Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/08BE03Kosijer

PODRŠKA ZA UPRAVLJANJE SPRING CLOUD GATEWAY KOMPONENTOM U MIKROSERVISNOM EKOSISTEMU

SUPPORT FOR MANAGING SPRING CLOUD GATEWAY COMPONENT IN THE MICROSERVICE ECOSYSTEM

Stevan Kosijer, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu su predstavljene osobine, osnovne komponente, prednosti i mane mikroservisne arhitekture. Obrazložena je funkcija komponente koja služi za kontrolu ulaznog saobraćaja u mikroservisnom ekosistemu. Opisana je implementacija proširenja koje omogućava direktnu manipulaciju rutama i filterima ove komponente. Nad realnim sistemom koji podržava dato proširenje izvršena su testiranja performansi pri obavljanju operacija manipulacije rutama. Predstavljeni su rezultati testiranja kao i mogući pravci daljeg razvoja ovog rešenja.*

Ključne reči: *Mikroservisna arhitektura, Gateway komponenta, Spring Boot Admin, upravljanje mikroservisnim ekosistemom.*

Abstract – *Features, main components, advantages and disadvantages of microservice architecture are presented in this paper. The function of the component used to control the inbound traffic in the microservice ecosystem is explained. An extension implementation that allows direct manipulation of routes and filters of this component is described. A real system that supports this extension has been subjected to performance tests when performing route manipulation operations. Test results are presented as well as possible directions for further development of this solution.*

Keywords: *Microservice achitecture, Gateway component, Spring Boot Admin, manipulating of microservice ecosystem.*

1. UVOD

Trenutno postoji mnoštvo tehnologija, arhitektura i praksi koje služe za rešavanje problema vezanih za određivanje i postavku infrastrukture softverskih rešenja. Mikroservisna arhitektura se izdvaja kao moguće rešenje za transformaciju i skaliranje velikih softverskih sistema. Razvoj mikroservisne arhitekture potstakao je izradu mnogobrojnih alata čija je namena olakšanje rada i razvoja mikroservisnih sistema.

Njihova primena se javlja u svim fazama razvoja mikroservisne arhitekture. Većinu ovih alata i projekata razvila je zajednica otvorenog koda. Jedan od popularnijih je *Spring Boot Admin* projekat.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Slavica Kordić, vanredni prof.

Spring Boot Admin projekat je kreiran sa idejom da ponudi jednostavan način registrovanja i upravljanja svim komponentama u mikroservisnoj arhitekturi koje su napravljene upotrebom *SpringBoot* okvira za razvoj veb baziranih aplikacija. Projekat *Spring Boot Admin* nudi centralizovano mesto sa uređenim korisničkim interfejsom za uvid u trenutno stanje sistema i mogućnost manipulacije podešavanja određenih komponenti sistema. Ono što nedostaje u ovom projektu je podrška za *Spring Cloud Gateway* komponentu.

Podrška za *Spring Cloud Gateway* komponentu podrazumeva proširenje korisničnog interfejsa *Spring Boot Admin* projekta, koje bi bilo u stanju da prepozna prisustvo *Gateway* komponente u mikroservisnom ekosistemu i ponudi dodatnu specijalizovanu stranicu putem koje je moguće vršiti pregled trenutnog stanja definisanih ruta i filtera i manipulaciju njihovih konfiguracija. Manipulacija rutama i filterima bi omogućila kontrolu ulaznog toka *HTTP* saobraćaja u mikroservisnom ekosistemu, jer se na osnovu njihovih definicija formiraju pravila koja služe za upravljanje saobraćajem. *Spring Cloud Gateway* komponenta predstavlja posrednika u komunikaciji mikroservisnog ekosistema sa spoljnim svetom. Iz ovog razloga ona ima bitnu ulogu u aspektima sigurnosti samog sistema.

Podložna je čestom nadgledanju i sitnim izmenama kako bi se omogućio ili zabranio ulaz u sistem sa spoljne strane. Smatra se da bi ovo proširenje doprinelo u aspektima brzine izvršenja akcija manipulacije konfiguracijama i smanjenju broja koraka potrebnih za izvršavanje ovih akcija. Bez ovakvog rešenja manipulacija konfiguracionim podacima na *Gateway* komponenti svodi se na direktne izmene podataka u izvornom programskom kodu komponente i njeno potpuno restartovanje, što nije efikasno. Pored *Uvoda* i *Zaključka*, ovaj rad sadrži još tri poglavlja. U poglavlju *Mikroservisna arhitektura* predstavljene su teorijske osnove evolucije mikroservisne arhitekture, prednosti i mane mikroservisa i neke od najbitnijih komponenti neophodnih za rad i funkciju mikroservisa. U poglavlju *Spring Boot Admin projekat* predstavljen je *Spring Boot Admin* projekat sa osnovnim svojstvima i metodama koje obezbeđuju njegovu funkciju za nadgledanje i upravljanje servisima u mikroservisnoj arhitekturi. U poglavlju *Podrška za Spring Cloud Gateway komponentu* predstavljena je implementacija podrške za *Gateway* komponentu, osnovni entiteti i njihova funkcija, kao i analiza razlike u performansama pre i posle implementacije rešenja. Ovaj rad obuhvata deo rezultata koji su prezentovani u master radu.

2. MIKROSERVISNA ARHITEKTURA

Mikroservisi predstavljaju tehniku razvoja softvera koja služi za konstrukciju aplikacije kao kolekcije slabo povezanih servisa [1]. Svaki servis je samostalan i trebalo bi da implementira jednu funkciju poslovne logike, odnosno skup bliskih funkcionalnosti. Osnovne karakteristike mikroservisa su: postojanje komponenti koje su zadužene za pojedinačnu funkcionalnost, organizovanje i podela komponenti po poslovnim potrebama, decentralizovano upravljanje i manipulisanje podacima, fleksibilnost i robustnost. Mikroservisna arhitektura je dizajnirana da prevaziđe izazove, neuspehe i probleme većih aplikacija. Njenom upotrebom postiže se veća otpornost sistema na greške i nezavisnost komponenti u sistemu.

Zbog nezavisnosti komponenti različite grupe ljudi koje rade na razvoju projekta mogu da se fokusiraju isključivo na specifičan deo aplikacije. Mogu da upotrebe svojstven stil programiranja, bez brige da će ugroziti bilo koga drugog. Umesto deljenja jedne baze podataka, kao što je slučaj kod monolitne arhitekture, svaki mikroservis poseduje zasebnu bazu podataka. Postojanje posebne baze podataka za svaki mikroservis je krucijalno i doprinosi smanjenju zavisnosti među servisima. Svaki servis može da poseduje i različit tip baze podataka, baš onaj koji najbolje odgovara njegovim potrebama. Svaki mikroservis se može smatrati zasebnom komponentom. Za osnovni rad i funkciju implementacije mikroservisne arhitekture, izdvojene su *discovery*, *configuration* i *gateway* komponenta i predstavljene u narednim potpoglavljima.

2.1. *Discovery* komponenta

Mikroservisima je neophodna međusobna komunikacija da bi delili informacije ili sinhronizovano izvršavali složenije funkcionalnosti. Klijent za otkrivanje servisa (*discovery client*) u osnovi treba da omogući registraciju i rezoluciju instanci servisa. Pokrenuta instanca servisa, postupkom registracije servisa signalizira svoju dostupnost *discovery* servisu. Kada je instanca servisa dostupna, druge instance servisa koriste postupak rezolucije servisa da bi pronašli datu instancu servisa na mreži. Rezolucija predstavlja proces vraćanja fizičke mrežne lokacije pojedinačne instance servisa.

Ove dve osnovne operacije, međutim, obuhvataju širi spektar sofisticiranog ponašanja neophodnog za funkciju mikroservisa. Rezultat ovog ponašanja je mogućnost međusobne komunikacije svih instanci u sistemu.

2.2. *Configuration* komponenta

U mikroservisnoj arhitekturi se uglavnom svi servisi razmeštaju po različitim lokacijama i različitim serverima. Svaki servis ima svoju datoteku svojstava u koju se beleže karakteristike kao što su konfiguracija baze podataka, *URL*-ovi vezani za samo okruženje i svojstva vezana za taj mikroservis. Ako se svaka od ovih datoteka svojstava čuva zajedno sa samim servisom, na istom okruženju, može doći do različitih problema.

Na primer, ako treba da se izvrši čak i najmanja promena, kao što je izmena korisničkog imena za bazu podataka ili neka sitna izmena u nekom od *URL*-ova, prvo se mora pronaći lokacija instance ovog mikroservisa, zatim se mora pristupiti njegovom serveru i okruženju, napraviti konkretna izmena i na kraju ponovo izgraditi i restartovati cela instanca servisa. Ako se ovakav pristup ponovi za svaku od instanci servisa, može se pokazati kao vrlo spor i zahtevan. Iz ovog razloga u mikroservisima je uvedena nova komponenta, pod imenom centralni konfiguracioni servis ili configuration komponenta. Centralni konfiguracioni servis je odgovoran za obezbeđivanje datoteka svojstava za svaki instancu servisa koja je registrovana na klijentu za otkrivanje mikroservisa.
U trenutku pokretanja, on preuzima sve datoteke svojstava sa putanje navedene u svojoj konfiguraciji i skladišti ih u memoriji. Ova putanja može biti fizička putanja na sistemu ili putanja do nekog repozitorijuma kao što je *Git*. *Configuration* komponenta postiže veću dinamičnost i brzinu sistema. Kontrola samih konfiguracija postaje preglednija i centralizovana.

2.3. *Gateway* komponenta

Klijenti aplikacije zasnovane na mikroservisima moraju da pristupaju i lociraju pojedinačne instance servisa. Zbog dinamičnosti koja je prisutna u implementaciji mikroservisne arhitekture ovo može biti izazov. Pored pristupa, potrebno je i kontrolisati saobraćaj upućen od strane klijenta. *Gateway* služi kao jedinstvena ulazna tačka za mikroservise i sve vrste klijenata. On opslužuje sve zahteve i preusmerava, rutira ili balansira do relevantne instance mikroservisa, na osnovu jedinstvene uloge i specifičnih svojstava svakog tipa mikroservisa.

Da bi obezbedio javni *API*, *Gateway* mora na pouzdan način da odredi lokacije i proveri dostupnost mikroservisa. Zbog toga u mikroservisnom ekosistemu uvek postoji implementiran neki mehanizam pronalaženja i logičkog imenovanja mikroservisa. *Gateway* ima i sigurnosnu ulogu jer pruža mogućnosti za validaciju i autorizaciju sadržaja *HTTP* zahteva koje obrađuje. Pruža i mehanizme za otklanjanje grešaka u slučajevima kada se prosleđeni *HTTP* zahtev ispostavi nevalidan.
Uz postojanje *Gateway* komponente mikroservisi interno mogu koristiti jedan optimizovan binarni protokol ili mehanizame za sinhronu i asinhronu komunikaciju i na taj način garantovati konstantnu isporuku.

3. *SPRING BOOT ADMIN* PROJEKAT

Spring Boot Admin predstavlja veb aplikaciju koja je nastala kao proizvod potreba *open source* zajednice. Njena glavna svrha jeste da obezbedi način za registrovanje i nadgledanje komponenti u arhitekturama kao što je mikroservisna arhitektura. Ona takođe rešava i problem modifikacije konfiguracija samih komponenti bez zaustavljanja njihovog rada. Pruža interfejs putem koga je moguće izvršavati akcije nadgledanja i upravljanja. *Spring Boot Admin* je prilagođen za rad sa ostalim *SpringBoot* aplikacijama. *SpringBoot* aplikacije su aplikacije nastale primenom *SpringBoot* okvira za razvoj *Java* baziranih aplikacija. *SpringBoot* okvir sadrži sveobuhvatnu infrastrukturnu podršku za razvoj mikroservisa [2]. *SpringBoot* aplikacije putem aktuator krajnjih tačaka pružaju informacije koje su neophodne za rad *Spring Boot Admin* projekta. Aktuator krajnje tačke proširuju funkciju aplikacija izlaganjem određenih informacija o aplikaciji, koje je potom moguće upotrebiti za svrhe upravljanja aplikacijom. Informacije koje

aktuator krajnje tačke pružaju mogu se iskoristiti u nadgledanju aplikacije, prikupljanju metrika, razumevanju *HTTP* prometa ili stanja baze podataka. Metrike se na primer mogu odnositi na iskorišćenost memorije u *Java* virtuelnoj mašini, iskorišćenost procesora, broj trenutno otvorenih datoteka u aplikaciji i slično.

4. PODRŠKA ZA *SPRING CLOUD GATEWAY* KOMPONENTU

Gateway komponenta kontroliše tok spoljnog *HTTP* saobraćaja na ulasku u sistem. U narednim potpoglavljima predstavljeni su entiteti kojima se *gateway* konfiguriše, njihova uloga i načini za njihovo podešavanje. Na osnovu postavljenih vrednosti ovih entiteta *gateway* komponenta bazira svoje ponašanje. Predstavljena su i proširenja dodata u trenutnu implementaciju *Spring Boot Admin* projekta koja omogućavaju manipulaciju nad *gateway* komponentom.
Ova proširenja podrazumevaju stranicu u projektu *Spring Boot Admin*, koja nudi načine za pregled i manipulaciju konfiguracija *gateway* komponente. Ovo je prva implementacija ovog tipa u oblasti mikroservisa i *gateway* komponente, tako da u određenoj meri olakšava dosadašnji način rada.

4.1. Rute i filteri

Svaka *gateway* komponenta u svojoj konfiguraciji sadrži listu ruta. Lista ruta je presudna za donošenje odluke koji zahtev treba proslediti u sistem, a koji ne. Ruta se sastoji od imena za identifikaciju, predikata rute, filtera i redosleda izvršavanja. Predikat rute predstavlja vrednost u vidu regularnog izraza. Filteri rute predstavljaju validacije i transformacije *HTTP* zahteva koje je potrebno izvršiti prilikom obrade zahteva koji pristiže na *gateway* komponentu. Filteri mogu biti lokalni ili globalni. Lokalni filteri su vezani za definiciju rute. Globalni filteri su nezavisni od odabira rute i izvršavaju se nad svakim pristiglim zahtevom.
Kada zahtev od klijenta stigne do *gateway* komponente, u prvom koraku se nad zahtevom izvršava set definisanih globalnih filtera. Oni na primer mogu vršiti sigurnosne provere, kao što je provera da li zahtev u zaglavlju sadrži sve neophodne informacije ili token sa pravima.
Filteri na osnovu ovih i sličnih drugih validacija u određenim situacijama mogu transformisati zahtev, proslediti zahtev dalje na obradu ili vratiti odgovor sa upozorenjem ili greškom. U drugom koraku *gateway* komponenta nad prosleđenim zahtevom izvršava poređenje URL-a iz zahteva sa predikatima svake od ruta u definisanoj listi ruta.
Odabira se prva ruta čiji jedan predikat ispunjava uslov. Poredi se regularni izraz iz predikata rute sa vrednošću *URL*-a iz zaglavlja zahteva. Iz razloga što postoji ovakav vid poređenja, postoji i podatak o redosledu izvršavanja u ruti. Na osnovu ove vrednosti se utiče na prioritete među rutama.
Nakon što je ruta odabrana nad zahtevom se izvršava niz njenih definisanih lokalnih filtera. Lokalni filteri vrše transformacije nad vrednostima u zahtevu i pripremaju ga za dalje prosleđivanje i finalnu realizaciju. Nakon ovog koraka, ukoliko je sve prošlo uspešno, dati zahtev zaista stiže na odredište i izvršava željenu poslovnu logiku u nekoj od instanci servisa.

4.2. Dodavanje i brisanje definicije rute

S obzirom da definicije ruta predstavljaju vrlo dinamičan podatak, koji je sklon čestim izmenama, javila se potreba za implementacijom rešenja koje bi omogućilo da se ovaj resurs menja dok je instanca servisa aktivna. Da bi ovako nešto bilo moguće, *gateway* komponenta mora imati podršku za direktnu izmenu konfiguracija. Ova podrška je omogućena dodavanjem aktuator krajnjih tačaka.
U ovom slučaju one pružaju mogućnost dodavanja i brisanja trenutnih definicija ruta na instanci *gateway* servisa. Na ovaj način se izbegava potreba za eksternom izmenom konfiguracije i restartom instance servisa radi osvežavanja verzije. Dodavanje definicije rute je omogućeno unosom podataka u *JSON* formatu koji predstavlja definiciju jedne rute sa svim neophodnim poljima.
Nakon uspešno izvršene akcije dodavanja, lista sa rutama koja se prikazuje na stranici se automatski osvežava i prikazuje novododatu rutu. Brisanje se može izvršiti takođe u listi ruta. Rezultati obe akcije se gotovo istovremeno primenjuju i na samu komponentu i menjaju način njenog rada. Iz ovog razloga se eliminiše potreba za direktnim izmenama u programskom kodu servisa, kreiranjem nove verzije servisa i zaustavljanjem rada instance servisa radi osvežavanja trenutne verzije novom verzijom.

4.3. Poređenje performansi

Razlika koja se postiže uvođenjem podrške za *gateway* komponentu ogleda se u izmeni koraka koje je neophodno izvršiti da bi se određena promena konfiguracije primenila na instanci *gateway* servisa. Uz mikroservisnu arhitekturu obično se primenjuje neka od modernih metodologija za razvoj i isporuku softverskih rešenja. Ovakve metodologije nalažu da postoji jasno definisana sekvenca procesa koje je neophodno izvršiti da bi se neka promena propagirala na radno okruženje. Pod radnim okruženjem podrazumeva se funkcionalan mikroservisni ekosistem koji je pokrenut na serverima.
Sekvenca procesa koju je potrebno izvršiti zavisi od projekta do projekta. U idealnom slučaju bilo bi dobro u potpunosti zaobići izvršavanje ovih procesa, što implementacija podrške za *gateway* komponentu u vidu stranice u *Spring Boot Admin* projektu i omogućava. U minimalnom skupu ovi procesi podrazumevaju kompajliranje i izgradnju datoteka sa programskim kodom nad kojim se vrše izmene, zatim pokretanje skupa testova nad uspešno izgrađenim programskim kodom, postavljanje nove verzije servisa na repozitorijum sa verzijama i finalno proces zaustavljanja rada trenutne verzije instance servisa na radnom okruženju i postavljanje i pokretanje nove. Svaki od ovih procesa ima određeno trajanje koje zavisi od obimnosti servisa, jačine hardvera na kom se izvršavaju procesi i veličine i kompleksnosti skupa testova. Skup testova može podrazumevati *unit* testove, integracione testove, *end-to-end* testove i druge [3].
Za izvršavanje testova neophodno je imati osposobljen i pokrenut čitav kontekst aplikacije. Zbog toga se česte izmene u konfiguraciji ne isplate s obzirom na vreme koje je potrebno da se primene i vreme koje prođe, a da blokiraju i stvaraju zavisnost u radu sistema.
Zavisnost se stvara tako što se privremeno zaustavlja rad trenutne instance *gateway* servisa, radi osvežavanja

novom. Testiranje u slučaju ovakvih promena nema smisla, jer se u testovima pokriva funkcionalni deo aplikacije, a konfiguracije se svakako izostavljaju ili se koristi specifičan set testnih konfiguracija. Za potrebe izvršavanja velikog dela procesa uključenih u proceduru izmene konfiguracija na gateway komponenti pre implementacije podrške, korišten je alat za automatizaciju *Jenkins*. *Jenkins* je alat koji pomaže u automatizaciji procesa koji se izvršavaju tokom razvoja softvera.

Ovi procesi podrazumevaju izgradnju projekta, testiranje, *deploy* i *release* proces. *Deploy* proces podrazumeva zaustavljanje rada trenutne instance servisa na radnom okruženju i pokretanje nove instance sa novom verzijom. *Release* proces podrazumeva kreiranje nove izvršive verzije servisa i njeno postavljanje na repozitorijum sa verzijama. U nekim slučajevima ovi procesi se izvršavaju manuelno, ali za potrebe mikroservisne arhitekture u kojoj je izvršavanje ovih procesa dosta učestalo, neophodna je primena nekog od alata za automatizaciju. *Jenkins* je pokrenut na zakupljenoj *AWS* (*Amazon Web Services*) instanci servera. Tip *AWS* instance servera je *c4.2xlarge*, tip procesora *2.9 GHz Intel Xeon E5-2666 v3*, sadrži 8 *CPU-a* i 15 *GB RAM* memorije. *Gateway* servis zauzima 68.9 *MB*, sadrži 5 integracionih i 158 *unit* testova.

Za izmenu konfiguracije rute i propagiranje promene na radno okruženje pre implementacije podrške za *gateway* komponentu, izvšeni su sledeći koraci: izmena datoteke sa konfiguracijama i postavljanje na git repozitorijum (2 minute i 15 sekundi), izgradnja programskog koda projekta (21 sekunda), vreme izvršavanja testova (2 minute i 7 sekundi), kreiranje izvršive *jar* datoteke i postavljanje na repozitorijum sa verzijama (40 sekundi), *deploy* (1 minut i 5 sekundi), pokretanje servisa i registracija instance na *discovery* servis (41 sekunda).

Ukupno trajanje procesa izmene konfiguracije rute je 7 minuta i 9 sekundi. Za izmenu konfiguracije rute i propagiranje promene na radno okruženje nakon implementacije podrške za *gateway* komponentu, izvršeni su sledeći koraci: otvaranje stranice sa podrškom za *gateway* komponentu (10 sekundi), dodavanje nove rute kroz formu na stranici (42 sekunde). Vreme potrebno da se izmeni konfiguracija rute i da se ta izmena zaista primeni na radnom okruženju je 52 sekunde. Postignuta je optimizacija što se tiče broja neophodnih koraka i vremena koje je potrebno da se izvrše zadaci prilikom izmene definicije rute na *gateway* servisu.

Nakon implementacije podrške zadatak je moguće izvršiti približno sedam puta brže nego ranije. Pored ove razlike takođe treba napomenuti da je u mikroservisnom ekosistemu svaki trenutak prestanka rada neke od bitnih instanci servisa zapravo blokirajući za čitavo radno okruženje. S obzirom da je u datom primeru reč o servisu koji u potpunosti kontroliše dolazni saobraćaj u sistemu, to znači da prestanak rada ove instance stvara i privremeni prestanak rada čitavog sistema. Iz tog razloga je korisna implementacija rešenja koje pruža način da se isti zadatak izvrši bez obustavljanja rada bilo koje instance.

5. ZAKLJUČAK

Na osnovu rezultata dobijenih prilikom testiranja performansi rešenja dolazi se do zaključka da stranica za upravljanje *gateway* komponentom može imati široku primenu u sistemima sa mikroservisnom arhitekturom. Pruža načine za brži rad prilikom nadgledanja podataka i smanjuje broj koraka za izvršavanje određenih akcija. Olakšava način za sticanje uvida u trenutno stanje i definiciju globalnih filtera i omogućava pregled i modifikaciju definicija ruta.

Pored toga, modifikacije primenjene na definicijama ruta se propagiraju na instancu servisa istog trenutka, dok je instanca servisa i dalje pokrenuta. U potpunosti se gubi potreba za izmenom konfiguracija u programskom kodu, pravljenjem nove verzije servisa i zaustavljanjem rada trenutne instance radi osvežavanja novom verzijom. Proces modifikacije definicije rute je ubrzan za sedam puta, a da se pri tome nikad ne zaustavlja rad instance *gateway* servisa. Isto se odnosi i na akciju brisanja definicije rute, koja takođe ne zahteva osvežavanje verzije instance servisa i samim tim ne blokira izvršavanje čitavog mikroservisnog ekosistema. Novi pravci razvoja podrazumevaju unapređenje dela *gateway* stranice zaduženog za unos nove definicije rute. Način unosa definicije rute može ponuditi formu sa boljom strukturom, koja je robustna i predlaže korisnicima moguće podatke za unos.

Deo stranice sa prikazom globalnih filtera je moguće proširiti tako da omogućava i detaljniji pregled svake od definicija globalnih filtera. Na ovaj način bi se sprečila potreba za pretraživanjem programskog koda da bi se izvršio pregled određenog globalnog filtera. S obzirom da je upotrebom aktuator krajnjih tačaka moguće saznati informacije o trenutno dostupnim krajnjim tačkama na bilo kom od servisa, ovaj podatak bi se mogao iskoristiti u implementaciji novog proširenja koje bi omogućilo korisnicima da na još brži način dodaju novu definiciju rute. Sistem bi mogao na osnovu postojećih definicija ruta, trenutnog stanja sistema i podataka o krajnjim tačkama servisa koje prikupi upotrebom aktuatora da predvidi neke od mogućih novih definicija ruta i da ih predloži korisniku. Ovo proširenje bi moglo biti od velike koristi jer je sam proces unosa novih definicija ruta obično redundantan i podložan korisničkim greškama.

6. LITERATURA

[1] Chris Richardson, *Microservices Patterns*, 1st *edition*, Manning Publications, 2018.

[2] John Carnell, *Spring Microservices in Action*, 1st *edition*, Manning Publications, 2017

[3] Dinesh Rajput, *Hands On Microservices – Monitoring and Testing*, Packt Publishing, 2018

Kratka biografija:

kontakt:
stkosijer@gmail.com

Stevan Kosijer rođen je 24.9.1994. godine u Vojniću, Hrvatska. Živi u Novom Sadu, završio je Osnovnu školu „Đorđe Natošević“ u Novom Sadu. Društveni smer gimnazije „Isidora Sekulić“ završio takođe u Novom Sadu. Godine 2013. upisuje Fakultet tehničkih nauka u Novom Sadu, na studijskom programu Računarstvo i automatika. Godine 2017. uspešno diplomira i nastavlja sa master studijama na istom fakultetu. Položio je sve ispite propisane planom i programom.