OBVLADOVANJE RAZLIČIC V MIKROSTORITVENI ARHITEKTURI

Luka Pavlič[1], David Zakelšek[2]
[1]Univerza v Mariboru, FERI, Koroška cesta 46, 2000 Maribor
[2]MOTIONDATA VECTOR Software GmbH, Feldkirchner Straße 11-15, A – 8054 Seiersberg
luka.pavlic@um.si, david.zakelsek@motiondata-vector.com

Povzetek

Mikrostoritvena arhitektura postaja vodilen načrtovalski stil sodobnih informacijskih rešitev. Zaradi svojih lastnosti prinaša veliko novosti iz vidika načrtovanja, implementacije, kot tudi nameščanja informacijskih rešitev. Kljub temu, da rešuje mnoge načrtovalske zagate, ki jih poznamo v t.i. monolotnih sistemih, ponuja razvijalcem tudi nove izzive. Eden izmed njih je zagotovo ustrezno naslavljanje izzivov, povezanih z uspešnim obvladovanjem različic posameznih mikrostoritev, ki morajo kljub različni dinamiki razvoja tvoriti homogeno celoto.

Ustrezno obvladovanje različic mora torej omogočiti nadaljnji razvoj in hkrati nespremenjeno delovanje obstoječe rešitve. Ustrezen mehanizem vpeljave različic poraja mnogo vprašanj in odločitev, ki jih moramo sprejeti že v začetni fazi razvoja, saj so kasneje spremembe izjemno otežkočene. Obstajajo različni pristopi za vzpostavitev tovrstnega okolja - vsak ima svoje prednosti in slabosti. Predhodna analiza in izbira pristopa imata pomembno vlogo, pri tem kako bomo v prihodnje opravljali vzdrževanje in nadaljnji razvoj tovrstne storitvene arhitekture.

V prispevku zato sistematično predstavimo problematiko vodenja različic mikrostoritev. Predstavimo in primerjamo različne pristope ter nakažemo, kako in kdaj izbrati ustrezen pristop oz. kombinacijo le-teh.

Abstract

VERSION MANAGEMENT IN MICROSERVICE ARCHITECTURE

The microservice architecture is becoming a leading architectural style in a modern information solution development. It brings novelties in fields such as system design, implementation, as well as finished information solutions deployment. However, it also brings several new challenges. One of them are challenges, associated with the successful management of individual microservice versions. Microservices, even in different versions, must form a homogeneous information solution.

An adequate version control must therefore allow continuous development and operations. The decisions, related to the version management should be made at the early phases of the development, since changes in this domain are more difficult during the development or even an operations phase. Several approaches exist - each with its advantages and disadvantages. This is why a preliminary analysis and careful selection of adequate approach play an important role in a long term project success.

In this paper, we systematically present the issue of managing microservice versions. We also present and compare different approaches and indicate how and when to choose the appropriate approach or a sound combination of them.

Ključne besede

Mikrostoritve, mikrostoritvena arhitektura, obvladovanje različic.

Key words

Microservices, microservice architecture, versions management.

UVOD

Informacijsko rešitev ob upoštevanju smernic mikrostoritvene arhitekture gradi množica neodvisnih mikrostoritev. Neodvisnost gre pri mikrostoritvah preko meje izvorne kode, in se nanaša tudi na izvajalno okolje ter razvojno ekipo, ki za določeno mikrostoritev skrbi. Zaradi stalnega razvoja in ločenega nadgrajevanja posameznih mikrostoritev se obvladovanje različic v sklopu tega pristopa kaže v novi luči. Sploh, ko govorimo o informacijskih rešitvah, ki služijo različnim strankam ter se posledično pričakuje, da različice mikrostoritev delujejo hkrati za različne stranke, pri čemur določene stranke ne želijo novosti, druge stranke pa so pripravljene investirati v posodabljanje obstoječih in dostavo novih funkcionalnosti.

Vsaka mikrostoritev ima natančno določen programski vmesnik, ki ga uporabljajo različni odjemalci, npr. zunanji sistemi, spletne in mobilne aplikacije ter druge storitve. Posamezen odjemalec je integriral določeno mikrostoritev v nekem časovnem obdobju in se vezal na točno določeno različico. Ta se zanaša na njeno konsistentno delovanje glede na definiran vmesnik. V celotnem življenjskem ciklu je potrebno mikrostoritev večkrat nadgraditi, razširiti ali odpraviti napake.

Pri zasnovi mikrostoritvene arhitekture je razmislek o upravljanju različic zelo pomemben. V primerjavi z monolitnimi sistemi imamo namreč opravka z več neodvisnimi gradniki sistema. Neodvisnost dojemamo tudi v smislu več razvojnih ekip z lastnimi razvojnimi cikli in kadencami ter tipično tudi večjim številom strank z lastno vizijo investicij v razvoj in nadgrajevanje sistema. Težko se izognemo temu, da bi večina mikrostoritev imela svoj lasten cikel razvoja in posledično upravljanja različic. Celoten nabor mikrostoritev mora kljub podpori neenakih različic in neodvisnega razvoja omogočati nemoteno delovanje, ki ga dosežemo z ustrezno medsebojno komunikacijo mikrostoritev, uporabniških vmesnikov ter drugih gradnikov sistema.

Članek predstavlja skrčen pregled razmislekov in pristopov, ki so v splošnem na voljo, ko se odločamo o vodenju različic mikrostoritev. V naslednjih dveh poglavjih uvodoma predstavimo izzive, ki izhajajo iz odločitve, da bomo dopustili neskladen razvoj različnih mikrostoritev informacijske rešitve. V nadaljevanju pa povzamemo glavne uveljavljene pristope pri obvladovanju različic ter jih medsebojno primerjamo.

IZZIVI PRI VODENJU RAZLIČIC MIKROSTORITEV

Upravljanje različic mikrostoritev predstavlja izziv predvsem iz vidika zagotavljanja stabilnih komunikacijskih vmesnikov, ki so ključ do šibke sklopljenosti v celoviti rešitvi. Potrebno je zagotoviti natančna pravila komuniciranja ter celovito upravljanje različic, kot je npr. vodenje matrik združljivosti različic mikrostoritev. Že uvodni razmislek o različicah mikrostoritev postreže s kar nekaj dilemami, kaj različica mikrostoritve sploh je:

- Se različica nanaša zgolj na vmesnik mikrostoritve ali tudi na njeno implementacijo?
- Je sprememba zaledja mikrostoritve (v smislu npr. spremenjenih podatkovnih entitet) tudi samodejno že razumljena kot nova različica mikrostoritve?
- Je smiselno govoriti o različicah takrat, ko so spremembe (bodisi vmesnika, implementacije, interne strukture ipd.) povsem združljive?

- Je potrebno razlikovati med združljivimi spremembami (takšnimi, ki omogočajo obstoječim odjemalcem nemoteno delo) in nezdružljivimi (terjajo spremembo odjemalcev)?

Različica mikrostoritve obsega vse našteto: tako združljive ali nezdružljive spremembe vmesnika, njene implementacije, interne strukture in zgolj dopolnitve oz. manjše nadgradnje. Kljub temu, da želimo zagotoviti hkratno delovanje več različic iste mikrostoritve, kar je dodaten tehnično-organizacijski izziv, bodo morali odjemalci v določenem časovnem obdobju omogočati podporo več različicam iste mikrostoritve. Doseganje te zmožnosti ni enostavno, še posebej, kadar odjemalec uporablja celoten nabor storitev. K vodenju različic prištevamo tudi vodenje življenjskega cikla različic. Da se izognemo neracionalnemu upravljanju (in hkratnemu posodabljanju) številnih različic iste mikrostoritve, moramo v tem sklopu določiti tudi način, kako, kdaj in na kakšen način zastarele različice mikrostoritev upokojiti.

Če povzamemo, so poleg naštetih še številni drugi izzivi vodenja različic mikrostoritev [10]:

- izbira primernega identifikatorja različic,
- določitev ustrezne semantike identifikatorja različic (združljive / nezdružljive spremembe, ločeno vodenje različic vmesnika in implementacije ipd.),
- hkraten razvoj in dopolnjevanje več različic hkrati,
- obvladovanje zunanjih odvisnost posamezne različice mikrostoritve (npr. podatkovne shrambe ipd.),
- vključitev nove različice z nezdružljivimi spremembami v delujoč ekosistem mikrostoritev,
- kontinuirano delovanje obstoječih odjemalcev ob menjavi različic,
- dokumentiranje sprememb,
- ločeno upravljanje različic vmesnika in implementacija,
- možnost skaliranja le določene različice oz. ločeno določanje pravil, vezano na različice,
- ohranjanje šibke sklopljenosti med mikrostoritvami, ne oziraje na različice,
- sistematična upokojitev zastarelih različic.

PREGLED UVELJAVLJENIH PRISTOPOV

Poslovno okolje pogosto diktira, da vodimo različice na nivoju mikrostoritve in se izognemo potrebi po zagotavljanju enake različice za vse storitve, skratka vodenju različice sistema. Glede na poslovne potrebe bomo določene mikrostoritve namreč hitreje nadgrajevali in uvajali nove, druge bodo posledično dlje časa ostale na prvotni različici.

Za izvedbo številnih sprememb in zagotovitev neprekinjenega delovanja je torej potrebno vpeljati različice. Le-te lahko vodimo na različne načine in z njimi označujemo opravljene spremembe. V obstoječi različici storitve lahko izvajamo spremembe, kot so manjši popravki

ki ne spremenijo delovanja. Sam način izvedbe zahteve ter že obstoječa definicija podatkov se ne sme spreminjati. Grobe spremembe, kot so spreminjanje tipov ali imen atributov, v obstoječi različici niso združljive, saj porušijo komunikacijo z obstoječimi odjemalci. Tovrstne spremembe moramo predstaviti in ponuditi v novi različici. Takšen pristop omogoča nemoten razvoj storitve, hkrati pa skrbi za nemoteno delovanje integriranih sistemov. Na ta način lahko sočasno obstajata dve ali več različic določene storitve. Starejše različice bodo sčasoma zastarele in jih več ne bo mogoče uporabljati. Takšne različice je potrebno postopoma upokojiti in izvzeti iz uporabe. Preden to izvedemo, moramo zagotoviti, da so vsi odjemalci preklopili na novejše različice. Hkratno delovanje več različic odjemalcem omogoča prestop na novejšo, preden je starejša različica odstranjena. Tako zagotovimo mehko migracijo brez izpada delovanja.

Pregledali in zbrali smo vodilne pristope vodenja različic. Pristopi so medsebojno komplementarni in redko uporabljeni ločeno eden od drugega. Združimo jih lahko v več kategorij, ki so podrobneje predstavljene v nadaljevanju [1,2].

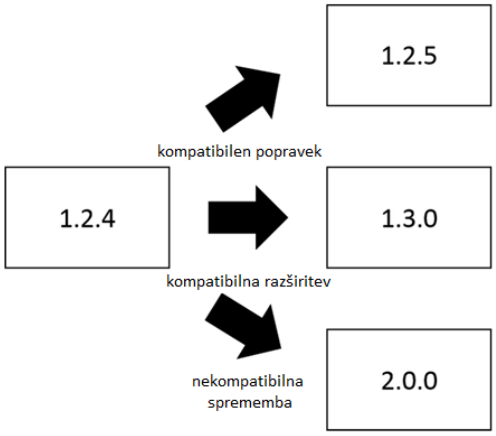
Zgradba in semantika identifikatorja različice

Identifikator različice se izbere v zgodnji fazi razvoja. Lahko uporabimo en identifikator za vse nivoje ali pa kombiniramo identifikatorja glede na primernost. Najpogosteje uporabljeni identifikatorji so [2]:

- pozitivno celo število,
- kombinacija več pozitivnih celih števil,
- datum.

Identifikator predstavlja pravilo, kako bo zasnovana različica. Dobre prakse kažejo na kombinacijo treh pozitivnih celih števil (angl. MAJOR.MINOR.PATCH). Vsako število ima svoj pomen, glede na opravljene spremembe jih tudi večamo. Zadnj ptevilo se nanaša na odpravljanje napak, kadar je storitev že v produkcijskem okolju. Popravki ne spreminjajo vmesnika, zato komunikacija teče nemoteno naprej. Vmesno število tipično predstavlja dodano funkcionalnost. Povečamo ga ob vsaki nadgradnji vmesnika in storitve, ki ne vpliva na delovanje obstoječe komunikacije. Prvo število pa predstavlja krovno različico. V kolikor dodane spremembe spreminjajo definicijo obstoječega vmesnika različice, to predstavlja nekompatibilno spremembo s trenutno različico vmesnika in s tem povečano prvo številko. Primer uporabe takšnega identifikatorja in njegov pomen je predstavljen na sliki 1.

Nekoliko manj priljubljen je sistem uporabe datuma uvedbe spremembe, ki je tipično pisan kot dolga številka (npr. YYYYMMDD). Uporablja se predvsem v aplikacijah s časovnimi omejitvami. Nekateri sistemi vežejo na tovrstni identifikator svoj cikel podpore uporabnikom. Za večje različice, ki jih predstavljajo leta, lahko na primer nudijo nekajletno podporo, za manjše različice, ki jih predstavljajo meseci in dnevi, pa je lahko podpora precej krajša. Identifikator koledarja v primerjavi z identifikatorjem celih števil sam po sebi ne pove veliko, zato je uporaben v kombinaciji s številskimi različicami.



Slika 1: Spreminjanje identifikatorja različic [10]

Upravljanje različic na strani mikrostoritve

V primeru, da se odločimo za vodenje različic na nivoju mikrostoritve, tipično ob vsaki zahtevi preverimo, s katero različico želi komunicirati odjemalec ter zahtevo predamo v obdelavo točno določeni različici iste mikrostoritve. Pri tem imamo več možnosti, kako izbrati ustrezno mikrostoritev [3]:

- različica v URL (npr. https://domena.com/api/v2.0.0/),
- različica v glavi zahteve,
- različica na nivoju vmesnika,
- različica na nivoju entitet.

Eden izmed priljubljenih načinov je vodenje različic v URL. Ta pristop dodaja identifikator različice v sam URL storitve, zato jo iz zahtev enostavno prepoznamo. Tudi pri beleženju je dovolj, da shranimo URL zahtevka, iz katerega prepoznamo različico. Pristop vodenja različic je zelo učinkovit, kadar uporabljamo usmerjevalnik (API Gateway), ki preko URL različice usmerja promet na posamezne storitve. Različica v URL je lahko standarden del naslova storitve ali podan kot URL parameter. Slednji dodaja fleksibilnost k razvoju, saj je možno različice, ki niso del naslova, dinamično upravljati [4].

Prednost pristopa kodiranja različice v glavo zahtevka je, da naslov in lokacija vira ostaneta nespremenjena v vseh različicah. To po eni strani poenostavi implementacijo odjemalca, saj API ostaja semantično enak, po drugi strani pa (delno) prelaga odgovornost izbire različice na odjemalca. Slabost nespremenjenega URL-ja glede na različice je semantična neuporabnost. Razvijalec na prvi pogled ne more vedeti, katera različica je bila zahtevana. Zagotoviti je potrebno beleženje, ki shranjuje tudi različico glave za lažji pregled dogajanja in iskanja napak.

Vodenje različic na nivoju vmesnika

Programski vmesnik ali API je definicija, kako uporabiti določeno storitev. Predstavlja vrste klicev in zahtev, ki jih je mogoče opraviti, ter opisuje podatkovne strukture, ki se prenašajo.

Natančno določa obliko podatkov in pravila, ki jih je potrebno upoštevati za zagotovitev komunikacije s storitvijo. Predstavlja pogodbo za doseganje komunikacije pri razvoju odjemalca in tudi same storitve. Programski vmesnik se skozi čas spreminja in razvija, pri čemer nam upravljanje sprememb in zagotavljanje neprekinjene komunikacije omogoča vpeljava različic. Pri upravljanju le-teh moramo določiti natančna pravila glede spreminjanja obstoječe različice vmesnika ter vpeljave nove različice. Možni pristopi so sledeči [8]:

- isti vmesnik za več mikrostoritev,
- isti vmesnik za posamezno mikrostoritev,
- vmesnik za posamezno različico mikrostoritve.

Vodenje različic vmesnika je neposredno povezano z ustreznim identifikatorjem različice in njegovo semantiko. Ko govorimo o spremembah vmesnika se torej pogovarjamo o spremembah na drugem ali celo prvem nivoju strukturiranega identifikatorja. Različice vmesnika, kot pogodbe med mikrostoritvami, so neločljivo povezane z združljivimi ali nezdružljivimi spremembami mikrostoritev. Če povzamemo spremembe, ki jih lahko uvajamo v vmesnik, so le-te sledeče [8].

Dovoljene spremembe vmesnika v obstoječi različici:

- iz vmesnika nismo ničesar odstranili,
- načina in pravil obdelave nismo spremenili,
- neobveznih atributov nismo spremenili v obvezne,
- vse, kar smo dodali, mora biti neobvezno.

Nezdružljive spremembe vmesnika:

- sprememba metapodatkov (parametri v glavi),
- sprememba v URL, vključno s parametri in njihovo semantiko,
- spremembe v predstavitvi podatkov,
- preimenovanje atributov v predstavitvi podatkov,
- sprememba relacij med viri podatkov.

Vodenje različic na nivoju implementacije

Implementacija storitve zagotavlja nabor funkcionalnosti, ki jih definira vmesnik. V splošnem imamo dve možnosti [9]:

- skupna izvorna koda za vse različice posamezne mikrostoritve,
- izvorna koda (samostojen projekt) za posamezno različico mikrostoritve.

Dobre prakse kažejo na izoliranost posamezne različice ter nemoten razvoj glede na druge različice. Za vsako potrebujemo podatkovne entitete in v določenih primerih tudi dostopne

točke, ki so dostopne zgolj znotraj posamezne različice. V implementaciji moramo izbrati pristop, kako bomo zasnovali različice v kodi. Več različic iste funkcionalnosti lahko združimo v enem projektu, kar pomeni, da moramo projekt dobro razčleniti na sklope in tako z arhitekturnimi prijemi onemogočiti poseganja v predstavitveni nivo med različicami. Precej lažje to dosežemo, kadar ustvarimo samostojen projekt za vsako različico. Velika prednost je odvisnost od uporabljenih knjižnic, ki so vezane na projekt. Tako ima vsaka različica kot samostojen projekt svoje odvisnosti, ki jih lahko spreminjamo brez vpliva na druge različice. V skupnem projektu, kjer vsaka različica uporablja enake knjižnice, lahko menjava in upravljanje teh pripelje do težav v kompatibilnosti določenih različic. Obstaja možnost, da bodo novejše različice knjižnic nekompatibilne s starejšimi implementiranimi različicami storitve. V kolikor knjižnice vsebujejo grobe spremembe, je le-te potrebno prilagoditi na celoten projekt. Glede na njegovo zasnovo je to potrebno storiti za eno ali vse različice, ki se nahajajo v projektu.

Upravljanje različic na strani odjemalca

Odjemalec tipično prevzame vlogo orkestratorja. Pomeni, da mora klicati več poslovnih funkcionalnosti (vsako v določeni različici) za izvedbo poslovnega procesa. Izbira pristopa upravljanja različic na strani odjemalca je v precejšni meri odvisna od same zasnove vmesnika mikrostoritve ter odločitve, ali bomo podpirali več različic odjemalcev hkrati. Od tega zavisi uporaba enega ali več možnih pristopov [6,7]:

- podpora odjemalca eni različici mikrostoritve,
- podpora odjemalca več različicam mikrostoritve,
- uporaba posrednikov (dostopna točka, povratni posrednik),
- podpora več dostopnim točkam iz odjemalca.

Za uporabo mikrostoritev mora odjemalec najprej poznati naslove vseh mikrostoritev. V kolikor je uporabljen posrednik zahtev (angl. reverse proxy), je celoten nabor mikrostoritev dosegljiv na enem naslovu, kar zagotovo olajša konfiguracijo odjemalca. Kadar nimamo centralne dostopne točke ali odjemalec deluje v vlogi orkestratorja, mora le-ta poznati naslove za vsako posamezno mikrostoritev. Glede na zasnovo arhitekture je lahko vsaka različica predstavljena kot samostojna mikrostoritev, ki jo mora odjemalec poznati.

Ko poznamo storitve in način vodenja različic, lahko pričnemo z uporabo posameznega vmesnika. Kljub podobnosti več različic je potrebno implementacijo izvesti brez odvisnosti. Nobena različica naj ne deli podatkovnih virov z drugo različico. V večini primerov je nastala nova različica, saj vsebuje spremembe, ki so nezdružljive s prejšnjo različico. Lahko se zgodi, da isti atribut z enakim imenom v eni in drugi različici ne predstavlja iste informacije. Na ta način moramo razmišljati tudi pri integraciji različice na strani odjemalca. Za vsako različico izdelamo svoje predstavitve podatkovnih virov (entitet), ki jih v prihodnosti neodvisno razvijamo. To je potrebno ponoviti za vsako mikrostoritev in njene različice.

PRIMERJAVA PRISTOPOV

Na podlagi opravljenega pregleda pristopov upravljanja različic posameznega nivoja lahko za posamezen pristop opredelimo njegove prednosti in slabosti. Tovrstna primerjava nam olajša odločanje za izbiro najprimernejšega pristopa glede na naše potrebe.

Pri izbiri identifikatorja različic smo primerjali pristope celega števila, kombinacije celih števil ter datum kot identifikator (Tabela 1). Celo število je enostaven identifikator, ki ga lahko uporabljamo na več nivojih in je primeren pristop za upravljanje API različic. Manj primeren je pri upravljanju različic implementacije, saj se izgubi informacija o pomembnosti spremembe. Boljšo preglednost nad spremembami nudi kompleksnejši kombiniran identifikator, ki je primernejši pri upravljanju različic na nivoju implementacije, manj primeren pa je za upravljanje API različic, saj bi vsaka MINOR ali PATCH sprememba pomenila novo različico API vmesnika. Posledično bi nastalo veliko novih različic v zelo kratkem času, kar nasprotuje dobrim praksam. Datum kot tretji identifikator je primeren za specifične vrste storitev, ki so vezane na časovne omejitve. Lahko ga uporabimo kot identifikator API različice. Sam po sebi ne nosi nobene informacije o opravljenih spremembah, zato nudi slabši pregled in ni primeren za uporabo na nivoju implementacije.

Tabela 1:Izzivi, ki jih rešuje identifikator različice

	Pozitivno celo število	Kombinacija več pozitivnih celih števil	Datum
Primernost identifikatorja na nivoju API	✓	×	✓
Primernost identifikatorja na nivoju implementacije	×	✓	×
Poenostavljena implementacija	×	✓	×

Na nivoju storitev smo primerjali pristope upravljanja različic v naslovu, glavi zahtevka, na nivoju vmesnika in upravljanja različic na nivoju entitet (tabela 2). Različica v naslovu je del vmesnika in je enostavno prepoznavna. Omogoča neodvisnost med različicami ter njihov neodvisen razvoj. Upravljanje različice je zahtevnejše v primerjavi z različico v glavi zahtevka. Prednost različice v zahtevku je tudi neodvisnost od API, vendar tovrstna različica ni več enostavno prepoznavna in ni nujno neodvisna, saj si lahko posamezne različice delijo podatkovne entitete ali vsebujejo druge odvisnosti. Sama različica je lahko del API, kar predstavlja lažjo preglednost in modularnost ter zahtevnejše upravljanje. Vodenje različic na nivoju entitet omogoča lažje upravljanje in neodvisno predstavitev na podatkovnem nivoju. Sam razvoj je skupen vsem različicam, kar onemogoči upravljanje različice kot samostojno storitev

Tabela 2: Izzivi, ki jih rešujejo pristopi na strani storitev

	Različica v URL	Različica v glavi zahteve	Različica na nivoju API	Različica na nivoju entitet
Upravljanje več različic hkrati	✓	✓	✓	✓
Večje število mikrostoritev	✓	×	✓	×
Enostavna vpeljava nove različice	✓	×	×	✓
Enostavna upokojitev različice	✓	×	✓	×
Skaliranje različice	✓	×	✓	×
Šibka sklopljenost različic	✓	×	✓	×
Poenostavljena implementacija	×	✓	✓	✓

Nadalje smo primerjali pristope na nivoju odjemalca. Odjemalec lahko podpira samo eno različico, kar olajša konfiguracijo dostopa, vendar ga omejuje na točno določeno različico funkcionalnosti. Pristop več različic mikrostoritve predstavlja bolj fleksibilno alternativo, saj omogoča načrtovan ali celo avtomatiziran prehod med različicami. Za podporo več različic je potrebna konfiguracija za vsako podprto različico. Število konfiguracij se z vsako dodatno dostopno točko ali novo različico povečuje, kar poveča kompleksnost uporabe odjemalca. Kot bolj prijazen pristop se ponuja centralna dostopna točka (reverse proxy), za katero se nahajajo vse mikrostoritve v vseh svojih različicah. Sama uporaba ter konfiguracija odjemalca je v tem pristopu zelo poenostavljena. (tabela 3)

Samo zasnovo odjemalca nam olajša vzorec usmerjanja prehoda, ki omogoča konfiguracijo zgolj ene dostopne točke, kar zmanjša konfiguracijo in olajša uporabo celotnega nabora storitev ter različic. Kadar odjemalec kliče več storitev in podatke prikazuje na grafičnem uporabniškem vmesniku, je priporočljiva uporaba vzorca kompozicije uporabniškega vmesnika.

Tabela 3: Izzivi, ki jih rešujejo pristopi na strani odjemalca

	Podpora ene različice mikrostoritve	Podpora več različic mikrostoritve	Ena dostopna točka (reverse proxy)	Več dostopnih točk
Upravljanje več različic hkrati	✓	×	✓	✓
Večje število mikrostoritev	×	✓	×	✓
Enostavna vpeljava nove različice	×	✓	✓	✓
Enostavna upokojitev različice	✓	×	✓	✓
Skaliranje različice	✓	×	✓	✓
Šibka sklopljenost	✓	×	✓	✓

ZAKLJUČEK

Čeprav mnogi avtorji delijo mnenje, da je uporaba različic mikrostoritev s vklopu enotne informacijske rešitve antivzorec (slaba praksa: na prvi pogled dobra ideja, ki pa se v praksi ne obnese dobro), se srečamo z informacijskimi rešivami, kjer se različicam ne moremo izogniti. V takem primeru je zelo pomemben poglobljen razmislek, kako identificirati različice, kakšno napredovanje različic dovoliti na nivoju posamezne mikrostoritve ter navsezadnje kako zagotoviti skladno delovanje celotnega ekosistema mikrostoritev in uporabniških vmesnikov kljub temu, da določene mikrostoritve hkrati obstajajo v več, morda celo nezdružljivih različicah.

V prispevku smo predstavili problematiko vodenja različic mikrostoritev. Podali smo možnosti, ki jih imamo tako na nivoju mikrostortitve, odjemalca, ko tudi vmesnika, da bi omogočili ustrezno vodenje različic mikrostoritev.

Predstavljeni pristopi niso izključujoči temveč so komplementarni – ustrezno vodenje različic na podanem projektu bo mogoče le ob skrbno izbranem naboru predstavljenih pristopov in tehničnih rešitev.

VIRI IN LITERATURA

[1] Sergiuoltean, „Microservice versioning“, Learn. Write. Repeat., jan. 05, 2018. https://sergiuoltean.com/2018/01/05/microservice-versioning/ (pridobljeno sept, 2021).

[2] W. T. Technologies, „Versioning Management in Microservices“, Medium, avg. 01, 2018. https://walkingtreetech.medium.com/versioning-management-in-microservices-8221ec592499 (pridobljeno sept, 2021).

[3] K. Brown, „How to design and version APIs for microservices (part 6)“, sep. 10, 2019. https://www.ibm.com/cloud/blog/rapidly-developing-applications-part-6-exposing-and-versioning-apis (pridobljeno sept, 2021).

[4] P. Sturgeon, „API Versioning Has No ‚Right Way‘“. https://apisyouwonthate.com/blog/api-versioning-has-no-right-way (pridobljeno sept, 2021).

[5] „Contracts, Addressing, and APIs for Microservices“, Google Cloud. https://cloud.google.com/appengine/docs/standard/python/designing-microservice-api (pridobljeno sept, 2021).

[6] S. Fidanov, „Supporting multiple API versions“, Terlici. https://www.terlici.com/2014/09/23/multiple-api-version.html (pridobljeno sept, 2021).

[7] „asp.net Versioning: REST API versioning with ASP.NET Core - DEV Community“. https://dev.to/99darshan/restful-web-api-versioning-with-asp-net-core-1e8g (pridobljeno sept, 2021).

[8] Y. Arimatsu, Y. Ishida, K. Noda, in T. Kobayashi, „Enriching API Documentation by Relevant API Methods Recommendation Based on Version History“, v 2018 IEEE Third International Workshop on Dynamic Software Documentation (DySDoc3), sep. 2018, str. 15–16. doi: 10.1109/DySDoc3.2018.00014.

[9] N. Terence, „Microservices in practice: From Architecture to Deployment“, Cuelogic Technologies Pvt. Ltd., maj 27, 2019. https://www.cuelogic.com/blog/microservices-in-practice-from-architecture-to-deployment (pridobljeno sept, 2021).

[10] ZAKELŠEK, David, 2021, Pristopi k obvladovanju različic mikrostoritev : magistrsko delo [na spletu]. Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko.