Razvoj aplikacije u oblaku uz pomoć Spring Cloud programskog paketa

Janković, Mihovil

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAZVOJ APLIKACIJE NA OBLAKU UZ POMOĆ SPRING CLOUD PROGRAMSKOG PAKETA

Završni rad

Mihovil Janković

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Mihovil Janković
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina	R 4358, 22.07.2019.
OIB Pristupnika:	65381733880
Mentor:	doc. dr. sc. Tomislav Galba
Sumentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Razvoj aplikacije u oblaku uz pomoć Spring Cloud programskog paketa
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Potrebno je opisati što je Spring programski paket i na primjeru jednostavne arhitekture implementirati elemente Spring Clout komponente kao što su API poveznik (eng. API Gateway), konfiguracije (eng. Configuration), Spring Boot aplikaciju, pronalazak servisa (eng. Service discovery) itd.
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Mihovil Janković
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. studenta, godina upisa:	R 4358, 22.07.2019.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj aplikacije u oblaku uz pomoć Spring Cloud programskog paketa**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Galba

i sumentora izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. Uvod

Programerska industrija doživjela je značajan rast posljednjih godina, a modernizacija svakodnevnog života dovodi do pojednostavljenja mnogih rutinskih zadataka. Osim što olakšava naše svakodnevne obaveze, ova transformacija također doprinosi lakšem pisanju programskog kôda i svega vezano s programiranjem. Tako u današnje vrijeme postoje aplikacije temeljene na oblaku (engl. *Cloud*) što pruža mnoge prednosti. Neke od glavnih prednosti su smanjivanje troškova održavanja aplikacije i smanjenje upletenosti upravljanja programskim kôdom, također transparentne su korisnicima i pružateljima resursa. Ideja aplikacije se zasniva na tome da koristeći *Spring* programski okvir i funkcionalnosti iz *Spring Cloud* programskog paketa korisnik upozna načine korištenja tih elemenata. Aplikacija predstavljena u završnom radu ugrubo se sastoji od 3 dijela. Prvi dio koji služi za dohvaćanje podataka sa vanjskog API-ja. Drugi dio koji služi za procesuiranje podataka, spremanje podataka u bazu podataka i za pristup podatcima iz baze podataka. Treći dio služi za komunikaciju između svih dijelova aplikacije. U radu je objašnjen način izrade jednostavne aplikacije korištenjem Java programskog jezika i programskog okvira *Spring* te korištenje *Spring Cloud* programskog paketa. U prvom dijelu rada opisat će se korištene tehnologije koje će također biti uspoređene s postojećim alternativama. U drugom dijelu rada opisuju se alati koji su odabrani za stvaranje ovog programskog rješenja. Zaključno, detaljno je opisano programsko rješenje te je prikazan rezultat pravilnog rada programa.

1.1. Zadatak završnog rada

Potrebno je opisati što je *Spring* programski paket i na primjeru jednostavne arhitekture implementirati elemente *Spring Cloud* komponente kao što su API poveznik (eng. API *Gateway*), konfiguracije (eng. *Configuration*), *Spring Boot* aplikaciju, pronalazak servisa (eng. *Service discovery*) itd.

2. Pregled područja teme

2.1. Slična rješenja

U ovom potpoglavlju navest će se i usporediti proizvodi koji koriste neke od tehnologija potrebne za izradu ovog rada. Aplikacija opisana u ovom radu ima specifične zahtjeve te je problematično pronaći proizvode koji koriste tehnologije koje se potpuno podudaraju sa ovim radom. Stoga, dani proizvodi su uspoređeni sa aplikacijom po značajkama koje su iste i u proizvodu i u aplikaciji.

- Alibaba - *Alibaba Group*, ogromni kineski konglomerat e-trgovine, koristi niz najsuvremenijih tehnologija, uključujući arhitekturu mikroservisa i *Spring* programski okvir. Ove tehnologije poboljšavaju skalabilnost, modularnost i održivost njihove platforme. Kao i aplikacija opisana u radu, e-trgovina *Alibaba* izgrađena je na arhitekturi mikroservisa. Ova arhitektura omogućuje dekompoziciju sustava na manje, neovisne servise koje komuniciraju putem API-ja, što dovodi do bolje fleksibilnosti i skalabilnosti. *Alibaba* također koristi *Spring* programski okvir za različite dijelove svojih aplikacija. *Alibaba* koristi mehanizme otkrivanja usluga slične našoj upotrebi Eureka poslužitelja. Otkrivanje usluga ključno je u arhitekturi mikroservisa kako bi se učinkovito locirale i komunicirale različite instance servisa.
- Zalando - Zalando, europska tvrtka za e-trgovinu koja se usredotočuje na modu i način života, također slijedi modernu praksu razvoja. Njihova platforma koristi arhitekturu mikroservisa i Spring tehnologije za podršku svojim operacijama u području online maloprodaje. Zalando-ova infrastruktura strukturirana je oko mikroservisa kao aplikacija opisana u završnom radu. Ovaj pristup omogućava im stvaranje manjih, specijaliziranih servisa koji se mogu neovisno razvijati i skalirati. Upotreba Spring Frameworka od strane Zalanda pruža svestrane alate za izgradnju i održavanje kompleksnih, skalabilnih aplikacija, što je ključno za veliku e-trgovinsku platformu.
- PayPal - I PayPal i aplikacija opisana u ovom radu koriste Spring programski okvir za izgradnju svojih *backend* usluga. Ova zajednička komponenta osigurava da obje aplikacije imaju koristi od Spring ekosustava, uključujući značajke poput ubacivanja ovisnosti, upravljanja transakcijama i aspektno orijentiranog programiranja. I jedna i druga aplikacija implementiraju arhitekturu mikroservisa. Ovaj arhitekturni pristup uključuje razbijanje monolitne aplikacije na manje, neovisne servise koje se mogu razvijati, implementirati i skalirati pojedinačno. To omogućuje fleksibilnost, jednostavno održavanje i skalabilnost. Dok se PayPal bavi financijskim transakcijama, korisničkim računima i poviješću transakcija, aplikacija opisana u radu usredotočuje se na prikupljanje, prijenos i pohranu

podataka o financijskim tržištima. Obe aplikacije zahtijevaju učinkovita rješenja za upravljanje podacima, uključujući integraciju s bazom podataka i obradu podataka. Obe aplikacije mogu koristiti *Eureku* za otkrivanje usluga.

2.2. Eureka server

Jedna od ključnih funkcionalnosti *Spring Cloud* programskog okvira je *Eureka* poslužitelj. Djeluje kao centralizirani registar koji prati sve dostupne usluge unutar sustava. Omogućuje uslugama da se registriraju na poslužitelju i pruža mehanizam za druge usluge da ih otkriju i komuniciraju s njima. Sve registrirane usluge pružaju informacije o jedinstvenom identifikatoru, *hostu i portu Eureka* poslužitelju. Usluge koje trebaju komunicirati s drugim uslugama mogu postaviti upit poslužitelju *Eureka* kako bi dobile lokaciju (*host* i *port*) željenih instanci usluge. Poslužitelj *Eureka* pruža registar dostupnih instanci usluga, što klijentima omogućuje dinamičko otkrivanje i povezivanje s njima [1].

Tehnologije koje se koriste umjesto *Eureka* poslužitelja su:

- HashiCorp Consul - Za razliku od *Eureka* poslužitelja koji je uglavnom korišten za *Java* aplikacije, *Consul* podržava više platformi, *Consul* pruža ugrađenu podršku za provjeru zdravlja dok se *Eureka* poslužitelj oslanja na *Spring Boot Actuator* za odrađivanje te provjere. *Eureka* poslužitelj koristi *Spring Cloud Config* za upravljanje konfiguracijom dok *Consul* to odrađuje samostalno.
- Apache ZooKeeper -*Eureka* poslužitelj ne pruža napredne značajke poput zaključavanja i praćenja događaja koje se nalaze u *ZooKeeperu. ZooKeeper* se koristi i za distribuirano zaključavanje i upravljanje konfiguracijom dok *Eureka* poslužitelj služi samo za registraciju i otkrivanje usluga. Za razliku od *ZooKeepera Eureka* poslužitelj ima jednostavnu arhitekturu i radi na osnovi *peer-to-peer* komunikacije.

2.3. Spring Cloud Gateway

Spring Cloud Gateway je komponenta unutar sustava koja se ponaša kao centralizirana pristupna točka za klijentske zahtjeve [2]. Prima zahtjeve od klijenata i prosljeđuje ih ostalim servisima unutar arhitekture. Dopušta definiranje pravila usmjeravanja direktnim zahtjeva specifičnim mikroservisima temeljenih na kriterijima kao URL obrasci, parametri upita, itd. Integrira mehanizme Service Discoverya kao što su Eureka ili Consul kako bi mogao dinamički otkriti potrebne mikroservise i usmjeriti zahtjeve prema njima. Koristeći Spring Cloud API Gateway postiže se pojednostavljenje klijentske arhitekture, poboljšava se skalabilnost aplikacije i potiče se centraliziranje sigurnosnih uvjeta preko više mikroservisa u nekom sustavu.

Tehnologije koje se koriste umjesto *Spring Cloud Gatewaya* su:

- *Netflix Zuul* - Za razliku od *Spring Cloud Gatewaya* koji koristi deklarativni pristup konfiguraciji putem YAML-a ili programskog kôda, *Netflix Zuul* koristi konfiguracijske datoteke na temelju *Groovyja* ili YAML za konfiguraciju usmjeravanja i filtriranja. Nadalje, *Spring Cloud Gateway* može koristiti reaktivne tehnologije poput *Reactora* i *Spring WebFlux-a* za obradu velikog broja zahtjeva u stvarnom vremenu dok *Zuul* nije optimiziran za zahtjeve sa visokim opterećenjem.
- *Nginx* - Nginx je web i proxy server koji se može koristiti za posredovanje web aplikacija, mikroservisa i statičkih datoteka dok je *Spring Cloud Gateway* specijaliziran za upravljanje prometom između mikroservisa. *Spring Cloud Gateway* se integrira sa *Java* aplikacijama dok se *Nginx* koristi sa aplikacijama neovisno o programskom jeziku.

2.4. Spring Cloud Config

Spring Cloud Config je komponenta *Spring Cloud* programskog okvira koja pruža vanjsko upravljanje konfiguracijom [3]. Omogućuje pohranu i upravljanje konfiguracijskim svojstvima za aplikaciju na centraliziranoj lokaciji. Aplikacije konfiguracijskim datotekama mogu dinamički pristupati tijekom izvođenja.

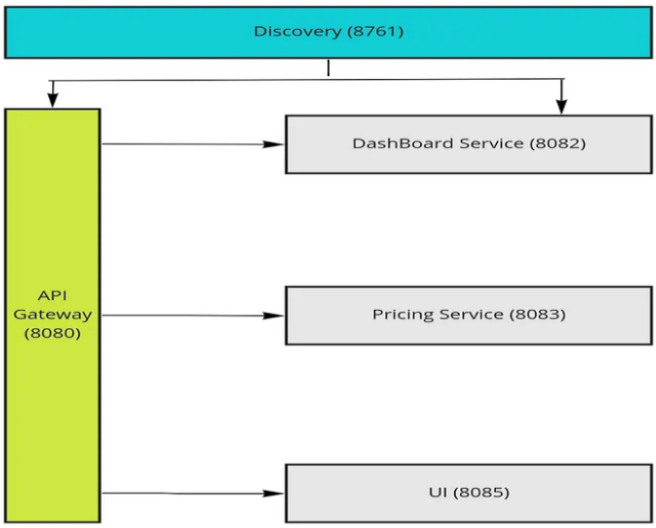
Tehnologije koje se koriste umjesto *Spring Cloud Configa* su:

- *Kubernetes ConfigMaps and Secrets* - Za razliku od Spring Cloud Configa koji je vanjski servis i integrira se s aplikacijama putem REST API-ja *Kubernetes ConfigMaps and Secrets* su dio *Kubernets* platforme i specifični su za infrastrukturu. Aplikacije svoju *Spring Cloud Config* konfiguracijsku datoteku dohvaćaju putem HTTP zahtjeva dok svoje *Kubernets ConfigMaps and Secrets* preuzimaju iz datoteka u kontejnerima.
- *AWS Systems Manager Parameter Store* - Konfiguracijske datoteke *Spring Cloud Configa* čuvaju se izvan aplikacije, a parametri *AWS Systems Manager Parameter Store-a* se čuvaju u AWS-ovoj infrastrukturi.

2.5. Arhitektura mikroservisa

Prema [4] arhitektura mikroservisa omogućuje izgradnju distribuiranog sustava koji se sastoji od labavo (eng. Loosely coupled) povezanih mikroservisa koji se implementiraju neovisno. Svrha takve arhitekture je da se aplikacija rastavi na manje autonomne cjeline koje rade zajedno kako bi se postigla funkcionalnost aplikacije. Svaki od mikroservisa predstavlja određenu funkcionalnost

aplikacija te dozvoljava neovisan razvoj, implementaciju i testiranje. Mikroservisi međusobno komuniciraju putem definiranih API sučelja i protokola.



Slika 2.1. *Prikaz mikroservisne arhitekture*

Na slici 2.1. [5] prikazan je primjer mikroservisne arhitekture. Vidi se API Gateway koji je ulazna točka aplikacije koja navodi svaki zahtjev za resurs ili rutu prema određenom mikroservisu. *Discovery* servis koji sadrži sve podatke o mikroservisima unutar sustava služi kao kontrolor koji nam govori koji od mikroservisa rade ,a koji ne. Ostali servisi su dijelovi poslovne logike sustava.

Neke od ostalih arhitektura koje se koriste umjesto arhitekture mikroservisa:

- Monolitna arhitektura – U monolitnoj arhitekturi aplikacija se izgrađuje kao jedna cjelina za razliku od arhitekture mikroservisa koja aplikaciju dijeli na male neovisne servise. Monolitna arhitektura je puno lakša za testiranje zbog toga što se aplikacija testira kao jedna cjelina, a kod mikroservisne arhitekture treba obratiti pozornost na interakciju između servisa. Također, monolitna arhitektura je lakša za izgradnju od mikroservisne arhitekture.
- Heksagonalna arhitektura – Heksagonalna arhitektura poznata kao *Ports and Adapters* arhitektura dijeli sustav u labavo (eng. Loosely coupled) povezane komponente, kao što su jezgra aplikacije, baza podataka, korisničko sučelje itd. Razlika između heksagonalne i arhitekture mikroservisa je ta što heksagonalna arhitektura se fokusira na odvajanje poslovne logike od ostatka programskog koda dok arhitektura mikroservisa odvaja aplikacije u mikroservise koji se mogu neovisno i samostalno izvršavati.

3. Korištene tehnologije

3.1. *Spring* programski okvir

Spring je programski okvir otvorenog kôda osmišljen za olakšavanje razvoja Java aplikacija temeljen na *Java Beansima* [6]. Prvu verziju napisao je Rod Johnson te ju je objavio u svojoj knjizi „Expert One-on-One: J2EE Design and Development“ [7] u listopadu 2002. godine. Programski okvir je izdan pod Apache 2.0 licencom u lipnju 2003. godine. Spring Framework podijeljen je na module. Aplikacije mogu odabrati koji moduli su im potrebni. Glavni su moduli jezgrenog spremnika, koji uključuju konfiguracijski model i mehanizam ubrizgavanja ovisnosti. Osim toga, Spring Framework pruža temeljnu podršku za različite arhitekture aplikacija, uključujući slanje poruka, transakcijske podatke i postojanost te web. Također uključuje *Spring* MVC *web framework* temeljen na *Servletu* i, paralelno, *Spring WebFlux* reaktivni web framework. Iako *Spring* ne posjeduje modele programiranja, vrlo je popularan u Java zajednici kao alternativa, čak i zamjena za Enterprise *JavaBean* EJB model. Neke od značajki i mehanizama *Spring* programskog okvira su:

- jednostavniji Java kôd
- korištenje anotacija
- ubacivanje ovisnosti
- omogućuje definiranje ovisnosti između objekata pomoću XML konfiguracije u fazi kreiranja objekata
- AOP (*Aspect oriented programming)*
- *Spring Container* – brine o objektima koje aplikacija koristi tijekom cijelog životnog ciklusa aplikacije

Kada govorimo o *Spring* programskom okviru, važno je znati ne samo što on radi, već i koja načela slijedi. Vodeća načela jesu:

- *Spring* programski okvir daje prednost davanju slobode programerima da odgode važne dizajnerske odluke. Na primjer, omogućuje prebacivanje između pružatelja postojanosti i rukovanje različitom infrastrukturom i API integracijama trećih strana kroz konfiguraciju, izbjegavajući potrebu za promjenom vašeg programskog koda.
- *Spring* vrijednosti se mijenjaju i ne nameću jedan "pravi način" obavljanja stvari. Priznaje širok raspon funkcionalnih potreba i usvaja različite pristupe kako se projekti mogu ostvariti.

- *Spring* pažljivo upravlja svojim razvojem kako bi smanjio ometajuće promjene između inačica. Podržava određene verzije Java Development Kit-a (JDK) kao i biblioteke trećih strana kako bi se osiguralo ispravno održavanje aplikacija i biblioteka izgrađenih na temelju *Springa*.
- *Spring* Framework stavlja snažan naglasak na održavanje visoke kvalitete programskog koda, uključujući detaljnu i ažurnu *javadoc* dokumentaciju. Posebno se ističe kao projekt s ispravno uređenom bazom programskog koda izbjegavajući kružne smetnje između umetaka.

3.2. *Spring Cloud* programski paket

Spring Cloud je programski okvir za stvaranje robusnih aplikacija u oblaku. *Spring Cloud* olakšava razvoj aplikacija pružajući rješenja za mnoge uobičajene probleme s kojima se susreće pri prelasku u distribuirano okruženje [8]. Također, pruža alate za bržu izradu nekih čestih obrazaca u raspodijeljenim sustavima i fokusira se na pružanje kvalitetnih rješenja za redovne probleme s kojima se susreće u programiranju.

3.3. Apache Maven

Apache Maven je alat za stvaranje središnjeg repozitorija svih JAR datoteka na računalu [9]. Također, koristi se za kreiranje i upravljanje projektima pisanim u C#, *Ruby, Scali* i ostalim jezicima. *Maven* adresira dva aspekta stvaranja programa, način pisanja i njegove ovisnosti(eng. *dependency)*. Alternativa *Mavenu* bi bio *Gradle*.

3.4. PostgreSQL

PostgreSQL, poznat i kao *Postgres*, sustav je otvorenog kôda za upravljanje relacijskim bazama podataka (RDBMS) koji naglašava proširivost, stabilnost i usklađenost sa SQL-om. Izvorno je razvijen na Kalifornijskom sveučilištu Berkeley 80-ih godina prošlog stoljeća i od tada je postao jedan od najpopularnijih i najmoćnijih sustava baza podataka. *PostgreSQL* slijedi model relacijske baze podataka koji omogućuje pohranu i organizaciju podataka u tablicama. Podržava razne vrste podataka uključujući numeričke, tekstualne, boolean, datum/vrijeme i JSON. Također, *PostgreSQL* osigurava ACID svojstva (*Atomicity, Consistency, Isolation, Durability).* Nadalje, pruža podršku i za geoprostorne podatke. Omogućava njihovo pohranjivanje, indeksiranje i postavljanje upita, što ga čini popularnim izborom za geografske informacijske sustave (GIS). Prema svemu navedenom, *PostgreSQL* je značajkama bogat i vrlo prilagodljiv sustav baze podataka koji može podnijeti širok raspon aplikacija i radnih opterećenja. Njegova kombinacija

pouzdanosti, proširivosti i usklađenosti sa *SQL*-om učinila ga je popularnim izborom za male projekte i za velike poslovne aplikacije.

3.5. Java

Java je objektno orijentirani programski jezik poznat po svojoj platformskoj neovisnosti. Stvorio ga je James Gosling a izdao *Sun Microsystems* 90-ih godina prošlog stoljeća [10]. *Java* aplikacije prevode se u *bytecode* koji može raditi na bilo kojem *Java* virtualnom stroju, odnosno, na bilo kojoj platformi s kompatibilnim *Java* virtualnim strojem. Java se široko koristi u razvoju web stranica, razvoju mobilnih aplikacija (Android) i razvoju poslovne razine softvera zbog svoje robusnosti, sigurnosnih značajki te izrazito razvijene zajednice i brojnih resursa za učenje. Slijedi filozofiju "napiši jednom, pokreni bilo gdje", čime postaje svestran izbor za različite potrebe razvoja softvera.

3.6. IntelliJ IDEA

IntelliJ je integrirano razvojno okruženje (IDE) za razvijanje računalnog softvera u *Javi, Kotlinu, Groovyu* i ostalim jezicima. *IntelliJ* je program tvrtke *JetBrains* kreiran 2001. godine i bio je jedan od prvih *Java* IDE-ova s naprednom navigacijom programskog kôda i integriranim sposobnostima refaktoriranja programskog kôda. *IntelliJ* pruža određene dodatne mogućnosti poput nadopune programskog kôda gdje *IntelliJ* analizira kontekst programskog kôda i predlaže koju metodu, funkciju ili varijablu treba dopisati.

3.7. Liquibase

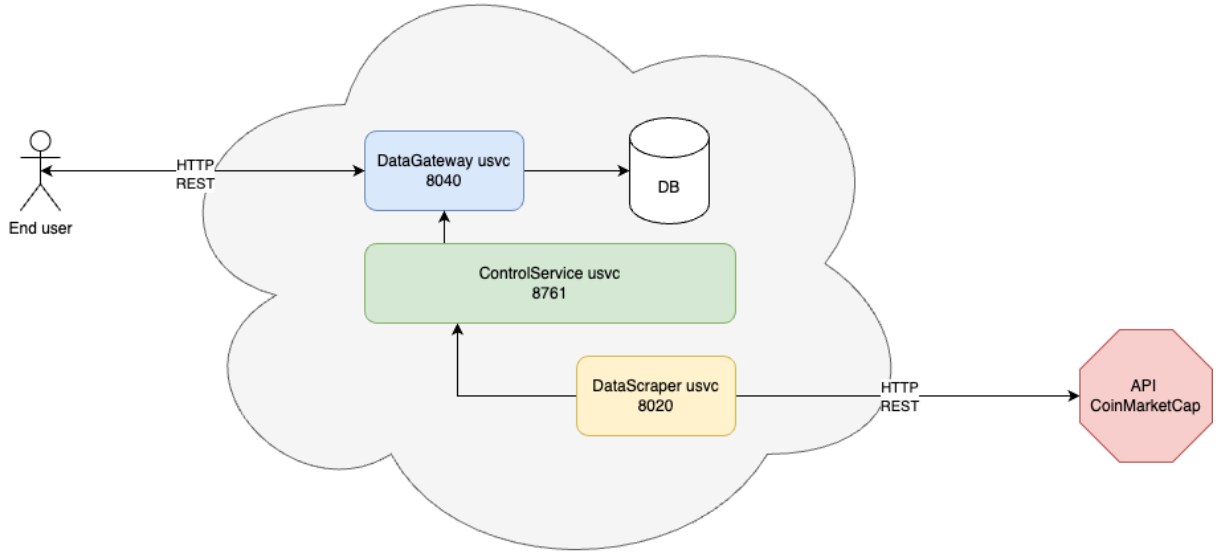
Liquibase je alat otvorenog kôda koji pomaže *programerima* pri održavanju i mijenjanju shema baze podataka. Također, omogućuje generiranje SQL upita za različite baze podataka, kontrolira migracije i vraćanje na zadnje stanje pričuvne kopije. Pojednostavljuje suradnju, kontrolu verzija i razvoj baze podataka unutar aplikacijskog kôda.

3.8. DBeaver

DBeaver je alat za upravljanje bazama podataka. Omogućuje korisnicima povezivanje i rad s različitim vrstama baza podataka, uključujući MySQL, PostgreSQL, Oracle, i mnoge druge. DBeaver ima jednostavno korisničko sučelje s potporom za više platformi te nudi funkcionalnosti SQL uređivača (eng. editor), vizualizacije podataka te mogućnost prijenosa i izvoza podataka. Također podržava različite vrste baza podataka i dodatke za proširenje funkcionalnosti.

4. Implementacija rješenja

Sa slike 4.1. vidi se način rada aplikacije. U ekosustavu aplikacije nalaze se 3 mikroservisa i baza podataka. Izvan ekosustava nalaze se korisnici i API. DataScraper pomoću HTTP upita dohvaća podatke te ih šalje DataGateway mikroservisu. DataScraper i DataGateway mogu komunicirati zato što ControlService služi kao *Eureka* poslužitelj na kojem su oba mikroservisa registrirana. DataGateway u bazu podataka upisuje podatke koje mu je proslijedio DataScraper. Korisnik do podataka dolazi preko DataGateway mikroservisa.



Slika 4.1. *Prikaz arhitekture rada*

Aplikacija počinje s Eureka poslužiteljem, koji služi kao središnji registar servisa. Svi mikroservisi u aplikaciji, poput Data Scrapera, Data Gatewaya i drugih, registriraju se na Eureka poslužitelj prilikom pokretanja. Ova registracija uključuje ime mikroservisa i mrežnu lokaciju (adresu i priključak). Mikroservisi šalju periodične „otkucaje srca“ ili provjere zdravlja Eureka poslužitelju kako bi pokazali da ispravno rade. Drugi mikroservisi ili korisnici šalju upite Eureka poslužitelju kako bi otkrili dostupne usluge, koristeći samo ime usluge, ne i specifične detalje o *host-u* i *port-u*. Ako je registrirano više primjeraka mikroservisa s istim imenom, Eureka poslužitelj može olakšati opterećenje, usmjeravajući zahtjeve dostupnim instancama. Kada mikroservis ne reagira ili se isključi radi održavanja, prestaje slati „otkucaje srca“. Eureka poslužitelj automatski ga uklanja iz registra kako bi se izbjeglo korištenje nedostupnih usluga. *DataScraper* je mikroservis posvećen dohvaćanju financijskih podataka s vanjskih API-ja, svakih 5 minuta dohvaća podatke. ScrapeService, koji je dio DataScrapera, rukuje dohvaćanjem podataka. *TransferService* koji je također dio *DataScrapera*, djeluje kao veza za prijenos podataka Data Gatewayu na daljnju

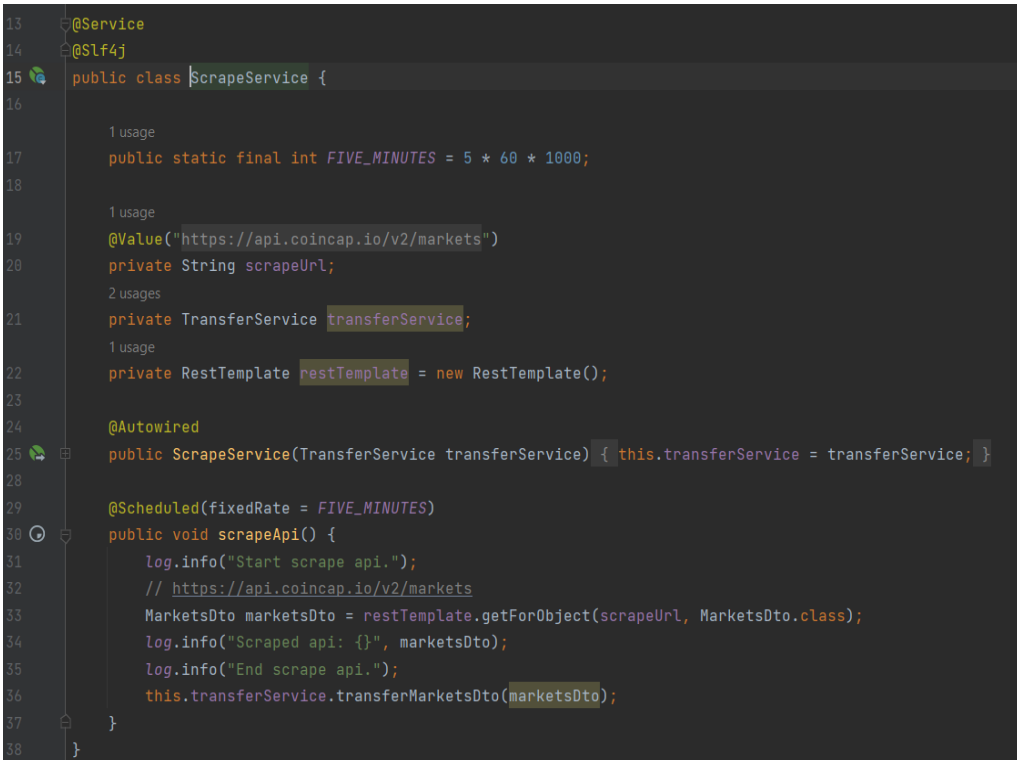
obradu. Data Gateway je središnja stanica za primanje, obradu i pohranu podataka o tržištu. Pretvara JSON podatake u strukturirane domenske objekte prikladne za obradu. Podaci o tržištu se pohranjuju u bazu podataka. Market Repository služi kao sučelje između aplikacije i baze podataka. Osigurava integritet podataka i pruža pouzdan način dohvaćanja pohranjenih podataka o tržištu za daljnju upotrebu ili analizu.

Dakle, Prvo se inicira zahtjev za podacima kako bi se dobili financijski podaci. Ovaj zahtjev može biti upućen bilo kojem mikroservisu. Ne mora se znati točna lokacija mikroservisa s kojeg zahtijevamo podatke nego se upućuje upit Eureka poslužitelju koji sadrži ime usluge. Potom Eureka poslužitelj vraća mrežnu lokaciju (*host* i *port*) odgovarajućeg mikroservisa. Pomoću lokacije koja se dobije od Eureka poslužitelja korisnik može slati zahtjev za podacima izravno mikroservisu. Mikroservis potom primi zahtjev za podacima te ga obrađuje. Zahtjev može uključivati dohvaćanje financijskih podataka s vanjskih API-ja, izvođenje transformacija podataka ili bilo koju drugu relevantnu operaciju. Nakon obrade, mikroservis šalje odgovor koji sadrži tražene financijske podatke nazad korisniku. Sada korisnik primljene financijske podatke može koristiti za različite svrhe, poput prikazivanja korisnicima ili provođenja analize.

4.1. DataScraper

DataScraper je mikroservis sastavljen od dva servisa. *ScrapeService* je odgovoran za dohvaćanje podataka sa API-ja, a *TransferService* služi za prosljeđivanje podataka u *DataGateway.*

- ScrapeService - Klasa označena anotacijom *@Service*, što ukazuje da se radi o *Spring* servisnom *beanu* kojim bi trebao upravljati *Spring* spremnik. Koristi *Lombok* anotaciju @Slf4j za generiranje *loggera*. Klasa ovisi o klasi *TransferService* koja je automatski povezana pomoću oznake *@Autowired* u konstruktoru. Klasa ima zakazanu metodu *scrapeApi*() anotiranu s *@Scheduled*. Ova je metoda predviđena za pokretanje s fiksnim vremenom svakih pet minuta (FIVE_MINUTES). Izvodi sljedeće korake: Bilježi informativnu poruku koja ukazuje na početak procesa dohvaćanja podataka sa API-ja. Izrađuje HTTP GET zahtjev za *scrapeUrl* za dohvaćanje tržišnih podataka iz navedenog API-ja. Odgovor se deserijalizira u objekt *MarketsDto* pomoću *restTemplate.getForObject*(). Bilježi informativnu poruku koja sadrži podatke s API-ja i poruku koja označava kraj procesa. Poziva metodu *transferMarketsDto*() usluge *TransferService* za prijenos izdvojenih tržišnih podataka.

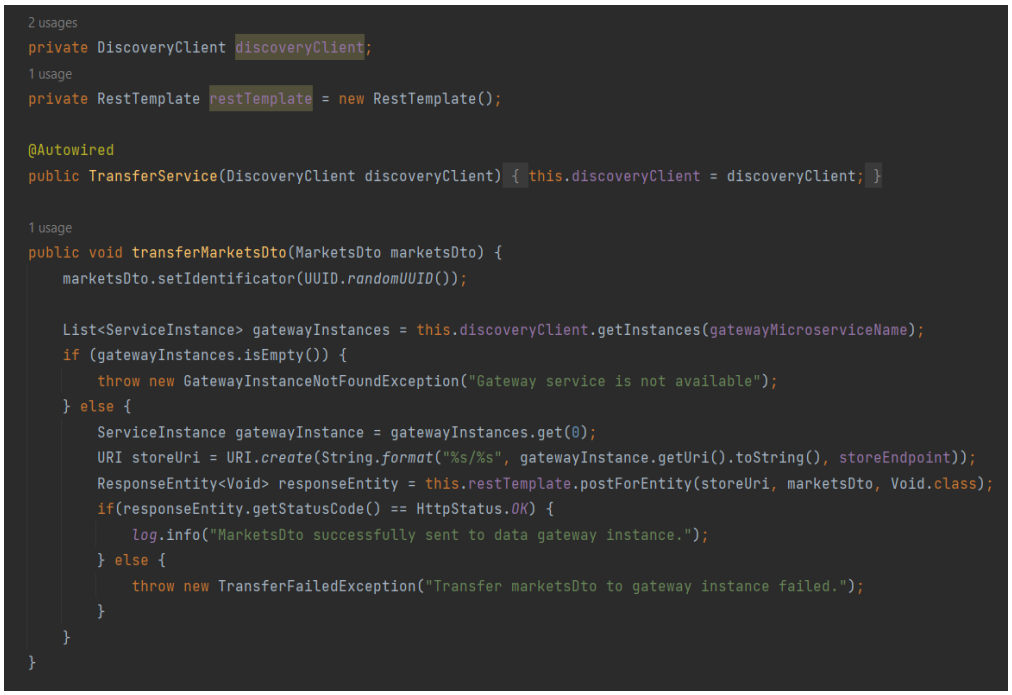


```java
@Service
@Slf4j
public class ScrapeService {

    1 usage
    public static final int FIVE_MINUTES = 5 * 60 * 1000;

    1 usage
    @Value("https://api.coincap.io/v2/markets")
    private String scrapeUrl;
    2 usages
    private TransferService transferService;
    1 usage
    private RestTemplate restTemplate = new RestTemplate();

    @Autowired
    public ScrapeService(TransferService transferService) { this.transferService = transferService; }

    @Scheduled(fixedRate = FIVE_MINUTES)
    public void scrapeApi() {
        log.info("Start scrape api.");
        // https://api.coincap.io/v2/markets
        MarketsDto marketsDto = restTemplate.getForObject(scrapeUrl, MarketsDto.class);
        log.info("Scraped api: {}", marketsDto);
        log.info("End scrape api.");
        this.transferService.transferMarketsDto(marketsDto);
    }
}
```

Slika 4.2. *Klasa ScrapeService*

Ukratko, *ScrapeService* povremeno dohvaća podatke sa API-ja pomoću *RestTemplatea*, dohvaća tržišne podatke, bilježi dohvaćene podatke i prenosi ih *TransferServiceu* na daljnju obradu.

- *TransferService* – Klasa koja je zadužena za prijenos podataka do *DataGateway* mikroservisa. Klasa je označena s anotacijom *@Service* kako bi se naznačilo da je to *Spring* servisni *bean*. Klasa ima nekoliko varijabli instance (*gatewayMicroserviceName* i *storeEndpoint*) koje su označene s anotacijom *@Value* za ubacivanje vrijednosti iz vanjske konfiguracije. Klasa ima konstruktor koji prihvaća *DiscoveryClient* objekt i automatski ga povezuje pomoću oznake *@Autowired*. To omogućuje klasi da koristi *DiscoveryClient* za otkrivanje instanci mikroservisa . Metoda *transferMarketsDto* prima *MarketsDto* objekt kao parametar i odgovorna je za njegov prijenos do instance API-ja. Unutar metode generira se jedinstveni identifikator (UUID) i postavlja na objekt *marketsDto*. *DiscoveryClient* koristi se za dohvaćanje instanci mikroservisa pristupnika (*gatewayMicroserviceName*). Ako nijedna instanca nije dostupna, izbacuje se iznimka *GatewayInstanceNotFoundException*. Ako su instance dostupne, prva se instanca dohvaća s popisa i dobiva se njezin URI. URI se konstruira korištenjem URI-ja instance pristupnika i vrijednosti *storeEndpoint*.



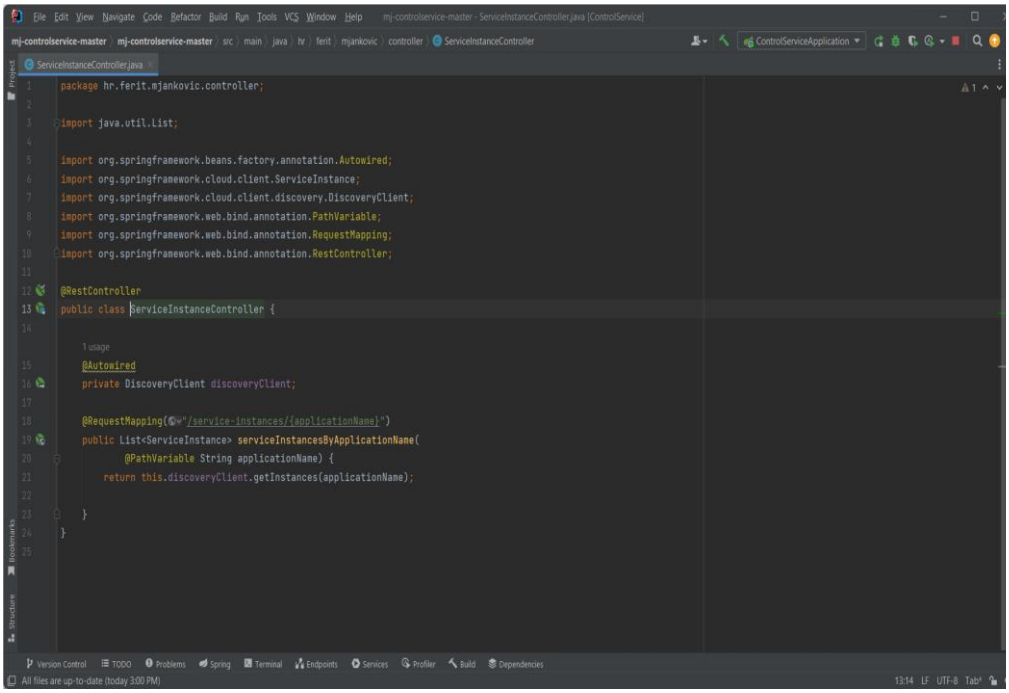
Slika 4.3. *TransferService*

Ukratko, isječak programskog kôd*a* sa slike 4.3. definira uslugu koja prenosi objekt *MarketsDto* na instancu pristupnika podataka upućivanjem POST zahtjeva određenoj krajnjoj točki na URI instance pristupnika. Obrađuje slučajeve kada nema dostupnih instanci pristupnika ili ako prijenos ne uspije.

- *ServiceInstanceController* - *ServiceInstanceController* predstavlja klasu *Spring* MVC kontrolera što znači da je to rukovoditelj za web zahtjeve. @RestController anotacija označava da je ova klasa RESTful kontroler koji obrađuje HTTP zahtjeve i automatski serijalizira odgovore kao JSON ili XML. @Autowired anotacija omogućuje automatsko umetanje ovisnosti *DiscoveryClient beana*. *DiscoveryClient* je *Spring Cloud* sučelje koje pruža mogućnost postavljanja upita registru usluga (*Eureka* poslužitelj) i dohvaćanja informacija o registriranim uslugama i njihovim instancama. *@RequestMapping*("/service-instances/{applicationName}"): određuje a ova metoda obrađuje HTTP zahtjeve s određenim URL uzorkom. U ovom slučaju, preslikava zahtjeve na /service-instances/{applicationName} URL. Metoda koristi *DiscoveryClient* za dohvaćanje popisa *ServiceInstance* objekata za zadani *ApplicationName*. Objekti *ServiceInstance* predstavljaju instance navedene usluge registrirane u registru usluga (Eureka poslužitelj).

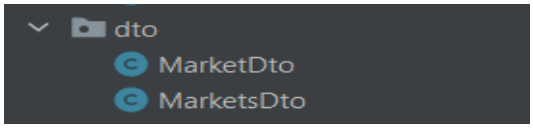
Metoda zatim vraća popis *ServiceInstance* objekata kao odgovor, koji će biti serijaliziran kao JSON ili XML na temelju sadržaja zahtjeva.

Dana transfer objects - DTO (objekti prijenosa podataka) koriste se za prikaz podataka koji se trebaju prenijeti između različitih slojeva ili komponenti aplikacije. DTO su objekti posebno dizajnirani za prijenos podataka, također djeluju kao spremnik za podatke i pružaju standardizirani način razmjene informacija.



Postoje dvije klase DTO-a:

- *MarketDTO* - predstavlja informacije povezane s određenim tržištem. Može sadržavati atribute kao što su identifikator tržišta, vremenska oznaka i druge relevantne podatke specifične za to tržište.
- *MarketsDTO* - predstavlja zbirku ili popis višestrukih unosa na tržište ili informacija. Može sadržavati popis *MarketDTO* objekata ili neke skupne podatke koji se odnose na više tržišta. *MarketsDTO* pruža pregled tržišnih podataka.*ControlService*



Slika 4.5. *Prikaz Data Transfer objekata*

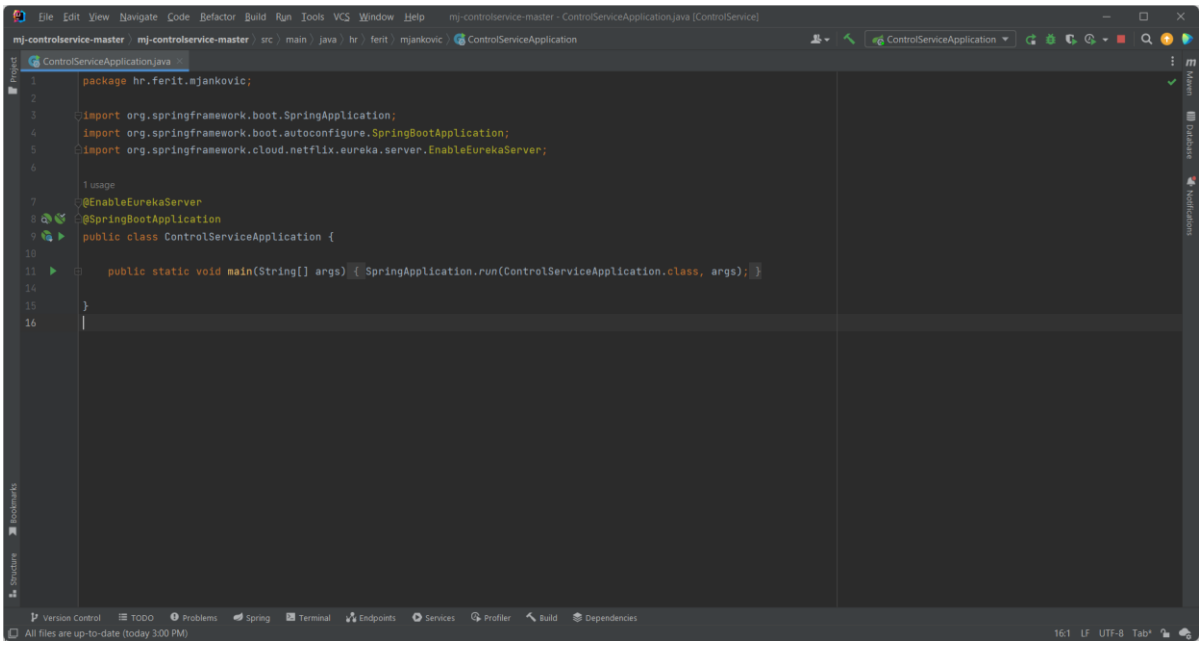
```java
public class MarketDto implements Serializable {
    private String exchangeId;
    private String rank;
    private String baseSymbol;
    private String baseId;
    private String quoteSymbol;
    private String quoteId;
    private String priceQuoute;
    private String priceUsd;
    private String volumeUsd24Hr;
    private String percentExchangeVolume;
    private String tradesCount24Hr;
    private Instant updated;
}
```

```java
public class MarketsDto implements Serializable {
    private List<MarketDto> data;
    private Instant timestamp;
    private UUID identificator;
}
```

Slika 4.6. *Prikaz DTO-a*

4.2. *ControlService*

Klasa sa slike 4.7. predstavlja glavnu klasu *Spring Boot* aplikacije s funkcijom *Eureka* poslužitelja. Klasa *ControlServiceApplication* označena je s anotacijom *@EnableEurekaServer*. Ova anotacija omogućuje funkcionalnost *Eureka* poslužitelja u aplikaciji, dopuštajući mu da funkcionira kao



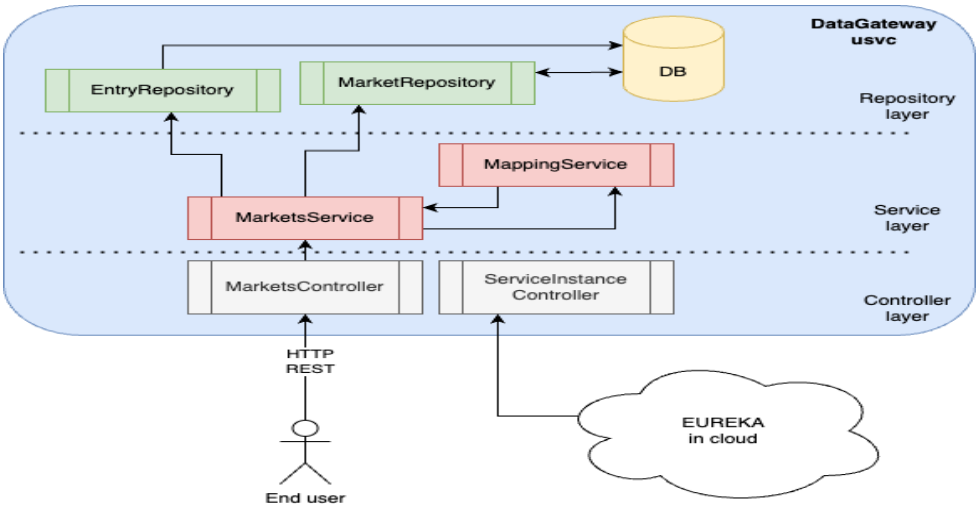
Slika 4.7. *Prikaz ControlServiceApplicationa*

registar usluga i poslužitelj za otkrivanje. Klasa je također označena anotacijom *@SpringBootApplication* koja kombinira nekoliko anotacija uključujući *@Configuration*, *@EnableAutoConfiguration* i *@ComponentScan*. To znači da je ova klasa ulazna točka za *Spring*

Boot aplikaciju i izvodi potrebnu konfiguraciju i skeniranje komponenti. Unutar *ControlService* mikroservisa također se nalazi *ServiceInstanceController klasa.*

4.3. DataGateway

Ova aplikacija implementira API *gateway* tehnologiju iz *Spring Cloud* programskog paketa te služi kao *proxy* za prosljeđivanje zahtjeva ostalim mikroservisima i koordinira komunikaciju između njih. U narednim odjeljcima bit će istražena detaljna analiza ključnih komponenata, klasa i funkcionalnosti našeg DataGatewayja.



Slika 4.8. *Prikaz slojeva DataGateway mikroservisa*

Sa slike 4.8. se vidi da ovaj mikroservis ima 3 sloja.

- *Controller layer* – U njemu se nalaze MarketsController i ServiceInstanceController koji služi kao pristupni kontroler za ostale servise.
- *Service layer* – Servisni sloj koji služi za operaciju nad podacima.
- *Repository layer* – Sloj za pristup i operacije nad bazom podataka.

```java
public class MarketsController {

    3 usages
    private MarketsService marketsService;

    @Autowired
    public MarketsController(MarketsService marketsService) { this.marketsService = marketsService; }

    @PostMapping(value = "/infos")
    public ResponseEntity<Void> storeMarketsInfos(@RequestBody MarketsDto marketsDto) {
        log.info("MarketsDto ready for save {}", marketsDto);
        try {
            marketsService.storeMarketData(marketsDto);
        } catch (Exception ex) {
            log.error("Exception while saving MarketsInfo object", ex);
            return ResponseEntity.badRequest().build();
        }
        return ResponseEntity.ok().build();
    }

    @GetMapping(value = "/infos/{marketUuid}")
    public ResponseEntity<MarketsDto> getMarketByUuid(@PathVariable("marketUuid") String marketUuid) {
        log.info("Fetch MarketsDto for uuid {}", marketUuid);
        MarketsDto marketsDto = new MarketsDto();
        try {
            marketsDto = marketsService.getMarketByUuid(marketUuid);
        } catch (Exception ex) {
            log.error("Exception while getting MarketsDto object", ex);
            return ResponseEntity.badRequest().build();
        }
        return ResponseEntity.ok().body(marketsDto);
```


Slika 4.9. *Prikaz klase MarketsController*

- *MarketsController* - Isječak programskog kôda sa slike 4.9. predstavlja klasu *Spring* REST kontrolera pod nazivom *MarketsController*. Definira krajnje točke za pohranjivanje i dohvaćanje tržišnih informacija predstavljenih *MarketsDto* objektima. Klasa *MarketsController* označena je anotacijom *@RestController*, što ukazuje da se radi o *Spring* REST kontroleru koji obrađuje HTTP zahtjeve i vraća JSON odgovore. Klasa ima varijablu instance *marketsService* tipa MarketsService, koja je automatski povezana pomoću oznake *@Autowired*. To omogućuje klasi da koristi *MarketsService* za pohranu i dohvaćanje tržišnih podataka. Bilježi informacijsku poruku o primljenom objektu. Klasa MarkesController poziva metodu *storeMarketData* definiranu u klasi *MarketsService* koja pohranjuje tržišne podatke u bazu podataka. Metoda *getMarketByUuid* označena je s *@GetMapping* i preslikava HTTP GET zahtjeve na krajnju točku

/markets/infos/{marketUuid}. Očekuje varijablu puta {marketUuid} koja predstavlja UUID tržišta. Bilježi informacijsku poruku o dohvaćanju *MarketsDto* za navedeni UUID. Poziva *getMarketByUuid* metodu *marketsService* za dohvaćanje tržišnih podataka. Ova klasa definira REST kontroler koji pruža krajnje točke za pohranjivanje i dohvaćanje tržišnih informacija pomoću *MarketsDto* objekata. Koristi *MarketsService* za izvođenje operacija i bilježi relevantne informacije tijekom procesa.

- *MarketsService* - Služi za kontroliranje poslovne logike koja upravlja podatcima sa marketa, dohvaća podatke iz DTO-a i sprema ih u bazu podataka pomoću *MarketRepositoryja* i *EntryRepositoryja*. Također koristi *MappingService* za pretvorbu DTO podataka u potrebni oblik (*Market* ili *Entry*). Vidljivo je na slikama 4.10. i 4.11.

```
public class MarketsService {

    3 usages
    private MarketRepository marketRepository;
    3 usages
    private EntryRepository entryRepository;
    3 usages
    private MappingService mappingService;

    @Autowired
    public MarketsService(MarketRepository marketRepository, EntryRepository entryRepository, MappingService mappingService) {
        this.marketRepository = marketRepository;
        this.entryRepository = entryRepository;
        this.mappingService = mappingService;
    }

    1 usage
    public void storeMarketData(MarketsDto marketsDto) {
        Market market = new Market();
        market.setIdentificator(marketsDto.getIdentificator().toString());
        market.setTimestamp(LocalDateTime.ofInstant(marketsDto.getTimestamp(), ZoneOffset.UTC));

        Market storedMarket = marketRepository.save(market);
        log.info("Stored market for identificator {}", storedMarket.getIdentificator());

        List<Entry> marketEntries = mappingService.toEntries(marketsDto.getData(), storedMarket);

        entryRepository.saveAll(marketEntries);
        log.info("Stored entries for market {}", storedMarket.getIdentificator());
        log.info("Number of stored entries: {}", marketEntries.size());
```

Slika 4.10. *Prikaz klase MarketsService*

```
    }

    1 usage
    public MarketsDto getMarketByUuid(String marketUuid) {
        MarketsDto marketsDto = new MarketsDto();
        marketsDto.setIdentificator(UUID.fromString(marketUuid));

        Market market = marketRepository.findByIdentificator(marketUuid);
        if (market == null) {
            log.info("Market with identifier {} not found.", marketUuid);
            return null;
        }

        marketsDto.setTimestamp(Instant.ofEpochMilli(market.getTimestamp().toEpochSecond(ZoneOffset.UTC)));
        marketsDto.setData(mappingService.toMarketDtos(entryRepository.findAllByMarket(market)));

        log.info("Ready to return data about market with id {}", marketUuid);
        return marketsDto;
    }
```

Slika 4.11. *Prikaz klase MarketsService*

- *MappingService* - Pruža metode za pretvorbu između *MarketDto* objekata i *Entry* objekata. Pruža mogućnost transformacije podataka po potrebi slojeva aplikacije. Metoda *toEntries* dohvaća listu *MarketDto* objekata i *Market* objekt kao parametar te pretvara svaki objekt *MarketDtoa* u Entry objekt i spaja ga sa svojstvima *Market* objekta. Na isti način radi i *toMarketDtos* samo suprotno. Na slici 4.11. vidimo toEntries metodu.

```
2 usages
@Service
public class MappingService {

    1 usage
    public List<Entry> toEntries(List<MarketDto> marketDtos, Market storedMarket) {
        List<Entry> marketEntries = new ArrayList<~>();
        marketDtos.forEach(marketDto -> {
            Entry marketEntry = new Entry();
            marketEntry.setExchangeId(marketDto.getExchangeId());
            marketEntry.setRank(marketDto.getRank());
            marketEntry.setBaseSymbol(marketDto.getBaseSymbol());
            marketEntry.setBaseId(marketDto.getBaseId());
            marketEntry.setQuoteSymbol(marketDto.getQuoteSymbol());
            marketEntry.setQuoteId(marketDto.getQuoteId());
            marketEntry.setPriceQuoute(marketDto.getPriceQuoute());
            marketEntry.setPriceUsd(marketDto.getPriceUsd());
            marketEntry.setVolumeUsd24Hr(marketDto.getVolumeUsd24Hr());
            marketEntry.setPercentExchangeVolume(marketDto.getPercentExchangeVolume());
            marketEntry.setTradesCount24Hr(marketDto.getTradesCount24Hr());
            marketEntry.setUpdated(LocalDateTime.ofInstant(marketDto.getUpdated(), ZoneOffset.UTC));
            marketEntry.setMarket(storedMarket);

            marketEntries.add(marketEntry);
        });

        return marketEntries;
    }
```

Slika 4.12. *Prikaz metode toEntries*

Sve korištene anotacije prikazane su u tablici 4.1.

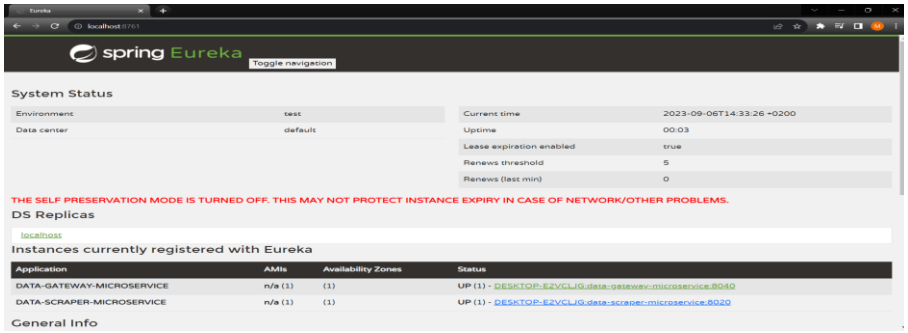
Tablica 4.1. *Popis i obrazloženje anotacija korištenih u završnom radu*

Anotacija	**Kratko objašnjenje**
@RestController	Koristi se za označavanje klase kao *RESTful* kontrolera što znači da klasa barata HTTP zahtjevima.
@RequestMapping	Koristi se za mapiranje HTTP zahtjeva specifičnim metodama u kontroleru. Definira URL i HTTP metodu(GET,POST, itd.).

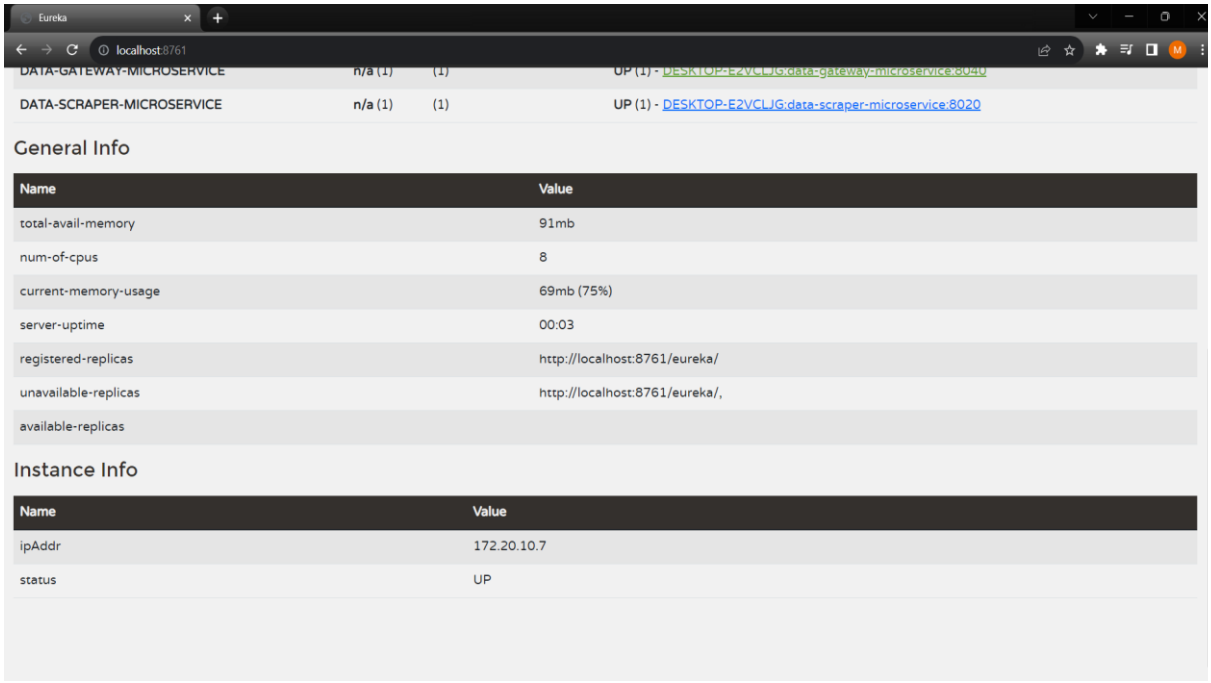
@Slf4j	Koristi se u svrhe *logganja*.
@AutoWired	Koristi se za direktno ubrizgavanje ovisnosti u programski kôd, definira konstruktor ili *setter* metodu za *Spring* ovisnost.
@PostMapping	Koristi se za baratanje HTTP POST zahtjevima specificira URL path do metode koja treba biti pozvana kada je primljen POST zahtjev.
@GetMapping	Koristi se za baratanje HTTP GET zahtjevima specificira URL putanju do metode koja treba biti pozvana kada je primljen GET zahtjev.
@Data	Koristi se za generiranje *gettera* i *settera*.
@Service	Koristi se za naglašavanje da je klasa servisna komponenta u poslovnom sloju najčešće korištena za definiranje logike i operacije koje barataju poslovnim zadatcima.
@SpringBoot Application	Koristi se za označavanje glavne klase *spring* aplikacije u poslovnom sloju, omogućuje automatsku konfiguraciju i skenira za komponente u aplikaciji.
@EnableEurekaServer	Koristi se za pokretanje *Eureka server* funkcionalnosti u *Spring Cloud* aplikaciji.

4.5. Testiranje

Slika 4.13. prikazuje *Eureka* poslužitelj te instance koje su registrirane na poslužitelju. Poslužitelj je lokalan na portu 8761. Također na slici 4.14. nalaze se dodatne informacije o poslužitelju.

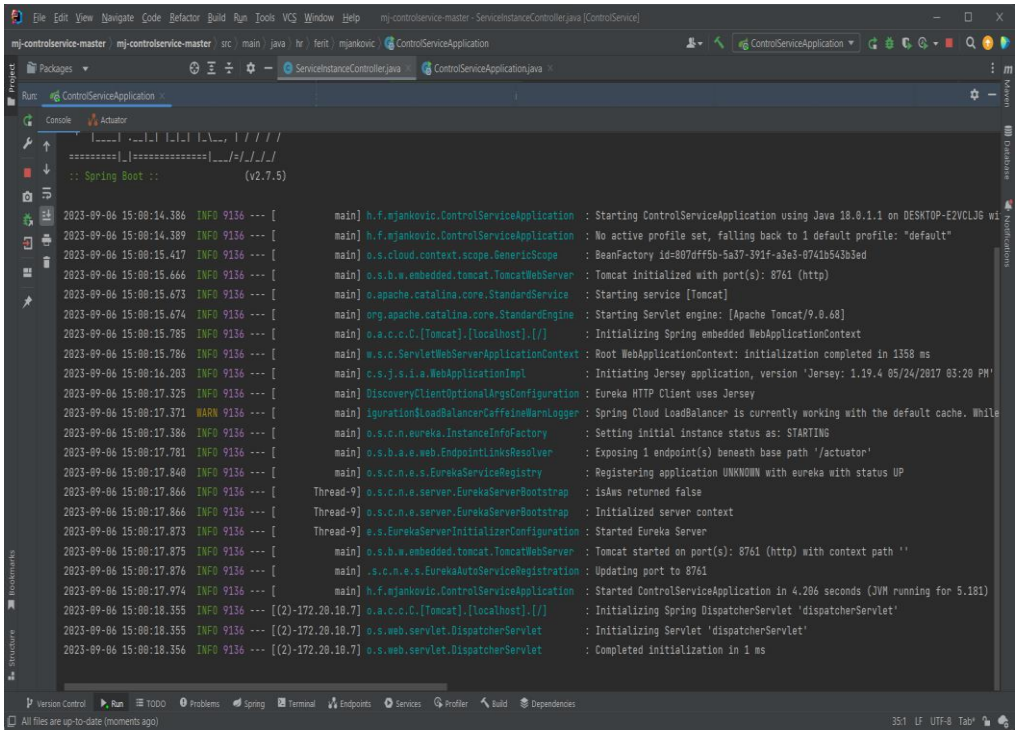


Slika 4.13. *Eureka server*



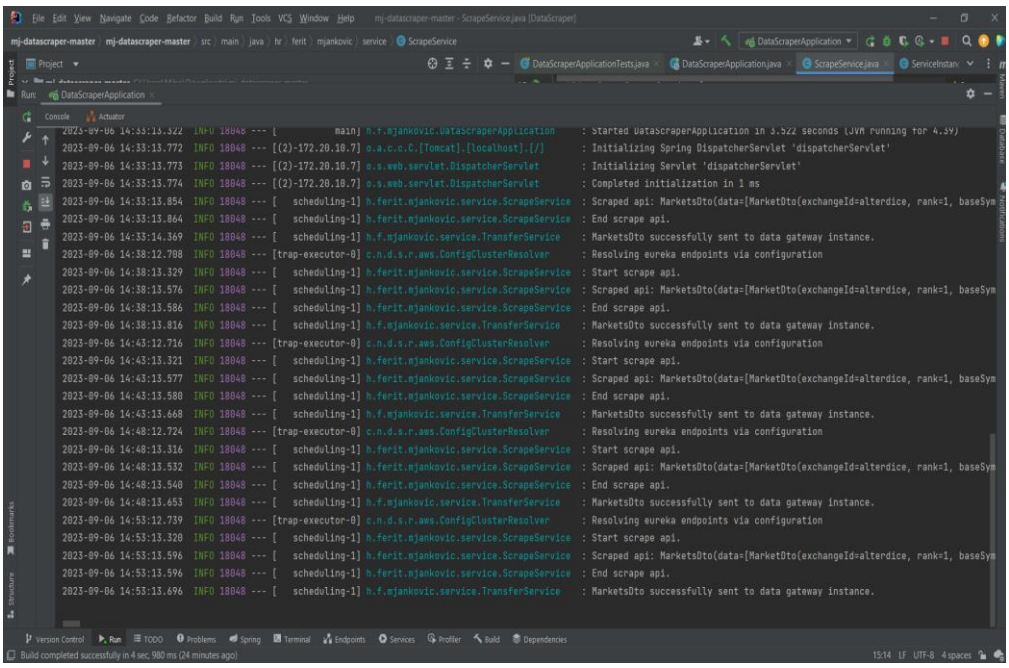
Slika 4.14. *Prikaz dodatnih informacija o serveru*

Uspješno pokretanje Eureka servera također vidimo i na ispisu terminala sa slike 4.15.



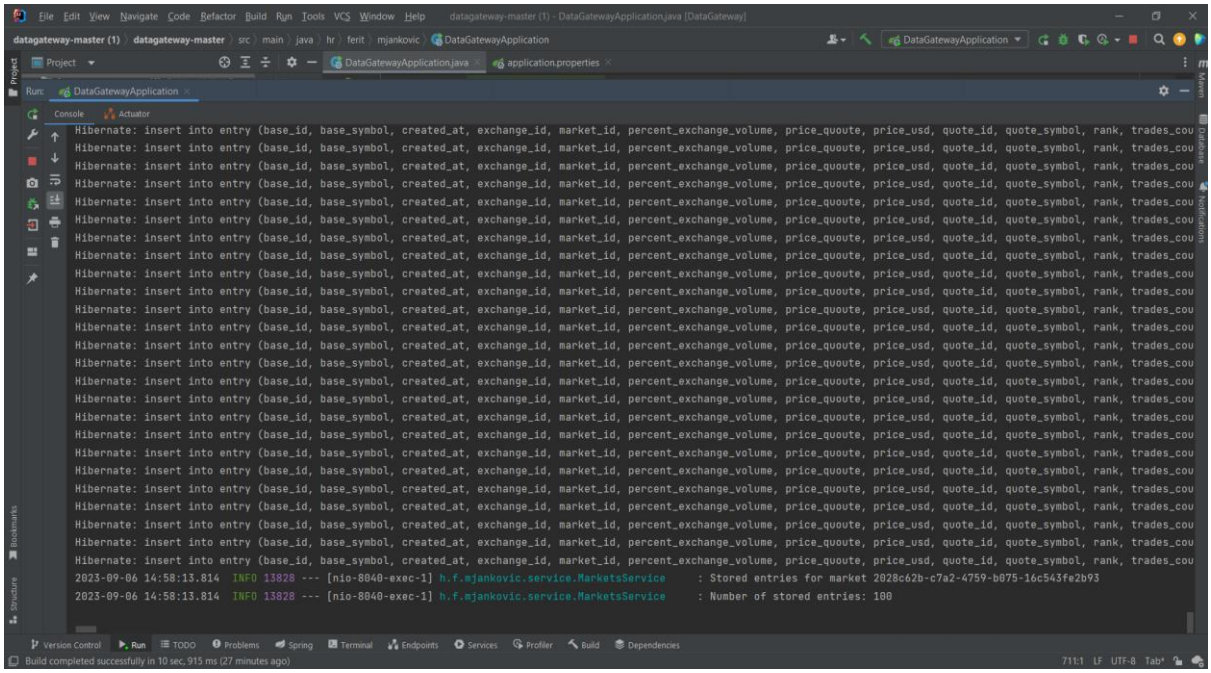
Slika 4.15. *Prikaz uspješnog pokretanja Eureka poslužitelja*

Uspješno dohvaćanje podataka sa API-ja vidimo na slici 4.16.



Slika 4.16. *Prikaz uspješnog dohvaćanja podataka*

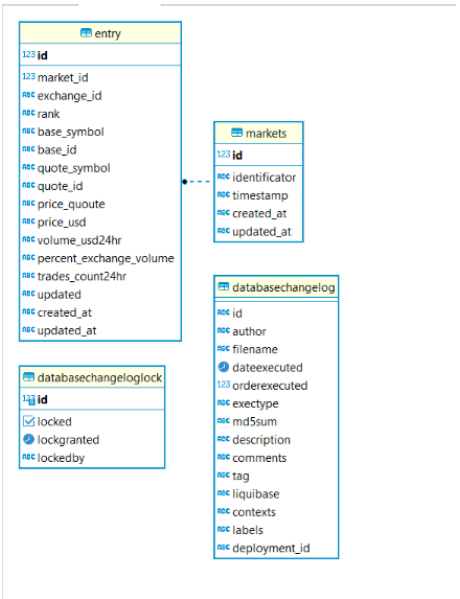
Posljednji korak koji potvrđuje ispravnost aplikacije je upis podataka u bazu što se vidi na slici 4.17



Slika 4.17. *Prikaz uspješnog upisivanja u bazu podataka*

4.6. Baza podataka

Baza podataka postavljena je na lokalnu mrežu i pristupa se na *portu* 5432. U bazi podataka postoje podatci o tržištu kriptovaluta te ostali podatci o vremenski bitnim činjenicama. Tablica *entry* i tablica *markets* povezane su preko stranog ključa id.



Slika 4.18. Prikaz baze podataka

Entitet 1: Entry

Ovaj entitet predstavlja podatke o kriptovaluti, ključan je aspekt naše aplikacije za praćenje financijskih podataka. Služi kao temeljna podatkovna struktura za pohranjivanje informacija o raznim financijskim tržištima i njima povezanim podatkovnim točkama. Uključuje sljedeća polja:

- `id` (primarni ključ): jedinstveni identifikator za svaki unos tržišnih podataka, automatski generiran korištenjem strategije identiteta.
- `market_id` (odnos više prema jednom): Referenca na pridruženo tržište, korištenjem odgode učitavanja i povezano preko stranog ključa "market_id".
- `exchangeId`: polje koje predstavlja identifikator razmjene.
- `rank`: Polje koje pohranjuje informacije o rangu.
- `baseSymbol`: Polje za osnovni simbol.
- `baseId`: Polje za osnovni identifikator.
- `quoteSymbol`: Polje za simbol navodnika.
- `quoteId`: Polje za identifikator ponude.
- `priceQuoute`: Polje za ponudu cijene.
- `priceUsd`: Polje za cijenu u USD.
- `volumeUsd24Hr`: polje za 24-satni volumen u USD.
- `percentExchangeVolume`: polje za postotak volumena razmjene.
- `tradesCount24Hr`: polje za broj trgovanja u zadnja 24 sata.
- `updated`: polje za vremensku oznaku koja pokazuje kada su tržišni podaci posljednji put ažurirani.
- `createdAt` (Vremenska oznaka stvaranja): Polje koje predstavlja vremensku oznaku stvaranja, koja je postavljena pri početnoj izradi i ne može se ažurirati.
- `updatedAt` (vremenska oznaka ažuriranja): polje koje predstavlja vremensku oznaku kada su tržišni podaci posljednji put ažurirani.

Entitet 2: Entitet tržišta

Ovaj entitet bilježi informacije o tržištu te vremenskoj oznaci i sastoji se od sljedećih polja:

- `id` (primarni ključ): jedinstveni identifikator za svaki unos vremenske oznake, generiran korištenjem strategije identiteta.
- ` timestamp` (Polje vremenske oznake): Polje za pohranu vremenske oznake, koja predstavlja određenu točku u vremenu.

- `identificator`: Polje za držanje identifikatora.
- `createdAt` (Vremenska oznaka stvaranja): polje koje predstavlja vremensku oznaku stvaranja, postavljeno pri početnoj izradi i ne može se ažurirati.
- `updatedAt` (vremenska oznaka ažuriranja): polje za vremensku oznaku kada su podaci zadnji put ažurirani.

Ovi entiteti i njihova odgovarajuća polja dizajnirani su za upravljanje i pohranjivanje tržišnih podataka i informacija o vremenskim oznakama, s odgovarajućim komentarima za specificiranje odnosa baze podataka, ograničenja i ponašanja.

Odnosi:

Entry - Markets odnos:

Entitet MarketDataEntity ima odnos Više-prema-Jedan s entitetom Tržište, povezan stranim ključem market_id. Ovaj odnos implicira da se više unosa podataka o tržištu može povezati s jednim tržištem, dok je svaki unos podataka o tržištu povezan s jednim tržištem.

5. Zaključak

Unutar ovog rada istražen je *Spring* programski paket i njegove ključne komponente kroz primjer jednostavne arhitekture. *Spring Cloud* je pokazao svoju moć u olakšavnju razvoja mikroservisnih aplikacija, pružajući alate za rješavanje složenih izazova u razmjeni podataka, konfiguraciji, pronalasku servisa i još mnogo toga. Kroz implementaciju se vidi kako API poveznik (API Gateway) može djelovati kao centralna točka za upravljanje zahtjevima i preusmjeravanje prometa prema odgovarajućim servisima. Konfiguracija omogućuje jednostavno upravljanje postavkama aplikacije, a *Spring Boot* ubrzava razvoj i olakšava upravljanje infrastrukturom. Najvažnija komponenta, *Service Discovery,* omogućuje dinamičko registriranje i otkrivanje servisa, čineći aplikaciju skalabilnom i otpornom na kvarove. Ovaj rad naglašava kako *Spring Cloud* može biti ključan alat za razvoj modernih, distribuiranih aplikacija. *Spring Cloud* pruža snažan okvir za izgradnju skalabilnih i pouzdanih mikroservisnih rješenja.

Literatura

[1] Service Discovery: Eureka Clients, dostupno na: https://cloud.spring.io/spring-cloud-netflix/multi/multi__service_discovery_eureka_clients.html [30.8.2023.]

[2] Spring Cloud Gateway, dostupno na: https://spring.io/projects/spring-cloud-gateway [30.8.2023.]

[3] L.Verma, Medium, Spring Cloud Config – Understanding the Basics, 2020., dostupno na: https://medium.com/swlh/spring-boot-microservices-developing-config-as-a-service-fa4866085086 [28.8.2023.]

[4] C. Richardson, What are microservices, dostupno na: https://microservices.io/ [28.8.2023.]

[5] L. Laxmikant, Microservice arhitecture with Spring Boot, 2020., dostupno na: https://lakshyajit165.medium.com/microservice-architecture-with-spring-boot-732d58b1c695 [27.8.2023.]

[6] Spring, Spring Framework Overview, dostupno na: https://docs.spring.io/spring-framework/reference/overview.html [5.9.2023.]

[7] R. Johnson, Expert One-on-One: J2EE Design and Development, Wrox, 2002.

[8] Spring, Spring Cloud, dostupno na: https://spring.io/projects/spring-cloud [28.8.2023.]

[9] Apache Maven Project, Maven, dostupno na: https://maven.apache.org/what-is-maven.html [29.8.2023.]

[10] Java, What is Java technology and why do I need it?, dostupno na: https://www.java.com/en/download/help/whatis_java.html [27.8.2023.]

Sažetak

Unutar servisa prikazano je registriranje klijenata na *Eureka* poslužitelj te njihovo međusobno komuniciranje. Svaki od registriranih klijenata može saznati koji je još klijent registriran na poslužitelj. Ukratko *Spring Cloud* elementi poput Eureka poslužitelja, *Spring Cloud Gatewaya* i *Spring* konfiguracije pripomažu u stvaranju skalabilnih mikroservisnih aplikacija. Pružaju mogućnost dinamičkog dodavanja i uklanjanja servisa. Također *Spring Config* pruža lako upravljanje konfiguracijskim datotekama. Takva arhitektura i način stvaranja aplikacija olakšava razvoj i održavanje aplikacija.

Ključne riječi: API Gateway, *Eureka* poslužitelj, mikroservisi, prikupljanje podataka, pohrana podataka

Abstract

Title: Cloud application development using Spring Cloud software package

Within the service, the registration of clients on the Eureka server and their communication with each other are demonstrated. Each of the registered clients can discover which other clients are registered on the server. In short, Spring Cloud elements like the Eureka server, Spring Cloud Gateway, and Spring configuration aid in creating scalable microservices applications. They provide the capability for dynamic addition and removal of services. Additionally, Spring Config offers easy management of configuration files. Such an architecture and development approach streamline the development and maintenance of applications.

Keywords: API Gateway, Eureka server, microservices, data collection, data storage

Životopis

Mihovil Janković rođen je 16.02.2001. godine u Slavonskom Brodu. Završava Osnovnu školu Antun Mihanović u Slavonskom Brodu 2015.godine te upisuje Gimnaziju Matije Mesića, prirodoslovno-matematički smjer. Nakon završetka srednje škole i položene mature upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski sveučilišni studij računarstva.

_____

Potpis autora