna rešitev je zapisana v obliki JSON dokumenta.

Tako načrtovano rešitev smo implementirali z uporabo programskega jezika Python. Pri tem smo si pomagali z različnimi podpornimi knjižnicami: Click za implementacijo ukaznega vmesnika rešitve, luguru za zapisovanje dnevniških zapisov, jsonschema za validacijo vhodnih JSON dokumentov, prettytable za izpis rešitve v tabelarični obliki znotraj ukaznega vmesnika ter knjižnico NiaPy [33], ki vsebuje množico implementiranih algoritmov

delujočih po vzorih iz narave katere smo spridoma pouporabili za reševanje našega problema avtomatskega načrtovanja delovanega časa. Implementirano rešitev smo z uporabo knjižnjice PyInstaller zapakirali v izvedljivo datoteko (.exe), ki vključuje izvajalno Python okolje z vsemi potrebnimi knjižnicami, s čimer smo olajšali izvajanje razvite rešitve znotraj operacijskega sistema Windows.

4.3 Prikaz delovanja in ovrednotenje rešitve

Za namen interakcije z razvito rešitvijo smo razvili vmesnik z ukazno vrstico preko katerega lahko podamo vhodne parametre v skladu s katerimi je nato razvita rešitev pognana. Rešitev podpira tri zagonske parametre (zastavice) in sicer **-f** (ali **--file**) s katero lahko definiramo pot do vhodne JSON datoteke, **-t** (ali **--time**) s katero lahko definiramo maksimalen dovoljen čas izvajanja in **-o** (ali **--output**) s katero lahko definiramo pot kamor naj se shrani končna rešitev podanega problema avtomatskega načrtovanja delovnega časa.

Slika 3 prikazuje zagon razvite rešitve z uporabo vseh treh zagonskih parametrov.

```
.\kdajdelam-schedule.exe --file .\primeri\trgovina.json --time 1 --output rezultat.json
```

Slika 3: Primer zagona razvite rešitve.
Vir: lasten.

Z namenom objektivnega ovrednotenja in vpogledom v delovanje razvite rešitve smo opravili analizo delovanja na treh primerih realnih podjetij. Podjetja so si med seboj različna glede na panogo (veterinarstvo, trgovina, gostinski lokal) kot tudi glede na potrebe načrtovanja delovnega časa. V vseh izvedenih in analiziranih primerih so upoštevane omejitve pri načrtovanju urnika (glej Tabela 2).

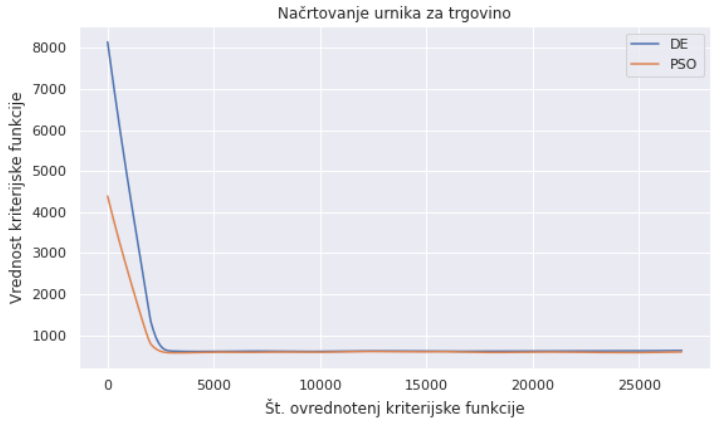
Za vsak analiziran primer smo izvedli dva zagona pri čemer smo v enem zagonu zadano nalogo reševali z uporabo algoritma diferencialne evolucije (angl. Differential Evolution, DE), v drugem pa z uporabo algoritma optimizacije z rojem delcev (angl. Particle Swarm Optimization, PSO).

4.3.1 Primer načrtovanja delovnega časa za trgovino

Primer načrtovanja delovnega časa za trgovino je najenostavnejši izmed vseh treh. Trgovina ima 8 zaposlenih in obratuje vse 6 dni v tednu od 8.00 – 6.00, torej 22 ur na dan. S tega vidika je ta primer še posebej specifičen. Načrt delovnega časa se je v tem primeru načrtoval za obdobje od 6.4.2020 do 11. 4.2020, torej za 1 delovni teden. Tabela 3 prikazuje najbolje oblikovan načrt delovnega časa ob uporabil algoritma DE in PSO. Iz tabele je razvidno, da je rešitev z uporabo algoritma DE oblikovala bolj optimalen načrt delovnega časa, saj je ta dosegel manj kazenskih točk kot tisti načrtovan z uporabo algoritma PSO. Poleg skupnega števila kazenskih točk, so iz tabele razvidne tudi dodeljene kazenske točke za posamezno kršenje omejitve. Kot lahko vidimo, sta oba algoritma bila enako kaznjena za kršitev omejitve za nepravilno vrsto dela (80 točk, kar se po tabeli kazenskih točk pretvori v 8 ur), pri ostalih pa pride do očitnega razhajanja v prid algoritmu DE.

Tabela 3: Prikaz kršitev najboljše načrtovanega delovnega časa za trgovino

Tip kazni	DE	PSO
Kazen za nepravilno vrsto dela	80 (8 ur)	80 (8 ur)
Kazen za prekoračitev delovne izmene	0	0
Kazen za nepravilno delovno mesto	0	50 (10 ur)
Kazen za kršenje dnevnega počitka	120 (6 ur)	320 (16 ur)
Kazen za kršenje tedenskega počitka	0	0
Kazen za umestitev zaposlenega v drugo poslovno enoto	0	0
Skupaj kazni	200	450



Slika 4: Potek zmanjševanja vrednosti kriterijske funkcije (kazni) pri načrtovanju urnika za trgovino.
Vir: lasten.

Ker je reševanje tako kompleksnih problemov lahko pogosto časovno zahtevna naloga, je pogosto bolj pomembno dobiti dovolj uporabno rešitev v čim krajšem času, kot pa dobiti malenkost boljšo rešitev vendar je za to potrebno veliko več časa. Slika 4 prikazuje potek optimizacije avtomatskega načrtovanja delovnega časa z algoritmom DE in PSO. Na *x* osi so števila ovrednotenj kriterijske funkcije na *y* osi pa vrednost te pri posameznem ovrednotenju. Iz vrednosti v začetni fazi optimizacije je razvidno, da so rešitve algoritma PSO bolj optimalne kot rešitve algoritma DE. Vidimo lahko, da oba algoritma največji del optimizacije načrta delovnega časa opravita v začetnih 2500 ovrednotenjih kriterijske funkcije, kar s stališča pridobitve čim boljše rešitve v čim krajšem času, nobenemu izmed algoritmov ne prinaša posebne prednosti. Na dolgi rok algoritem DE sicer uspe malenkostno izboljšati rešitev v primerjavi z algoritmov PSO, vendar ne občutno.

Slika 5 prikazuje primer izpisa tedenskega načrta dela ustvarjenega s pomočjo razvite rešitve ob uporabi optimizacijskega algoritma DE. Izpis je zgolj demonstracijske narave, končna rešitev pa je izvožena v obliki JSON dokumenta, ki omogoča nadaljno strojno obdelavo in poljuben izpis.

Monday (06.04)	Tuesday (07.04)	Wednesday (08.04)	Thursday (09.04)	Friday (10.04)	Saturday (11.04)
[08:00–16:00]:Zaposleni 1	[08:00–16:00]:Zaposleni 1	[08:00–16:00]:Zaposleni 4	[08:00–16:00]:Zaposleni 8	[08:00–16:00]:Zaposleni 4	[08:00–16:00]:Zaposleni 4
[08:00–16:00]:Zaposleni 2	[08:00–16:00]:Zaposleni 2	[08:00–16:00]:Zaposleni 5	[08:00–16:00]:Zaposleni 6	[08:00–16:00]:Zaposleni 5	[08:00–16:00]:Zaposleni 5
[08:00–16:00]:Zaposleni 3	[08:00–16:00]:Zaposleni 5	[08:00–16:00]:Zaposleni 2	[08:00–16:00]:Zaposleni 5	[08:00–16:00]:Zaposleni 2	[08:00–16:00]:Zaposleni 2
[14:00–20:00]:Zaposleni 4	[14:00–20:00]:Zaposleni 8	[14:00–20:00]:Zaposleni 8	[14:00–20:00]:Zaposleni 1	[14:00–20:00]:Zaposleni 8	[08:00–16:00]:Zaposleni 6
[14:00–20:00]:Zaposleni 5	[14:00–20:00]:Zaposleni 3	[14:00–20:00]:Zaposleni 1	[14:00–20:00]:Zaposleni 2	[14:00–20:00]:Zaposleni 6	[14:00–20:00]:Zaposleni 1
[20:00–06:00]:Zaposleni 6	[20:00–06:00]:Zaposleni 6	[14:00–20:00]:Zaposleni 3	[14:00–20:00]:Zaposleni 3	[14:00–20:00]:Zaposleni 1	[14:00–20:00]:Zaposleni 8
[20:00–06:00]:Zaposleni 7	[20:00–06:00]:Zaposleni 7	[14:00–20:00]:Zaposleni 6	[14:00–20:00]:Zaposleni 7		[14:00–20:00]:Zaposleni 3
	[20:00–06:00]:Zaposleni 6				[14:00–20:00]:Zaposleni 7

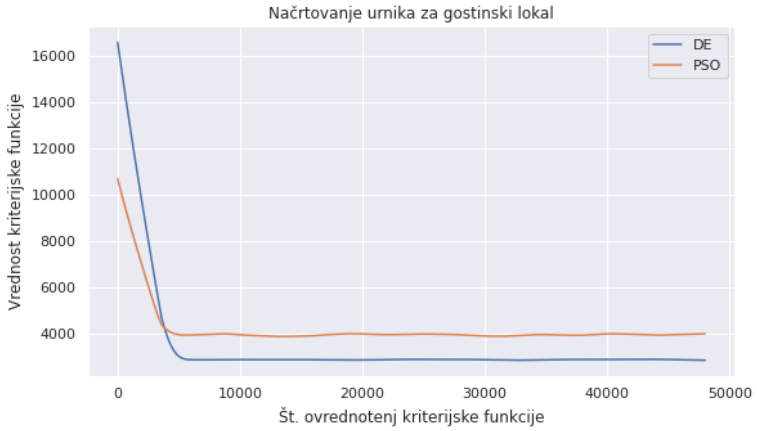
Slika 5: Primer avtomatsko načrtovanega delovnega časa za en teden ob uporabi algoritma DE.
Vir: lasten.

4.3.2 Primer načrtovanja delovnega časa za gostinski lokal

Gostinski lokal ima skupno 6 zaposlenih in obratuje vse dni v tednu od 7.00 do 22.00 ure. Načrt delovnega časa se je načrtoval za obdobje med 30.12.2019 in 2.2.2020 (5 polnih tednov) kar lahko pretvorimo v 150 mest v katere je bilo potrebno dodeliti 6 zaposlenih. Specifika pri načrtovanju urnika za ta primer je prekrivanje posameznih izmen dela ter različne dolžine posameznih izmen. Tabela 4 prikazuje najboljši načrt delovnega časa ob uporabi algoritma diferencialne evolucije in algoritma optimizacije z rojem delcev. Iz tabele je razvidno, da je algoritem DE pri enakih pogojih uspel načrtovati optimalnejši delovni čas, saj je seštevek kazni manjši kot pri rešitvi pridobljeni z uporabo algoritma PSO.

Tabela 4: Prikaz kršitve najboljše načrtovanega delovnega časa za gostinski lokal

Tip kazni	DE	PSO
Kazen za nepravilen tipa dela	1405 (140,5 ur)	1400 (140 ur)
Kazen za prekoračitev delovne izmene	0	0
Kazen za nepravilno delovno mesto	0	0
Kazen za kršenje dnevnega počitka	210 (10,5 ur)	380 (19 ur)
Kazen za kršenje tedenskega počitka	0	0
Kazen za umestitev zaposlenega v drugo poslovno enoto	0	0
Skupaj kazni	1615	1780



Slika 6: Potek zmanjševanja vrednosti kriterijske funkcije (kazni) pri načrtovanju urnika za gostinski lokal. Vir: lasten.

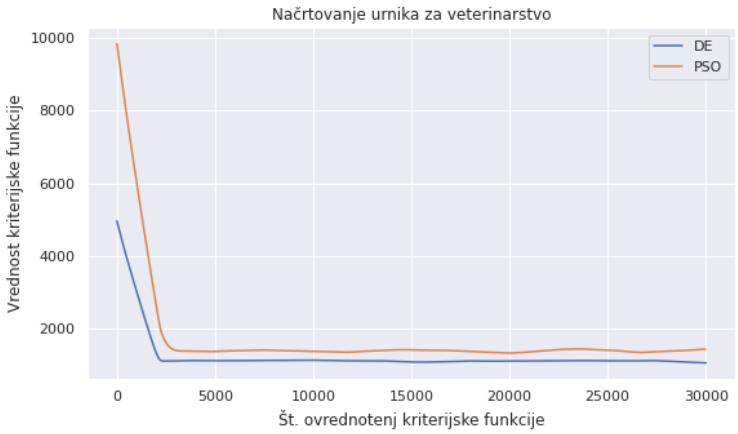
Iz Slika 6 lahko razberemo da sicer algoritem PSO v začetni fazi delovanje načrtuje bolj optimalne rešitve kot algoritem DE, vendar oba glavnino optimizacije opravita do približno 5000. ovrednotenja kriterijske funkcije. Posledično sta, gledano iz vidika pridobitve čim boljše rešitve v čim krajšem času, praktično enakovredno. Iz krivulij je razvidno, da v nadaljevanju DE algoritem načrtuje bolj optimalne rešitve kot algoritem PSO.

Primer načrtovanja delovnega časa za **za veterinarstvo**Tretji primer je primer načrtovanja delovnega časa za veterinarstvo. Veterinarstvo ima 5 zaposlenih in posluje od ponedeljka do petka med 8.00 do 20.00. Načrt delovnega časa se je oblikoval za obdobje med 1.4.2020 in 30.4.2020 (22 delovnih dni) kar lahko pretvorimo v 110 mest v katere je bilo potrebno dodeliti 5 zaposlenih. Iz Tabela 5 je razvidno, da je optimalnejši načrt dela sestavil optimizacijski algoritem DE. Sestavljen urnik je bil kaznjen zgolj za eno kategorijo kršitev in sicer je kršil omejitev za prekoračitev delovne izmene v obsegu 600 točk kar se pretvori v 60 ur. Rešitev pridobljena z algoritmo PSO je sicer v tej kategoriji kršitev dosegla manjšo kazen vendar je poleg te pridelala še znatno vsoto kazenski točk zaradi kršenja omejitve dnevnega počitka v obsegu 320 točk oz. 16 ur.

Tabela 5: Prikaz kršitve najboljše načrtovanega delovnega časa za veterinarstvo

Tip kazni	DE	PSO
Kazen za nepravilen tipa dela	0	0
Kazen za prekoračitev delovne izmene	600 (60 ur)	540 (54 ur)
Kazen za nepravilno delovno mesto	0	0
Kazen za kršenje dnevnega počitka	0	320 (16 ur)
Kazen za kršenje tedenskega počitka	0	0
Kazen za umestitev zaposlenega v drugo poslovno enoto	0	0
Skupaj kazni	600	860

Za razliko od poteka zmanjševanja vrednosti kriterijske funkcije v prejšnih dveh primerih je v tem primeru iz grafa razvidno, da so rešitve pridobljene z algoritmod DE skozi celoten potek optimizacije vidno boljše kot tiste pridobljene z uporabo algoritma PSO. Podobno kot pri prejšnjih dveh primerih, je tudi v tem točka kjer je opravljen večji del optimizacije načrta delovnega časa podobna pri obeh algoritmih.



Slika 7: Potek zmanjševanja vrednosti kriterijske funkcije (kazni) pri načrtovanju urnika za veterinarstvo. Vir: lasten.

5 Zaključek

Reševanje industrijskih problemov z uporabo računske inteligence se je in se še vedno izkazuje kot zelo uspešno pri reševanju raznovrstnih problemov. Ne glede na napredek pristopov, metod in tehnik računske inteligence pa se še vedno v veliki meri, pri vpeljavi teh soočamo z številnimi izzivi. Zahtevnost vpeljave nam sicer lajšajo programske knjižnjice in paketi, ki imajo v naprej definiran nabor raznovrstnih algoritmov kar nam omogoča lažjo eksperimentiranje pri snovanju rešitve za naš cilnji problem. V prispevku smo predstavili najpogostejše skupine algoritmov, ki se tipično uporabljajo za naslavljanje industrijskih problemov. Podali smo smernice za izbiro primernih algoritmov ter se dotaknili izzivov s katerimi se pogosto soočamo. Vpeljavo računske inteligence smo prikazali na primeru avtomatksega načrtovanja delovnega časa. Razvita rešitev se izkaže s svojo prilagodljivostjo glede na zahteve, želje in potrebe končnega uporabnika, rešitve načrtov dela, ki jih oblikuje pa so smiselne in izvedljive. Analizirane primere smo izvedli z uporabo dveh različnih optimizacijskih algoritmov, pri čemer se je v vseh primerih algoritem diferencialne evolucije izkazal za uspešnejšega kar še enkrat več potrjuje njegovo vsesplošno uporabnost.

Literatura

[1] Fister, I. (2017). Algoritmi računske inteligence za razvoj umetnega športnega trenerja. Univerza v Mariboru (Slovenia).
[2] Blum, C., & Merkle, D. (Eds.). (2008). Swarm intelligence: introduction and applications. Springer Science & Business Media.
[3] Terano, Toshiro, Kiyoji Asai, and Michio Sugeno. Fuzzy systems theory and its applications. Academic Press Professional, Inc., 1992.
[4] Engelbrecht, A. P. (2007). Computational intelligence: an introduction. John Wiley & Sons.
[5] Eiben, A. E., & Smith, J. E. (2003). Introduction to evolutionary computing (Vol. 53, p. 18). Berlin: springer.
[6] Dasgupta, Dipankar. "Advances in artificial immune systems." IEEE computational intelligence magazine 1.4 (2006): 40-49.
[7] FISTER, Iztok, 2019, Avtomatsko načrtovanje in vrednotenje klasifikacijskih cevovodov v bioinformatiki [na spletu]. Univerza v Mariboru, Fakulteta za zdravstvene vede.

[8] Fister Jr, I., Yang, X. S., Fister, I., Brest, J., & Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. arXiv preprint arXiv:1307.4186.

[10] Hornby, Gregory, et al. "Automated antenna design with evolutionary algorithms." Space 2006. 2006. 7242.

[11]Ghoddusi, Hamed, Germán G. Creamer, and Nima Rafizadeh. "Machine learning in energy economics and finance: A review." Energy Economics 81 (2019): 709-727.

[12] Tayarani-N, Mohammad-H., Xin Yao, and Hongming Xu. "Meta-heuristic algorithms in car engine design: A literature survey." IEEE Transactions on Evolutionary Computation 19.5 (2014): 609-629.

[13] Nikolos, Ioannis K., Eleftherios S. Zografos, and Athina N. Brintaki. "UAV path planning using evolutionary algorithms." Innovations in intelligent machines-1. Springer, Berlin, Heidelberg, 2007. 77-111.

[14] Gao, Shangce, et al. "A state-of-the-art differential evolution algorithm for parameter estimation of solar photovoltaic models." Energy Conversion and Management 230 (2021): 113784.

[15] Saavedra-Moreno, B., et al. "Seeding evolutionary algorithms with heuristics for optimal wind turbines positioning in wind farms." Renewable Energy 36.11 (2011): 2838-2844.

[16] Jursa, René, and Kurt Rohrig. "Short-term wind power forecasting using evolutionary algorithms for the automated specification of artificial intelligence models." International Journal of Forecasting 24.4 (2008): 694-709.

[17] Podgorelec, Vili, et al. "Digital Transformation Using Artificial Intelligence and Machine Learning: An Electrical Energy Consumption Case." International Conference "New Technologies, Development and Applications". Springer, Cham, 2022.

[18] Fister, Iztok, Iztok Fister Jr, and Dušan Fister. Computational intelligence in sports. Cham: Springer, 2019.

[19] Pena-Reyes, Carlos Andrés, and Moshe Sipper. "Evolutionary computation in medicine: an overview." Artificial Intelligence in Medicine 19.1 (2000): 1-23.

[20] Slowik, Adam, and Halina Kwasnicka. "Evolutionary algorithms and their applications to engineering problems." Neural Computing and Applications 32.16 (2020): 12363-12379.

[21] Odili, Julius, et al. "A comparative evaluation of swarm intelligence techniques for solving combinatorial optimization problems." International Journal of Advanced Robotic Systems 14.3 (2017): 1729881417705969.

[22] Zhang, Shuzhu, et al. "Swarm intelligence applied in green logistics: A literature review." Engineering Applications of Artificial Intelligence 37 (2015): 154-169.

[30] Gondro, Cedric, and B. P. Kinghorn. "Application of evolutionary algorithms to solve complex problems in quantitative genetics and bioinformatics." Guelph: University of Guelph (2008).

[31] Tzanetos, Alexandros, Iztok Fister Jr, and Georgios Dounias. "A comprehensive database of Nature-Inspired Algorithms." Data in brief 31 (2020): 105792.

[32] Wolpert, David H., and William G. Macready. "No free lunch theorems for optimization." IEEE transactions on evolutionary computation 1.1 (1997): 67-82.

[33] Vrbančič G, Brezočnik L, Mlakar U, Fister D, Fister I. NiaPy: Python microframework for building nature-inspired algorithms. Journal of Open Source Software. 2018 Mar 22;3(23):613.

Strojno učenje za boljše javne storitve: zgodnja identifikacija iztokov iz omrežja pitne vode

Sašo Karakatič[1], Špela Pečnik[1], Grega Vrbančič[1], Rok Kukovec, Matej Levstek[2], Aleš Erker[2], Vili Podgorelec[1]

[1] Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
saso.karakatic@um.si, spela.pecnik@um.si, grega.vrbancic@um.si, rok.kukovec@um.si, vili.podogrelec@um.si

[2] Mariborski vodovod, javno podjetje, d.o.o., Maribor, Slovenija
matej.levstek@mb-vodovod.si, ales.erker@mb-vodovod.si

Sinopsis Pitna voda je redek in dragocen vir, zato je ustrezno upravljanje tega vira bistveno za družbeni in gospodarski razvoj vsake države ali regije, saj je čista pitna voda uporabljena v vseh ključnih sektorjih gospodarstva, kot so kmetijstvo, industrija, energetika in promet. Pametne naprave so spremenile skoraj vse vidike našega življenja – in tudi pri upravljanju vodovodnega omrežja ni nič drugače. Moderni vodovodni sistemi so že opremljeni z napravami za sprotno beleženje pretokov skozi posamezne odseke vodovodnega omrežja. Prav tako so pogosti tudi sistemi, ki informacije o pretokih v obliki nadzornih sistemov prikazujejo v realnem času in s tem omogočajo spremljanje celotnega omrežja v vsakem trenutku. Še vedno pa za odkrivanje nepravilnosti v vodovodnem omrežju skrbijo bodisi človeški kontrolorji, ali pa primitivni sistemi na osnovi pravil. Pri človeških kontrolorjih je problem, da sta njihova čas in pozornost omejena. Prav tako pa je kontrolorja veščega identifikacije iztokov potrebno dodobra izobraziti. Obstoječi avtomatski sistemi pa so preveč primitivni, da bi zaznali kaj več kot nenadne ogromne iztoke – manjši iztoki, ki na dolgi rok naredijo veliko škode, pa ostanejo neodkriti. V tem članku je predstavljen sistem za zgodnje odkrivanje iztokov, ki temelji na tehnikah napredne analize podatkov in strojnem učenju. Sistem, ki bo opisan temelji na različnih tehnikah, kot so gručenje podobnih odsekov omrežnega sistema ter napoved pričakovane porabe vode določenega področja glede na letni čas in porabo v ostalih podobnih odsekih. Prvi rezultati kažejo, da je sistem zelo koristno orodje, ki kontrolorjem pomaga pri identifikaciji iztokov, ter te samodejno išče tudi v času, ko človeški kontrolorji niso na delovnem mestu.

Ključne besede:
Strojno učenje
Iztoki pitne vode

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022

1 Uvod

Pitna voda je naravni vir in dobrina, katere pomen in potreba v svetu stalno narašča, zato učinkovito upravljanje celotnega vodovodnega sistema, od vodnih virov do končnih odjemalcev, postaja ključnega pomena. Izgubljena voda pomeni nepotrebno obremenjevanje vodnih virov in nepotrebno porabo električne energije, potrebne za črpanje in distribucijo vode do odjemalcev. Vodne izgube je potrebno zniževati in nadzorovati, saj pomenijo prihranek na vodnih virih z racionalizacijo stroškov in investicij ter zmanjšanje posegov v okolje in prostor v trajnostnem razvoju sistema. Predstavljajo izziv v vseh vodovodnih sistemih, v industrijskih območjih pa povzročajo tudi izpad proizvodnih procesov. V svetu se zaradi zastarele vodovodne infrastrukture izgubi v povprečju od 25 % do 50 % vode. V državah Evropske unije se ti deleži gibljejo od 2 % pa tudi do 60 %, v nekaterih evropskih državah pa tudi do 85 %. Učinkovito zmanjševanje vodnih izgub je tako eden glavnih izzivov vseh upravljalcev vodovodnih sistemov.

Mednarodna združenja za vodo in njihove smernice priporočajo zamenjavo najmanj dveh procentov najbolj dotrajane vodovodne infrastrukture na leto. Glede na trend v zadnjih letih vedno manjših vlaganj v vodovodno infrastrukturo in posledično manjšega števila rekonstrukcij cevovodov, tega cilja nismo dosegali, zato vodne izgube predstavljajo vedno večji izziv. Stremimo k temu, da stanje vodovodnega sistema, tudi z uporabo in preizkušanjem novih tehnologij in pristopov, optimiziramo.

Dodaten izziv predstavlja pomanjkanje zadostnih količin primernih kadrovskih virov, ki bi se ukvarjali z aktivnim pregledom iztekanj na terenu in z njihovim lociranjem. Rezultati novih tehnologij bi tako raziskovalce na terenu lahko dodatno obremenili, po drugi strani pa pripomogli pri učinkovitejših oz. natančneje usmerjenih pregledih.

1.1 Predlagana rešitev s strojnim učenjem

Identifikacije iztokov iz vodovodnega omrežja navadno poteka, kar ročno preko pregleda vzorcev porabe pitne vode posameznih merilnih področij. Je pa tak ročen pregled problematičen, saj se večji iztoki ugotovijo prepozno, manjši pa so mnogokrat spregledani. Hkrati pa se mnogokrat najde lažen iztok vode, ki nepotrebno preobremeni ekipe na terenu. Zato je efektivna identifikacija spretnost, ki jo operater mora razviti (največkrat dolgotrajen postopek ob obilici pomoči izkušenejših operaterjev in ekip na terenu), da se mu ne izmuznejo iztoki in, da ne sproži preveč lažnih alarmov.

Tako delo, identifikacija iztokov iz vodovodnega omrežja, je tako idealno situirano za avtomatizacijo, saj je delo monotono, ter zahteva obilo spretnosti in izkušenj. Do določene mere so postopki avtomatizacije identifikacije iztokov že vpeljani v raznorazne sisteme nadzora vodovodnega omrežja. Ampak največkrat je avtomatska identifikacija implementirana le v obliki preprostih pravil, ki so zapisana v stilu: »če je poraba ta teden dvakrat večja od porabe prejšnjega tedna, sproži alarm«. Taka naivna pravila sprožijo preveč lažnih alarmov in izpustijo preobilico pravih iztokov. Posledično so ob poplavi takih nepravilnih signalov naivni avtomatski sistemi operaterjem le v breme in ne v pomoč. [1]

Prav z namenom nadgradnje sistemov avtomatske identifikacije iztokov iz vodovodnega omrežja, je potrebno najti alternativne implementacije, ki ne delujejo po naivnih pristopih. Algoritmi strojnega učenja so ena izmed potencialnih rešitev, ki lahko pomagajo pri tem je »inteligentnejši« identifikaciji iztokov. Namreč, ne temeljijo na naivnih pravilih, ki bi jih vnaprej definirali ljudje (strokovnjaki ali programerji) ampak omogočajo sprotno prilagajanje pravil, hkrati pa imajo sposobnost delovanja po vzorcih iz vodovodnega omrežja, ki jih z naivnimi pravili ni možno opisati. [1]

V tem prispevku je opisan sistem, ki prav z uporabo algoritmov strojnega učenja uspe efektivno identificirati iztoke iz vodovodnega omrežja pitne vode, ter s tem pomaga operaterjem sprejemati boljše odločitve za sprožitev alarmov ter posledično bolj efektivno porabo časa ekip na terenu.

2 Iztoki pitne vode

Več kot 1.672 km razvejanega vodovodnega omrežja nas zavezuje k odgovornemu vzdrževanju celotnega sistema. Nepredvideni dogodki (prelomi cevi, puščanja…), ki povzročajo iztok oz. vodne izgube se večinoma pojavljajo zaradi dotrajane vodovodne infrastrukture. V analizah izgub vode sledimo smernicam mednarodnih združenj IWA (International Water Association), AWWA (American Water Works Association) in WHO (World Health Organization), ki obravnavajo obvladovanje vodnih izgub z upoštevanjem metod v analizi in kontroli sistema. V ta namen se tudi na območju RS teži k spremljanju vodne bilance po metodologiji IWA, kar se zahteva tudi po Uredbi o oskrbi s pitno vodo (Ur.l. 88/2013). Iz vodne bilance je razvidno, da je vtok v sistem enak vsoti delov prodane in neprodane vode, ki je vsota deležev:

- Neobračunane avtorizirane porabe (razlika med odčitki na števcih in prodano vodo).
- Navideznih izgub, ki so delno:
  - Neavtorizirana poraba (priključki na črno ali javna raba) in
  - posledica nenatančnih meritev (slaba merilna mesta oz. vodomeri, neustrezna kvaliteta, slaba proizvodnja, vzdrževanje ali dimenzioniranje) in
- Dejanskih izgub vode, ki so odraz stanja vodovodnega sistema oz. omrežja z vodooskrbnimi objekti in se pojavljajo na:
  - Vodih surove vode in sistemih za obdelavo vode,
  - transportnih in razdelilnih vodih in
  - priključkih do merilnega mesta.

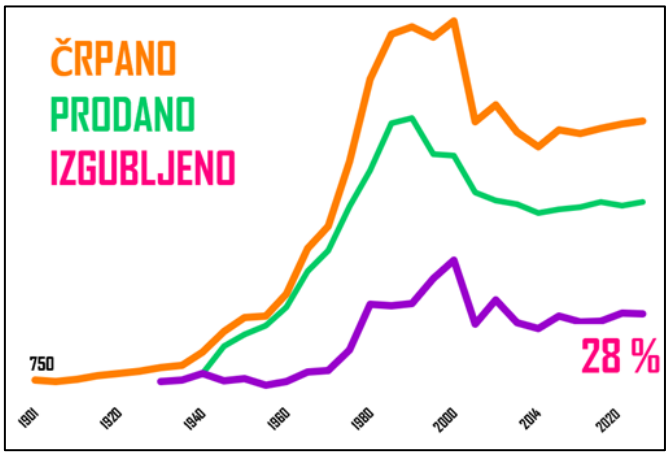
Metodologija IWA priporoča uporabo različnih načinov za analizo vodnih izgub (izgube na odjemalca, izgube na km cevovoda, UARL, CARL, ILI...). Zaradi lažjega prikaza in primerjave z drugimi sistemi se po izvedbi vodne bilance določijo obstoječe letne dejanske vodne izgube (CARL – Current Annual Real Losses). Za zmanjševanje dejanskih vodnih izgub je pomembno ugotavljati tudi neizogibne letne izgube (UARL – Unavoidable Annual Real Losses), ki predstavljajo minimalne podzemne izgube, katere je skoraj nemogoče odkriti z uporabo običajnih metod in naprav. Za ugotavljanje uspešnosti reševanja dejanskih izgub v vodooskrbnem sistemu se uporabi kazalnik iztekanja ILI (Infrastructure Leakage Index), ki predstavlja razmerje med obstoječimi dejanskimi letnimi izgubami in neizogibnimi letnimi izgubami in prikazuje, kako dobro je sistem vzdrževan. Skladno s tem je sistem Mariborskega vodovoda smiselno razdeljen po blokih:

- Blok I: Mestna občina Maribor in občine Miklavž, Hoče - Slivnica in Duplek,
- Blok II: Občini Ruše in Selnica ob Dravi,
- Blok III: Občine Pesnica, Kungota, Šentilj, Lenart, Sveta Ana v Slov. goricah, Sv. Jurij v Slov. goricah, Sv. Trojica v Slov. goricah, Benedikt, Cerkvenjak in Gornja Radgona.

Tabela 1: Kazalnik ILI in delež izgubljene vode v letu 2021.

Opis	Črpano	V omrežje [m3]	Prodano [m3]	Vodne izgube [m3]	Dej.izgube CARL [m3]	UARL [m3]	Vodne izgube [%]	ILI
BLOK I	10.638.182	10.546.383	7.440.228	3.106.156	3.012.971	602.785	29,5	5,0
BLOK II	650.623	644.804	489.870	154.934	150.286	70.385	24	2,1
BLOK III	2.551.280	2.538.187	1.950.856	587.331	569.712	423.272	23,1	1,3
SKUPNO	13.840.085	13.729.374	9.880.953	3.848.421	3.732.969	1.116.036	28	3,3

Kazalnik ILI nam daje zadovoljiv podatek, da je sistem kot celota dobro vzdrževan. Kaže, da so vodne izgube v sistemu kot celoti, dobro upravljane, vendar obstaja potencial za izboljšave. Stanje sistema na področju Blokov II in III je zadovoljivo, nadaljnje zmanjševanje izgub na tem delu je lahko ekonomsko neupravičeno, posebno pozornost pa je potrebno nameniti področju Bloka I. Višji kazalnik ILI pripisujemo predvsem dejstvu, da je to urbano območje večje poselitve z večjo gostoto ter več gospodarskimi/industrijskimi odjemalci, kar posledično pomeni večji odjem vode na km cevovoda. Stremimo k temu, da sistem čim bolje vzdržujemo s sistemsko analizo in vlaganji tako v infrastrukturo, kot v razvoj ter s tem znižujemo vodne izgube ter porabo električne energije.



Slika 1: Količina črpane, prodane in izgubljene vode po letih.
Vir: lasten.

»Dejanske izgube« so rezultat slabega stanja in okvar na omrežju in kažejo na potrebo po investicijah v omrežje. Izgube so posledice okvar na ceveh, hišnih priključkih in armaturah. Nujna je opredelitev strukture izgub z določitvijo dejanskih vodnih izgub. Te je potrebno analizirati v bilanci vode znotraj obravnavanih območij (občine, bloki in zaključena merilna območja oz. DMA) na osnovi celovitega informacijskega sistema, centralne baze podatkov, programske opreme in strokovnih služb.

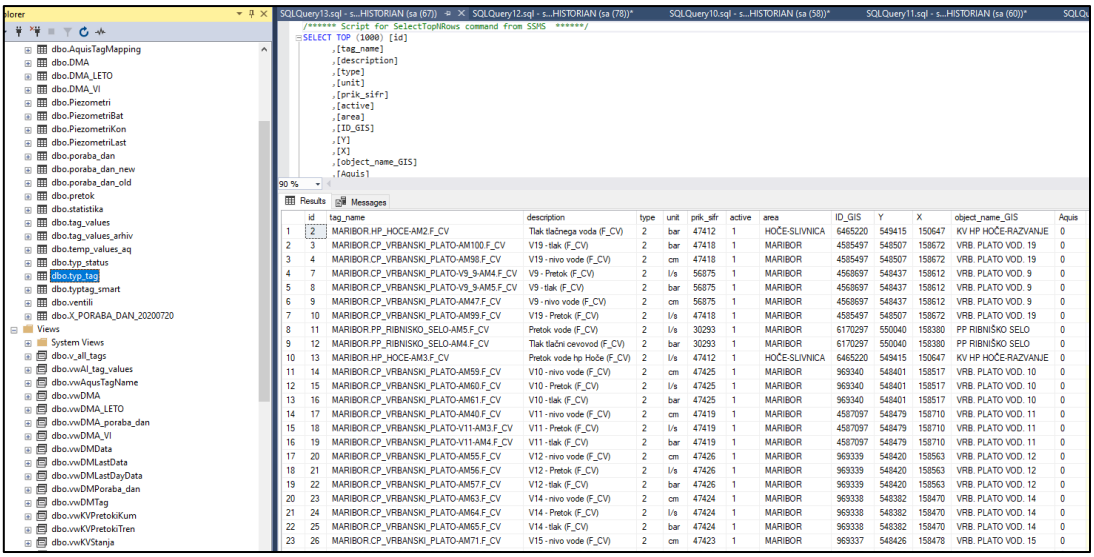
»Navidezne izgube« obvladujemo v sklopu sistematične zamenjave vodomerov na merilnih mestih pri odjemalcih in na merilnih točkah na omrežju s sodobnejšimi in natančnejšimi. Zamenjava poteka za območje celotnega sistema, sočasno z uvedbo daljinskega odčitavanja vodomerov, vendar je dinamika in realizacija vezana in odvisna od razpoložljivih sredstev. Od leta 2012 je potekala zamenjava vodomerov vedno intenzivneje. Danes imamo na sistemu že več kot 99 % vodomerov na daljinsko odčitavanje.

Zniževanje izgub na spojnih vodih (priključnem omrežju) je vezano na program zamenjave vodomerov. Program zamenjav vodomerov poteka v ciklusu petih let. Nadzorstvo nad izgubami vode za obvladovanje izgub, ob zamenjavi vodomera predvideva sočasen pregled priključka z ročnim detektorjem izgub. Slednje zahteva letni pregled 8.000 priključkov.

Zniževanje vodnih izgub je mogoče le ob izpolnjevanju trajnostne in razvojne naravnanosti izvajanja dejavnosti oskrbe s pitno vodo. Program zniževanja vodnih izgub temelji na sodobni informacijski tehnologiji, sodobnem sistemu nadzora in vodenja proizvodnih procesov in distribucije pitne vode in preizkušanju novih pristopov.

2.1 Dosedanji način iskanja iztokov

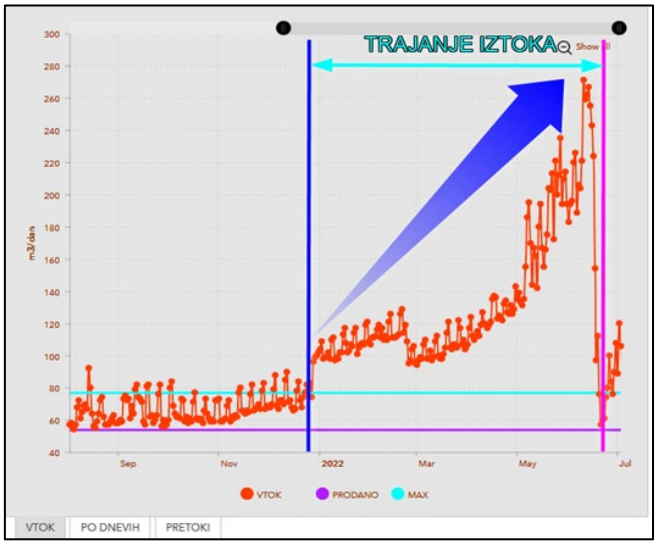
Mariborski vodovod ima za namene stalnega nadzora vodovodnega sistema vzpostavljeno centralno bazo podatkov, v kateri so zbrane vse meritve v sistemu, tako iz sistema daljinskega upravljanja (SDU), kot tudi podatki kontrolnih merilnikov z GSM/GPRS prenosom. Za namene iskanja nepredvidenih dogodkov v omrežju, ki povzročajo vodne izgube, se obdelujejo podatki o količini vode, trenutnih pretokih in tlakih. S celotnega vodovodnega sistema tako v bazi zberemo in obdelamo več kot 36.000.000 posameznih meritev na leto.



	id	tag_name	description	type	unit	prik_sifr	active	area	ID_GIS	Y	X	object_name_GIS	Aquis
1	2	MARIBOR.HP_HOCE-AM2.F_CV	Tlak tlačnega voda (F_CV)	2	bar	47412	1	HOČE-SLIVNICA	6465220	549415	150647	KV HP HOČE-RAZVANJE	0
2	3	MARIBOR.CP_VRBANSKI_PLATO-AM100.F_CV	V19 - tlak (F_CV)	2	bar	47418	1	MARIBOR	4585497	548507	158672	VRB. PLATO VOD. 19	0
3	4	MARIBOR.CP_VRBANSKI_PLATO-AM98.F_CV	V19 - nivo vode (F_CV)	2	cm	47418	1	MARIBOR	4585497	548507	158672	VRB. PLATO VOD. 19	0
4	7	MARIBOR.CP_VRBANSKI_PLATO-V9_9-AM4.F_CV	V9 - Pretok (F_CV)	2	l/s	56875	1	MARIBOR	4568697	548437	158612	VRB. PLATO VOD. 9	0
5	8	MARIBOR.CP_VRBANSKI_PLATO-V9_9-AM5.F_CV	V9 - tlak (F_CV)	2	bar	56875	1	MARIBOR	4568697	548437	158612	VRB. PLATO VOD. 9	0
6	9	MARIBOR.CP_VRBANSKI_PLATO-AM47.F_CV	V9 - nivo vode (F_CV)	2	cm	56875	1	MARIBOR	4568697	548437	158612	VRB. PLATO VOD. 9	0
7	10	MARIBOR.CP_VRBANSKI_PLATO-AM99.F_CV	V19 - Pretok (F_CV)	2	l/s	47418	1	MARIBOR	4585497	548507	158672	VRB. PLATO VOD. 19	0
8	11	MARIBOR.PP_RIBNISKO_SELO-AM5.F_CV	Pretok vode (F_CV)	2	l/s	30293	1	MARIBOR	6170297	550040	158380	PP RIBNIŠKO SELO	0
9	12	MARIBOR.PP_RIBNISKO_SELO-AM4.F_CV	Tlak tlačni cevovod (F_CV)	2	bar	30293	1	MARIBOR	6170297	550040	158380	PP RIBNIŠKO SELO	0
10	13	MARIBOR.HP_HOCE-AM3.F_CV	Pretok vode hp Hoče (F_CV)	2	l/s	47412	1	HOČE-SLIVNICA	6465220	549415	150647	KV HP HOČE-RAZVANJE	0
11	14	MARIBOR.CP_VRBANSKI_PLATO-AM59.F_CV	V10 - nivo vode (F_CV)	2	cm	47425	1	MARIBOR	969340	548401	158517	VRB. PLATO VOD. 10	0
12	15	MARIBOR.CP_VRBANSKI_PLATO-AM60.F_CV	V10 - Pretok (F_CV)	2	l/s	47425	1	MARIBOR	969340	548401	158517	VRB. PLATO VOD. 10	0
13	16	MARIBOR.CP_VRBANSKI_PLATO-AM61.F_CV	V10 - tlak (F_CV)	2	bar	47425	1	MARIBOR	969340	548401	158517	VRB. PLATO VOD. 10	0
14	17	MARIBOR.CP_VRBANSKI_PLATO-AM40.F_CV	V11 - nivo vode (F_CV)	2	cm	47419	1	MARIBOR	4587097	548479	158710	VRB. PLATO VOD. 11	0
15	18	MARIBOR.CP_VRBANSKI_PLATO-V11-AM3.F_CV	V11 - Pretok (F_CV)	2	l/s	47419	1	MARIBOR	4587097	548479	158710	VRB. PLATO VOD. 11	0
16	19	MARIBOR.CP_VRBANSKI_PLATO-V11-AM4.F_CV	V11 - tlak (F_CV)	2	bar	47419	1	MARIBOR	4587097	548479	158710	VRB. PLATO VOD. 11	0
17	20	MARIBOR.CP_VRBANSKI_PLATO-AM55.F_CV	V12 - nivo vode (F_CV)	2	cm	47426	1	MARIBOR	969339	548420	158563	VRB. PLATO VOD. 12	0
18	21	MARIBOR.CP_VRBANSKI_PLATO-AM56.F_CV	V12 - Pretok (F_CV)	2	l/s	47426	1	MARIBOR	969339	548420	158563	VRB. PLATO VOD. 12	0
19	22	MARIBOR.CP_VRBANSKI_PLATO-AM57.F_CV	V12 - tlak (F_CV)	2	bar	47426	1	MARIBOR	969339	548420	158563	VRB. PLATO VOD. 12	0
20	23	MARIBOR.CP_VRBANSKI_PLATO-AM63.F_CV	V14 - nivo vode (F_CV)	2	cm	47424	1	MARIBOR	969338	548382	158470	VRB. PLATO VOD. 14	0
21	24	MARIBOR.CP_VRBANSKI_PLATO-AM64.F_CV	V14 - Pretok (F_CV)	2	l/s	47424	1	MARIBOR	969338	548382	158470	VRB. PLATO VOD. 14	0
22	25	MARIBOR.CP_VRBANSKI_PLATO-AM65.F_CV	V14 - tlak (F_CV)	2	bar	47424	1	MARIBOR	969338	548382	158470	VRB. PLATO VOD. 14	0
23	26	MARIBOR.CP_VRBANSKI_PLATO-AM71.F_CV	V15 - nivo vode (F_CV)	2	cm	47423	1	MARIBOR	969337	548426	158478	VRB. PLATO VOD. 15	0

Slika 2: Centralna baza podatkov.
Vir: lasten.

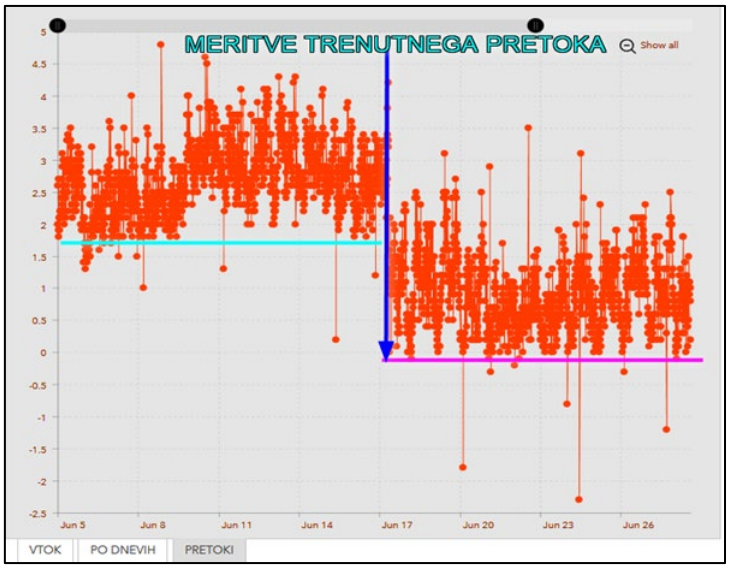
Celotno vodovodno omrežje je dolgo več kot 1.672 km, zato ga je kot celoto nemogoče učinkovito obvladovati. Iz tega razloga omrežje delimo na *»zaključena merilna območja«* (District Metering Area - DMA), v katerih na dnevni ravni merimo količine vtokov, iztokov in trenutne pretoke, jih preračunamo in primerjamo s povprečno količino prodane vode. Trenutno imamo vzpostavljenih 136 takšnih merilnih območij. Na ta način dobimo seznam območij z največjimi razlikami med dobavljeno in povprečno prodano vodo (m3), na podlagi katerega se prioritetno določajo območja pregledov omrežja. Preračuni so narejeni na dnevni ravni. Podatke vode v DMA tudi grafično vizualiziramo in dnevno spremljamo trende. V primeru sprememb trenda, najprej preverimo morebitna redna vzdrževalna dela in nato po potrebi ukrepamo, saj nam povišani trendi po vsej verjetnosti nakazujejo nepredviden iztok vode, ki se pojavi zaradi lomov cevi, puščanj na spojnih elementih ali priključkih končnih odjemalcev.



Slika 3: Primer začetka puščanja in sanacije.
Vir: lasten.

Z vedno boljšimi podatki kontrolnih vodomerov, smo aplikacijo razvili tudi s preračuni po trenutnih pretokih (l/s), na 15 minutni ravni. Vsi vodomeri nam meritev trenutnih pretokov ne omogočajo, zato je zaključenih merilnih območij, ki jih spremljamo po trenutnem pretoku, nekoliko manj, kot tistih, ki jih spremljamo po dobavljeni količini vode, izraženi v dnevnem volumnu (m3). Pri iskanju vodnih izgub so med najpomembnejšimi kazalci

meritev minimalnega nočnega pretoka (Night Line Measurement – NLM) med 2:00 in 4:00 uro, ko je odjem vode najmanjši. Na ta način precej natančneje ocenimo količino izgubljene vode v določenem zaključenem območju.



Slika 4: Meritev trenutnega pretoka – puščanje in normalizacija po odpravi defekta.
Vir: lasten.

Območjem s povišano količino dobavljene vode sledi raziskava na terenu oziroma *»aktivna kontrola iztoka«*. Za namen raziskav vodovodnega omrežja na terenu imamo dve ekipi raziskovalcev, ki sta opremljeni s sodobno opremo za natančno lociranje puščanj. Raziskovalci uporabljajo geofon, senzorje šuma in korelatorje, s katerimi je možno natančno locirati prelom cevi oz. puščanje.

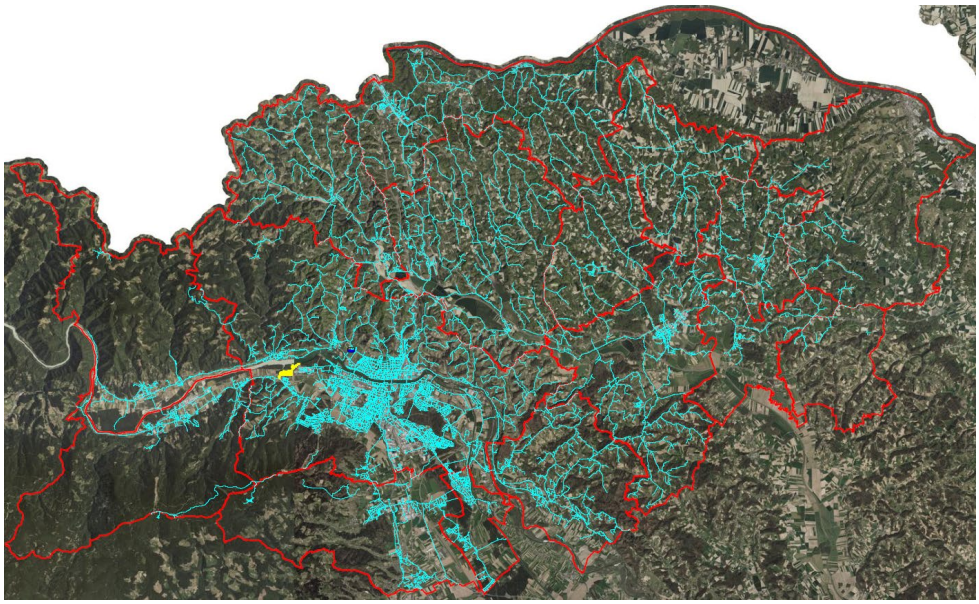
Senzorji šuma se zelo učinkovito obnesejo pri iskanju okvar, saj se zaradi večjega števila (komplet senzorjev) lahko pokrije večji del omrežja, ponekod celotno zaključeno merilno območje. Na litoželeznih, jeklenih in pocinkanih ceveh senzorji delujejo zelo dobro tudi na večjih razdaljah, medtem ko na ceveh iz PVC niso tako zanesljivi, prav tako jih je potrebno montirati bližje.

Slika 5: Loggerji šuma se z magnetom pripnejo na vodovodno cev/priključek/ventil.
Vir: lasten.

Senzorji šuma delujejo samostojno, njihova nadgradnja pa so senzorji šuma z možnostjo korelacije oz. korelatorji. Iskanje okvare je z njimi še učinkovitejše, saj omogočajo analizo šumov med posameznimi senzorji, izvedbo korelacije in s tem poenostavijo postopek lociranja puščanja. Po izvedbi korelacije tako v primeru defekta natančneje določimo razdaljo defekta od posameznega korelatorja.

3 Obstoječa infrastruktura

Vodovodni sistem Mariborskega vodovoda je največji enovit sistem za oskrbo z vodo v Sloveniji in letno načrpa približno 14.000.000 m3 vode. Obsega 1.672 km omrežja, 49 vodnjakov, 130 zbiralnikov vode, 134 prečrpalnih postaj, 7.770 hidrantov in 50.061 vodomerov. Fizično povezuje in združuje skoraj 200.000 prebivalcev severovzhodne Slovenije. V celoti ali vsaj delno s pitno vodo oskrbuje odjemalce v 16 občinah in sicer iz Mestne občine Maribor, Občine Ruše, Selnica ob Dravi, Hoče-Slivnica, Miklavž na Dravskem polju, Duplek, Pesnica, Šentilj, Kungota, Lenart, Sveta Ana, Benedikt, Sveta Trojica v Slovenskih goricah, Sveti Jurij v Slovenskih goricah, Cerkvenjak in Gornjo Radgono.



Slika 6: Vodovodni sistem je dolg 1.672 km in oskrbuje 16 občin na področju SV Slovenije.
Vir: lasten.

Celotna infrastruktura je nacionalnega pomena za dolgoročno ohranjanje zdravja in naravnih virov. Sistem, ki je izpostavljen tako obsežni infrastrukturi mora delovati tako, da ohranja kakovost vode, kot najpomembnejši vir življenja in stalno zmanjšuje stroške vzdrževanja in izgube vode zaradi dotrajanosti ali prelomov. Zraven vodovodne infrastrukture vsebuje »celovit informacijski sistem podjetja«, ki povezuje tako tehnični sistem za podporo upravljanja omrežja, kot tudi poslovni informacijski sistem, namenjen točni evidenci vseh poslovnih dogodkov na omrežju in sistemu, obračunu storitev in podporo vsem drugim procesom v podjetju.

Geografski informacijski sistem povezuje vodovodno omrežje in objekte ter drugo opremo v evidenci celotnega sistema, s tehničnim elektronskim arhivskim sistemom, opremljenim z digitaliziranimi načrti in skicami omrežja.

V procesih črpanja in distribucije vode se uporablja sodobna tehnologija in tehnika, ki se povezuje v sistem daljinskega nadzora in upravljanja vodooskrbnih objektov in omrežja. Tehnologija omogoča spremljanje in optimiziranje pretokov, tlakov in posledično zniževanje vodnih izgub. Sistem je v procesu nenehnega posodabljanja, v smislu vključevanja vedno več objektov in kontrolnih točk omrežja v daljinsko upravljanje.

Meritve z vodovodnih objektov se zbirajo v centralni bazi podatkov, ki zajema podatke sistema daljinskega upravljanja (SDU), kot tudi podate kontrolnih merilnikov z GSM/GPRS prenosom.

Zbrane podatke dnevno analiziramo in jih vizualiziramo v nadzorni aplikaciji »WaterLoss MBV«, ki povezuje »Centralno bazo meritev« in »geografski informacijski sistem«. Zajema prikaz vseh 136 zaključenih merilnih območij (DMA), s prikazom področja, karakteristik omrežja (število priključkov, število priključenih oseb, dolžino omrežja, povprečen tlak), grafični prikaz z dnevnimi vtoki in primerjavo med dobavljeno in povprečno prodano vodo, na podlagi katerih se določajo prioritetna območja pregledov omrežja. Aplikacija omogoča tudi analizo zaključenih merilnih območij po minimalnih nočnih pretokih območjih z nameščenimi vodomeri, ki to omogočajo.

Odločitve sprejemamo tudi na podlagi trendov dobavljene količine vode – nenadna povišana količina dobavljene vode je razlog za alarm in posledično raziskavo omrežja na terenu oz. lociranje defekta (prelom cevi, puščanje).

Slika 7: Aplikacija za nadzor zaključenih merilnih območij »WaterLoss MBV«.
Vir: lasten.

4 Strojno učenje za iskanje iztokov

Iskanje iztokov iz vodovodnega omrežja bi po laičnem pregledu lahko izvajali ročno in postopoma ob vizualnem pregledu porabe pitne vode. Vsak nenaden dvig porabe pitne vode bi verjetno nakazoval pojav izpusta vode. Pa je res temu tako? Kaj pa, če je razlog za nenaden dvig vode v resnici zunanji pojav, ki odraža dejansko nenadno povečanje porabe pitne vode? Nekaj takih pojavov se redno in periodično dogaja tekom celotnega leta. En tak primer je, da ob prvem skoku temperatur v koncu pomladi ali začetku poletja nekateri domači porabniki, ki predstavljajo posameznike, ki živijo v hišah, začnejo s polnjenjem domačih bazenov iz vodovodnega omrežja pitne vode. Drugi primer je, da ob dolgotrajnem pojavu suše zalivajo domači vrt kar z vodo iz vodovodnega omrežja.

Prav tako je zmotno mišljenje, da pri iztokih vode pride do nenadnega in velikega dviga porabe pitne vode. Nekateri iztoki se začnejo z neznatnim povečanjem porabe, ki ga ni možno zaznati niti ročno, niti z nobeno tehniko strojnega učenja. Se pa tako povečanje porabe akumulira na dolgi rok. Če je iztok stalen in v res majhnih količinah, je tak iztok skoraj nemogoče zaznati, saj lahko predstavlja le spremembo pri porabniku (DMA-ju): mogoče rojstvo novega otroka, novo domačo žival ali novo napravo, ki porabi nekaj vode. Če pa se majhni iztoki postopoma povečujejo (kot je mnogokrat primer v realnosti), pa je tak konstanten dvig možno zaznati.

Seveda taki dvigi porabe niso pokazatelj iztoka iz vodovodnega omrežja. Nepremišljena reakcija interventne ekipe na terenu bi take ekipe le preobremenila. Prav zato je pomembno, da se za identifikacijo izlivov uporabijo naprednejše tehnike, kot pa le gol in laičen pregled porabe pitne vode.

Kot že namigujejo prejšnji odstavki, je identifikacija iztokov močno povezana s kvalitetno napovedjo porabe pitne vode. Če vemo napovedati kakšna bi morala biti poraba v določenem obdobju, lahko pregledamo tudi razliko med predvideno porabo in dejansko porabo. Vzorci razlike med tema vrednostma skozi določeno časovno obdobje pa povedo, če je razlika skladna z iztoki, ali pa gre za kakšen drugi pojav (mogoče se je v prazno hišo končno naselila družina).

4.1 Napovedovanje porabe

Edini podatek, ki ga z zadovoljivo mero zanesljivosti lahko zagotovimo v skorajda celotnem omrežju Mariborskega vodovoda je dejanska poraba porabnikov na določenem DMA-ju v časovnem rezu. V trenutnem sistemu se podatki o porabi v DMA-ju beležijo v petnajst minutnih intervalih. Analiza premajhnih intervalov lahko s seboj prinese preveč šuma: bodisi zaradi napak v merjenju, ali zaradi nenavadnih vzorcev porabe, ki niso povedni za napovedovanje. Posledično je analiza na tako majhnih intervalih neprimerna in je potrebna agregacija porabe v večje časovne intervale. V našem primeru so se enourni intervali izkazali za primerne – niso predolgi, saj so vzorci porabe čez dan razvidni, in niso prekratki, saj ni preveč šuma. [3]

Pri napovedovanju porabe pitne vode imamo izziv, ki mu pravimo univariatna analiza časovnih vrst. Namreč, podatki o pretekli porabi tvorijo podatke v strukturi časovne vrste. Ker pa imamo opravek le z eno vrednostjo, tj. dejansko porabo v časovnem rezu, imamo opravek z univariatno analizo.

Pri napovedovanju gibanja časovnih vrst se uporabljajo številne, dobro poznane tehnike, ki so bile implementirane v tem eksperimentu in katerih opisi sledijo v nadaljevanju.

4.1.1 Drseče povprečje

Metoda **drsečega povprečja** (angl. *moving average* ali *rolling mean*) je klasična metoda analize univariatnih časovnih vrst, kjer se za vsak časovni korak izračuna trend, ki temelji na preteklih podatkih. Ta metoda se mnogokrat uporablja tudi pri glajenju gibanja časovnih vrst in prav to glajenje laično obravnavamo kot trend gibanja časovne vrste. [4]

Izračun drsečega povprečja se naredi s pomočjo konvolucijskega filtra, ki za določeno točko v času pregleda k prejšnjih časovnih korakov in za vrednosti v teh k korakih izračuna povprečje. Formalna definicija izračuna vrednosti drsečega povprečja za najnovejši časovni korak je sledeča.

$$DP_n = \frac{v_{n-k+1} + v_{n-k+2} + \cdots + v_n}{k} = \frac{1}{k} \sum_{i=n-k+1}^{n} p_i$$

Kjer v_n predstavlja vrednost v najnovejšem časovnem koraku, v_{n-k+1} predstavlja vrednost za k korakov nazaj, vmes pa so vse vrednosti med tema korakoma. Vrednost k predstavlja velikost okna konvolucijskega filtra. Večje ko je okno filtra, bolj gladka bo krivulja, ampak hkrati se lahko manjši vzorci gibanja spregledajo. Manjše ko je okno filtra, manj gladka je krivulja, ampak bolj je izračun dovzeten za šume, ki predstavljajo lažne vzorce.

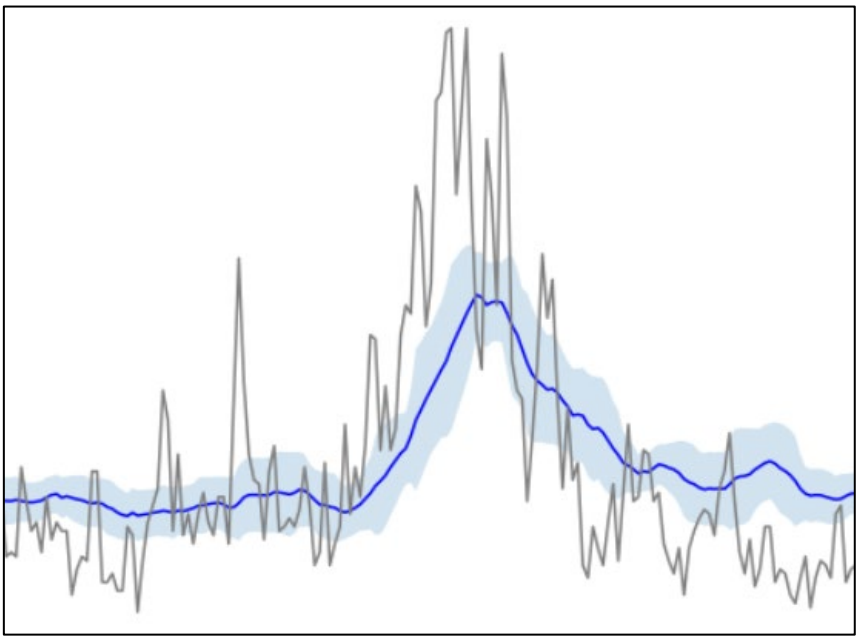
Iz preprostega drsečega povprečja lahko izpeljemo številne različice. Ena izmed pogostih je vsekakor **uteženo drseče povprečje**, kjer novejši podatki prispevajo večji delež k izračunu povprečja, kot pa starejši. To uteževanje je lahko linearno ali celo eksponentno (v tem primeru govorimo o eksponentnem uteženem drsečem povprečju).

Poleg drsečega povprečja lahko izračunamo tudi druge drseče opisno statistične vrednosti. Zelo prav nam pride tudi **drseč standardni odklon** (ali druga mera variabilnosti), ki nam kaže kako velika volatilnost je bila v oknu konvolucijskega filtra. Slika 14 prikazuje drseče povprečje in drseč standardni odklon na podatkih porabe pitne vode. [4]

4.1.2 Dekompozicija časovnih vrst

Ko pa imamo za en DMA izračunan splošni vzorec porabe (bodisi z drsečim povprečjem ali kakšno drugo metodo), pa lahko nadaljujemo z postopkom, ki mu pravimo dekompozicija časovnih vrst. Od dejanske porabe pitne vode lahko odštejemo trend in ostane nam poraba, ki se ne sklada s splošnim vzorcem. Pravimo, da smo časovno vrsto razdelili na dva sestavna dela oz. na dve kompoziciji. Prav zato postopku pravimo dekompozicija.

Ni nujno, da se ustavimo tukaj, lahko z dekompozicijo nadaljujemo: ostanek ponovno s konvolucijo zgladimo, da dobimo vzorec porabe, ki predstavlja razliko od splošnega vzorca. Ta korak smo naredili le zaradi glajenja, ne za iskanje novega vzorca. Tokrat pa razlika med zglajenim ostankom in dejansko porabo predstavlja izredne nenavadnosti oz. celo nenadne iztoke pitne vode. [5]



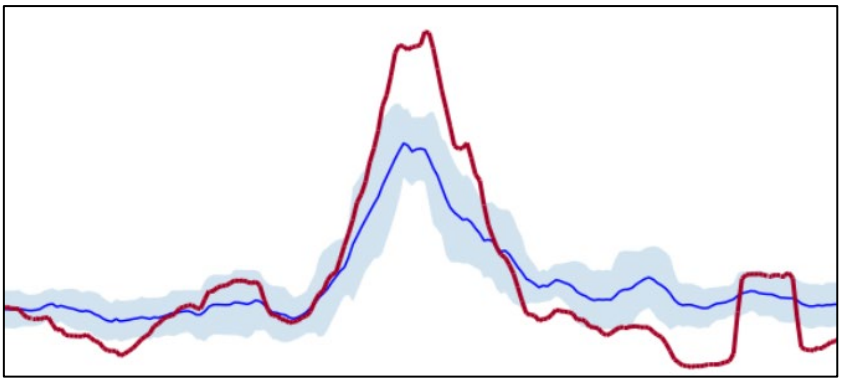
Slika 8: Drseče povprečje in drseč standardni odklon.
Vir: lasten.

V primeru, da imamo podatke preko več časovnih period (leto, mesec, teden…), lahko prvo komponento splošnega vzorca izračunamo celo le z enim DMA mestom. Zaradi spreminjajoče se narave porabe vode posameznih DMA-jev in nenehno spreminjajočega se vodovodnega omrežja, analiza podatkov čez daljše obdobje (čez obdobje dveh let) ni smiselno, saj pridemo do pojava premika podatkov (angl *data shift*) (na katerega vplivajo bodisi premiki kovariatov ali premiki koncepta), ko začnejo že naučeni modeli identifikacije iztoka delovati slabše.

4.1.3 Splošni vzorci porabe pitne vode

Taka drseča povprečja in standardne odklone lahko za več časovnih vrst tudi agregiramo – najlažje kar s povprečenjem preko dimenzije časa. Ta trik smo uporabili tudi pri implementaciji našega sistema, da smo dobili splošen vzorec porabe celotnega območja Mariborskega vodovoda.

Prav primerjava splošnega vzorca porabe pitne vode in vzorca porabe pitne vode enega DMA-ja pa predstavlja pomembno informacijo o odstopanju porabe posameznega DMA-ja napram pričakovani porabi. Prav razlika med vzorcem splošne porabe in porabo enega DMA-ja pa predstavlja pomemben indikator, ki ga lahko interpretiramo kot odstopanje, ki zna nakazovati na nevarnost iztoka pitne vode na posameznem DMA-ju. Razlika med splošno porabo in porabo posameznega DMA-ja je na spodnji sliki 15 prikazana v obliki obarvane črte vzorca porabe.



Slika 15: Splošen trend porabe pitne vode celotnega območja Mariborskega vodovoda (modro), ter trend porabe pitne vode enega DMA-ja (črno).
Vir: lasten.

Bolj ko je linija temna, večje odstopanje porabe pitne vode za izbran DMA opažamo in s tem je tudi večja nevarnost za potencialni iztok. Velja tudi obratno, bolj ko je svetla črta, bolj je poraba pitne vode izbranega DMA-ja skladna s porabo celotnega območja Mariborskega vodovoda in manjša je nevarnost za iztok na izbranem DMA-ju.

Seveda, neprimerno je interpretirati vsak manjšo obarvanje črte trenda posameznega DMA-ja. Čeprav vse uporabljene metode agregacije oz. izračuna trenda do določene mere odstranijo šume, so ti še vedno lahko prisotni. Namreč, prevelika robustnost na šume (odstranjevanje vseh nihanj) bi lahko izbrisala dejanske vzorce. Prav zato pa se interpretirajo le odtenki črt na nekoliko daljše časovno obdobje, ne le nekaj dnevna potencialna nevarnost za iztok. Po prvih preizkusih je interpretacija stalne nevarnosti v zadnjih petih dneh že dober pokazatelj nevarnosti za iztok, kar je vsekakor krajše časovno obdobje, kot pa mora preteči od iztoka po klasičnem ročnem sistemu iskanja iztokov.

4.2 Gručenje podobnih porabnikov

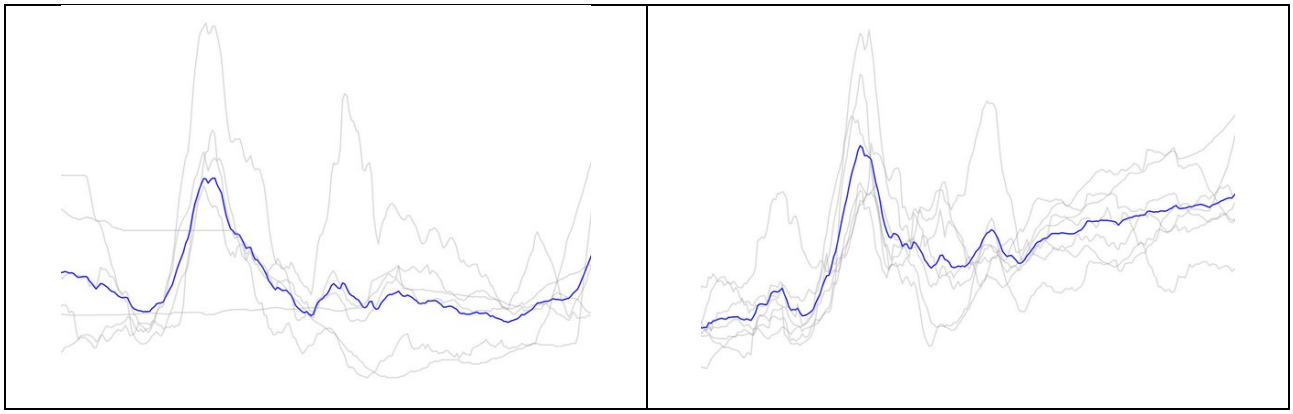
Vsekakor pa primerjava porabe vsakega DMA-ja s porabo na celotnem območju Mariborskega Vodovoda ni najbolj pravilna. Namreč, vsa območja si niso enakovredna po porabi. Nekatera območja (npr. center) so preplavljena z mešanimi porabniki – tako domačimi porabniki v hišah, kot stanovanjih, med katere se vmešajo tudi storitvene dejavnosti. To je vsekakor v nasprotju z industrijskimi območji (npr. Melje) ali območji, kjer prevladujejo porabniki, ki stanujejo le v hišah. Ker Mariborski Vodovod pokriva nekoliko širše območje, kot le mestno občino, se nekateri porabniki ukvarjajo tudi s kmetijskimi dejavnostmi. Vsak tak tip porabnika pitne vode ima nekoliko sebi specifične vzorce porabe. V industrijskih conah v času kolektivnih dopustov in v splošnem izven delovnega časa beležijo manjšo porabo pitne vode. Porabniki, ki živijo v družinskih hišah in imajo nekaj zelenice si ob prvem pojavu vročih dni največkrat polnijo domače bazene kar s pitno vodo. Prav tako pa območja s pretežno domačimi porabniki doživljajo bodisi dvige čez vikend (če družine preživijo te doma), čez popoldan (po službi), ter spuste čez praznike ali celo vikende (če te preživijo drugje kot doma) in ob nedoločenem času tudi čez poletje (ko gredo na dopust). Vsi ti dvigi in spusti porabi pitne vode so del vsakdanjega življenja in se ne smejo zamenjevati za nenavadnosti ali celo za pokazatelj iztoka pitne vode. [6]

Prav zaradi tega pa je tudi nesmiselno primerjati porabo posameznega DMA-ja s prav celotnim območjem Mariborskega vodovoda. Da res ugotovimo, če so nenadni dvigi ali spusti porabe nekaj pričakovanega ali pa gre za nekaj nenavadnega, bi bilo prav, da posamezne DMA-je primerjamo z območji, ki so v splošni porabi temu DMA-ju podobni – pri tem pa ignoriramo tiste, ki so v splošni porabi pitne vode popolnoma drugačni.

V namen iskanja podobnih pa se lahko uporabi tehnika strojnega učenja, ki se imenuje gručenje. Z gručenjem se iz podatkov najde vnaprej definirano število gruč DMA-jev, za katere velja sledeče:

- Tisti DMA-ji, ki so v isti gruči, imajo podobne profile porabe pitne vode, ter

– tisti DMA-ji, ki so v drugih gručah, se v profilu porabe pitne vode razlikujejo.



Slika 16: Dve različni gruči DMA-jev glede na trende porabe pitne vode.
Vir: lasten.

Gručenje podobnih časovnih vrst lahko poteka na več načinov. Za namen našega sistema sta bila implementirana dva. Prvi pristop je **k-mean gručenje za časovne vrste**, kjer se med posameznimi časovnimi koraki izračuna absolutna razdalja med porabami. Seštevek teh razdalj prikazuje razliko med porabama dveh DMA-jev. Cilj je, da se v isto gručo podajo taki DMA-ji, a je ta seštevek razdalj med njimi manjši napram seštevkom razdalja do DMA-jev v drugih gručah. Slika 16 prikazuje primer rezultata takega gručenja.

Drugi pristop, ki smo ga implementirali za namen gručenja, pa je gručenje s pomočjo **evolucijskega algoritma** s pomočjo NiaPy knjižnice [2]. Gručenje merilnih območij izvedemo na način, da v prvem koraku posamezna merilna območja naključno razporedimo v n gruč. V naslednjem koraku izračunamo povprečen standardni odklon porabe vode po dnevih za merilna območja znotraj posamezne gruče ter vrednosti standardnih odklonov združimo z uporabo pristopa utežene vsote. Utežene vsote vseh gruč seštejemo in to vrednost v nadaljevanju uporabimo kot cenitveno funkcijo (angl. *fitness function*) pri uporabi algoritma jDE [7], temelječega na diferencialni evoluciji s katerim skušamo poiskati takšno razporeditev merilnih območij v n gruč, da bo vrednost cenitvene funkcije čim manjša. Algoritem izvede 5000 iteracij, najboljšo najdeno razdelitev merilnih območij, to je rešitev, ki je dosegla najnižjo vrednost cenitvene funkcije pa uporabimo kot končno rešitev.

5 Zaključek

Identifikacija iztokov pitne vode je vedno bolj pereč problem ob vse pogostejših pojavih hudih sušnih obdobij, predvsem v poletnih mesecih. Zanesljivost identifikacije pri ročnem pregledu podatkov ali čakanju na namige o iztoku iz okolja, prinaša preveč nepredvidljivosti in nepovratne škode na okolje, družbo in finance lokalnih skupnosti. Prav zato je predlagan sistem za samodejno identifikacijo s pomočjo algoritmov strojnega učenja zelo primeren za uporabo ob obstoječih sistemih nadzora vodovodnih sistemov.

S prispevkom smo prikazali kako lahko prav zastavljen sistem učinkovito pomaga kontrolorjem pri identifikaciji iztokov in s tem na daljši rok ohrani vire pitne vode kar se da učinkovito porabljene. Vsekakor pa tak sistem ne nadomešča človeške odločitve za posredovanje na terenu – je le pomoč, ki v obliki »človek nad strojem« (angl. *human over the loop*) pomaga pri odločitvah. Za še učinkovitejše in dobro integrirane sisteme pa so seveda potrebne še nadaljnje raziskave – tako algoritmov strojnega učenja za zanesljivejšo in hitrejšo identifikacijo iztokov, kakor za izkoriščanje še več podatkov, ki bi lahko tako identifikacijo naredilo še bolj zanesljivo.

Literatura

[1] ADEDEJI Kazeem B., YSKANDAR Hamam, BOLANLE Tolulope Abe, in ADNAN M. Abu-Mahfouz »Towards achieving a reliable leakage detection and localization algorithm for application in water piping networks: An overview.« v IEEE Access, peta številka 2017, str. 20272-20285.

[2] VRBANČIČ Grega, BREZOČNIK Lucija, MLAKAR Uroš, FISTER Dušan, in FISTER Iztok »NiaPy: Python microframework for building nature-inspired algorithms«, b Journal of Open Source Software številke 3.23, 2018, str. 613.

[3] LIM Bryan in STEFAN Zohren »Time-series forecasting with deep learning: a survey. « v Philosophical Transactions of the Royal Society, številke 2194 v 2021.

[4] HANSUN Seng »A new approach of moving average method in time series analysis.«, v zborniku konference 2013 conference on new media studies (CoNMedia), 2013, str. 1-4.

[5] DAGUM Estela Bee »Time series modeling and decomposition.« v Statistica, številka 70 v 2010, str. 433-457.

[6] AGHABOZORGI Saeed, ALI Seyed Shirkhorshidi, in TEH Ying Wah »Time-series clustering–a decade review.« v Information systems, številka 53 v 2015, str. 16-38.

[7] BREST Janez, Greiner Sašo, BOSKOVIC Borko, MERNIK Marjan in ZUMER Viljem »Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems«, v IEEE Transactions on evolutionary computation št. 6, leta 2006, str. 646-657.

OTS 2022 Sodobne informacijske tehnologije in storitve

Zbornik petindvajsete konference, Maribor, 7. in 8. september 2022

Marjan Heričko, Tina Beranič (ur.)

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
marjan.hericko@um.si, tina.beranic@um.si

Povzetek V zborniku petindvajsete konference OTS 2022 so objavljeni prispevki strokovnjakov s področja informatike, v katerih so predstavljene izkušnje in, skozi konkretne projekte, dokazano uspešni pristopi in dobre prakse naslavljanja arhitekturnih izzivov, povezanih z vpeljavo umetne inteligence in strojnega učenja, zasnove in vpeljave rešitev poslovne inteligence, učinkovite obdelave velepodatkov in podatkovnih tokov, apliciranja vzorcev pri zasnovi mikrostoritvenih in brezstrežniških arhitektur, uporabe zmožnosti tehnologij veriženja blokov in decentraliziranih aplikacij, digitalne preobrazbe in posodobitve IKT rešitev na osnovi funkcionalnosti, ki jih omogočajo digitalne denarnice, apliciranja agilnih metod ter izboljšanjem odzivnosti pri zagotavljanju učinkovite podpore poslovnim procesom, integracije sistemov in razvoja podpornih storitev pametnega doma, telemedicine in zelenega prehoda, razvoja in optimizacije mobilnih, spletnih in oblačnih rešitev ter uporabe sodobnih in aktualnih, tudi odprtokodnih, tehnologij pri razvoju naprednih informacijskih rešitev in storitev.

Ključne besede:
informatika
informacijske tehnologije
programsko inženirstvo
informacijske rešitve
digitalna preobrazba
razvoj mobilnih in spletnih rešitev
poslovna inteligenca
umetna inteligenca in strojno učenje
obdelava velepodatkov in podatkovnih tokov
agilne metode
tehnologije veriženja blokov
kibernetska varnost

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

Beyond Now

bintegra
Svetovanje in IT rešitve

COMTRADE GAMING

databox

impol
Aluminium Industry

INEA

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.
INFORMATIKA

MIKROpis

msg life ODATEAM

niceHash

NovaKBM

NOVUMRGI

OPERATO
member of ELES Group

Pošta Slovenije

RAZVOJNI CENTER IKTS ŽALEC

RC IRC Celje, d.o.o. / Ltd.

SRC

tridens
technology

3fs

MEDIJSKA POKROVITELJA

Računalniške novice
www.racunalniske-novice.com

VEČER

Grow your Career with us

Beyond Now.

Beyond Now is an international leading ecosystem orchestration and digital platform provider, powering organizations to launch new services at speed and grow revenue in an era of cloud, IoT, AI and 5G by utilizing our digital platform and SaaS BSS.

We're building our team with the best people, and that's where YOU come in:

- Headquarters in **Graz, London, Dublin, USA**
- Foundation year: **1989**
- **220** people

Check out our open positions – Seniors, Juniors, Interns are welcome

https://beyondnow.hirehive.com/

- SOFTWARE ENGINEEER
- SOFTWARE/SOLUTION ARCHITECT
- IT ADMIN
- PRODUCT OWNER
- SCRUM MASTER PROJECT MANAGER
- CLOUD ENGINEER
- QUALITY ENGINEER

What's in it for you?

- NO CORE TIME HOME OFFICE
- MEAL VOUCHER
- JOB TICKET
- FREE ENTRANCE SCHWARZL-LAKE
- FREE PARKING
- EVENTS & GIFTS
- ONBOARDING & BUDDY
- TRAINING & DEVELOPMENT

See what our team has to say about why they love working at Beyond Now:

"[What's exciting about Beyond Now is] 'Working with bleeding edge technologies - using the latest technical solutions to meet customers' needs.'

(Manuel, Developer)

"'Cliché to say, I know, but it's a really good, relaxed and flexible working environment.'

(Matic, Product Owner)

"'[there are] Opportunities within [the] team to take on responsibilities and grow and succeed together'

(Sandra, Project Manager)

"'You just need the will to learn and to develop and you will find your spot in the company.'

(Nusa, Documentation Engineer turned Quality Engineer)

Check out the way we work!

https://www.beyondnow.com/en/company/careers/work-at-beyond/

We are looking forward to your online application!

Contact:
Angelika Koelbl-Theussl, Katarina Pranjic
T: + 43 3136 207 526, + 43 316 8003 1047
Seering 6, 8141 Premstätten
welcome@beyondnow.com

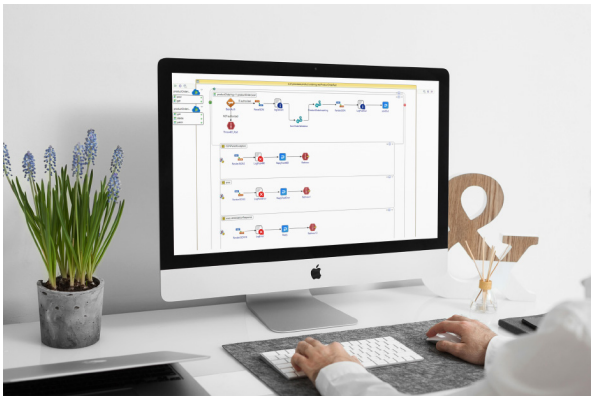
www.beyondnow.com

Apply now!

bintegra

Ponujamo odlično programsko opremo in IT rešitve.

Kadarkoli, kjerkoli in na katerikoli napravi.



Napredne integracijske rešitve

Agilna integracija, ki združuje integracijske tehnologije, agilne tehnike razvoja in platforme v oblaku za izboljšanje hitrosti in varnosti razvoja programske opreme, je ključna podlaga za uspešno digitalno preobrazbo.
Zgradite, uporabite in podprite zunanje in notranje API-je za pospešitev razvoja novih storitev.

Pomagamo vam izbrati pravo tehnologijo, integrirati aplikacije, podatke in informacije za avtomatizacijo procesov in izboljšanje uporabniške izkušnje.

Inovativni uporabniški portali

Skupaj z vami učinkovito rešujemo vaše izzive na naslednjih področjih:

STRANKE » Omogočite svojim strankam, da sami rešijo svoje težave brez pomoči zastopnika v živo. Ponudite jim inovativni samopostrežni portal. Predstavite ustreznejšo vsebino s personalizacijo za vsako skupino strank.

PRIHODEK » Povečajte spletni prihodek s trgovino B2B ali B2C. Ustvarite trajnostne konkurenčne prednosti in izkoristite nove priložnosti za prihodke.

INTERNI PROCESI » Posodobite svoj intranet in s tem povečajte produktivnost za zaposlene v pisarni, na daljavo, zmanjšajte preobremenitev z informacijami in preklapljanje konteksta, izboljšajte izkušnje zaposlenih, ohranite institucionalno znanje in hitro vključite nove zaposlene.

Bihub – 360° upravljanje komunikacije s strankami

Socialni mediji so eden najhitreje rastočih kanalov, ki jih potrošniki uporabljajo za interakcijo z blagovnimi znamkami.

Z rešitvijo **Bihub** se lahko pogovarjate s strankami preko e-pošte in družbenih medijev na enem mestu, lahko pa celotno komunikacijo dodelite tudi drugim kolegom ali ekipam.
Poenostavite lahko upravljanje družbenih medijev in ga uspešno integrirate v svoj CRM. Poskrbite, da ne boste zamudili nobenega vprašanja, potencialne stranke ali pritožbe strank.

www.bintegra.com

info@bintegra.com

COMTRADE GAMING

About

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 15 years of experience.

Comtrade Gaming's strengths are the development of technology solutions based on industry standards and legislative requirements. We are always looking for new talent who wants to join our team of 300+ colleagues. Get more info on our Careers page.

Scan the QR code for more information.

Who are we looking for:

- Developers
- Architects
- Test automation engineers
- System engineers
- Database developers
- Business analysts
- Project managers
- Support engineers
- QA engineers etc.

Technologies

For our projects, we use various technologies. Among most frequently used are **.NET, Angular, MS SQL, Java, PostgreSQL, Cloud solutions, JavaScript, CSS/HTML5 etc.**

databox

Business Analytics Platform

Databox is a cloud-based business analytics platform helping companies around the world track and improve their performance.

Easiest Setup SMALL BUSINESS FALL 2021 · Leader WINTER 2022 · Easiest To Use SMALL BUSINESS 2021

Recognized as Leading Data Visualisation Software on G2 Crowd

We are hiring!

Grow your career in the booming industry of the future.
Check out open positions at databox.com/careers

- Complex challenges
- Top-notch technology
- Customized career path
- 140 members across the globe
- People-first company culture

impol
Aluminium Industry




5x
povečanje obsega prodaje
v zadnjih 30 letih

20 %
80 %
2.500
zaposlenih

400 mio €
investicij v zadnjih
10 letih

5.
največji slovenski
izvoznik

262.500
ton proizvedenih
izdelkov na leto

Trajnostno delovanje
in odgovoren odnos
do zaposlenih




Slovenska Bistrica | Slovenija | www.impol.si

```
a = '507313975456'
N,E = ''
for (i=0; i<len(a); i=i+2):
     N[i/2] = a[i]-1
     E[i/2] = a[i+1]+1
print ("Meet us at "+(N/1e4)+", "+(E/1e4))
```

future@inea.si

Oblikujemo in uresničujemo

celostne rešitve, ki presegajo skupek svojih sestavnih delov - avtomatizirane proizvodne linije, specializirane naprave za industrijski strojni vid, pametne sisteme za upravljanje s porabo in distribucijo energije ter mnoge druge raznolike visokotehnološke izzive.

Podpiramo in omogočamo

z vzajemnim spoštovanjem in temeljitim strokovnim znanjem, ki ga vzpodbujamo in negujemo. Tako kot naše rešitve se tudi mi nenehno izboljšujemo ter nadgrajujemo z višino ambicij in širino izkušenj. Na tej filozofiji temeljijo vsi naši podvigi.

Negujemo in združujemo

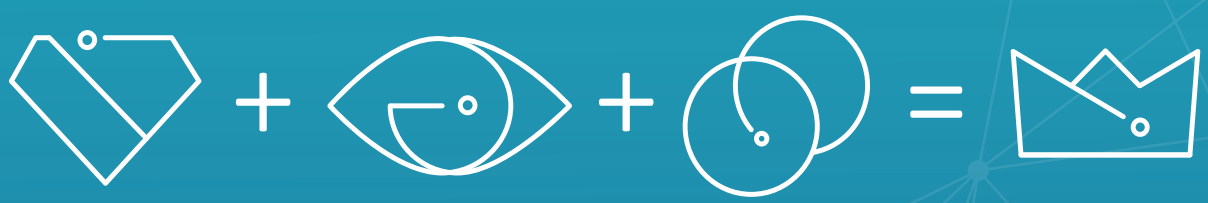
s pristnim odnosom, strastjo in neustavljivim zanosom. Pot od zasnove do rešitve je osnovana na medsebojnem zaupanju, ki ga gradimo s pomočjo robustne strukture lastništva zaposlenih in akademskih razvojno-raziskovalnih povezav.

spoznaj nas

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.

INFORMATIKA

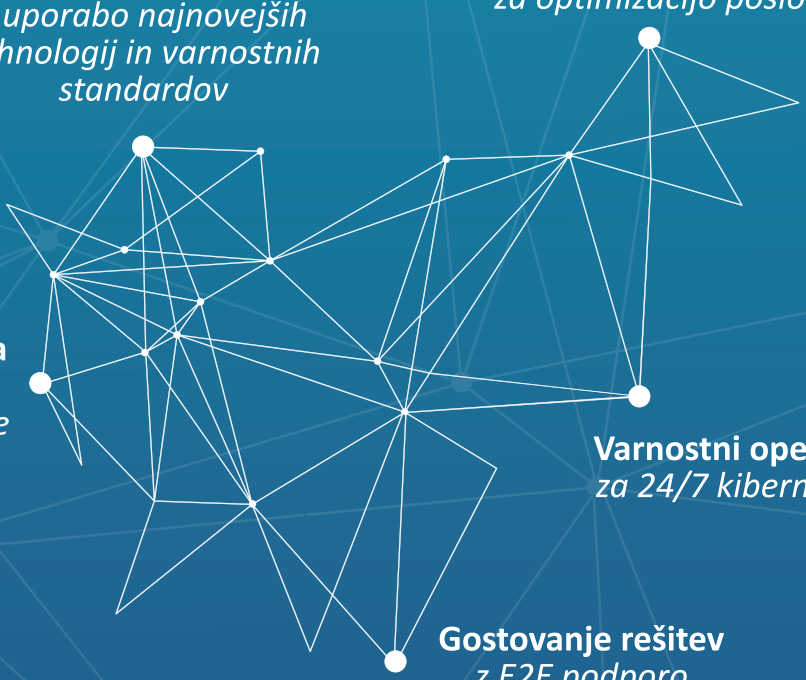
pravi partner za vas



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

Connected mHealth

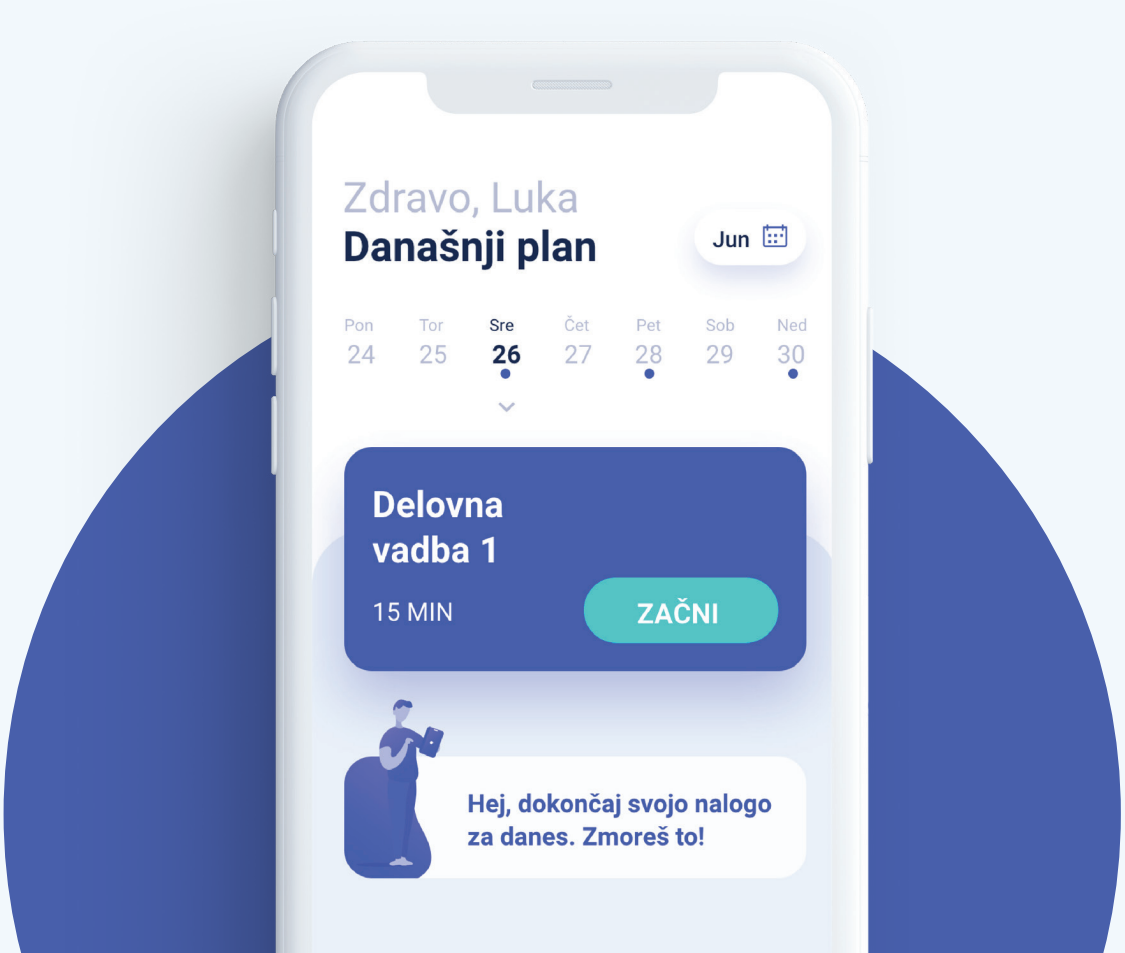
Digitalna platforma leta 2022

Nagrada na AACVPR (American Association of Cardiovascular and Pulmonary Rehabilitation)

Connected mHealth digitalna platforma, razvita za namene fizioterapije / rehabilitacije na daljavo, **bo letos, Septembra 2022, na Floridi, prejela vse-ameriško priznanje skupaj s kliniko Mayo Clinic.** V sklopu raziskave Connected mobile-Health for Rehabilitation at MayoClinic-Arizona, izvedene na kliniki Mayo Clinic so namreč uporabili rešitev Connected mHealth za dodeljevanje vadb oziroma izvajanje rehabilitacije na daljavo in prišli do izjemnih rezultatov, ki bodo imeli velik vpliv na prihodnost fizioterapije / rehabilitacije. Več: https://www.aacvpr.org

connectedmhealth.com

2022 AWARDS



.expertise .innovation .partnership

- razvoj celostnih informacijskih sistemov za zavarovalnice in njihove agente
- rešitve v oblačnih storitvah v mikrostoritveni arhitekturi z uporabo vsebnikov Docker
- digitalizacija procesov
- razvoj mobilnih aplikacij
- agilen razvoj
- več kot 20 let zanesljiv in inovativen partner

msg.Symass

Kontakt:
msg life odateam d.o.o.
Titova cesta 8
SI-2000 Maribor
Tel: + 386 2/ 2356 229
E-Mail: Andrej.Kline@msg-life.com

www.msg-life.si

ZA NOVE STRANKE
9
MESECEV
BREZPLAČNO
VODENJE

POVEŽITE NAJPOMEMBNEJŠE BANČNE STORITVE ZA ODLIČNO CENO

v Paketu Komplet.

Zmagovalna kombinacija storitev vključuje **vodenje računa**, **najboljšo spletno in mobilno banko v Sloveniji**, **nižjo obrestno mero** ob sklenitvi stanovanjskega kredita ter še več ugodnosti.

Sklenite Paket Komplet prek mobilne banke mBank@Net in **izkoristite ugodno ponudbo zdaj**.

NovaKBM

080 17 70
nkbm.si/paket-komplet

NOVUM RGI

www.novum-rgi.com

ENTERPRISE JAVA APPLICATION DEVELOPMENT

AGILE SOFTWARE DEVELOPMENT

INTERNATIONAL PROJECTS

IT SOLUTIONS FOR INSURANCE BUSINESS

CLOUD SERVICES FOR INSURERS

OPERATO
member of ELES Group

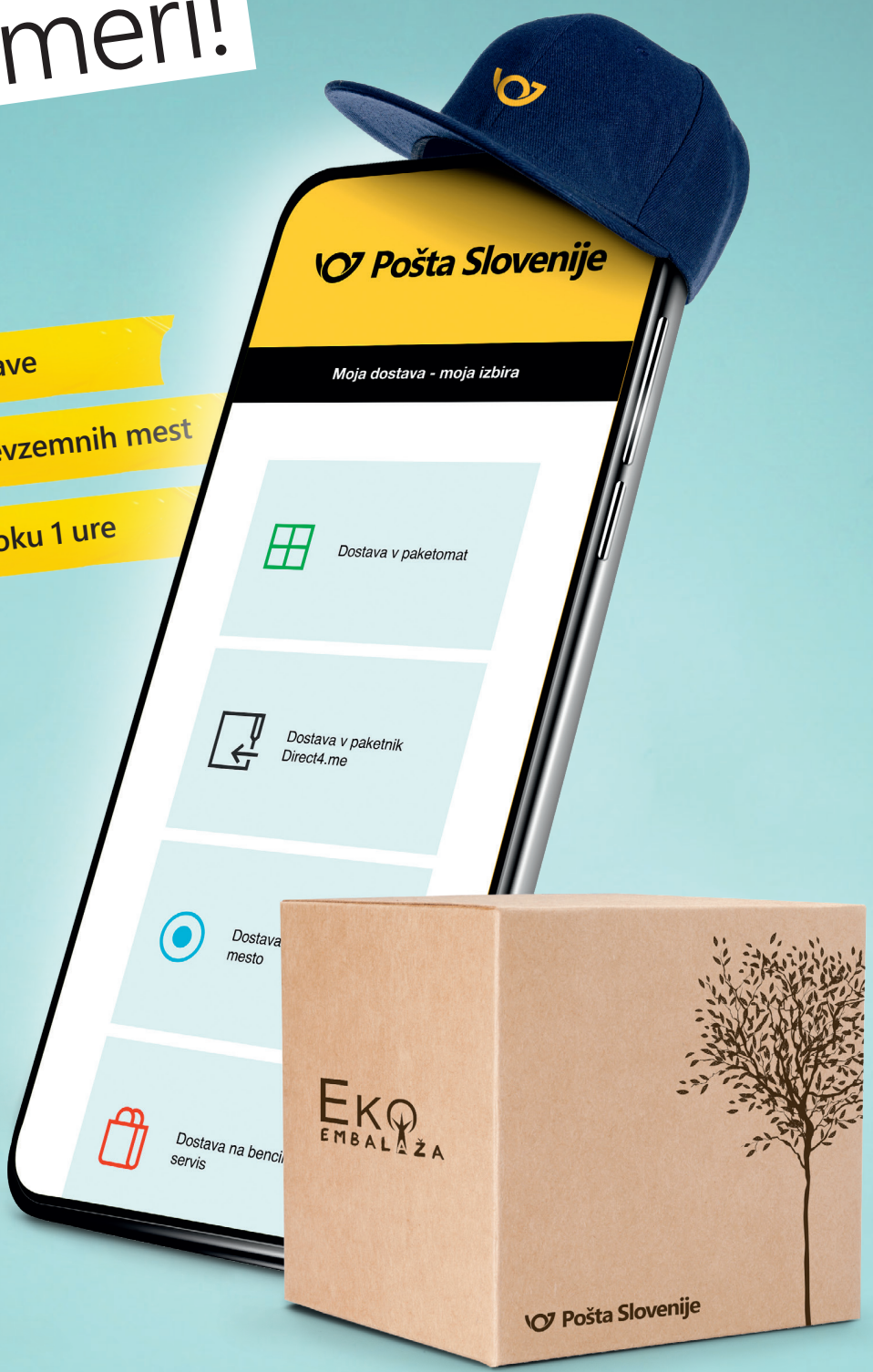
NUJNO OBVESTILO ZA VSE ŠTUDENTE!

Obveščamo te, da bo 7. in 8. septembra dovolj mrzlega piva in tople kave, ker **NE BO** zmanjkalo elektrike.
Sistem SUMO skrbi za naše in druga omrežja v Evropi.

Če te zanima IT v energetiki in delo v mednarodnem start-upu, te vabimo, da nam pišeš na info@operato.eu, ali nas poišči na **OTS2022 konferenci**.

Izberi dostavo po meri!

- 7 načinov dostave
- Več kot 1.100 prevzemnih mest
- Dostava tudi v roku 1 ure



POENOSTAVI SPLETNO NAKUPOVANJE Z IZBIRO DOSTAVE, KI TI NAJBOLJ USTREZA

Izberi enega od več kot 1.100 alternativnih prevzemnih mest in s spletnim orodjem "Sledenje pošiljk" spremljaj, kako blizu je tvoj paket.

Več nasvetov za enostavne spletne nakupe na www.posta.si/enostavna-dostava

KRAFTWERK

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

#MiSmoSRC

Storitve na področju informatike in vrhunske IT opreme že 35 let.

Družinam prijazno podjetje z veliko mero fleksibilnosti in 360+ prijaznimi sodelavci.

SRC je slovensko podjetje, ki deluje v Ljubljani, Mariboru, Naklem in Beogradu.

Kaj delamo

- Različne sodobne **poslovne rešitve po naročilu** – večinoma spletne aplikacije in portale (JS, .NET, AngularJS...)
- Napredno **upravljanje s podatki in vsebinami** (BigData, poslovna analitika, CMS in DMS).
- **Integracije** med različnimi poslovnimi sistemi, podatkovnimi bazami in aplikacijami (EAI z Rest API, webhooks...)
- **Digitalizacija in avtomatizacija** poslovanja z uporabo BPM in RPA platform, uvajanjem AI, forecastin-gom in podobno.
- Napredne **storitve informacijske varnosti**, vključno s Cloud Managed Security in SOC-om.
- **Upravljanje infrastrukture** – oblačne (imamo lasten poslovni oblak), hibridne in tudi on-prem.
- **Informatiziramo zdravstveni sistem** (od HIS in laboratorijskih sistemov do upravljanja naročanja in čakalnic).
- **Razvojno - raziskovalne projekte** z implementacijo novih tehnologij (npr. s področja IoT, AI in podobno).

Kako delamo

- Projekte razvijamo z uporabo sodobnih agilnih metod (predvsem Scrum, včasih tudi Kanban).
- Naloge imamo jasno opredeljene v Jiri, uporabljamo Teamse in Webex - zato tudi delo od doma ni problem.
- Z implementacijo DevOps metod in orodji povezujemo agilni razvoj, produktno vodenje in operacije.
- Vedno skrbimo za to, da je naročnik zadovoljen, zato imamo s strankami trajno dobre poslovne odnose.

SRC

SRC d.o.o. | Tržaška c. 116, 1000 Ljubljana | 01 600 700 | src@src.si | www.src.si

GROW Your Bussines with INNOVATION

ACCELERATE TIME-TO-MARKET

MONETIZATION

Monetize Anything-as-a-Service

Tridens Monetization is an innovative, real-time, convergent charging, and monetization platform for any industry, any business model, and any revenue stream in today's digital economy.

CHARGE & DRIVE

Electric Vehicle Charging

Tridens Charge & Drive is an all-in-one software solution for Charge Point Operators (CPO) and e-Mobility Service Providers (EMP).

DRIVE & PAY

Pay-As-You-Drive Insurance

Tridens Drive & Pay is a pay-as-you-go software solution for insurance companies. Enabling them to charge their customers for subscription and usage-based consumption services.

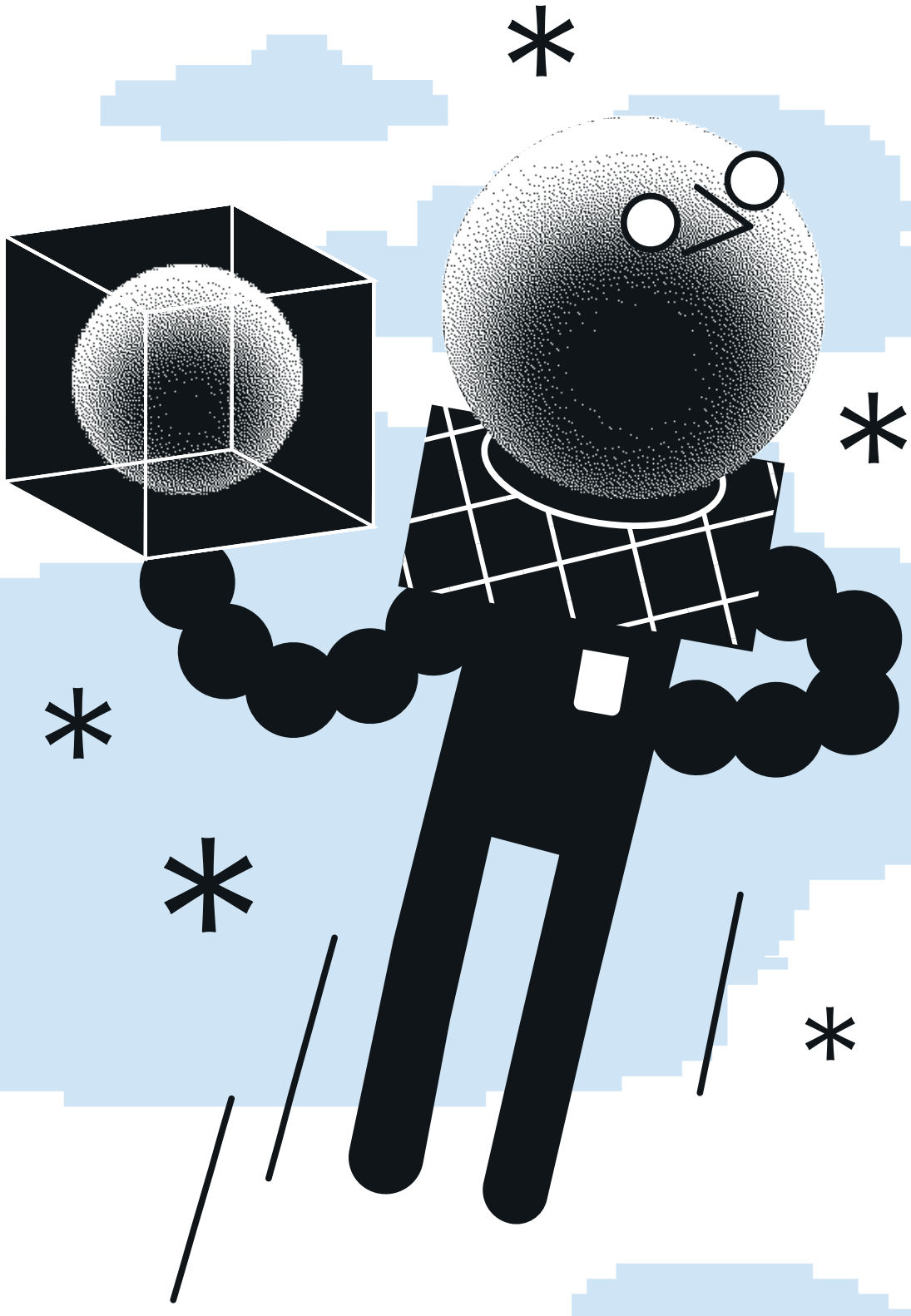
Contact Us

tridens technology

Tridens d.o.o.
Zagrebška cesta 22
2000 Maribor
Slovenia

www.tridenstechnology.com
+386 31 627 462
+386 59 010 975
info@tridenstechnology.com

3fs CLOUD INNOVATION PARTNER



3fs.si
3fs.si/careers

AKCIJA

ZA UDELEŽENCE

KONFERENCE OTS2022

Prejmite sveže novice iz sveta računalništva vsakih 14 dni v vaš poštni nabiralnik!

Letna naročnina na revijo Računalniške novice (21 številk) skupaj z darilom za samo

33,50 €

IZBIRATE LAHKO MED:

Ključavnica Masterlock 8391 (90 cm)

USB ključ Apacer 3.1 128 GB

Kinetic Reactor

01 620 88 00 **narocnine@stromboli.si**

Tehten, kritičen.
Časnik, ki reče bobu bob.

EKSKLUZIVNO ZA OTS 2022

BERITE VEČER

14 DNI BREZPLAČNO

SPLOŠNA

Poskenirajte kodo in izberite tiskano ali digitalno naročnino na Večer

ATHENA
EUROPEAN UNIVERSITY

EVROPSKA UNIVERZA

ATHENA stremi k digitalni preobrazbi družbe, kjer se podpira **razvoj** pravičnega, trajnostnega in varnega **digitalnega gospodarstva**.

Projekt združuje partnerske institucije in njihove deležnike tako, da vzpostavlja sodelovalno okolje **visokega šolstva, gospodarstva in širše družbe**.

7 VISOKOŠOLSKIH INSTITUCIJ

127 820
Študentov

10 645
Zaposlenih, vključno s 6 539 raziskovalci

63
Fakultet

234
Raziskovalnih skupin

PRILOŽNOSTI ZA PARTNERJE OTS

MEDNARODNO POVEZOVANJE

Učinkovita regionalna omrežja omogočajo priložnosti za povezovanje in sodelovanje na mednarodni ravni. Podjetjem so na voljo vključevanja v mednarodne raziskovalne in industrijske projekte ter v inovacijske izzive.

ŠTUDENTSKI IZZIVI IN PRAKSE

Platforma PRAXIS ponuja "virtualni trg", kjer lahko podjetja, visokošolske institucije ali raziskovalne skupine objavljajo izzive, prakse ali bodoče projekte, na katere se prijavljajo študentje iz vse Evrope in širše.

PRENOS ZNANJA IN TEHNOLOGIJ

Inovacijski ekosistemi za učinkovit prenos znanja in tehnologij na partnerskih institucijah omogočajo deljenje inovacij in rezultatov raziskav z industrijo in družbo ter povezovanja z lokalnimi skupnostmi.

http://athenauni.eu
athena-WP4@um.si

Univerza v Mariboru

European Commission

#EuropeanUniversities

Building the universities for the future

GENERALNI POKROVITELJ

inova iT

POKROVITELJI

Beyond Now.

bintegra
Svetovanje in IT rešitve

COMTRADE GAMING

databox

impol
Aluminium Industry

INEA

INFORMACIJSKE STORITVE IN INŽENIRING, D.O.O.
INFORMATIKA

MIKROpis

msg ODATEAM

niceHash

NovaKBM

NOVUMRGI

OPERATO
member of ELES Group

Pošta Slovenije

RC IRC Celje, d.o.o. / Ltd.

RAZVOJNI CENTER
IKTS
ŽALEC

SRC

tridens
technology

3fs

MEDIJSKA POKROVITELJA

Računalniške novice
www.racunalniske-novice.com

VEČER

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

INŠTITUT ZA INFORMATIKO