Razvoj web aplikacije za agregaciju ponude studentskih restorana u mikroservisnoj arhitekturi

Ferenac, Teo

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

Teo Ferenac

Razvoj web aplikacije za agregaciju ponude studentskih restorana u mikroservisnoj arhitekturi

Završni rad

Pula, rujan, 2022. godine

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

Teo Ferenac

Razvoj web aplikacije za agregaciju ponude studentskih restorana u mikroservisnoj arhitekturi

Završni rad

JMB: 0303092166, redoviti student

Studijski smjer: Računarstvo

Predmet: Dinamičke web aplikacije

Znanstveno područje: Tehničke znanosti
Znanstveno polje: Računarstvo
Znanstvena grana: Informacijski sustavi
Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan, 2022. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani _____, kandidat za prvostupnika _____ ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, _____ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom

_____

_____

koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____

Potpis

_____

Sažetak

U ovom radu opisat će se aspekti mikroservisne arhitekture, *scraping* podataka i agregacije podataka. Objasnit će se kako mikroservisi rade i kako se razlikuju od monolitne arhitekture, mane i prednosti istih te kad je bolje koju koristiti i zašto. Isto tako, opisat će se što su *scraping* i agregacija podataka, kad se koriste, kako i zašto. Također će se pokazati kako napraviti implementaciju mikroservisne arhitekture pomoću dijagrama slučaja uporabe i klasnog dijagrama.

Github link: Teo1121/MenzaAPI (github.com)

Ključne riječi: mikroservisi, mikroservisna arhitektura, web aplikacija, scraping podataka, agregacija podataka

Summary

This paper will cover the concepts of microservices, data scraping and data aggregating. It will explain how microservices work and how they differ with monolith architecture, when to use which and why. Likewise, it will cover what data scraping and data aggregating is, when it is used and why. It will also demonstrate how to make a working application using the microservice architecture, use case and class diagrams.

Github link: Teo1121/MenzaAPI (github.com)

Key words: microservices, microservice architecture, web application, data scraping, data aggregation

Sadržaj

1. Uvod

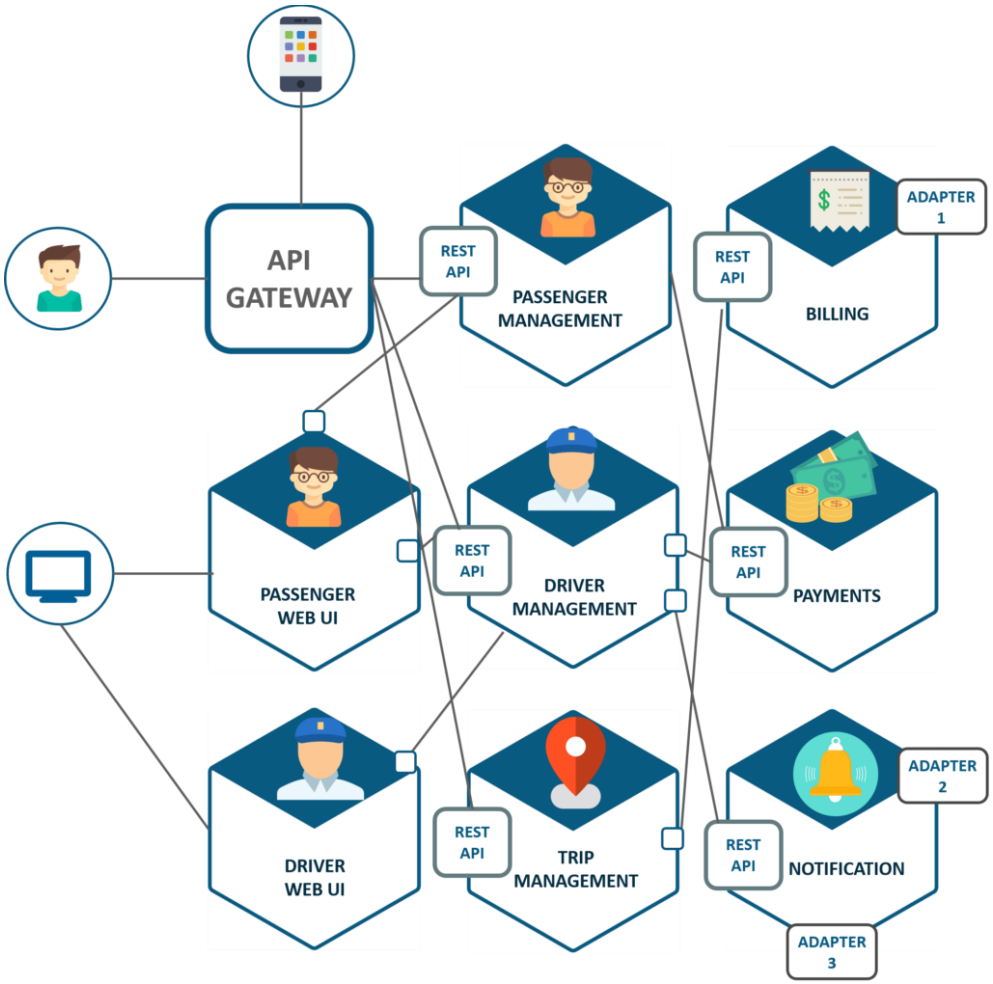
Termin mikroservisa je prvi put opisao dr. Peter Rodgers na svojoj prezentaciji o računarstvu u oblaku 2005. godine. Protivio se konvencionalnom razmišljanju razvijanja softvera i dao je ideju mikro-web servisa. No, prva implementacija mikroservisa se dogodila 1997. godine kad je IBM objavio *Enterprise Java Bean* ili EJB. To je mali servis koji se koristi u izgradnji modularnog poslovnog softvera. No bio je kompliciran, težak za pretragu i otklona greški i najvažnije od svega, dostupan samo u Java programskom jeziku. Zbog toga se razvila servisno orijentirana arhitektura ili SOA čiji je cilj razvoj samostalnih servisa pomoću različitih programskih jezika. Mikroservisna arhitektura je podskup servisno orijentirane arhitekture koja se prvi put pojavila 2011. godine na konferenciji softverskih arhitektura.

Osim mikroservisa, druga popularna arhitektura je monolitna. Ona je prevladavala razvojem poslovnog softvera te je i danas vrlo dobar izbor za njihov razvoj. Postoje mnoge razlike između njih, a u ovom radu opisat će se koje su i za koji projekt je bolja koja arhitektura.

Osim tih razvojnih arhitektura važno je znati kako se prikupljaju i agregiraju podaci s interneta. Prvi softver za prikupljanje podataka s interneta bio je *web crawler*, to jest softver koji prolazi kroz sve web stranice da bi ih indeksirao. Godine 1993. je prvi *web crawler* krenuo s indeksiranjem World Wide Weba. Uz njegovu pomoć je nastao prvi web pretraživač temeljen na *crawler* indeksiranju zvan JumpStation. Tek nakon razvoja BeautifulSoup biblioteke 2004. godine je *web scraping* postao dostupniji svim developerima. BeautifulSoup je Python biblioteka koja parsira HTML kod u strukture bolje razumljive za programere.

2. Mikroservisi

Mikroservisi su nezavisno postavljeni servisi definirani za rješavanje specifičnog zadatka. To znači da svaki mikroservis radi na svom zadatku bez potrebe rada drugog mikroservisa. Na taj način mogu se raditi izmjene na jednom mikroservisu bez potrebe ponovnog postavljanja cijelog sustava. Skup mikroservisa koji zasebno rade na rješavanju zajedničkog cilja zove se mikroservisna arhitektura.



Slika 1 Primjer mikroservisne arhitekture

Na slici 1 prikazan je primjer mikroservisne arhitekture za upravljanje transportnim sustavom. Svaki zadatak kod upravljanja takvim sustavom je razdvojen i pojednostavljen s ciljem da se, neovisno o drugim servisima, implementira mikroservis za rješavanje tog zadatka. Naravno neki mikroservisi ne mogu u potpunosti riješiti svoj zadatak bez vanjskih informacija drugih mikroservisa. U tim situacijama potrebno je implementirati komunikaciju između mikroservisa. Da bi se

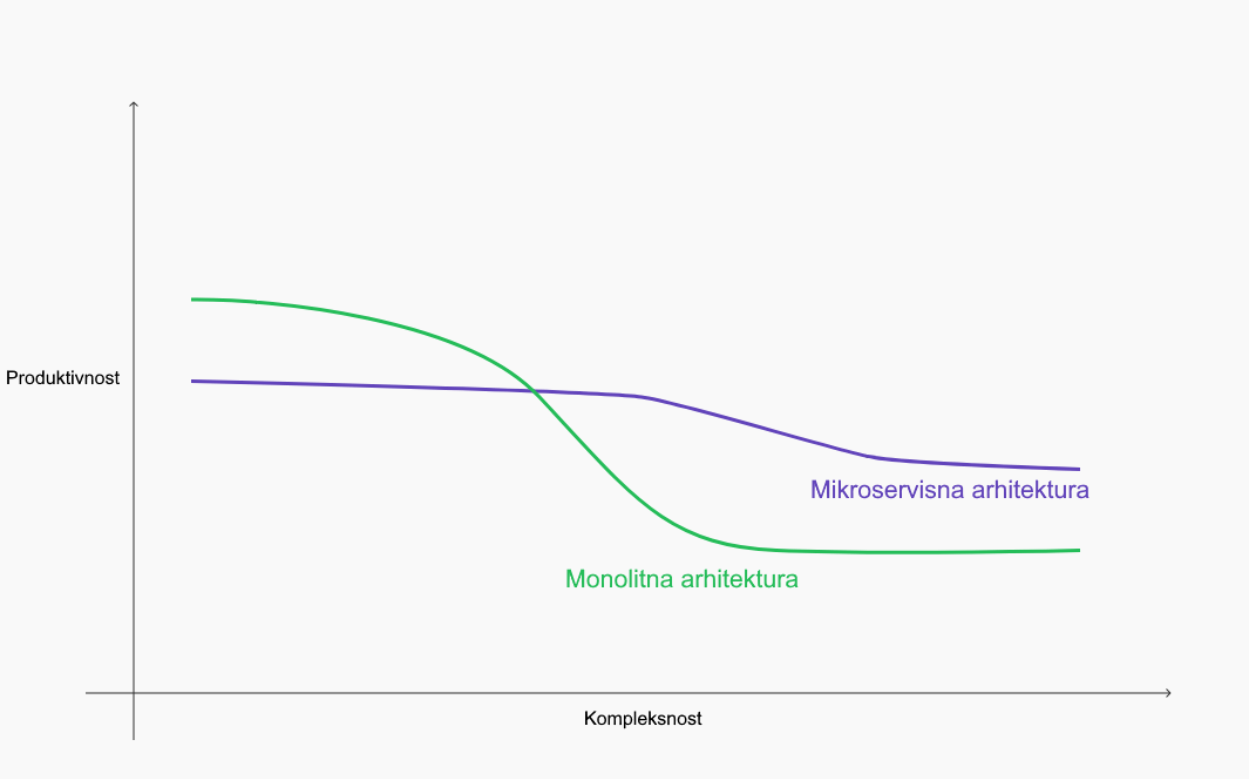
sačuvala nezavisnost mikroservisa implementira se *loosely coupled* ili labavo spojena komunikacija mikroservisa. Pojednostavljeno, znaci da izmjenom jednog mikroservisa ne mora se mijenjati druge mikroservise koje moraju komunicirati s njim.

3. Monolitna arhitektura

Prije populariziranja mikroservisne arhitekture, za gotovo svaki projekt se koristila monolitna arhitektura. Monolitna arhitektura je definirana kao jedna izvedba servisa u kojem su spojene sve komponente potrebe za uspješno obavljanje rada. Za razliku od mikroservisa, komponente monolita moraju biti pisane istom tehnologijom jer se zajedno nalaze i pokreću u jednoj aplikaciji. To znači da komponente monolita mogu direktno zvati metode drugih komponenti bez potrebe za komunikacijom preko mreže. Međutim, ukoliko se želi napraviti promjena na jednoj komponenti potrebno je ponovno kompiliranje i pokretanje cijelog sustava. To čini monolitnu arhitekturu odličnim izborom za manje kompleksnije probleme kao što se može vidjeti na slici 2. Mikroservisna arhitektura je također dobar izbor za zadatke niske kompleksnosti, ali više je ciljana na rješavanje kompleksnijih zadataka s većom mogućnošću proširenja. Prednost monolitne arhitekture naprema mikroservisnoj jest njena jednostavnija implementacija, manja latencija zbog direktne komunikacije komponenti i lakše zapisivanje greški svih komponenti zajedno.

Mnoge firme još koriste monolitnu arhitekturu. Postojeće firme ne žele napraviti izmjenu cijelog postojećeg monolitnog sustava na novu arhitekturu ako postojeći sustav radi ispravno s manjim poteškoćama kod dogradnje novih značajka, a razvoj novog sustava bi oduzeo puno vremena i kapitala firmama. Također, nove firme nemaju dovoljno kapitala za izradu sustava pomoću mikroservisne arhitekture koja zahtijeva vise posla za razvoj funkcionalnog proizvoda naprema monolitnoj arhitekturi.

Unatoč tome postoje mnoge firme s dovoljno kapitala i potrebom za proširenje koje su spremne uložiti u ažuriranje sustava iz monolitne u mikroservisnu arhitekturu.



Slika 2 Ovisnost produktivnosti i kompleksnosti (Microservices Architecture: An Introductory Guide - Edvantis)

4. Ekstrakcija i agregacija podataka

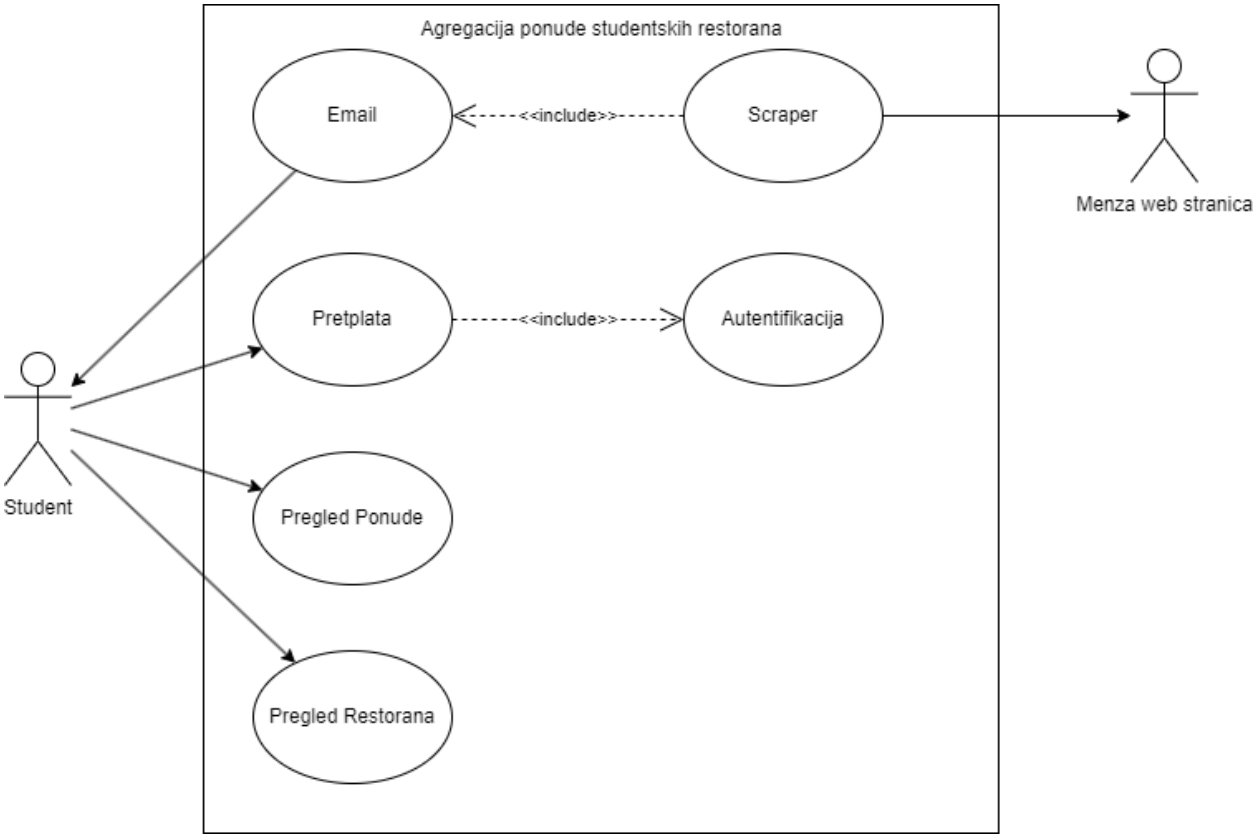
Web ekstrakcija podataka ili *web scraping* je proces izvlačenja informacija s web stranica pomoću softvera. Kako bi dobio tražene podatke šalje zahtjev web serveru za HTML izvorni kod web stranice. Ako ga uspješno dobije, pomoću HTML *tagova* izdvaja i preuređuje podatke da dobije tražene informacije. U slučaju da se treba *scrapati* više stranica s istog web servera, *scraper* će zatraži izvorni kod svih tih stranica i ograničiti širinu propusnog pojasa web serveru. Zbog toga mnogi web serveri blokiraju *scrapere* pomoću automatske detekcije botova ili CAPTCHA provjerom.

Kod potražnje podataka s više izvora dolazi do potrebe njihovog zajedničkog spajanja u bazi podataka, to jest agregacije. Naravno podaci s različitih izvora neće biti istog formata zbog čega dolazi do potrebe za uniformno definiranu vrstu u koju se

mogu formatirati podaci sa svih izvora. U slučaju da jednom izvoru fali neki podatak koji je dostupan u drugim izvorima, taj se podatak zapisuje kao prazan, to jest nepostojeći.

5. Model slučaja uporabe

Za lakši razvoj projekta izrađen je dijagram slučaja uporabe (engl. *use case diagram*) koji prikazuje klijente i njihove moguće interakcije s aplikacijom.



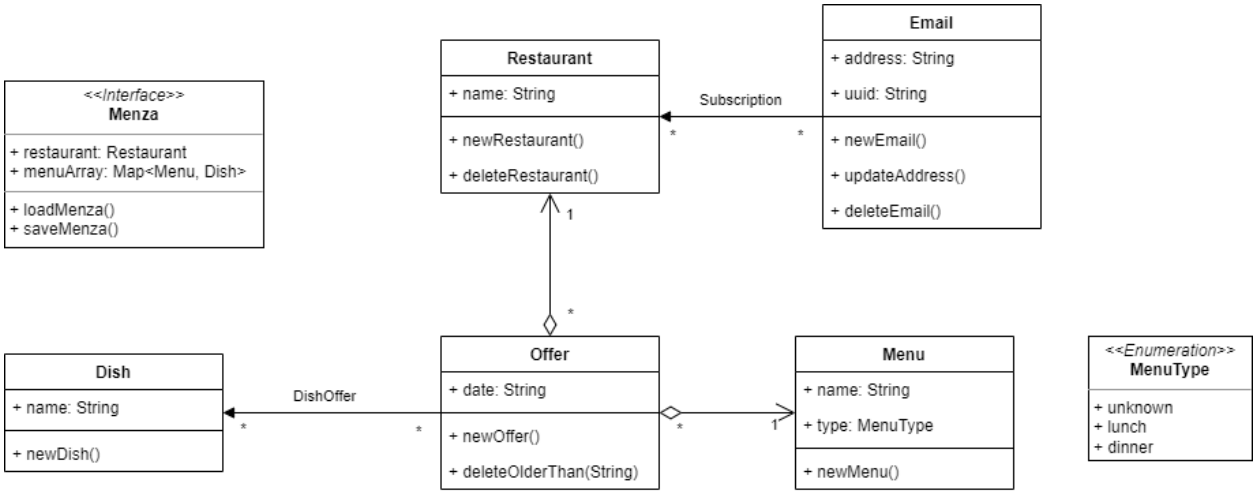
Slika 3 Dijagram slučaja uporabe

Kao što se može vidjeti na slici 3 aplikacijom se služe dvije vrste korisnika. Student može od aplikacije zatražiti popis svih dostupnih restorana, pregled najnovije ponude traženog restorana i preplatiti se na nove ponude odabranog restorana. Kod preplate potrebno je da se student autentificira s jedinstvenom lozinkom koju stvori kad se prvi put prijavi, to jest registrira. Registracija je jedino potrebna ako se student želi prijaviti na pretplatu novih ponuda putem emaila. *Scraper* servis periodično traži promjene u ponudi restorana. U slučaju promjene, ona se bilježi u bazi podataka i prosljeđuje emailu da pošalje mail svim preplaćenim korisnicima. Za implementaciju

više restorana samo se treba stvoriti novi *scraper*, a u rijetkim slučajevima moguće je da isti *scraper* komunicira s više restorana jer dijele sličan dizajn web stranice. Zbog tog jednostavnog načina nadogradnje novih menzi je korištena mikroservisna arhitektura.

6. Klasni model

Za lakšu vizualizaciju rada sustava sastavljen je klasni dijagram (engl. *class diagram*) u kojem se nalaze reprezentacije klasa s atributima i metodama korištenim u projektu.



Slika 4 Klasni dijagram

Email klasa reprezentira korisnika. Atribut address sadrži email adresu korisnika na koju se šalju mailovi, a uuid je jedinstvena oznaka korisnika kojom se on i samo on služi za prijavu i odjavu pretplate novih ponuda restorana. Metode služe za stvaranje novog te brisanje i ažuriranje postojećeg Email zapisa.

Klasa Restaurant reprezentira restoran na koji se korisnik može preplatiti. Vezan je s Email klasom kroz pomoćnu Subscription klasu koja sadržava jedan par identifikatora za Restoran i Email.

Dish je klasa koja reprezentira jedno jelo. U slučaju da postoji jedno jelo kroz više restorana bilježi se samo jedno te se isto koristi za sve restorane.

Da bi menza zadovoljila veći broj studenata često nudi vise menija. Klasa Menu reprezentira te menije. Atribut type označava je li meni namijenjen za ručak ili večeru. Zbog mogućnosti da restoran ne nudi opciju razlikovanja ručka i večere, npr.

cjelodnevni meni, umjesto booleana se koristi popisivač (engl. *enumerator*) MenuType koji u sebi ima dodatnu vrijednost unknown.

Offer klasa veže meni s restoranom i skupom jela koristeći pomoćnu DishOffer klasu. U njoj se zabilježuje vrijeme dobivanja novih podataka te se s njom mogu pregledati, sortirati i izbrisati ponude.

Menza u ovom dijagramu nije klasa već interface namijenjen preoblikovanju ponude u JSON format pomoću kojeg mikroservisi lakše komuniciraju međusobno i s korisnikom.

7. Korištene tehnologije

Zbog načina rada mikroservisne arhitekture svaki se mikroservis može implementirati pomoću različite tehnologije. Međutim, to je samo opcija izgradnje te se u praksi mikroservise jedne mikroservisne arhitekture pišu istom tehnologijom zbog lakšeg razvoja i zapošljavanja. U nastavku opisane su korištene tehnologije s razlozima i manama.

7.1. Programski jezik

Python se u današnjici mnogo razvio, te se koristi za izradu svakakvih rješenja za probleme svih razina teškoće. Od razvoja igara do strojnog učenja, Python je postao najpopularniji programski jezik na svijetu. Odličan je za početnike kao i za napredne znanstvenike s jednostavnom sintaksom i velikom zajednicom koja se stalno širi. Zbog tog razloga se on koristio kao glavni programski jezik u razvoju projekta.

Iako je Python dobar jezik za izradu mikroservisa postoje mane zbog kojih ga mnogo ljudi ne koriste. Najpoznatija mana jest njegova performansa, od deset najpopularnijih jezika Python je daleko najsporiji. Osim efikasnosti, Python također ima problem s konzumacijom radne memorije. Zbog njegove jednostavnosti implementirane su varijable bez vrste podataka što znatno utječe na količinu memorije potrebnu za izvođenje.

7.2. Baza podataka

Za bazu podataka koristi se SQLlite. Glavna razlika između SQLite-a i tipične SQL baze podataka kao mySQL je njihovo izvođenje. Baze podataka se tipično izvode zasebno kao jedan servis, dok je SQLite uključen u programu, te se zajedno s njim izvodi. Takav tip baze podataka je definiran kao ugrađena baza podataka (engl. *embedded database*), te se u projektu izvodi zajedno s ORM-om.

ORM ili *Object-relational mapping* je tehnika pretvorbe podataka pomoću objektno orijentirane programske arhitekture. To znači da se za spremanje i čitanje podataka ne mora koristiti specijalan upit, već se samo zove metoda s objektom, objekt se zatim spremi ili iščitani iz baze podataka, te se iščitani podaci vrate kao objekt ili kod pisanja podataka vraća potvrda da je objekt uspješno spremljen u bazu podataka.

SQLite je odličan za manje projekte s malim brojem modela, dolazi s Pythonom te ga je lagano implementirati i zauzima najmanje memorije od svih konkurentnih baza podataka.

Najveća mana i glavni razlog zašto se ne koristi u većim projektima je ta da ne podržava paralelno pisanje u bazu podataka. U slučaju više zahtjeva za pisanje može doći do velikog zastoja u bazi podataka. No, u projektu to nije značajno jer se velika količina podataka zapisuje samo kad se dobiju nove informacije od *scrapera*. Osim ukoliko se pokrene sto *scrapera* u isto vrijeme, ta mana gotovo nema utjecaj na izvoženje.

7.3. Komunikacija mikroservisa

Komunikacija između mikroservisa se izvodi pomoću jednog centralnog mikroservisa koji je zadužen za preusmjeravanje i po potrebi preoblikovanje zahtijeva, te bilježenje greški

Komunikacija se vrši pomoću gRPC-a. gRPC je Googleova implementacija RPC ili *Remote Procedure Call* okvira (engl. *framework*). Tim okvirom mogu se zvati metode drugih servisa skoro kao da su vlastite. Pomoću HTTP/2 transportnog protokola i Protocol Buffera za serijalizaciju podataka znatno je brzi od REST okvira.

Jedine su mu mane nedostatak zajednice i alata za testiranje HTTP/2 i Protocol Buffer protokola, kao što REST ima alat za testiranje zvan Postman.

7.4. Komunikacija s klijentom

Da se ograniči pristup unutarnjim metodama, kao što su pisanje i čitanje iz baze podataka, izrađeno je razdvojeno sučelje za komunikaciju s klijentom. Zbog HTTP/2 protokola koji još nije podržan na modernim web browserima te poteškoće kod testiranja gRPC-a, zamijenjen je s REST API. Iako je sporiji od gRPC-a, lakši je za implementirati pomoću Flask biblioteke te ime veću zajednicu i razne alate za testiranje.

8. Zaključak

Mikroservisi su u mnogim slučajevima bolja alternativa monolitima, no to ne znači da će se monolitna arhitektura prestati koristiti. Postoje projekti u kojima mikroservisna arhitektura nema prednost pred monolitnom, naprotiv monolitna arhitektura zna biti bolji izbor kod izrade mnogih aplikacija. Nema potrebe uvijek koristiti jednu arhitekturu samo zato što je ona sada u modi.

Kod izrade ozbiljne poslovne aplikacije, od velike je važnosti izrada potrebnih dijagrama za lakšu i kvalitetniju izradu traženog rješenja. One usmjeravaju developere kako bi krajnji proizvod trebao raditi i izgledati. Bez dobro definiranog plana za izradu projekta teško će developeri izraditi kvalitetan softver.

Literatura

Newman, Sam. *Monolith to microservices: evolutionary patterns to transform your monolith*. O'Reilly Media, 2019.

Kalske, Miika, Niko Mäkitalo, and Tommi Mikkonen. "Challenges when moving from monolith to microservice architecture." In *International Conference on Web Engineering*, pp. 32-47. Springer, Cham, 2017.

Newman, Sam. *Building microservices*. O'Reilly Media, 2021.

Monolith vs Microservices architecture: Split or not to split | Brocoders blog about software development, 4.9.2022

Nadareishvili, Irakli, Ronnie Mitra, Matt McLarty, and Mike Amundsen. *Microservice architecture: aligning principles, practices, and culture*. " O'Reilly Media, Inc.", 2016.

How to Make Microservices Communicate - DZone Microservices, 4.9.2022

Web Scraping History: The Origins of Web Scraping (scrapingrobot.com) 5.9.2022

Popis slika