



Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Departman za matematiku i
informatiku



Robert Sabo

Skalabilan mikroservis orijentisan sistem namenjen berzi kripto valuta

C r y p t O f f e r

-Diplomski rad-

Novi Sad, 2018

SADRŽAJ

PREDGOVOR	5
1. UVOD	7
Šta su kriptovalute?	7
Domen primene i cilj <i>CryptOffer</i> -a	8
<i>CryptOffer</i> kao zatvoren sistem kontrole zajedničkih resursa	9
Cilj razvoja <i>CryptOffer</i> -a	10
2. KORIŠĆENE TEHNOLOGIJE	11
2.1. Klijentski deo aplikacije	12
Angular	12
Bootstrap	20
2.2. Serverski deo aplikacije	21
Spring Boot	22
Baza podataka	25
Komunikacija	27
Sigurnost podataka	33
3. ARHITEKTURA I SARŽAJ SOFTVERA	36
3.1. Angular klijenska aplikacija	38

3.2.	CryptOffer	38
3.3.	CryptoAuthService	39
3.4.	CryptoEurekaServer	40
3.5.	CryptoAdvertising	41
3.6.	CryptoWallets	42
4.	IMPLEMENTACIJA SOFTVERA	43
4.1.	Angular klijentska aplikacija	43
4.2.	CryptOffer	46
4.3.	CryptoAuthService	49
4.4.	CryptoEurekaServer	50
4.5.	CryptoAdvertising	51
4.6.	CryptoWallets	54
5.	UPOTREBA APLIKACIJA	56
6.	ZAKLJUČAK	64
7.	LITERATURA	65
8.	REFERENCE	66
	BIOGRAFIJA	68

PREDGOVOR

CryptOffer je naziv cele aplikacije kao projekta. Naziv je izveden iz dva dela: *crypto* i *offer*. *Crypto* usmerava na domen upotrebe, odnosno na svet kriptovaluta, dok *offer* prevedeno sa engleskog znači ponuda. Spojem ova dva izraza dobijamo zanimljivo ime koje asocira na berzu kriptovaluta, a koristiće se kao naziv ovog projekta kroz ceo razvoj i dokumentaciju.

Diplomski rad sastoji se od sledećih poglavlja:

1. U *uvodnom* delu je opisan cilj i domen primene aplikacije. Čitalac se uvodi u osnove domena. Nakon toga se stavlja akcenat na objašnjenje funkcionalnog cilja aplikacije, te se objašnjava za šta se je namenjena. Na kraju uvoda je analizirana druga mogućnost upotrebe iste aplikacije.
2. Poglavlje *Korišćene tehnologije* opisuje sve tehnologije korišćene za izradu ovog rada. Tehnologije su podeljene na 2 dela: klijentske i serverske. Kod klijenskog dela su, uz opis razvojnog *framework*-a, objašnjeni i ključni delovi funkcionisanja istog. Sa serverske strane se stavlja akcenat na korišćene pomoćne biblioteke, kao i objašnjavanje njihovog rada. U ovom delu se nalaze objašnjenja funkcionisanja inovativnih biblioteka namenjenih za mikroservis orijentisan razvoj.
3. U poglavlju *Arhitektura i sadržaj mikroservisa* je data celokupna arhitektura mikroservisa kao i distribucija logike i implementacije. Prikazan je dijagram arhitekture mikroservisa te je objašnjena uloga i funkcionalnost svakog od njih. Bitna uloga ovog poglavlja se ogleda u razumevanju svrhe svakog mikroservisa i uloge tehnologije koje uključuje.
4. Poglavlje *Implementacija softvera* se sastoji od prikaza samog koda aplikacija. O svakom segmentu aplikacije su opisani najznačajniji i najkarakterističniji delovi implementacije istog. Osim karakterističnih implementacija, za svaki deo aplikacije je prikazan način upotrebe karakterističnih tehnologija i biblioteka koje koristi
5. *Upotreba aplikacije* sadrži upustvo za korišćenje aplikacije. Prvo je prezentovan način pokretanja kao i potrebna priprema za redovan rad celokupnog sistema. Nakon toga, uz slike korisničke perspektive,

objašnjen je rad aplikacije. Takođe, ovo poglavlje sadrži zanimljiv prikaz testiranja rada balansera zahteva (Ribbon-a) koji igra glavnu ulogu u skalabilnosti ove aplikacije.

6. U *zaključku* se nalazi lični utisak procesa razvoja ovog projekta kao i generalan zaključak stečen ovim radom.
7. U *literaturi* je spisak svih korišćenih literatura pri izradi projekta.
8. *Reference* predstavljaju smernice gde se nalaze detaljnije informacije o terminima koji su korišćeni a nisu obrađivani u ovom radu.

Mikroservis je tehnika razvoja *software*-a bazirana na servis orijentisanoj arhitekturi (SOA[1]). Osnovna ideja je podela sistema na nezavisne celine. Svaka celina predstavlja po jedan mikroservis koji je precizno definisan svojim interfejsom i poslovnom logikom koju izvršava. Osnovna prednost ovakve arhitekture je skalabilnost, što je iz godine u godinu sve neophodnije razvojem tehnologije i zainteresovanosti korisnika,. S druge strane, prednosti ovakve arhitekture je i modularnost u cilju lakšeg razumevanja, razvoja i testiranja.

1. UVOD

Drugom polovinom 2017. godine došlo je do ogromne popularizacije **kriptovaluta**. Najveću pažnju je privuklo kretanje cena kriptovaluta, potpuno očekivano, rastom popularnosti raste tražnja pa raste i sama cena, što je pogrešan pravac razmišljanja u sferi tehnologije: Ono što ovakvu pojavu čini dobrom, po društvo pre svega, jeste njena tehnologija odnosno način funkcionisanja. Implementacija i princip po kom rade kriptovalute dovodi do minimalnih mogućnosti korupcije i malverzacije.

Šta su kriptovalute?

Kriptovaluta je oblik digitalne imovine koja se koristi kao sredstvo razmene. Definiše se kao **digitalni novac**: Kriptovalute imaju sve potrebne karakteristike da budu novac odnosno sredstvo plaćanja. Fizička i pravna lica ih poseduju, koriste se za plaćanje dobara i usluga, i nemaju upotrebnu vrednost već se sva vrednost ogleda na tržištu (isti slučaj kao i novac). Prva razlika digitalnog novca od standardnog, što lično smatram i prednošću, je to što je univerzalan za sva lica bez obzira na nacionalnu, geografsku ili bilo koju drugu različitost. Svi mogu da koriste isti novac i svuda mu je vrednost relativno jednaka, što nije slučaj sa novcem (sa srpskim dinarima se trguje samo u Srbiji). Druga bitna razlika je arhitektura monetarnog sistema. Preciznije kriptovalute su decentralizovane, dok je redovna moneta centralizovan sistem. Razlika je što kod centralizovanog sistema jedna, centralna, jedinica je glavna i ona čuva podatke bez sumnje u tačnost, dok kod decentralizovanog sistema mnogo jedinica čuva isti podatak i validan je podatak (konzensus) koji većina tvrdi tačnim. Ovakav način organizovanja čini kriptovalute sigurnim, pouzdanim od korupcije i malverzacija, i nezavisnim od bilo kakve geografske ili nacionalne jedinice.

Kriptovalute koriste kriptografiju čime obezbeđuje svoje osnovne benefite:

- sigurnosti transakcija
- kontrola stvaranja dodatnih novčanih jedinica
- kontrola potvrde transfera valute (konzensus).

Domen primene i cilj *CryptOffer*-a

Cela prethodna priča je uvod u domen upotrebe sistema ovog rada i predstavlja alibi odabira ovakvog pravca razmišljanja. Kao što se i u samom naslovu napominje, u pitanju je berza kriptovaluta. Inicijalna ideja i na kraju realizacija je usmerena u pravcu oglasa. Naime, osnovni pravac su samo prosti kupoprodajni oglasi čiji su predmet trgovine isključivo kriptovalute. S obzirom da su kriptovalute finansijska sredstva ovaj oglasnik ipak predstavlja **berzu**. Ova ideja se razvila u modul po imenu *CryptoAds*.

Razvojem ideje, predviđen je još jedan vrlo koristan modul koji se lepo slaže sa berzom. Svako lice na berzi ima **javnu** i **privatnu** reprezentaciju. Javna je predstavljanje lica na samom tržištu, odnosno njegova ponuda i tražnja. Pod privatni deo spada ono što lice ne iznosi javnosti već služi sopstvenoj evidenciji. Tu se podrazumeva kalkulacija imovine lica, odnosno prikaz rezultata rezultat trgovine kao i odgovori na pitanja kao što su: koji koraci su bili dobri, koji ne, na kome mestu se napravila greška, potencijalni dobici i sl. Gore navedena inicijalna ideja oglasnika (*CryptoAds*) pokriva samo javnu prezentaciju, odnosno samu berzu. Međutim, *CryptOffer* ima ideju realizacija i privatnog dela.

CryptoWallets je modul *CryptOffer*-a čiji je zadatak (prethodno opisan) privatna reprezentacija lica na berzi. Ovaj modul ima za zadatak da korisniku pruži podatke o trenutnom rezultatu trgovine kriptovaluta. Korisnik u ovom modulu ima mogućnost pregleda, kako celokupnog rezultata, rezultata po valutama investiranja tako i rezultat i ocenu svake pojedinačne transakcije (kupovine ili prodaje).

Radi šire mogućnosti korišćenja specifikacije *CryptoWallets*-a dodaje se i funkcionalnost ručnog dodavanja kriptovaluta na stanje. Ova funkcionalnost daje smisao i korišćenju celog sistema i korisnicima koji ne žele da trguju već samo žele da prate stanje i kretanje uložених sredstava.

Kako bi *CryptoWallet* nudio realne podatke potrebno je da u realnom vremenu sistem ima informacije o trenutnim cenama kriptovaluta. Na taj način se pruža prikaz podataka u realnom vremenu. Obezbeđivanje ovih

podataka je iz realnog sistema berze što je realizovano eksternim izvorom podataka.

Ukratko sumirana ideja aplikacije:

- *CryptoWallets* prikazuje ulogovanom korisniku stanje kapitala unetog od strane korisnika
- *CryptoWallets* prikazuje ulogovanom korisniku stanje kapitala razmenjenog na *CryptoAds*
- *CryptoAds* vrši evidenciju oglasa i utiče na stanja *CryptoWallets*-a
- Interni izvor podataka o stanju i eksterni izvor podataka o realnim cenama

CryptOffer kao zatvoren sistem kontrole zajedničkih resursa

Inicijalna ideja razvoja *CryptOffer*-a je rađena u pravcu javno dostupnog sistema koji može koristiti ko god poželi. U tom slučaju *CryptoAds* sistem služi za stupanje u kontakt klijenata i razmene valuta na taj način. Problem koji se pojavljuje je da nakon obavljene transakcije unutar ovog sistema korisnici moraju i realnu promenu da izvrše: prodavac pošalje kriptovalute u kriptonovčanik kupca, a kupac uplati novac prodavcu.

Međutim, šta ako bi svi korisnici koristili isti kriptonovčanik? U tom slučaju, ukupna raspoloživa sredstva u kriptonovčanicima su svačija. *CryptOffer* savršeno dogovara ovom slučaju korišćenja: on bi imao ulogu da u svakom trenutku korisniku da podatak koliki je njegov udeo. Pored toga, pri razmeni valuta ne bi bilo potrebe prebacivati iz jednog kriptonovčanika u drugi jer su svi resursi na istom mestu samo je pitanje raspodele učešća.

Ovakav slučaj korišćenja gubi na smislu pri globalnoj upotrebi (kao otvoren javnodostupan sistem) jer je sam po sebi centralizovan a jedna od glavnih prednosti kriptovaluta je decentralizovanost. Uvođenje centralizovanog modula u decentralizovani sistem dovodi do apsurdna i isključuje u potpunosti benefite sistema kriptovaluta. Međutim, ukoliko bi se ovako zamišljen *CryptOffer* koristio u zatvorenom krugu korisnika, na primer unutar jedne

organizacije, cela ideja *CryptOffer*-a dobija mnogo na značaju, primeni i upotrebnoj vrednosti.

Primer upotrebe da jedna organizacija (preduzeće) sve svoje finansijske transakcije vrši u kriptovalutama. Tada modul *CryptoWallet* pruža stanje. Direktni unos transakcija u *CryptoWallet* su ulazi i izlazi novca iz organizacije (prodaja kriptovalute je isplata, npr.: radnik podiže gotovinski novac; kupovina je uplata kriptovaluta, npr.: klijent plaća usluge). *CryptoAds* modul bi, uz minimalna promene vezane za prava, bio zadužen za kretanje novca unutar organizacije (plata radnika bi bila samo prodaja valute od strane organizacije konkretnom radniku). U ovom primeru, *CryptOffer* jedini dodir sa eksternim promenama (realne promene na kriptonovcaniku) ima pri ulazu i izlazu digitalnog novca iz organizacije, sva ostala kretanja su u potpunosti podržana u samom sistemu *CryptOffer*-a koji pruža vrlo lak i jednostavan sistem prenosa vlasništva.

Cilj razvoja *CryptOffer*-a

Pored postizanja funkcionalnih zahteva, *CryptOffer* nosi sa sobom cilj edukativne prirode. Osnovna motivacija razvoja u mikroservis orijentisanom okruženju je sticanje iskustva i novih znanja u ovoj oblasti. Naime, ovaj način arhitekture je relativno nov u svetu informacionih tehnologija.

Ovim radom se vrši istraživanje u svetu mikroservisa. Ideja je otkrivanje korisnih i dobro upotrebljivih alata iz ove oblasti. Naime, s obzirom da je mikroservis orijentisana arhitektura još uvek relativno nova, većina razvijenih alata mogu biti upotrebljivi i korisni u nekim slučajevima, dok u drugim i nemaju preveliki značaj. Dakle, cilj ovog rada je analizirati određenog skupa tehnologija i alata te doći do zaključka u kojim slučajevima su oni upotrebljivi i potrebni a u kojima ne igraju značajnu ulogu.

2. KORIŠĆENE TEHNOLOGIJE

CryptOffer je, celokupno gledano, web aplikacija. Korisnici joj pristupaju i koriste je putem web-browser-a (pretraživač) putem adrese. Sve ostalo potrebno za rad aplikacije treba da je podešeno na serveru i krajnji korisnici nemaju dodira sa tim.

Web aplikacije su Klijent-Server orijentisane. To znači da cela struktura može podeliti na dva dela:

- Klijentski deo aplikacije
- Serverski deo aplikacije

Klijentski deo aplikacije predstavlja kod koji se izvršava u samom web-browser-u korisnika i on je najčešće zadužen da vizualizaciju, odnosno odgovarajući prikaz aplikacije. Takođe, pored prikaza služi i za koordiniranje pri unosu podataka. Zbog toga se ovaj deo zove *IO System* – Input-Output eng. odnosno ulaz-izlaz. Pored funkcionalnosti (IO), klijent ima za zadatak prilagođavanje i ulepšavanje radi lepšeg utiska samog korisnika (*user- experience*).

Serverski deo predstavlja aplikaciju koja se bavi posluživanjem podataka klijentima. Za razliku od klijenta, gde se kod izvršava u browseru svakog korisnika, ovaj kod se uglavnom izvršava na jednom mestu, na serveru. Osnovna uloga servera je obsluživanje klijenata. Pored toga, najčešće se server bavi celom poslovnom logikom aplikacije.

2.1. Klijentski deo aplikacije

Za izradu klijentskog dela web aplikacije se najčešće koristi kombinacija HTML-a[6], kao deskriptivnog jezika statičkog dela stranica, i JavaScript[4] kao programski jezik koji se izvršava od strane web-browser-a i koji dopunjava promenljive delove stranica. Ova kombinacija je najzastupljenija u svetu web klijentskih aplikacija, međutim, većina web aplikacija se ne sastoji samo od navedenog. Za lakši razvoj i održavanje, prvenstveno zbog preglednosti, koristi se *framework*[2] (prevodi se kao radni okvir). Za razvoj klijenta *CryptOffer*-a korišćen je Angular.

Angular

Angular je *framework* za razvoj aplikacija razvijen 2009. godine od strane Google-a[5]. U prvoj verziji, ovaj okvir je nazivan AngularJS, a počevši od druge verzije, koji se prvi put pojavljuje 2016. godine, samo Angular. Sufiks "JS" upućuje da se radi o JavaScript *framework*-u. Izbacivši "JS" iz imena, Google ukazuje da se uz razvoj web aplikacija, omogućava i razvoj mobilnih kao i desktop aplikacija, a time Angular postaje univerzalni okvir.

Angular *framework* napisan je u programskom jeziku TypeScript. Ovaj programski jezik razvijen je od strane Microsoft-a[7]. Njegova je svrha ispraviti nedostatke JavaScript-a. TypeScript kod nije izvršiv od strane web-browser-a pa se on prevodi (eng. compile) u JavaScript kod. TypeScript omogućava strogo tipiziranje (eng. strong typing) nad JavaScript kodom, tj. mogu se deklarirati tipove podataka, što nije slučaj kod JavaScript-a. Na taj način je omogućena provera nad tipovima podataka prilikom prevođenja koda kako ne bi došlo do greške prilikom izvršavanja. Tipska greška bi se primetila i u JavaScript-u (JS) i u TypeScript-u (TS). Razlika je što u JavaScript-u do greške dolazi tek pokretanjem aplikacije od strane web-browser-a (kaže se u runtime-u aplikacije), dok se u TypeScriptu tipska greška signalizira prilikom prevođenja (compiletime), za vreme samog kucanja koda. Pored strogo tipiziranja i otkrivanja grešaka prilikom prevođenja, značajna nadopuna JavaScript kodu jeste uvođenje mehanizama objektno-orijentiranog programiranja.

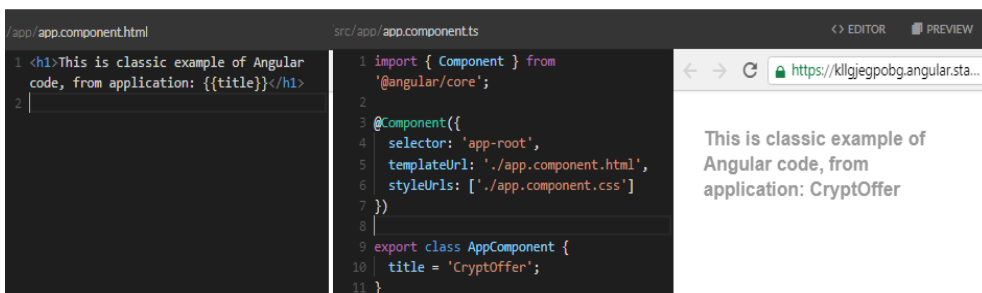
Angular aplikacija sastoji se od mnoštva komponenti koje čine stablo. Skup komponenti čini logičku celinu, odnosno modul. Svaka aplikacija razvijena u Angularu sastoji se od barem jednog modula, koji sadrži korensku komponentu. Ostale komponente se ugnježdavaju unutar korenske. Osim komponenti, Angular pruža i mnoge druge funkcionalnosti između kojih najznačajniju ulogu igraju: servisi i direktive. U okviru *CryptOffer*-a su pravljene komponente i servisi, dok su direktive korišćene predefinisane samim *framework*-om. Sve komponente i ceo sastav modula se navodi u okviru specijalne klase *app-module.ts*[8]. Sadržaj navedenih komponenti, servisa i drugih angular fajlova u ovaj fajl zaokružujemo celinu jednog modula.

Komponente u Angularu

Komponenta predstavlja osnovnu i najupotrebljeniju jedinicu unutar Angular projekta. Podrazumevana građa jedne komponente čine 4 fajla:

- Šablon (template) – *.html – HTML statički kod deskriptivnog tipa koji se dopunjava Angular specifičnim delovima kao dinamičko referenciranje za podatke iz klase. Angular specifični dinamički kod se navodi unutar duplih vitičastih zagrada (Kod 2.1.1. – levi deo). Prilikom pokretanja stranice, HTML kod će se interpretirati dobro poznatim načinom od strane web-browser-a, dok će dinamički deo biti zamenjen realnim vrednostima iz objekta klase.
- Klasa – *.ts – Kod objektno-orijentisane klase koja ima zadatak da dopuni template kako podacima tako i logikom. Svako polje i metoda definisane u klasi su vidljive i u šablonu komponente. Uvezana polja (eng. binded) su dvosmerno uvezana. U slučaju promene vrednosti u objektu klase, izmena će se reflektovati i na prikazu, kao i izmena od strane korisnika u nekom izmenjivom delu stranice direktno menja vrednost iste promenljive u objektu klase. Takođe, reakcija promena je automacka, svakakva promena se izvršava u oba smera u trenutku, bez potreba pozivanja dodatnih metoda kako bi se promena reflektovala (što nije slučaj kod većine tehnologija).

- CSS – *.css – fajl koji je namenjen za specificiranje stila komponente. Nije neophodno koristiti ovaj fajl, moguće je ceo stil definisati na nivou aplikacije, što je slučaj kod *CryptOffera*
- Test – *.spec.ts – fajl namenjen za testiranje komponente. (nije

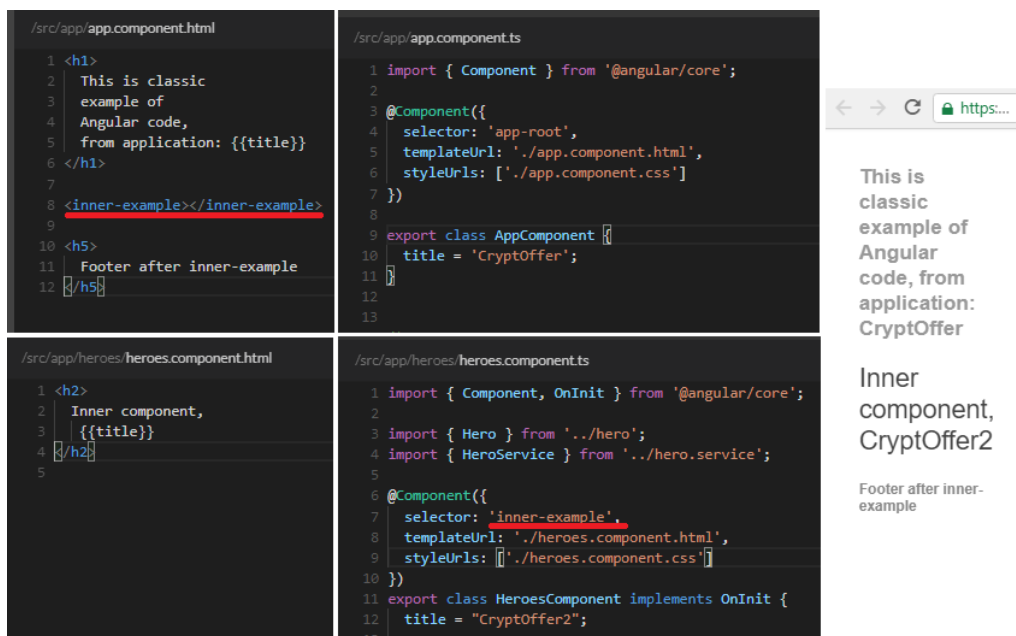


Kod 2.1.1.- Primer komponente u Angular-u. Levi deo je HTML sa Angular specifikacijom. Sredina je TypeScript klasa sa vrednost. Desno je predstavljen rezultat ovog koda

korišćeno niti izučavan za ovaj diplomski rad)

Glavni deo komponente je klasa. U okviru definisanja klase se navodi pored imena klase još i ime komponente – **selector** (služi za referenciranje putanja), link ka šablonu – **templateUrl** i CSS – **styleUrls** fajlovima (Kod 2.1.1. – srednji deo. U okviru pasusa su boldovana imena koja referenciraju na prezentaciju sa slike).

Međukomponentsko uvezivanje se definiše unutar šablona. Da bi se unutar jedne komponente prikazala druga, stavlja se element sličan HTML-tagu, osim što naziv taga predstavlja ime komponente (atribut selector sa Kod 2.1.1 – srednji deo). Primer uvezanih komponenti se može videti na Kod 2.1.2. (Primer je samo skiciran radi prezentacije. Konkretni delove implementacija u poglavlju 4. Implementacija softvera)



Kod 2.1.2- Ilustracija ugnjezdavanja komponenti. Ključne reči uvezanosti podvučene raznom crvenom crtom. Gornji 2 fajla predstavljaju korensku komponentu dok su donji fajlovi prezentacija inner-example komponente. Desno je prikazan rezultat izvršavanja koda.

Dakle, komponente služe za građenje vizuelnih delova aplikacije – popunjavanje statičkih stranica promenljivim podacima. Postavlja se pitanje iz kojih izvora se dobavljaju podaci komponentama.

Za generisanje svih tipskih fajlova Angular ima ugrađen metod generisanja. To znači, da za pravljenje komponente nije potrebno napraviti 4 fajla već je dovoljno pokrenuti odgovarajuću komandu u Command Prompt-u[9] (Kod 2.1.3):

```
>ng generate component dashboard
```

Kod 2-1.3 CMD comanda za generisanje Angular komponente. Ova naredba će izgenerisati komponentu po imenu 'dashboard-component' kako je to navedeno u komadni

Uloga servisa je dobavljanje i pružanje podataka komponentama. Za razliku od komponente, servis se sastoji od 2 fajla:

- Klasa – *.ts – TypeScript klasa koja je zadužena za logiku pružanja podataka
- Test – *.spec.ts – klasa za testiranje

Servisi unutar aplikacije su najčešće korištene od strane komponenti. Cilj je da komponente budu što jednostavnije, odnosno da se servisima prepuste svi složeniji zadaci kao što su dobavljanje podataka sa servera ili validacija korisničkog unosa. Naime, komponenta bi trebala biti posrednik između pogleda koji se iscrtava koristeći šablon komponente i logike aplikacije. Dobro napisana komponenta sadrži samo promenljive i metode za vezanje podataka, a sve netrivialne zadatke prenosi servisima. Ovakav stil razvoja aplikacije, prebacivanje logike aplikacije na service, te korištenje tih servisa unutar komponenti, značajno olakšava razvoj, nadzor i održavanje aplikacije.

Kao za komponentu, tako i za generisanje servisa postoji ugrađena naredba. To znači da se vrlo lako, jednom naredbom, izgenerise servis sa zadatim imenom. Primer generisanja servisa može se videti na kodu 2.1.4. Pored generisanja fajlova koji uključuju generisanje servisa, ovom naredbom se novogenerisani servis navodi i u okviru klase za registrovanje svih delova modula (app-module.ts). Na taj način ova naredba izvršava sav potreban posao generisanja novog servisa (komponente ili drugo).

```
>ng generate service auth
```

Kod 2-1.4 CMD komanda za generisanje Angular servisa. Ova naredba će izgenerisati servis po imenu 'auth-service' koj će biti registrovan u okviru modula.

Dependency Injection

Uvezivanje komponenti i servisa se vrši dobro poznatim paternu po nazivu *Dependency Injection* [10]. Ovaj patern služi za korišćenje objekta bez potrebe eksplicitnog instanciranja zahtevanog objekta. U različitim tehnologijama različito se navodi korišćenje ovog paternu, ali je suština upotrebe svugde ista. Jedan način je da klasi koja hoće da koristi objekat, kroz konstruktor dostavimo postojeću instancu traženog objekta. Angular naznake za uvezivanje komponenti i servisa koristi baš ovaj patern i ovaj primer. Naravno, na ovaj način je u Angularu moguće međusobno uvezivanje i drugih stvari, ne samo komponente i servisa, npr. međusobno uvezivanje servisa, i sl. Primer je prikazan u kodu 2.1.5.

Patern *Dependency Injection* se koristi i u SpringBoot mikroservisima ove aplikacije.

```
export class CoinDetailComponent implements OnInit {  
  
  constructor(  
    private route:ActivatedRoute,  
    private wallet:WalletService  
  ) {}  
  
  ngOnInit() {...}
```

Kod 2.1.5 – Dependency Injection – primer korišćenja implicitnog uvezivanja objekta koji CoinDetailComponent koristi: ActivatedRoute i WalletService. Nakon ovakvog navođenja u svakom delu koda ove klase možemo koristiti navedene servise i tako da nam ovaj princip garantuje prisutnost objekta.

Direktive predstavljaju izvršni kod u okviru HTML stranica komponenti. U okviru komponente je da šablon komponente sadrži deskriptivni deo – HTML i dinamički deo koji se menja realnim vrednostima iz objekta klase komponente. Međutim, ovde nije uzeto u obzir dinamičnost dekrriptivnog dela: šta ako nije tačno utvrđen broj redova tabele, na primer, ili ukoliko prikazivanje određene komponente treba da zavisi od vrednosti. Za rešavanje navedenih slučajeva u Angularu postoje direktive.

Direktive se navode kao atributi u okviru HTML tagova. Njihovo navođenje počinje znakom *. Najčešće korišene direktive su:

- `*ngFor` – govori Angularu da višestruko ispiše tag u kome se nalazi. Pored toga, ova direktiva ima i karakteristiku da svaki element iz mapirane kolekcije (u primeru je to `buyings`) dodeli promenljivoj koja će se koristiti u tom tagu (primer u kod 2.1.6)

```
<tr *ngFor="let b of buyings">
  <td>{{b.name }}</td>
</tr>
```

*Kod 2.1.6 – Primer directive `*ngFor`. Ovako naveden kod će ispisati za svaki objekat u kolekciji `buying` njegovo atribut `'name'`.*

- `*ngIf` – govori Angularu da se tag u kome je naveden prikazuje samo ako je logička vrednost navedenog izraza tačna (primer Kod 2.1.7)

```
<div *ngIf="message" class="alert-danger" role="alert">
  {{message}}
</div>
```

*Kod 2.1.7 – Primer directive `*ngIf`. Div tag i njegov sadržaj će biti prikazani samo ukoliko je promenljiva po imenu `'message'` logički tačna (sadrži vrednost)*

Zaključak o Angularu

U ovom ovom delu su navedene najistaknutije novonaučene karakteristike Angular *framework*-a koje su po sebi i svom načinu funkcionisanja došle do izražaja. Angular je mnogo moćna i prostran alat, dok navedene karakteristike predstavljaju samo osnove funkcionalnosti korišćene pri izradi ovog rada.

Pored navedenih karakterističnih stvari za Angular, u mogućnosti upotrebe stoje i mnogo drugih softverski dobro poznatih stvari, kao što su (navode se upotrebe u izradi *CryptOffera*):

- klase modela (entiteti) za grupisanje podataka
- interseptori za prestretanje poruka komunikacije
- mehanizam za navigaciju između stranica
- filteri prikazivanja podataka na stranicama (primer formatiranje ispisa datuma) i sl.

Bootstrap

Bootstrap je *framework*, otvorenog koda, za razvijanje mobilnih i web aplikacija koncentrisan na vizualni deo aplikacije. To je alat za razvoj uz pomoć HTML-a, CSS-a i JS-a.

Ideja kreiranja Bootstrap-a jeste da se olakša programeru vizuelizacija i stilsko sređivanje web aplikacije. Pomoću ugrađenog grid sistema za pozicioniranje elemenata stranice, obimnih ugrađenih komponenti i stilova, kao i mogućnosti dodataka zasnovanih na jQuery-u [11] ili JavaScript-u, stilizacija i vizualizacija web aplikacije je u mnogo olakšana.



Slika 2.1.8 – Prezentaciona slika koja upućuje na glavnu prednost Bootstrap-a: široka prilagodljivost razlicitim uređajima i rezolucijama

Bootstrap se može svesti na tri glavne datoteke:

- bootstrap.css – CSS framework
- bootstrap.js - JavaScript/jQuery framework
- glyphsicons – font (ikone za skup font-ova)

Bootstrap u sebi ima ugrađen set alata i biblioteka za vizuelno sređivanje web aplikacija. Primenuje se prilagodljivi dizajn u kojem se prikaz unutar web-browser-a prilagođava širini prozora, a za to je zaslužan njegov grid koncept rasporeda(eng. grid system). Unutar grid koncepta, prikaz se sastoji od redova, a svaki red od najviše 12 kolona čija se širina menja u zavisnosti od širine prozora. Stoga je prikaz jedne aplikacije prilagođen za uređaje različitih rezolucija: od visoko rezolucionarnih prezentacija do malih telefonskih prezentaija.

2.2. Serverski deo aplikacije

Serverski deo aplikacije je zadužen za poslušivanje podataka klijentu. Na poziv klijenta obrađuje primljeni zahtev. Zahtevi se kategorizuju u dve namene: zahtevanje podataka i unos podataka. Server se najčešće, za razliku od klijenskog koda, izvršava na jednom mestu i jedna aplikacija služi za obluživanje svih klijenata. Server koristi razne tehnologije kako bi najopštije i najuspešnije vršio svoj posao. Većinu tehnologija korišćenu u *CryptOffer* u će biti objašnjenje u ovom poglavlju.

U prethodnom pasusu je navedeno da najčešće jedna serverska aplikacija oblužuje sve klijente. Međutim, *CryptOffer* je primer web aplikacije gde serverski deo nije jedna aplikacija već skup koji uključuje nekolicinu aplikacija. Serverska logika distribuira se na različite celine kako bi se postigla skalabilnost. To je postignuto mikroservisnom orijentisanošću. Kako bi obezbedili sve potrebne funkcionalnosti skalabilnog mikroservis orijentisanog sistema potrebno je rešiti sledeće ključne probleme:

- Baze podataka i rad sa bazama— digitalna skladišta u kojima se čuvaju podaci kao i alati za manipulisanje tim podacima
- Bezbednost – podsistem odgovoran za ograničavanje dostupnosti podataka (podaci nisu javni)
- Komunikacija – mikroservisi treba međusobno da komuniciraju. Takođe, ovde se uključuje postizanje skalabilnosti.

Pre uvođenja dodatnih tehnologija potrebno je odabrati osnovu, odnosno tehnologiju u kojoj će se razvijati svaki pojedinačni mikroservis. Za to je odabran *SpringBoot*. Iako *CryptOffer* sve mikroservise razvija u istoj tehnologiji, to nije ograničenje. Mikroservisi jednog sistema ne moraju da budu u istoj tehnologiji. Arhitektura mikroservisa *CryptOffera* je opisana u poglavlju 3-Arhitektura i sadržaj mikroservisa. Ovo poglavlje opisuje pomoćne tehnologije koje koriste mikroservisi.

Spring Boot

Svaki od mikroservisa je napisan u programskom jeziku Java[3] koristeći *Spring Boot framework*. Spring Boot predstavlja vrlo uprošćenu osnovu za pokretanje kompleksnih stvari bazirano na principu podrazumevanih podešavanja. On predstavlja nadogradnju na *Spring framework* koji predstavlja moćan i napredan alat za razvoj širokog spektra serverskih funkcionalnosti web aplikacija. Spring kao alat ima manu koju *SpringBoot* nadogradnja rešava, a to je kompleksnost konfigurisanja u samom početku razvoja.

Spring je framework otvorenog koda koji predstavlja kontejner za izgradnju serverskog dela web aplikacija. Sve Springove komponente su pod nadzorom kontejnera koji je zadužen za njihovo efikasno korišćenje. Spring komponente su sledeće:

- Kontroler – predstavlja prezentacioni deo aplikacije. Zadužen je za komunikaciju sa klijentima, mapiranje i preusmeravanje klijentskih zahteva ka odgovarajućim servisima
- Servisi – predstavljaju poslovnu logiku servera. Zaduženi su za pružanje različitih usluga svima koji ih pozivaju (najčešće pozivanje od strane kontrolera ili međusobno pozivanje servisa)
- Repozitori – predstavlja logiku skladištenja podataka u bazu podataka.



Slika 2.2.1 Spring -> SpringBoot. Slikovit prikaz izmena i modifikacija Springa koja čini SpringBoot

Pored navedenih komponenti, u *Springu* ključnu ulogu igra i konfigurisanje aplikacije. Moćan alat ima širok spektar mogućnosti i funkcionalnosti što zahteva konfigurisanje u istim dimenzijama. Iskustvom je primećeno da u početnoj fazi razvoja *Spring* aplikacije većina konfiguracija se kopira i stavlja na podrazumevano. Ovaj problem je motivacija razvoja *SpringBoot* nadogradnje na *Spring*. Naime, *Spring Boot* predstavlja ubrzan početak razvoja *Spring* aplikacije. Za razliku od *Spring-a*, koji zahteva mnogo konfigurisanja, koje u startu razvoja aplikacije ne igra ulogu, *Spring Boot* se minimalno konfiguriše u početku. On postavi sve neophodne vrednosti konfigurisanja na podrazumevanu i na taj način obezbeđuje programeru da vrlo brzo dođe do prve radne izvršne verzija aplikacije.

Pored uprošćavanja konfiguracija, postoji još jedna bitna razlika između *Springa* i *SpringBoota*. *Spring* aplikacija, kao izvršna, postavlja na web server sa kontejnerom (Apache Tomcat [12] ili neki drugi), odnosno razvijena aplikacija zahteva podešavanje web servera koji će izvršavati aplikaciju. *Spring Boot*, ponovo principom podrazumevanog podešavanja, nosi sa sobom web server kao ugrađenu komponentu i time ne zahteva postavljanje aplikacije na web server. Dovoljno je samo pokrenuti aplikaciju, kao bilo koji jednostavan program, i *Spring Boot* će podići svoj web server na kom će se izvršavati kod aplikacije.

Spring je orijentisan ka komponentama koje svaka ima svoj zadatak i ulogu. Uvezivanje komponenti radi po već pominjanom principu *Dependency Injection* (Opisan u sklopu *Angulara*). Svaka komponenta se inicijalizuje od strane servera koji pamti instancu i dobavlja je drugim komponentama. U *CryptOffer* sistemu se *dependency injection* koristio anotiranjem polja komponente (pogledati Kod 2.2.1) Uvezivanje komponenti je moguće kroz konstruktor takođe (Kao i u *Angularu*).

```
@RestController
public class AdsAndWalletController {
    @Autowired
    AdAPIManager adAPIManager;
```

Kod 2.2.1 Primer Dependency Injecitona u Springu

Maven je alat za izgradnju (build) aplikacije. Zasnovan je na XML[13] fajlu koji opisuje sve ključne karakteristike aplikacije, potrebne za izgradnju. Fajl ključan za ovaj alat je *pom.xml* i smešten je u baznom direktorijumu projekta. Prilikom pokretanja izgradnje aplikacije, *Maven* analizira *pom.xml* fajl, prikupi podešavanja te izgradi traženi izvršivi fajl.

Najveći benefit upotrebe *Maven*-a se ogleda u načinu uvezivanja pomoćnih, već postojećih biblioteka. Naime, među zavisnostima, odnosno unutar `<dependencies>` taga *pom.xml* fajla, se navode potrebne biblioteke aplikacije. Pri izgradnji, *Maven* sam skine fajlove biblioteka koje su navedene kao zahtevane, a nisu prethodno već preuzete. Fajlovi se skidaju sa dobro poznatog servera registrovanih *Maven* biblioteka. Na ovaj način, navođenjem samo ključnih reči definisanja potrebne biblioteke *Maven* sređuje sve ostalo. Na primeru Kod 2.2.2 je pokazano šta je dovoljno da se navede u okviru kako bi se koristila biblioteka Feign.

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Kod 2.2.2 – Isečak iz pom.xml-a. Primer navođenja korišćenja biblioteke Feign iz Spring framework.cloud projekta. Ove 4 linije su dovoljne da bi Maven umeo da dovuče potrebne fajlove i tako igradi aplikaciju koja koristi Feign kao pomoćnu biblioteku.

Baza podataka

Svaka aplikacija u svojoj memoriji čuva određenu količinu podataka. Međutim, dešava se da količina podataka pretekne resurse radne memorije. U slučaju čuvanja podataka samo u radnoj memoriji aplikacije takođe predstavlja problem svaki prekid rada aplikacije (radna memorija se briše). Zbog ovih problema gotovo sve aplikacije poseduju bazu podataka kao mesto gde se skladište podaci na memoriji računara. *CryptOffer* koristi relacione baze podataka[14]. Za radnu verziju je odabrana *MySQL* baza. Alat za rad sa bazom je *JPA Persistence* koji omogućuje potpunu nezavisnost od tipa baze.

MySQL baza podataka

MySQL je aplikativni servis za upravljanje relacionim bazama podataka. Najpopularniji je izbor baze podataka za web aplikacije jer je potpuno besplatan. Podiže se kao servis aplikacija na serveru i omogućuje pristup sa drugih adresa što daje slobodu distribucije.

U okviru jednog MySQL servera podaci su grupisani prvenstveno u šeme (scheme). Jedna šema čini izolovan skup tabela. Karakteristika relacionih baza podataka je međusobno povezivanje tabela preko stranih ključeva (referenci). Tabele se uvezuju preko identifikacionih ključeva. Tabele imaju definisane kolone (polja) po imenu. Pored imena, kolone sadrže tip podatka kao i druga ograničenja i pravila kao što su: dozvoljavanje nepostojanja vrednosti, podrazumevana vrednost i sl.

Uobičajen rad sa podacima relacionih baza podataka se vrši koristeći struktuiranog jezika za upite (eng. Structured Query Language – SQL[15]). Ovim jezikom se pišu upiti za čitanje baze podataka ili modifikaciju podataka u bazi. Postoji mogućnost upotrebe baze kroz kod aplikacije direktnim pisanjem SQL upita, ali ovakav način pristupa ima mnogo nedostataka. Zbog toga se koriste ugrađene biblioteke za rad sa bazama podataka. *CryptOffer* koristi *JPA Persistence* sa implementacijom *Hibernate-a*[16].

JPA – Java Persistence API je biblioteka za rad sa bazama podataka. Predstavlja jednu od implemetacija mapiranja objekata na baze (eng.Object-relational mapping - ORM). JPA ima funkciju da na osnovu koda pravi tabele u bazi. Pored kreiranja, JPA pruža i potpunu mogućnost manipulisanja podacima baze samo kroz kod aplikacije, bez potrebe pisanja SQL upita. SQL upiti se automacki generisu na osnovu koda i programer najčešće nema potrebe da brine o tome. Slučaj kada se u kodu ipak definišu upiti je kod kompleksnijeg čitanja podataka.

JPA je orijentisan programiranju anotacijama. Običnu Java klasu anotacijom *@Entity* JPA pretvaramo u tabelu u bazi. Takođe, postoji širok opseg anotacija za konfigurisanje ograničenja tabela i kolona tabela. Primeri: za naznačavanja identifikacionog polja u tabeli staljva se anotacija *@Id*; jedinstvenost vrednosti polja na nivou tabele se postize anotacijom *@Unique*; i sl. Primer entitea baze podataka je dat u poglavlju 4 – Implementacija softvera na kodu 4.5.2.

Druga vrlo korisna stvar koje JPA nosi sa sobom su JPA Repository-i. Predstavljaju alat za upravljanje podacima baze. Bez korišćenja JPA bilo bi potrebno ručno implementirati klasu u kojoj bi se navodili svi potrebni upiti. Korišćenjem JPA upotrebljava se generička univerzalna implementacija za manipulisanje entitetima. Dovoljno je navesti interfejs koji nasleđuje *JpaRepository* i osnovne funkcionalnosti upravljanja podacima baze se mogu koristiti. Dodavanje funkcionalnosti van osnovnih ugrađenih je takođe prilično prost zadatak. Primer *JPARepostory*-a kako i širi opis upotrebe je predstavljen na Kod 4.5.3.

Metode koje počinju sa *findBy* automacki implementiraju pretragu po polju navedenom u nastavku naziva metode. Takođe je dozvoljeno koristiti logičke veznike u nastavku imena metode. Tako bi metoda po *findBySymbol* sa primera koda 4.5.3 predstavlja pretragu po polju *Symbol*.

Komunikacija

Kroz ceo ovaj rad se pominje komunikacije, slanje zahteva, dobavljanje podataka i sl. Svaki od tih sinonima označava isto, a to je tok razmene podatak između aplikacija. Način razmene podataka definiše se protokolom. U projektu *CryptOffer* sve komunikacije su realizovane preko REST protokola.

REST Protocol

Representational State Transfer (REST) predstavlja koncept u kome jedinstveni URL-ovi domena predstavljaju objekte kojima upravljamo kroz HTTP[17] metode. Pojednostavljeno rečeno, REST predstavlja protokol komunikacije koji se ograničava na HTTP protokolu i svi podaci se identifikuju adresama, odnosno URL-ovima. Kao najveću prednost REST-a čine potpuna platformska i jezička nezavisnost. Potpuno je nebitna tehnologija i platforma u kojoj je neki REST API napisan, čak i ne postoji mogućnost prepoznavanja u okviru dobijenog odgovora.

REST protokolom se najčešće razmenjuju podaci u JSON formatu što nije ograničenje. Potpuno je otvorena mogućnost razmene bilo kakvih svih tipa HTTP konteksta (XML, HTML, plain text, bytes...). Zahtevi u REST-u nose oznaku metode. REST metode su ograničene i predefinisane. Postoje preporuke za upotrebu metoda, međutim to je samo preporuka i konvencija, nikakvo funkcionalno ograničenje ne postoji u implementaciji – potuna sloboda definisanja. Metode i preporuke upotrebe su sledeće:

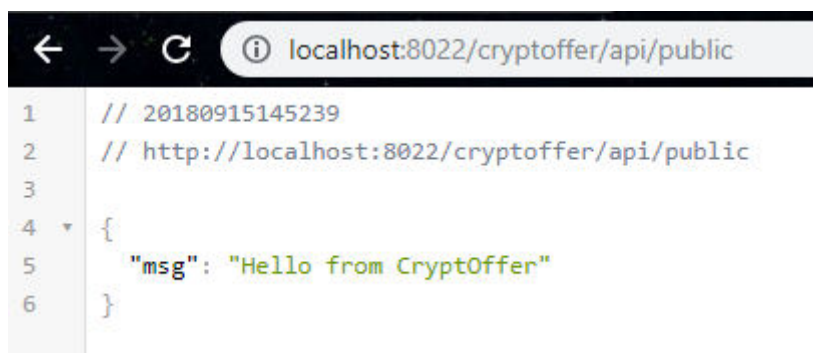
- GET – dobavljanje podataka
- POST – dodavanje podataka – upis
- PUT – izmena postojećih podataka
- DELETE – brisanje podataka

U okviru Springa definisanje REST-a je vrlo jednostavna. REST servisi se definišu u okviru REST kontrolera. U okviru klase koja se anotira sa *@RestController* metode anotirane sa *@RequestMapping* (*@GetMapping* –

za GET, `@PostMapping` za POST metode itd.). Jedan vrlo prost ali reprezentativan primer se vidi je pokazan na Kod 2.2.5 i rezultat na Slici 2.2.2

```
@RestController
public class HBController {
    @GetMapping("public")
    public String hello(){
        return "{ \"msg\": \"Hello from CryptoOffer\"}";
    }
}
```

Kod 2-2.3 Implementacija REST kontrolera u Springu. Pokazana je implementacija GET metode čiji je link '{linkAplikacije}/public' kako je navedeno u anotaciji.



Slika 2.2.2 Rezultat čitanja REST API-a implementiranog u Kod 2.2.3.

Navedeno je kako se pružaju podaci preko RESTa. U primeru (Slika 2.2.2) se vidi čitanje podataka ali kroz browser. To nije upotrebino u kodu, ne na ovaj način. U komunikaciji *CryptoOffer*-a postoje 2 mesta čitanja REST API-a koji je implementiran:

- Klijentska aplikacija čita podatke od servera – o ovome vodi računa Angular koji automacki mapira dobijen JSON[18] format na navedenu klasu.
- Mikroservisi čitaju podatke međusobnih poziva – za ovo je korišćena Feign biblioteka

Feign klient

Feign predstavlja biblioteku iz Spring Cloud [191649409687.□] projekta koja se koristi za čitanje REST API-a. Predstavlja vrlo uprošćen način čitanja podataka. Kod za čitanje REST resursa (REST API servisa) koristeći Feign izgleda prilično jednostavno i intuitivno. Sve što je potrebno je da se napravi interface čiji parametri i povratne informacije moraju da se poklapaju sa ulaznim i izlaznim parametrima REST API servisa. Pored toga, neophodno je navesti odgovarajuće anotacije kako bi Feign znao koju REST metodu i koji URL da gađa (@GetMapping("all" bi gađa metodu GET:

```
@FeignClient("CryptoWallets")
@RequestMapping("api/coins")
public interface WalletAPIManager {

    @GetMapping
    public List<Coin> coins(
        @RequestParam("username") String username);

    @GetMapping("{symbol}")
    public CoinDetailed coin(
        @PathVariable("symbol") String symbol,
        @RequestParam("username") String username);

    @GetMapping("all")
    public List<Coin> allcoins();

    @PutMapping
    public boolean put(Buying... b);
}
```

Kod 2-2.4 Primer Feign klienta za čitanje servisa CryptoWallets. Na ovaj način je realizovano svako čitanje REST API-a u okviru projekta CryptOffera.

CryptoWallets/api/coins).

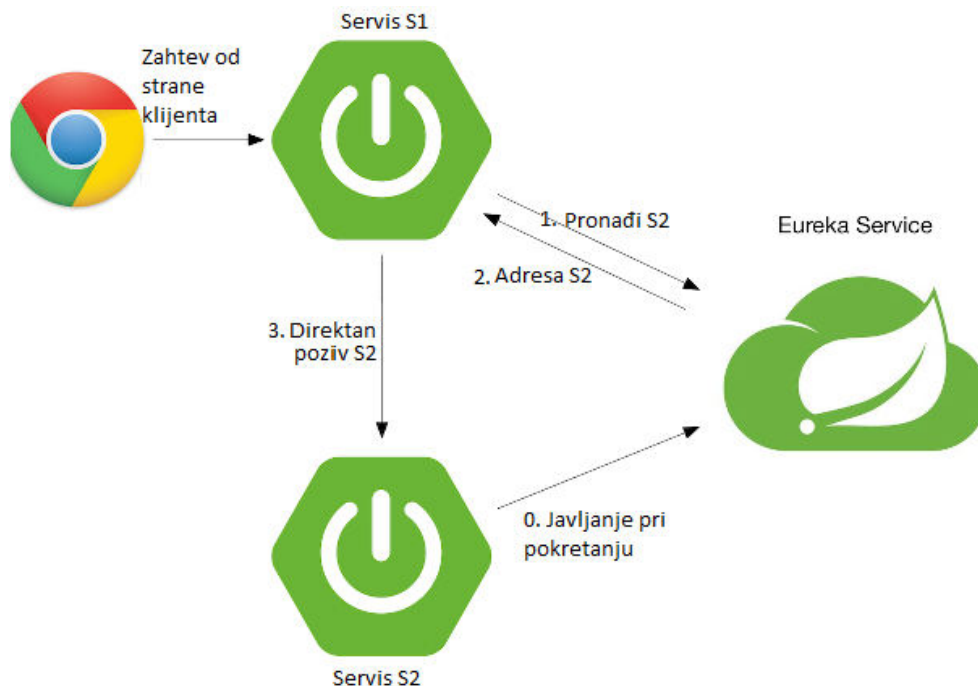
Na primeru Kod 2.2.4 je dat kod za čitanje mikroservisa *CryptoWallets*. Anotacija `@FeignClient` (“*CryptoWallets*”) označava da ovaj interfejs čita podatke sa mikroservisa koji je registrovan po imenu *CryptoWallets* (Više o registrovanim mikroservisima u sekciji o Eureka Serveru). Nakon toga je naveden, u okviru anotacije `@RequestMapping`, osnova adrese koju ovaj interfejs čita. Svaka metoda će gađati URL sa adresom registrovanog mikroservisa konkatenirano sa osnovom adrese i adresom konkretne metode. Metode su anotirane sa oznakom HTTP metode (GET, PUT, POST...) kao i opcionom adresom konkretne metode (ukoliko se ne navede gađa se metoda naznačena na interfejsu).

Eureka Server

Eureka Server je biblioteku iz Srping Cloud Netflix [23] projekta. To je, ustvari, biblioteka koja se stavlja u jedan samostalan servis mikroservis. Zadatak ovog Eureka Servera kao mikroservisa (u daljem tekstu: Eureka Server) je jedinstveno mesto registracija svih drugih mikroservisa.

Eureka Server za cilj ima centralizaciju komunikacije mikroservisa. Kako bi se mikroservisi lakše pronalazili, oni predstavljaju Eureka Klijente, koji se javljaju Eureka Serveru preko svog identifikacionog imena. Kada se mikrosevis javi pod imenom, kažemo da je registrovan na Eureka Server pod tim imenom. Registrovane adrese po imena Eureka Server pruža po zahtevu. Paralelna uloga iz društvenog života Eureka Severu bi bio živi imenik: javljaju mu se svi po imenu i ostavljaju svoju adresu, on ih pamti, a zatim zahtevu pitanju “gde se nalazi osoba XZ“ Eureka Server bi odgovorio sa adresom.

U klasičnom web server aplikacijama (monolitnim) ovo je suvišno. Međutim, kod mikroservis orijentisanih sistema Eureka Server mnogo olakšava distribuciju. Administratori koji vode računa o mikroservisima nemaju potrebe da paze na adrese servisa već je dovoljno da se pobrinu da se svaki mikroservis registruje Eureka Serveru. Ukoliko je servis registrovan, Eureka garantuje da će se svaki zahtev biti korektno usmeren odgovarajućem mikroservisu.



Slika 2.2.3 Tok poziva između mikroservisa uz upotrebu Eureka Servra

Tok poziva mikroservisa uz korišćenje Eureka servera je sledeći (Servis S1 poziva servis S2):

0. Svaki mikroservis se po podizanju javlja Eureka Serveru po imenu
1. Servis S1 pita Eureka Server za adresu servisa S2.
2. Eureka Server odgovara sa adresom na kojoj je dostupan servis S2
3. Servis S1 šalje poziv na konkretnu adresu koju je Eureka Server odgovorio.

Registracija mikroservisa ne zahteva implementaciju u kod već samo navođenje da će servis biti Eureka Klijent kao i navođenje imena servisa. Poziv servisa preko Eureka servera je pokriven na primeru Feign Client-a u Kodu 2.2.4 gde ime *CryptoWallets* (`@FeignClient("CryptoWallets")`) označava ime po kojem će se zahtevati adresa od Eureka servera (očekuje se da postoji servis koji se registrovao sa zadatim imenom). Dakle svaki poziv intefjsa iz primera će uključiti navedena 3 koraka.

Postavlja se pitanje, šta ako se više klijenata javi sa istim imenom? Ovo nije problem, ovo je prednost i jedna od motivacija uvođenja Eureka Servera u *CryptOffer* projekat.

Ribbon – Load Balancer

Ribbon je biblioteka takođe iz Spring Cloud Netflix[20] projekta. Potpuno ime ove biblioteke je *Ribbon the Client side Load Balancer* odnosno Ribbon kao balanser zahteva klijentske strane. Za zadatak ima obezbeđivanje skalabilnosti bez potrebe za bilo kakvom promenom u postojećom sistemu.

Ribbon radi uz podršku prethodno navedenih tehnologija: Feign i Eureka Server. Predstavlja alat za podelu posla između istih servisa podignutim na različitim hardverskim resursima u cilju poboljšanja performansi(nije neophodna hardverska distribucija za funkcionisanje). Naime, navedeno je da Eureka Server pamti registrovane mikroservise. Ribbon je alat koji obezbeđuje raspodelu posla između istoimeno registrovanih servisa.

Ribbon dobija na smislu u slučajevima kada su mikroservisi preopterećeni, kada upotreba jednog sistema postane masovna i postavljeni hardverski resursi ne stizu da obrade sve zahteve, Ribbon je taj koji rešava problem. Ukoliko u opterećenom mikroservis-orijentisanom sistemu postoji podrška za Ribbon, dovoljno je da se kod kritičnog (najopterećenijeg) mikroservisa pokrene još jednom na drugom hardveru. Novi mikroservis se javlja Eureka Serveru koji, sada, za isto ime poseduje registrovane 2 instance istofunkcionalnog mikroservisa. Nakon toga, svaki zahtev Eureka Serveru za dobijanje adrese tog mikroservisa obrađuje i Ribbon. Obrada Ribbon podrazumeva analizu trenutne opterećenosti (zauzetosti) instanci. Nakon obrade Ribbon-a Eureka Server odgovora sa fizičkom adresom mikroservisa koja je manje opterećena. Ovim dobijamo konstanto istu opterećenost oba hardvera i na taj način zahtevi bivaju mnogo brže obrađivani.

Za razliku od Eureka Servera, koji za svoju podršku zahteva samostalan mikroservis, Ribbon je integrisan sa Eureka Serverom i Eureka klijentima. To znači da Ribbon zahteva samo odgovarajuće konfigurisanje, ne i dodatni mikroservis za obavljanje svojih poslova.

Sigurnost podataka

Postavlja se pitanje, kako učiniti podatke aplikacije sigurnim. Ukoliko su podaci javni, mogu biti zloupotrebljeni protiv korisnika aplikacije. Takođe, bez imlementirane sigurnosti sistem ne može da garantuje ni tačnost podataka koje čuva, niko ne garantuje validnost unetih podataka. Iz navedenih razloga se uvodi sigurnost u komunikacioni tok podataka.

S obzirom da je *CryptOffer* mikroservis orijentisan, i bezbednost je implementirana u istom usmerenju. Za ovakve potrebe, kao savršeno odgovara *OAuth2* mehanizam sigurnosti.

OAuth2

OAuth2 predstavlja specifična pravila protoka podataka mrežom kako bi se obezbedila sigurnost podaka. Koristi se u web orijentisanim aplikacijama za sigurnost API-a podataka.

OAuth2 podržava različite tipove autorizacije. U terminologiji ovog protokola, autorizacioni tipovi se nazivaju Grant tipovi (grant type). Grant tipovi *OAuth2* protokola su :

- Autorizacioni kod (eng Authorization Code)
- Lozinka (eng Password)
- Klijentski kredencijali (eng Client Credentials)
- Implicitni (eng Implicit)

CryptOffer u svojoj implementaciji i trenutnoj verziji sadrži implementiran grant tip lozinke i klijentskih kredencijala. On se ugleda na principu logovanja korisnika na klijentskoj aplikaciji i serverska proverava. Osim kredencijala korisnika, pri logovanju se šalju i kredencijale klijentske aplikacije.

U *OAuth2* postoje 4 uloge:

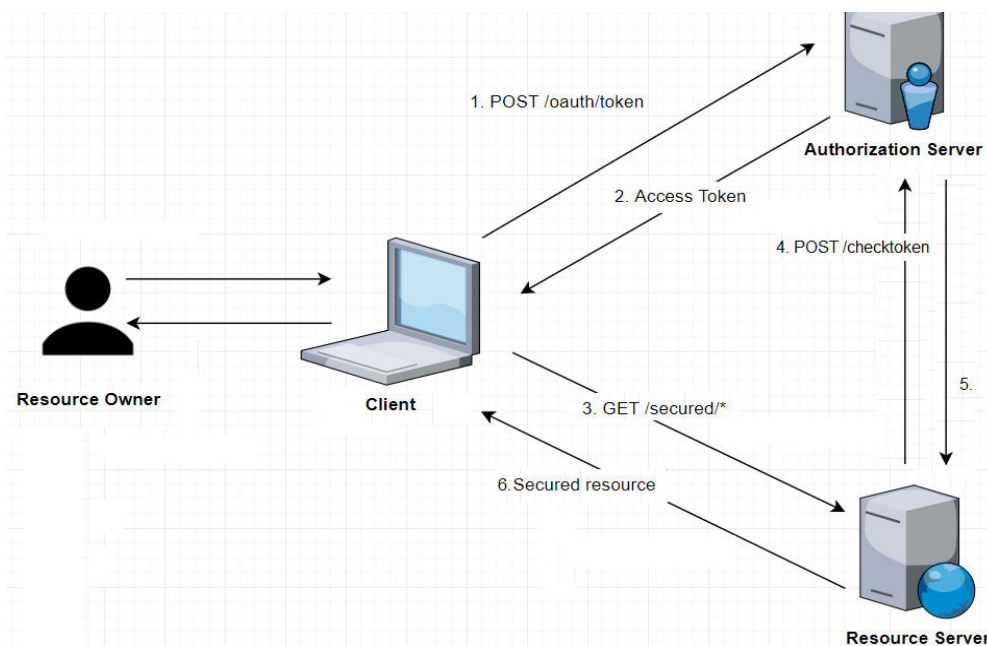
1. Korisnik (Resource Owener) – predstavlja osobu, dakle fizičko lice, koja će koristiti aplikaciju

-
2. **API Server (Resource Server)** – predstavlja serversku aplikaciju koja sadrži i čuva podatke koje sistem želi bezbedno da razmenjuje
 3. **Klijentska aplikacija (Client)** – klijentska web aplikacija koju korisnik koristi u svom web-browser i koja komunicira sa serverom
 4. **Autorizacioni Server (eng Authorization Server)** – serverska aplikacija koja ima za zadatak brigu o autorizaciji.

Ovaj alat je token orijentisan. Naime, autorizacioni server poseduje informacije o korisnika, izdaje tokene i proverava tačnost tokena. Ovo su glavni zadatci autorizacionog servera. Prilikom zahteva korisnika za podacima, proces bezbednog pružanja podataka sadrži 6 koraka pre prikaza podataka korisniku (Slika 2.2.4.):

1. **Klijent** šalje **autorizacionom server** zahtev za proveru kredencijala. Pored kredencijala korisnika, ovaj zahtev sadrži i kredencijale za verifikaciju aplikacije koja je definisana u kodu.
2. **Autorizacioni server** odgovara **klijentu**. U slučaju nevalidnih kredencijala, server vraća odgovor greške i proces se tu neuspešno završava. U suprotnom, ako su kredencijali u redu, autorizacioni server odgovara sa pristupnim tokenom (Access Token). Taj token klijent čuva u sesiji i dodaje se svakom zahtevu ka API Serveru.
3. **Klijent** šalje zahtev **API Serveru** za potrebnim podacima kako bi ob služio korisnika. U okviru zahteva prilaže i pristupni token kako bi garantovao da je klijent ima prava da pristupi podacima. Ukoliko pokuša da se pošalje zahtev bez tokena, ovde se neuspešno završava proces.
4. Kada API server dobije zahtev sa uključenim tokenom, mora prvo da proveriti da li je token zaista validan. Tada se **API server** obraća **autorizacionom serveru** za proveru validnosti tokena.
5. **Autorizacioni server** odgovara **API serveru** sa podatkom da li je token validan kao i osnovnim podacima korisnika čiji je ovo token. U slučaju da je klijentska aplikacija poslala zahtev sa nevalidnim tokenom, ovde se otkriva to i vraća klijentu odgovarajući odgovor.

6. Ukoliko do ovog koraka nije pronađena greška, to je dovoljan znak da se podaci mogu pružiti klijentu tako da ostanu potpuno bezbedni. Dakle, u ovom koraku **API Server** odgovara **klijentu** sa traženim podacima i klijent može potpuno sigurno da prezentuje korisniku



zahtev.

Slika 2.2.4 – Tok komunikacije pri zahtevanju korisnika podataka. Prilikom autorizacije se vrše svi navedeni koraci. Nakon uspešne autorizacije token se čuva u sesiji te se dodaje svakom zahtevu klijenta ka API serveru, izbegavaju se koraci 1. i 2.

Naveden proces predstavlja prvi zahtev korisnika. Kako je navedeno, nakon 2. Koraka klijent sačuva u sesiji pristupni token. Svaki sledeći zahtev korisnika klijent obrađuje na sledeći način: ako u sesiji postoji pristupni token, obratiću se direktno API serveru i priložiću token iz sesije; ako nemam token u sesiji, tražiću od korisnika da mi da kredencijale i vršim ceo postupak autorizacije. Dakle, nakon uspešne autorizacije (prvi zahtev), svaki sledeći zahtev se sastoji od koraka 3.-6.s

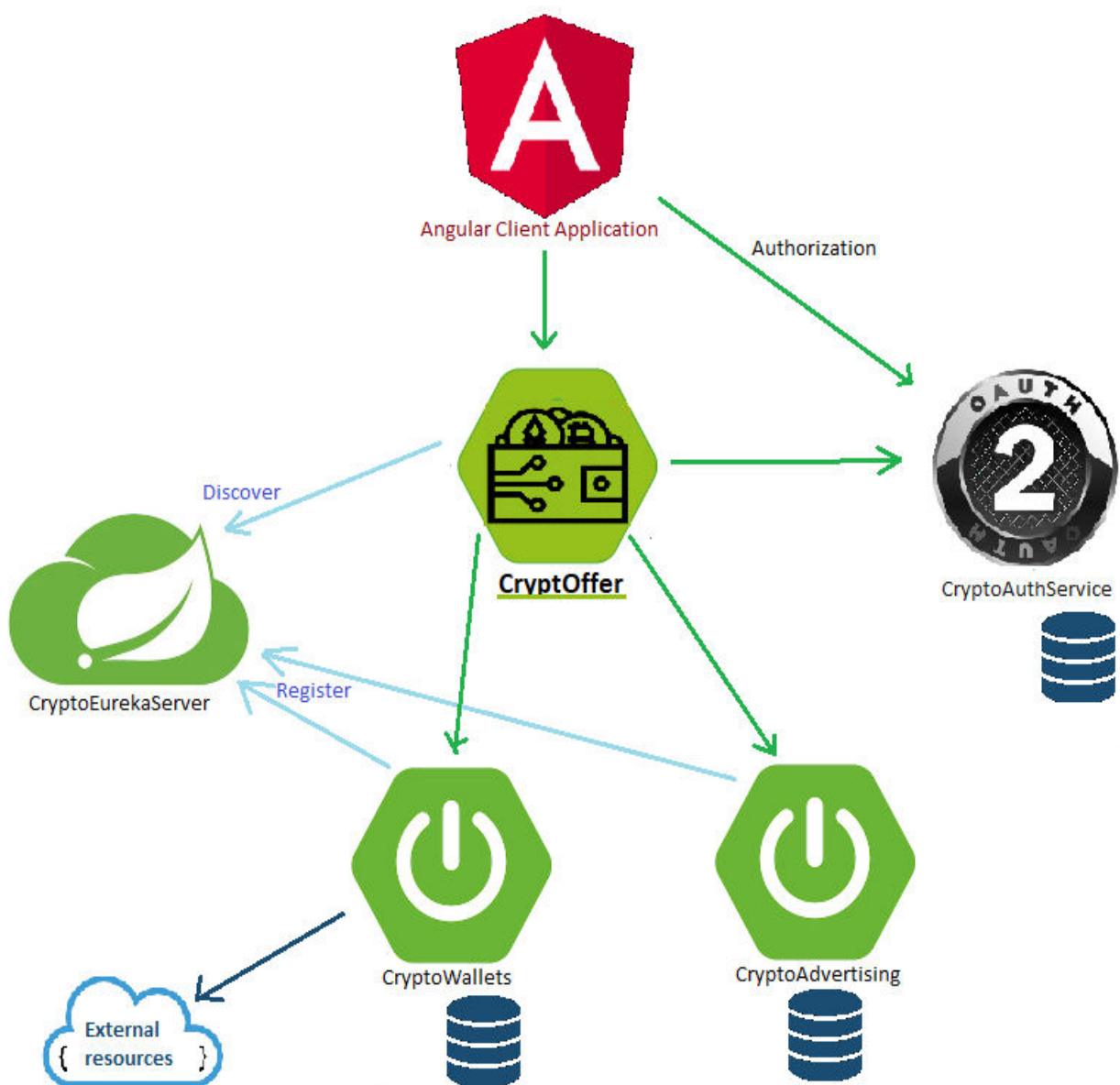
3. ARHITEKTURA I SARŽAJ SOFTVERA

U prethodnim poglavljima ovog rada je navedena ideja i cilj celokupnog sistema. Navedene i razjašnjene su tehnologije koje su korištene za implementaciju. Celokupna priča o korišćenim tehnologijama je bila uopštena. U ovom poglavlju se prikazuje način upotrebe navedenih tehnologija na konkretnom softveru aplikacije *CryptOffer*.

Uz primenu tehnologija na konkretnu aplikaciju ovde će biti opisana celokupna arhitektura aplikacije kao i detaljno objašnjene uloge i funkcije svakog od podsistema.

Skica celokupne arhitekture predstavljena je na Slici 3.1. Na slici je navedeno 6 celina:

1. Angular klijentska aplikacija – klijentska aplikacija
2. CryptOffer – centralna jedinica serverskog dela
3. CryptoAuthService – mikroservis zadužen za autorizaciju
4. CryptoEurekaServer – mikroservis zadužen za registraciju svih mikroservisa aplikacije
5. CryptoAdvertising – servis zadužen za logiku oglasa kupoprodajnih oglasa kriptovaluta
6. CryptoWallets – servis zadužen za evidenciju stanja kriptovaluta korisnika



Slika 2.2. Skica celokupne arhitekture aplikacije. Celokupna aplikaija se sastoji od klijentske aplikacije i serverskih 5 mikroservisa.

3.1. Angular klijenska aplikacija

Klijentski deo aplikacije zadužen je za prezentaciju. Kao što je navedeno u uopštenom opisu klijentske aplikacije tako i ovaj podprojekat ima zadatak da direktno komunicira sa korisnikom. Za zadatak ima prikaz podataka od servera i unos novih podataka. Cela implementacija je rađena u **Angular framework**-u opisanom pod tačkom 2.2 ovog rada.

Komunikacija ovog projekta je dvostrana, kao što je prikazano na slici 3.1. Angular klijent komunicira putem REST API-a sa *CryptOffer* mikroservisom koji je zadužen za pružanje svih potrebnih podataka klijentu. Drugi kanal komunikacije je sa *CryptoAuthService*-om za potrebe autorizacije. Komunikacija sa ovim mikroservisom je manje frekventan. U priči o bezbednosti *OAuth2* tehnologijom ova aplikacija predstavlja ulogu **klijenta**. To znači, da proziva autorizacioni server, konkretno *CryptoAuthService*, za proveru kredencijala i pribavljanje **pristupnog tokena**. Pristupni token se čuva u memoriji web aplikacije i dodaje se uz svaki zahtev kao **API serveru**, konkretno *CryptOffer*-u.

3.2. CryptOffer

Centralni server aplikacije predstavlja servis *CryptOffer*. Kao centralna jedinica predstavlja nosilac celokupne strukture aplikacije pa zbog toga ima isto ime ovaj servis kao i naziv celog projekta. Osim toga, *CryptOffer* mikroservis nosi ulogu *glavne saobraćajnice*. Upućen je u sve komunikacione kanale aplikacije.

Posao *CryptOffer*-a započinje po zahtevu klijentske aplikacije, odnosno Angular projekta ove aplikacije. Prva uloga, po dobijanju zahteva, je provera **bezbednosti**. Ukoliko se pristupa javnim podacima, obrada zahteva se nastavlja bez preusmeravanja ka **autorizacionom serveru**. Ukoliko se pristupa privatnim podacima (secure) proverava se da li postoji **pristupni token** u okviru zahteva. Ukoliko token postoji, šalje se zahtev *CryptoAuthService*-u kao **autorizacionom serveru** za proveru validnosti tokena i dobavljanje podataka o vlasniku tokena (korisniku koji je inicirao

zahtev). *CryptOffer* ima ulogu **API Servera** iz OAuth2 bezbednosnog sistema.

Nakon provere bezbednosti, ovaj servis preuzima obradu zahteva. Obrada zahteva je definisana u kontrolerima. Svi gotovo celokupan interfejs servisa koje ovaj mikroservis pruža u implementaciji čuvaju preusmeravanje na odgovarajući ili odgovarajuće mikroservise nižeg sloja. Ovde se ogleda uloga *CryptOffer*-a kao glavne saobraćajnice.

S obzirom da je korišćena tehnologija Eureka Servera sa podrškom Ribbon balansera zahteva, prvi korak usmeravanja zahteva ka mikroservisima nižeg sloja predstavlja **javaljanje Eureka Serveru**, konkretno *CryptoEurekaServer*-u. Ovaj zahtev se izvršava implicitno, što znači da u kodu nigde ne može da se pokaže implementacija ovog poziva. Jedino što se zadaje su konfiguracija i ime mikroservisa nižeg nivoa po kome će se tražiti registrovana adresa.

Po odgovoru Eureka Servera, *CryptOffer* šalje zahtev najmanje opterećenoj instanci traženog mikroservisa. Ovo nam garantuje *Ribbon* podrška. U slučaju da postoji samo jedna registrovana instanca traženog mikroservisa, slučaj je trivijalan i *Ribbon* ne igra ulogu. Po prijemu odgovora od prozvanog mikroservisa, *CryptOffer* šalje klijentu odgovor. Po potrebi u zavisnosti od zahteva, dodatno se obrađuje zahtev pa tek onda šalje odgovor.

3.3. CryptoAuthService

CryptoAuthService predstavlja jedini mikroservis, pored centralne jedinice (*CryptOffer*), koji poseduje konekciju sa web klijentom. Svrha komunikacionog kanala sa web klijentskim delom aplikacije se ugleda u ulozi ovog mikroservisa. *CryptoAuthService* predstavlja **autorizacioni server** iz uloga tehnologije OAuth2 bezbednosnog sistema.

Konkretno, uloga ovog mikroservisa se ugleda u 2 vrlo bitne metode: provera kredencijala i provera tokena. **Provera kredencijala** je zahtev upućen od strane web klijent aplikacije. Prva provera koja se implicitno vrši je provera **klijentskih** kredencijala (ne korisničkih već kredencijala klijenta

kao aplikacije). Ukoliko je zahtev poslat od očekivanog klijenta, ovaj korak će proći i prelazi se na proveru kredencijala **korisnika**. Ako su kredencijali u redu, generiše se pristupni token koji se šalje kao odgovor na zahtev. Takođe, isti token biva sačuvan u memoriji zajedno sa podacima o korisniku. Dužina čuvanja tokena se ograničava kroz konfiguraciju.

Druga bitana uloga autorizacionog servera se ugleda u **proveri validnosti tokena**. Kada se pogleda tok podataka kod sistema obezbeđenih ovom tehnologijom, prikazano na slici 2.2.4. i objašnjeno u poglavlju 2.2. podnaslov Sigurnost, dobija se jasna slika da se ovaj metod poziva vrlo često od strane **API servera**. Svaki zahtev koji pristupa privatnim podacima **API servera** mora da sadrži pristupni token i svaki put validnost tokena treba da se proverí. Uz proveru validnosti tokena **autorizacioni server** pruža podatak o korisniku vlasniku tokena.

Obe navedene metode su implicitne, dakle *OAuth2* tehnologija ih nosi u sebi, a *CryptoAuthService* ih samo konfiguriše, ne implementira.

Kako bi validno čuvao podatke o korisnicima, *CryptoAuthService* koristi bazu podataka. U bazi čuva osnovne podatke o korisniku: username koji mora biti jedinstven i email, kao i kriptovanu vrednost zadate lozinke. Za **kriptovanje** lozinke krišćen je *BcryptEncoder* [21].

Registracija novog korisnika se nalazi u implementaciji ovog mikroservisa. U pitanju je javna metoda koja po zahtevu sa prikupljenim podacima o korisniku dodaje u bazu podataka novog korisnika.

3.4. CryptoEurekaServer

CryptoEurekaServer predstavlja server registracije mikroservisa. Kao što je navedeno u opisu tehnologije Eureka Server, ovaj mikroservis ima za zadatak da pamti registovane instance živih mikroservisa.

Kada se vrši poziv između mikroservisa, prvo se od ovog mikroservisa zatraži adresa, a tak nakon toga se zahtev usmerava na konkretnu instancu mikroservisa. Tok podataka poziva između mikroservisa detaljno je opisan

u opisu same tehnologije Eureka Servera u poglavlju 2.2. podnaslov komunikacija. Slikov prikaz toga podataka je dat na slici 2.2.3.

Takođe, u okviru ovog mikroservisa se vrši logika prepoznavanja opterećenosti mikroservisa i preraspodela zahteva. Za to je zadužen *Ribbon*.

3.5. CryptoAdvertising

CryptoAdvertising predstavlja mikroservis koji ima funkcionalnosti **berze** kriptovaluta, odnosno kupoprodajnog oglasnika. Ovaj mikroservis je implementiran u slojevitom arhitektonskom stilu. To znači da je logika podeljena u 3 sloja:

- Prezantacioni sloj (Presentation layer)– predstavlja sloj prezentacije, odnosno logiku zaduženu za pružanje podataka.
- Sloj poslovne logike (Buissnes layer)– predstavlja logiku i poslovnu pamet mikroservisa. To znači da se na ovom sloju razvija specifičnost i zadatak.
- Sloj podataka (Database layer) – sloj zadužen za skladištenje podataka u bazu i rad sa podacima.

U prezentacionom sloju su pisani RESTfull API servisi koristeći ugrađene biblioteke SpringBoot-a.

Sloj podataka koristi prethodno opisanu tehnologiju *JPA Persistence*-a i *MySQL* bazu podataka. U bazi se čuva jedna tabela koja predstavlja sve oglase, kako trenutno aktivne tako i istorijske (ostvarene).

Karakteristika *CryptoAdvertising* mikroservisa je da on predstavlja **Eureka klijent**. To znači da se pri pokretanju javlja Eureka Serveru, konkretno servisu po nazivu *CryptoEurekaServer*, pod karakterističnim imenom. Dodatno, ovaj mikroservis mora da se javi i sa naznakom odobravanja *Ribbon*-a, kao balansera za zahteva.

3.6. CryptoWallets

Slično kao i *CryptoAdvertising*, *CryptoWallets* mikroservis ima karakteristiku **Eureka Klijenta**. Arhitektonski stil je takođe slojevit uz određene izmene, dakle i ovaj mikroservis sadrži prezentacioni sloj sa implementiranim RESTfull API servisima, sloj poslovne logike i sloj baze podataka. Međutim, ovde sloj poslovne logike ima kompleksniju implementaciju.

Specifičnost *CryptoWallets*-a se ogleda u korišćenju eksternih izvora podataka. Naime, on koristi **API berze** koji pruža realne podatke o trenutnom i istorijskom stanju u svetu kriptovaluta. Strani resurs (API berze) je pronađen istraživanjem i bez saradnje drugih lica. U pitanju je web servis po imenu *CryptoCompare*[22]. Detalji se dokumentacije API-a se mogu pronaći u referencama [23].

Korišćenje eksternih resursa kao nadogradnja informacija na interne podatke aplikacije zahteva određenu kompleksnost logike. Pa tako, na primer, podaci o trenutnom stanju su ažurni u realnom vremenu jer se trenutna cena valute uzima sa tržišta. Takođe, prilikom prikaza promene vrednosti u realnom vremenu za određenu valutu, podaci su stvarni jer se koriste API za istorijske cene sa istog izvora.

Pored korisnih podataka, integracijom ovog eksternog API servisa obrađen je i prikaz slikovitih simbola valuta. Kako svaka kriptovaluta ima svoj logo koji ga karakteristiše i po kome se prepoznaje, a oni se vrlo često menjaju i dodaju novi, koriste se sličice sa istog servera. Ovakav pristup prebacuje poverljivost korektnosti na korišćeni API servis i smanju potrebu održavanja (u slučaju pojavljivanja nove kriptovalute ili promena logoa postojeće, u *CryptOffer*-u nema potrebe za izmenama koda).

Osim eksternih resursa pomenuti su i interni podaci. *CryptoWallets* poseduje svoju bazu podataka u kojoj čuva sve promene na stanju novčanika po korisnicima. Za praćenje stanja je zadužena jedna tabela. Ona pamti sve promene na stanju. Za prikaze sumiranih podataka koristi se grupisanje promena stanja i na taj način se dobija celokupan rezultat.

deklarizovana na liniji 14).

Na liniji 20 desne strane iste slike je prikazan primer naznačavanja upotrebe servisa. Ova naznaka predstavlja prethodno opisan patern *Dependency Injection*-a. Tražen servis po imenu `WalletService` se koristi u metodi `ngOnInit` (linija 25).

```
const httpOptions = {headers: new HttpHeaders(  
    {'Access-Control-Allow-Origin': '*'} )};  
@Injectable({  
    providedIn: 'root'  
})  
export class WalletService {  
    coinsUrl :string = SERVER_URL+'coins';  
  
    constructor(  
        private http : HttpClient  
    ) {}  
  
    getCoins():Observable<Coin[]>{  
        return this.http.get<Coin[]>(this.coinsUrl,httpOptions)  
            .pipe(tap(t => {  
                console.log("coins fetched ");  
                console.log(t);  
            } )  
    );  
}
```

Kod 4.1.2 Kod servisa koji čita REST Api u Angularu.

U Kodu 4.1.2 je predstavljena logika čitanja REST API-a. Metoda `getCoins` se poziva u komponenti prikazanoj u Kodu 4.1.1. Ovaj servis koristi klasu `HttpClient` iz ugrađene biblioteke *Angular*-a. Metoda ne vraća direktno listu `Coin` objekata već, kako je navedeno, `Observable` listu. To znači pri pozivu ove metode, treba da se prijavi za osluškivanje na vraćeni objekat. Tek kada se zahtev izvrši (dobije odgovor), biće izvršen kod prijavljen za osluškivanje. Ovakav pristup predstavlja **reaktivno programiranje**[24] i koristi se kod svih servisa koji iščekuju podatke. Prilikom poziva REST API-a navedena je definicija metod: `GET`; tip u koji se rezultat konvertuje: kolekcija objekata tipa `Coin` (`Coin[]`); URL adresa i objekat tipa `HTTPOptions` u kome su definisani pomoćni delovi zahteva

```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {
  constructor(public auth: AuthService,
               private router : Router) {}

  intercept(request: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    var hasAuthReq:boolean = request.headers
      .get('Authorization') != null;
    if( !hasAuthReq && this.auth.isAuthenticated()){
      request = request.clone({
        setHeaders:
          {Authorization: `bearer ${this.auth.getToken()}`}
      });
    }
    return next.handle(request).do(
      (event: HttpEvent<any>) => {},
      (err: any) => {
        if (err instanceof HttpErrorResponse) {
          if (err.status === 401 || err.status === 403) {
            this.router.navigate(['/login/'+err.status]);
          }
        }
      }
    );
  }
}
```

Kod 4.1.3 Kod interseptora u Angularu. Za zadatak ima da presreće svaki izlazni zahtev i doda u opis zahteva autorizacioni kod. Pored toga, vidimo da presreće i svaku dolaznu poruku te u slučaju greške preusmera na login

poziva RESTa.

Još jedan prezentaciono zanimljiv segmet ovog dela aplikacije predstavlja autorizacija Kod 4.1.3. Kao što je rečeno, svaki zahtev se presreće i dodaje autorizacioni token. To očekuje da je korisnik prethodno ulogovan te da je token sačuvan u memoriji aplikacije. Ukoliko taj uslov važi, u opisu zahteva (HTTP Request Headers) se dodaje autorizacioni token. Pored presretanja izlaznih zahteva, ovaj kod obrađuje i odgovore na trivijlan način: u slučaju da se dobije odgovor sa status kodom 404 ili 403 prebacujemo se na login stranicu.

4.2. CryptoOffer

CryptoOffer sadrži nekoliko zanimljivih delova koji se tiču konfiguirisanja. Na Kod 4.2.1 se je prikazana konfiguracija 3 dosada pomenute tehnologije:

- `@EnableEurekaClient` - oznaka da je ovaj mikroservis jedan od Eureka Klijenata
- `@EnableFeignClients` - signalizacija da ovaj mikroservis sadrži Feign klijente koji će se locirati preko registrovanih imena (EurekaServera)
- `@EnableDiscoveryClient` - odobravanje balansera zahteva od strane Ribbon biblioteke

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
@EnableDiscoveryClient
public class CryptoOfferApplication {
    public static void main(String[] args) {
        SpringApplication.run(
            CryptoOfferApplication.class, args);
    }
}
```

Kod 4.2.1 Konfiguirisanje SpringBoot aplikacije CryptoOffer.

Pošto ovaj servis predstavlja **API Server** ulogu OAuth2 bezbednosnog sistema, i to zahteva konfigurisanje što je pokazano na kodu 4.2.2. Prva metoda sa koda predstavlja definisanje koje metode su javno dostupne a koje ne. Druga metoda sa istog koda predstavlja ključnu reč za identifikovanje servera kao **API Servera** sistema. Isti ključ mora da bude postavljen na **API Serveru** i u **autorizacionom serveru**. Treća metoda definiše način provere validnosti pristupnog tokena – u slučaju *CryptOffer*-a to je slanje zahteva

```
@EnableResourceServer
public class ResourceServerConfig extends
ResourceServerConfigurerAdapter{

    @Override
    public void configure(HttpSecurity http)throws Exception {
        http.authorizeRequests()
            .antMatchers("/public").permitAll()
            .antMatchers("/coins/all").permitAll()
            .antMatchers("/**").authenticated();
    }

    @Override
    public void configure(ResourceServerSecurityConfigurer res)
        throws Exception {
        res.resourceId("oauth2-resurce");
    }

    @Primary
    @Bean
    public RemoteTokenServices tokenService(){
        RemoteTokenServices tokenService =
            new RemoteTokenServices();
        tokenService.setCheckTokenEndpointUrl(
            "http://localhost:8044/api/oauth/check_token");
        tokenService.setClientId("trusted-auth-client");
        tokenService.setClientSecret("crypto-secret");
        return tokenService;
    }
}
```

Kod 4.2.2 Konfigurisanje servisa kao API Servera (Resource server) u OAuth2 bezbenosnom protokolu

autorizacionom serveru.

Osim konfiguracije `CryptoOffer` sadrži jednu zanimljivost koja je već prikazana u opisu tehnologija a to je Feign klijent. Podsetimo, Feign

```
@FeignClient("CryptoAdvertising")
@RibbonClient("CryptoAdvertising")
@RequestMapping("api/ads")
public interface AdAPIManager {

    @GetMapping
    public List<Ad> ads();

    @GetMapping("{symbol}")
    public List<Ad> ads(@PathVariable("symbol") String symbol);

    @GetMapping("notdone")
    public List<Ad> adsNotDone();
    ... }
```

Kod 4.2.3 Primer Feign klijenta. Ovako definisan klijent predstavlja dovoljnu definiciju čitanja REST API-a. Svaka metoda ovog interfejsa predstavlja po jednu metodu API servera koji se poziva

predstavlja način čitanja REST API-a između mikroservisaa.

U kodu 4.2.3 je predstavljen deo interfejsa za čitanje mikorservisa `CryptoAdvertising`. Vidimo da je u anotiranju samog interfejsa navedeno ime mikroservisa koji se pita. Tri navedene metode predstavljaju poziv 3 metode REST API kontrolera u okviru `CryptoAdveritsing` mikroservisa.

`CryptoOffer` kao mikroservis predstavlja cenralnu jedinicu i najčešće samo preusmerava zahteve. Međutim, jedna metoda je zanimljiva jer za dobijen jedan poziv ona upućuje 2 nova poziva. To je metoda za **realizaciju oglasa**, odnosno ostvarenja kupovine ili prodaje. U tom slučaju, `CryptoOffer` dobije zahtev realizacije i podatke o oglasu koji se realizuje(objekat tipa `Ad`). Na osnovu oglasa koji se realiuje se prave 2 objekta tipa promene stanja za servis `CryptoWallet` (objekti tipa `Buying`). U kodu 4.2.4. je prikazana logika pravljenja objekta promene stanja u novčaniku korisnika koji je pristao na oglas, na osnovu prosležene instacne oglasa (`Ad`). Novčana količina se

postavlja pozitivna ili negativna u zavistnosti od toga da li je u pitanja

```
@Service
public class BuyingManager {

    public Buying makeMeBuyingFromAd(Ad ad, String username){
        Buying b = new Buying();
        double amount = ad.getIsBuying()
                        ? -ad.getAmount()
                        : ad.getAmount();
        b.setAmount(amount);
        b.setSymbol(ad.getSymbol());
        b.setDate(new Date());
        b.setPrice(0.0);
        // will be calculated in WalletService
        b.setUsername(username);
        return b;
    }
}
```

Kod 4.2.4. Kod logike za kreiranje objekta promene stanja u CryptoWalletu (Buying) na osnovu oglasa iz CryptoAdvertising (Ad)

kupovina ili prodaja. Cena se postavlja na 0 jer će biti prepisana u okviru *CryptoWallet*-a (podsetnik: *CryptoWallet* zna realne cene koje uzima na sa eksternih izvora). Datum realizacije se postavlja trenutni.

4.3. CryptoAuthService

CryptoAuthService za zadatak ima implementaciju **autorizacionog servera**. To se vrši kroz konfigurisanje samo aplikacije. U kodu 4.3.1. je predstavljeno konfigurisanje ovog servisa za navedenu ulogu OAuth2 sistema. Prvo neophodno za ovaj postupak je dodavanje anotacija `@EnableAuthorizationServer`. Ovom anotacijom se daje do znanja *SpringBoot* da ovaj servis predstavlja autorizacioni server, što je cilj. Pored toga, neophodno je zadati konfiguracije servera, kao što su: kredencijali klijentske aplikacije, ključ (id) autorizacionog servera, dužina držanja pristupnog tokena aktivnim i dr. Navedeno konfigurisanje je prikazano takođe u okviru koda 4.3.1 unutar metode `configure(ClientDetailsServiceConfigurer)`.

```

@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends
    AuthorizationServerConfigurerAdapter {
    @Override
    public void configure(ClientDetailsServiceConfigurer clients)
    throws Exception {
        clients.inMemory().withClient("trusted-auth-client")
            .authorizedGrantTypes("client_credentials",
                                "password")
            .authorities("ROLE_CLIENT", "ROLE_TRUSTED_CLIENT")
            .scopes("read", "write", "trust")
            .resourceIds("oauth2-resource")
            .accessTokenValiditySeconds(3600)
            .secret(passwordEncoder.encode("crypto-secret"));
    }
}

```

Kod 4.3.1 Isečak iz klase za konfigurisanja CryptoAuthService-a kao autorizacionog servera.

4.4. CryptoEurekaServer

CryptoEurekaServer je jedan od najopisivanijih servisa ovog rada. Gotovo se u 80% paragrafa serverskog dela opisa bar jednom spomene Eureka Server. Međutim, ovaj servis sadrži ukupno jednu klasu.

```

@SpringBootApplication
@EnableEurekaServer
public class CryptoEurekaServerApplication {
    public static void main(String[] a) {
        SpringApplication.run(CryptoEurekaServerApplication.class, a);
    }
}

```

Kod 4.4.1. Jedina klasa servisa CryptoEurekaServer

Anotacije glavne klase `@EnableEurekaServer` upućuje da je ovaj mikroservis Eureka Server. Osim ove anotacije neophodno je i u konfiguracionom fajlu promenuti neku od podrazumevanih vrednosti (kodu 4.4.2.). Podsetnik, *SpringBoot* radi na principu porazumevanioih vrednosti: svi konfiguracioni parametri imaju podrazumevanu vrednost sve dok se ne navede drugačije. U slučaju Eureka Servera potrebno je promenuti podrazumevanu vrednost. U pitanju su parametri koji se moraju staviti na `false` a odgovaraju na pitanje: da li da se ovaj mikroservis javlja Eureka Serveru. S obzriom da je sam servis Eureka Server javljanje nema smisla.

```
eureka.client.register-with-eureka=false  
eureka.client.fetch-registry=false
```

Kod 4.4.2 Isečak iz application.properties fajla koji naznačava da servis ne treba da se javlja Eureka Serveru.

4.5. CryptoAdvertising

CryptoAdvertising predstavlja jednostavan mirkoservis sa konkretnim zadatkom. Za registrovanje EurekaServeru uz podršku Ribbon-a su potrebne anotacije `@EnableEurekaClient` i `@EnableDiscoveryClient` iznad glavne klase, kao što je to pokazano za *CryptOffer* mikroservis (Kod 4.2.1). Ime pod kojim se mikroservis javlja Eureka Serveru se zadaje u okviru konfiguracionog fajla po ključu `spring.application.name` (Kod 4.5.1). Pored imena mikrosevisa, u ovom fajlu su navedeni i dodatni parametri kao što su:

- `server.servlet.context-path` - kao putanja servisa nakon adrese
- `server.port` - kao port na kome će servis da se pokrene (0 – znači dodela porta odabirnom slučajnosti)
- `spring.datasource.*` - kao parametri pristupa bazi
- `spring.application.name` - ime aplikacije. Koristi se kao ime pod kojim se servis registruje EurekaServeru

Na Kod 4.5.1 prikazan je primer konfiguracionog fajla ovog mikrosevisa

```
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/cryptoads
spring.datasource.username=mysql
spring.datasource.password=password
server.servlet.context-path = /api
server.port = 0
spring.application.name=CryptoAdvertising
```

Kod 4.5.1 Fajl konfiguraije (application.properties) servisa CryptoAdvertising

CryptoAdvertising ima svoju bazu podataka u kojoj čuva sve oglase. Baza podataka je tipa *MySQL* a za rad sa bazom korišćena je *JPAPersistence* biblioteka. U kod 4.5.2 je pokazan primer entita. Ovako naveden entitet JPA pretvara u table u okviru baze podataka.

```
@Entity
@NamedQueries({
    @NamedQuery(name="Ad.getBySymbolNotDone",
        query="SELECT a FROM Ad a WHERE a.symbol=:symbol "
        + "AND a.adDone is null ORDER by a.date DESC"),
    @NamedQuery(name="Ad.getAll",query="SELECT a FROM Ad a "
        + "ORDER by a.date DESC"),
    @NamedQuery(name="Ad.getAllNotDone",query="SELECT a FROM Ad a "
        + "WHERE a.adDone is null ORDER by a.date DESC")
})
public class Ad {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String username;
    private String symbol;
    private double amount;
    private boolean isBuying;
    private Date date;
    private Date adDone;
    private String doneWithUser;

    public Ad() {}
}
```

Kod 4.5.2 Primer entiteta u JPA. Ovako napisanu klasu JPA posmatra kao tabelu u bazi.

Osim automackog mapiranja i generisanja entiteta na tabele, JPA nosi sa sobom ugrađene podrazumevane operacije nad entiteima. Korišćenje ovog benefita je opisano u sekciji 2.2. Serverski deo aplikacije u predelu JPA Persistence. U kodu 4.5.3 je dat primer `AdRepository` koji po nasleđivanju `JpaRepository` sadži osnovne metode(`save()`, `findById()`, `delete()`, `findAll()` i sl.). Metoda `findBySymbol` se implicitno implemetnira od strane JPA i vrši pretragu po polju `symbol`. Drugi način implementacije neuobičajenih upita jeste prikazan na metodi `getBySymbolNotDone` koja je anotirana sa `@Query`. Ova metoda očekuje mapiran upit po imenu, tzv. `NamedQuery`. `NamedQuery` se definišu na entitetu. Na primeru kod 4.5.2. postoje navedeni upiti među kojima je `getBySymbolNotDone`.

```
public interface AdRepository
    extends JpaRepository<Ad,Integer>{

    public List<Ad> findBySymbol(String symbol);

    @Query
    public List<Ad> getBySymbolNotDone(
        @Param("symbol") String symbol);
```

Kod 4.5.3 Primer JpaRepository-a. AdRepository nasleđuje JpaRepository i time postaju dostupne osnovne funkcionalnosti kao što su findById(..), findAll(), save(..) i slično.

Metoda findBySymbol samo po nazivu daje do znanja da je u pitanju filtriranje po atributu 'symbol'

Metoda getBySymbolNotDone je anotirana sa @Query što znači da je definicija upita ove metode navedena kao istoimeni NamedQuery u klasi entiteta Ad (čiji je ovo repository) (primer u okviru Kod 4.5.2)

4.6. CryptoWallets

CryptoWallets pored svoje baze sadrži čitanje eksternog izvora podataka. U pitanju je već pominjani API za realne cene valuta na tržištu. Za to je

```
@Service
public interface RESTReader {
    public SortedMap<String, Coin> getAllCoins();
    public double getCurrentPrice(String symbol);
    public Map<String, Double> getPricesFor(
        List<String> symbols);
    public double getHistoricalPrice(String symbol, Date ts);
    public SortedMap<Date, Double> getPriceForLast10Days(
        String symbol);
}
```

Kod 4.6.1 Interfejs čitača eksternog izvora podataka u okviru CryptoWallets servisa

zadužen REST Reader čiji je interfejs prikazan u kodu 4.6.1

Pošto se kombinuju 2 izvora podataka, u okviru ovog mikroservisa postoji servis (kao SpringBoot komponenta) koji se bavi dobavljanjem svih podataka i nazvan je *CoinManager*. Interfejs *CoinManager* servisa u potpunosti sakriva izvore podataka te ukoliko se posmatra samo interfejs ne može da se nasluti da su dobijeni podaci iz raličitih izvora. Međutim, u implementaciji ovog servisa se pozivaju dva različita izvora i rezultati se kombinuju. Primer metode koja spaja podatke različitih izvora je predstavljen u kodu 4.6.2.

Metoda *getMyCoins* (Kod 4.6.2) treba da vrati listu *Coin* objekata koji sarži sumirane jedne kupljene valute za korisnika. Ulazni parametar ove metode je korisničko ime korisnika koji zahteva podatke. Prvo se pročita podatak o svim transakcijama koji se na nivou upita grupiše po valutama. Nakon toga se sa eksternog API-a pročitaju trenutne cene za valute koje korisnik poseduje. Sledeći poziv se upućuje bazi podataka za čitanje sačuvanih linkova na logoe valuta. Svi prikupljeni podaci se spajaju, sortiraju po količini novca i tako vraćaju kao rezultat poziva ovog metoda.

```

@Override
public List<Coin> getMyCoins(final String username) {
    //reads from DB
    this.myCoinsMap = getMyCoinsWithoutPrices(username);

    List<String> myCoinsSymbols = new
        ArrayList<>(this.myCoinsMap.keySet());
    Map<String, Double> prices =
        restReader.getPricesFor(myCoinsSymbols);
    Map<String, CoinImageUrl> coinDets =
        dbReader.findBySymbol(myCoinsSymbols);

    for (Coin v : myCoins){
        v.setCurrentPrice(prices.get(v.getSymbol()));
        CoinImageUrl ciu = coinDets.get(v.getSymbol());
        v.setName(ciu.getName());
        v.setImageLink(ciu.getUrl());
    }
    Collections.sort(myCoins, new CoinAmountComparator());
    return this.myCoins;
}

```

Kod 4.6.2 Metoda CoinManagera koja vraća podatke za Dashboard komponentu u klijentskom delu aplikacije. Zanimljivost ove metode se ugleda u kombinovanju podataka sa različitih servisa. Sa restReader-a se čitaju cene, dok se iz baze čitaju valute koje korisnik poseduje kao i linkovi na slike valuta koje korisnik poseduje. Nakon toga se podaci spajaju i tako vraćaju pozivaču ove metode.

Grupisanje podataka na nivou upita je definisano u imenovanom upitu (NamedQuery) koji se definiše u entitu Buying po imenu “[Buying.getGrouped](#)”. Upit filtrira podatke po korisniku pre grupisanja kako bi se ograničio na transakcije jednog korisnika. Nakon grupisanja, se radi i

```

@NamedQuery(name="Buying.getGrouped",
    query="SELECT b.symbol, sum(b.input) as input, max(b.date) as date,"
    + "sum(b.amount) as amount FROM Buying b WHERE b.username=:username"
    + "GROUP BY b.symbol HAVING sum(b.amount) > 0")

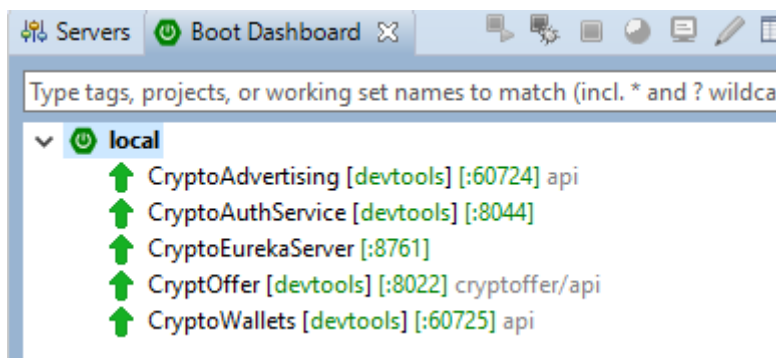
```

Kod 4.6.3 Upit grupisanja transakcija u sumirane podatke jedne valute. Ovaj upit zaista pruža trenutnu realno stanje po valutama.

filtriranje podataka čija je ukupna količina 0, u cilju izbacivanja prodatih valuta.

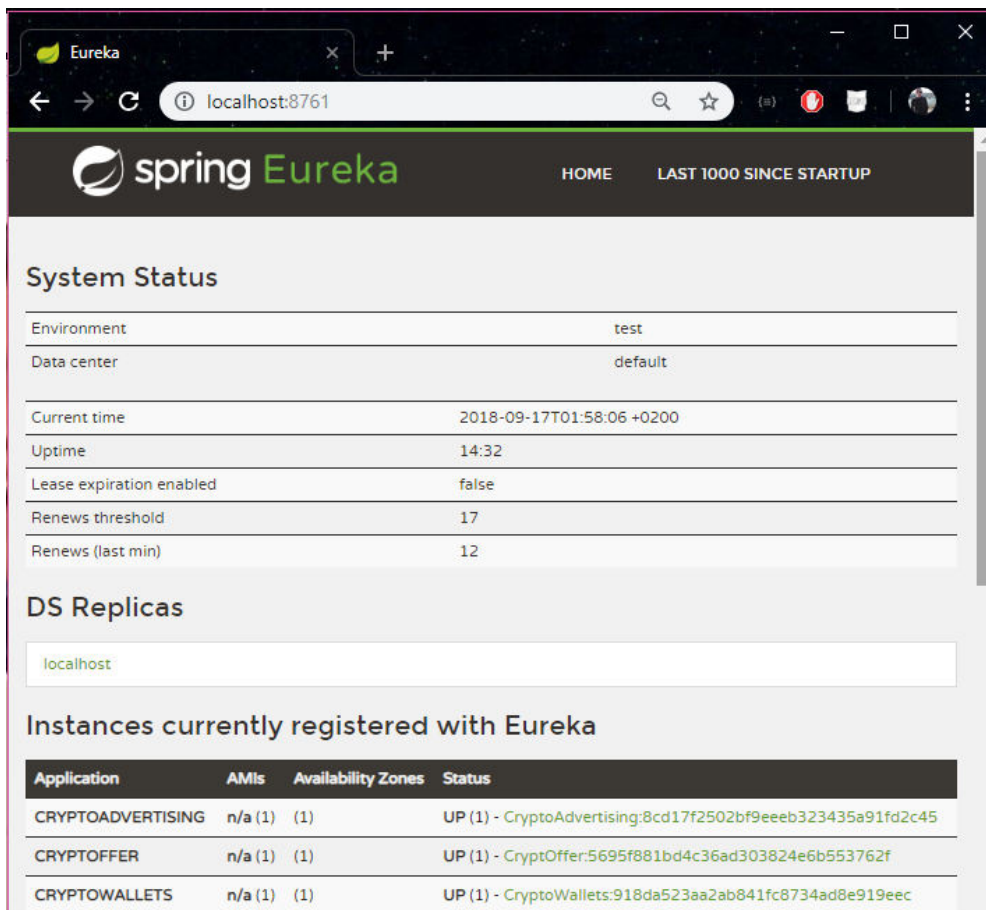
5. UPOTREBA APLIKACIJA

Pokretnje celokupnog sistema zahteva podizanje više aplikacija. Pre svega, podizanje svih serverskih mikroservisa. U pitanju su 5 Spring Boot aplikacija koje su izvršive. Primer stanja pokrenutih svih servisa u razvojnom okruženju prikazan je na slici 5.1



Slika 5-1 Slika primera pokrenutih servisa u razvojnom okruženju. U zagradama broječi zapis predstavlja port na kome aplikacija drži otvoren komunikacioni kanal RESTfull

Preporučivo je najpre aktivirati Eureka Server kako bi ostali mikroservisi po pokretanju uspešno izvršili registraciju. Kada se Eureka Server podigne, u okviru svoje implementacije sadrži prostu web aplikaciju koja služi za prikaz trenutnog stanja. Primer se nalazi na slici 5.2. Pristup ovoj web aplikaciji se dobija prostim otvaranjem adrese pokrenute aplikacije u web-brower-u (na primeru ovaj servis je podignut na portu 8761 lokalne adrese).



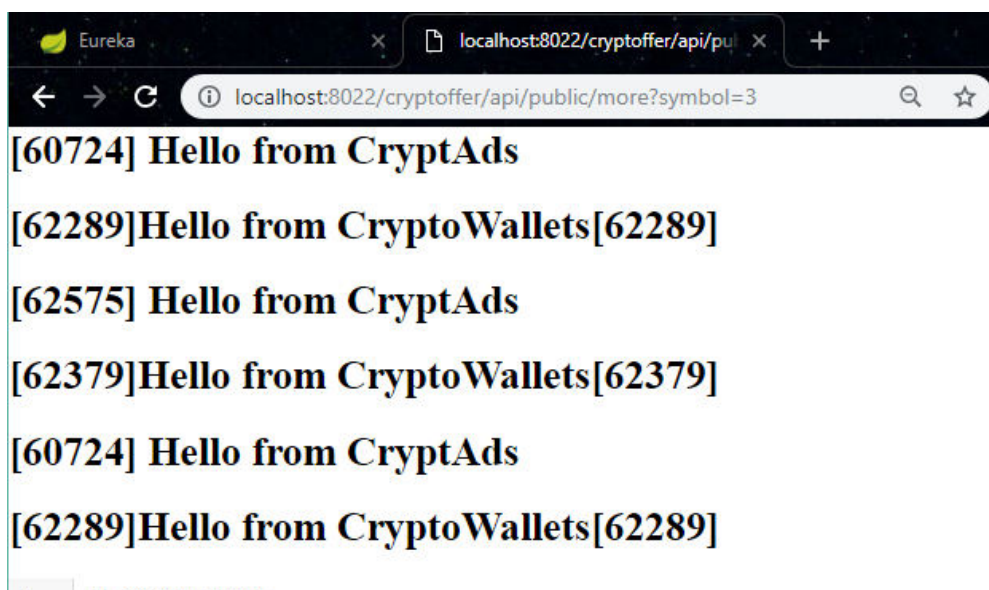
Slika 5.2. Ugrađen web klijent eureka servera koji služi za prikaz stanja.

Slika 5.3 predstavlja tabelu registrovanih mikroservisa u slučaju kada postoji više od jednog aktivnog servisa po imenu. Prvi red tabele daje do znanja da su aktivne 2 instance mikroservisa po imenu *CryptoAdvertising*. Treći red pokazuje da su ukupno bile registrovana 3 mikorservisa po imenu *CryptoWallets*, i da su trenutno 2 aktivna (Primer je simuliran radi prezentacije).

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
CRYPTOADVERTISING	n/a (2)	(2)	UP (2) - CryptoAdvertising:8688f45ffc232a870bd3530b7a740d27 , CryptoAdvertising:8cd17f2502bf9eeeb323435a91fd2c45
CRYPTOOFFER	n/a (1)	(1)	UP (1) - CryptoOffer:5695f881bd4c36ad303824e6b553762f
CRYPTOWALLETS	n/a (3)	(3)	UP (2) - CryptoWallets:fa5966619a9c123c7c2e21ebdceb3b8a , CryptoWallets:601c43c57c62896abfd08c70f33a7f74 DOWN (1) - CryptoWallets:918da523aa2ab841fc8734ad8e919eec

Slika 5.3. Isečak sa monitoring aplikacije EurekaServera u slučaju više aktivnih instance mikroservisa.

Kao reprezentativan primer upotrebe Ribbon-a napisane su metode na servisima CryptoOffer , CryptoAdveriting i CryptoWallets. CryptoOffer poziva 3 puta oba mikroservisa, konkatenira odgovore i na kraju vraća celokupni odgovor. CryptoAdvertising i CryptoWallets odgovaraju sa imenom aplikacije kao i portom na kojem je instanca mikroservisa aktivna. Odgovor na ovakav zahtev je predstavljen na slici 5.4.



Slika 5.4. Slika prezentacije raspodele zahteva od strane Ribbon-a.

Ovaj primer je napravljen samo za namene testiranja Ribbon balansera zahteva i reporezentativnog je karaktera. Dokaz rada raspodele zahteva se ovde očitovidi. CryptoWallets šalje 3 odgovora: 1.i 3. Odgovori su od mikroservisa koji je aktivan na portu 62289 dok je 2. odgovor sa porta 62379. Situacija je slična i za pozivanje CryptoAdveriting mikroservisa.

Nakon uspešnog startovanja serverskih servisa, za rad je neophodno pokrenuti i server koji pruža Klijentsku aplikaciju. Pokretanje servera za Angular aplikaciju se vrši u Command Prompt-u komandom navedenom u

```
>ng serve
```

Kod 5.1. CMD komanda za pokretanje Angular klijentske aplikacije

kodu 5.1

Nakon pokretanja, aplikacija je dostupna (po podrazumevanim podešavanjima) na portu 4200. Po otvaranju se dobija login ekran Slika 5.5.

Nakon uspešnog logovanja, korisnik biva prebačen na Dashboard (Slika 5.6). Na ovom ekranu vidimo celokupno trenutno stanje (leva tablea slike 5.6) kao i pregled podataka po kriptovalutama (desna tabela slike 5.6). Ovaj ekran

My wallet CryptoAds Reports Logout

CRYPTOFFER

Make full detailed view to you crypto finance.
Make offers and find others.
No fee!
Peer2peer market!

Login

Username

Password

Login

Register

Slika 5.5 Login ekran pokrenute klijentkse aplikacije

predstavlja sveukupni pokazatelj stanja kriptovaluta koje korisnik poseduje.

[My wallet](#)[CryptoAds](#)[Reports](#)[Logout](#)

CRYPTOFFER


Make full detailed view to you crypto finance.
Make offers and find others.
No fee!
Peer2peer market!

Current balance review


Total investment	2,863.68 €	
Current capital	4,319.43 €	
Profit	1,455.75 €	50.84 %

Add new transacion

My coins review


ETH

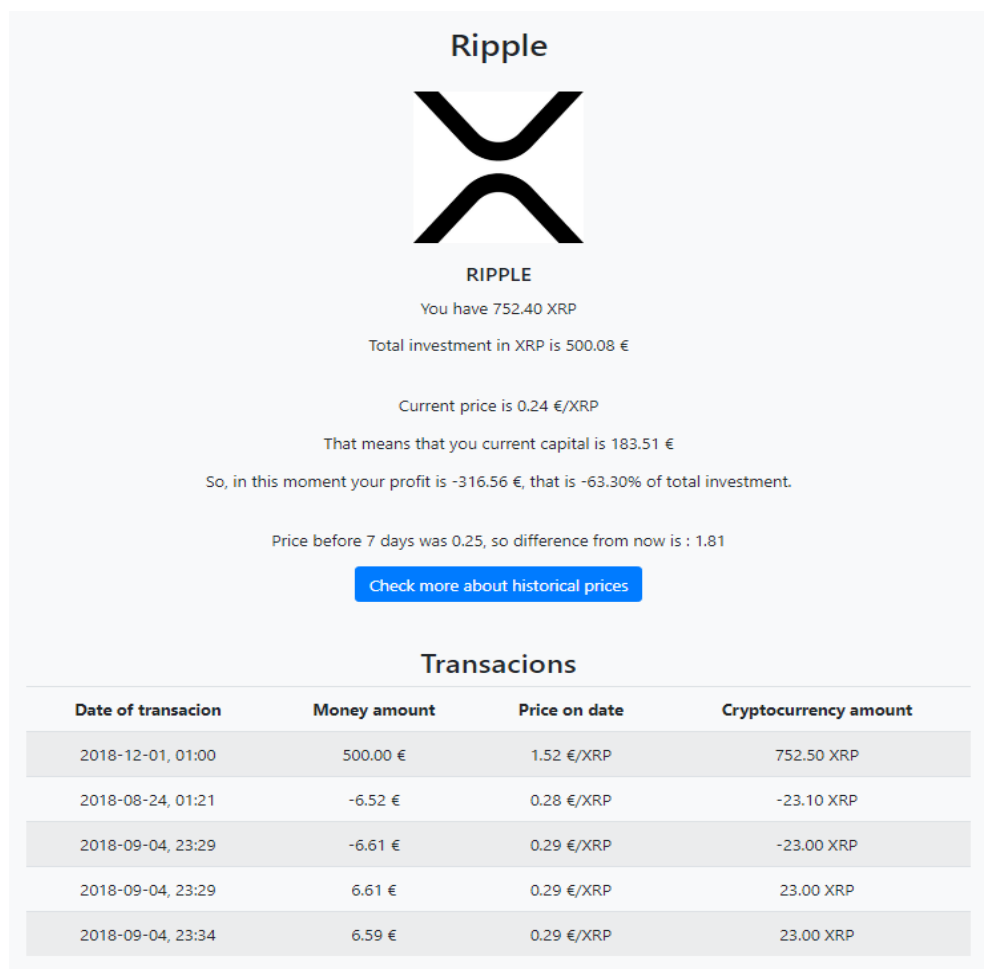
Amount:	0.12 ETH	
Total input:	1,064.76 €	
Current price:	192.21 €/ETH	
Current capital:	23.07€	
Profit:	-1,041.70 €	-97.83%


BTC

Slika 5.6. Dashboard ekran. Prikaz stanja kriptovaluta ulogovanog korisnika.

Za detaljniji prikaz stanja odabrane valute dovoljan je klik na željenu kriptovalutu i korisnik će biti prebačen na stranicu koja prikazuje detalje (Slika 5.7.). Na ovoj stranici dato je deskriptivno stanje odabrane

kriptovalute uz pokazatelje profita u pozitivnom ili negativnom smislu. Takođe, korisna informacija predstavlja cenu pre nedelju dana i odnos te cene sa trenutnom. Link za detalje kretanja cene se takođe nalazi na ovoj stranici. Ispod toga, data je tabela sa svim transakcijama ove valute.



Slika 5.7. Stranica za prikaz detalja jedne kriptovalute – konkretan primer: Ripple

Na svim stranicama je prisutan navigacioni bar koji služi za kretanje u okviru aplikacije. U navigacionom baru se nalazi dugme za prelazak na stranicu oglasa. Klikom na *CryptoAds* korisnik će biti prebačen na stranicu koja je prikazana na slici 5.8. Pretraga po valuti je implementirana serverski (ne

samo filtriranje od strane klijentske aplikacije). Posmatranjem moguće akcije za svaki oglas može se uočiti razlika (dugme pored opisa oglasa). Naime, ukoliko je pronađen oglas koji je postavljen od strane ulogovanog korisnika (na primeru je ulogvan korisnik sa korsinickim imenom ‘robii’), moguća akcija je brisanje oglasa. Za oglase postavljene od strane drugih korisnika, ponuđena akcija je prihvatanje i realizacija ponude.

Ads

Search
Clear

Found 14 ads

Buy/Sell	Symbol	Amount	Date of upload	Set by user	
🛒	XRP	22.00	2018-08-24, 00:08	jaki	Realize with my wallet
🛒	IOT	2.32	2018-06-28, 17:05	marko97	Realize with my wallet
🛒	ETH	2.32	2018-06-19, 01:46	robii	Delete
🛒	BTC	0.03	2018-06-19, 01:46	pane	Realize with my wallet
🛒	XRP	23.10	2018-06-17, 13:16	helen	Realize with my wallet
🛒	BTC	0.00	2018-06-17, 01:46	jaki	Realize with my wallet
🛒	XRP	23.00	2018-05-31, 01:46	robii	Delete
🛒	ETH	0.02	2018-01-31, 00:46	stef	Realize with my wallet

[Add new advertisement](#)

Slika 5.8 – Stranica prikaza oglasa za trenutno ulogovanog korisnika: robii.

6. ZAKLJUČAK

Na postavljanje pitanja: ‘da li ovakav projekat - web aplikacija zaista mora da bude podeljena po servisima?’ odgovor je negativan. Ovako zahtevana aplikacija se vrlo prostije može razviti u okviru jednog servisa. Međutim, to nije bio cilj ovog rada. Cilj ovog rada je istraživanje i učenje u pravcu mikroservis orijentisanog razvoja. Cilj je ostvaren, a rezultat se nalazi u ovom radu.

Motivacija mikroservis orijentisanog razvoja je trenutni trend u svetu informacionih tehnologija. Napredak hardverskih komponenti polako gubi na značaju zbog odnosa cene i dostignuća. Napredak softvera ima konstantne finansijke troškove, a svaki korak napredka čini osnovu za sledeći. Na taj način se softverska tehnologija eksponencijalno razvija. Zahteva i ideja je sve više, a hardver je skup. Iz ovih razloga se radi distribucija softvera po hardverima. Monolitan sistem mora da radi na jednoj hardverskoj komponenti. Distribuirani (mirkoservisni) sistem je raposređen na više harverskih komponenti te se pojačanje performansi postiže integracijom nove harderske komponente.

Osim pravca kretanja IT-a motivacija je i modularan razvoj. Mikroservis orijentisan razvoj zahteva strogu podelu po modulima. Projektni zadatak treba dobro podeliti kako na nezavisne celine. Kada se to uspešno uradi, razvoj je mnogo lakši i jednostavniji nego kod monolitnog sistema. Razlog je što se svaka komponenta pojedinačno obrađena i testira, pa je nakon toga integracija mikroservisa dosta bezbolnija.

Nepisan zahtev mikroservisa je kvalitetan monitoring. Jako je bitno da se u slučaju greške jako lako locira problem, identifikuje mesto problema. Zbog toga svaki mikroservis zahteva kvalitetnu signalizaciju tokom rada.

7. LITERATURA

1. <https://angular.io/>
2. <https://dzone.com/>
3. <https://spring.io/projects/spring-boot>
4. <http://cloud.spring.io/spring-cloud-aws/1.2.x/>
5. <https://getbootstrap.com/>
6. <https://www.wikipedia.org/>
7. <https://www.w3schools.com/>
8. <https://searchdatamanagement.techtarget.com>
9. <https://spring.io/guides/gs/service-registration-and-discovery/>
10. Blog:
 - REST in Peace: Microservices vs monoliths in real-life examples
 - korisnik: RDX;
 - sajt: <https://medium.freeCodeCamp.org>
11. Knjiga:
 - Mastering Spring Cloud
 - od: Piotr Minkowski;
 - izdavač: Packt Publishing
12. Knjiga:
 - REST with Spring
 - od: Baeldung
13. Knjiga:
 - Microservice Architecutre;
 - od Irakli Nadareishvili and Ronnie Mitra

Kod aplikacije ovog diplomskog rada se može pronaći na GitHub-u .

Serverski deo aplikacije

https://github.com/robertsabo0/AWS_CryptOffer

Klijentski deo aplikacije:

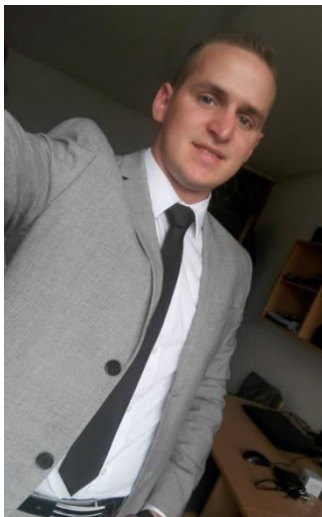
https://github.com/robertsabo0/CryptOffer_Client

8. REFERENCE

1. Service oriented architecture
 - https://en.wikipedia.org/wiki/Service-oriented_architecture
2. Framework
 - <https://whatis.techtarget.com/definition/framework>
3. Java
 - <https://www.ibm.com/developerworks/java/tutorials/j-introjava1/index.html>
 - JAVA 8 – Programiranje, pisac Yakov Fain, izdavač: Kompjuter biblioteka Beograd
4. JavaScript
 - <https://www.javascript.com/>
5. Google
 - <https://www.google.com/about>
6. HTML
 - Introducing HTML5, by. Bruce Lawson, Remy Sharp
 - <https://www.yourhtmlsource.com/starthere/whatishtml.html>
7. Microsoft
 - <https://www.microsoft.com/en-us/about>
8. *App.module.ts*
 - <https://angular-2-training-book.rangle.io/handout/modules/introduction.html>
9. Command Prompt
 - <https://www.lifewire.com/command-prompt-2625840>
10. Dependency Injection
 - <https://searchmicroservices.techtarget.com/definition/dependency-injection>
11. jQuery
 - https://www.w3schools.com/jquery/jquery_intro.asp
12. Apache Tomcat Server
 - <http://tomcat.apache.org/>
13. XML
 - https://www.w3schools.com/xml/xml_what.asp

-
- Java and XML, 3rd Edition – Solutions for Real-World Problems by: Brett McLaughlin and Justin Edelson
14. Relational Databases
 - <https://searchdatamanagement.techtarget.com/definition/relational-database>
 15. SQL
 - https://www.w3schools.com/sql/sql_intro.asp
 16. Hibernate
 - <http://hibernate.org/>
 17. HTTP
 - <https://searchwindevelopment.techtarget.com/definition/HTTP>
 - HTTP: The Definitive Guide (Definitive Guides) 1st Edition; by David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal and Sailu Reddy
 18. JSON
 - https://www.w3schools.com/js/js_json_intro.asp
 19. Spring Cloud
 - <http://projects.spring.io/spring-cloud/>
 20. Spring Cloud Netflix
 - <https://cloud.spring.io/spring-cloud-netflix>
 21. BCryptEncoder
 - <https://docs.spring.io/spring-security/site/docs/4.2.7.RELEASE/apidocs/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html>
 22. CryptoCompare
 - <https://www.cryptocompare.com/>
 23. CryptoCompare API docs
 - <https://min-api.cryptocompare.com/>
 24. Reactive programming
 - <https://dzone.com/articles/5-things-to-know-about-reactive-programming>
-

BIOGRAFIJA



Robert Sabo

Cara Lazara 34

Ridica 25280, Sombor

email: robert.sabo0@gmail.com

Kontakt: +381631164045

Zanimanje: student – Prirodno-Matematički
Fakultet u Novom Sadu; smer Informacione
tehnologije

Robert Sabo, rođen je 22. Septembra 1995. godine u Somboru, Srbija. Osnovnu školu “Petar Kočić” u Ridici završio je 2010. godine, iste godine upisuje srednju ekonomsku školu u Somboru. Srednju školu uspešno završava 2014. godine. Nakon završetka odlučuje da se upiše na akademske studije. 2014. godine upisuje Prirodno-matematički fakultet univerziteta u Novom Sadu, smer informacione tehnologije na departmanu za matematiku i informatiku.

UNIVERZITET U NOVOM SADU
PRIRODNO MATEMATIČKI FAKULTET
DEPARTMAN ZA MATEMATIKU I INFORMATIKU

KLJUČNA DOKUMENTACIJA

Redni broj:

RBR

Identifikacioni broj:

IBR

Tip dokumentacije:

TD

Monografska

dokumentacija

Tip zapisa:

TZ

Tekstualni

štampani materijal

Vrsta rada:

VR

Završni rad

Autor: Robert Sabo

AU

Mentor: dr Srđan Škrbić

MN

Naslov rada: Skalabilan mikroservis orijentisan sistem

NS namenjen berzi kriptovaluta

Jezik publikacije: Srpski/latinica

JP

Jezik izvoda: Srpski

JI

Zemlja publikacije: Srbija

ZP

Uže geografsko područje: Vojvodina

UGP

Godina: 2018

GO

Izdavač:

Autorski reprint

IZ

Mesto i adresa:
Trg

Prirodno – matematički fakultet,

MA

Dositeja Obradovića 4, Novi Sad

Fizički opis rada:

8 poglavlja, 79 strana,

FO

17 slika, 26 koda

Naučna oblast:

Informatika

NO

Naučna disciplina:

Web Aplikacije

ND

Predmetna odrednica/

Angular, Sprig Boot

Ključne reči:
Skalabilnost

Microsevice,

PO

UDK:

Čuva se:
matematiku

Biblioteka Departmana za

ČU

i informatiku, Novi Sad

Važna napomena:

Nema

Izvod

IZ:

Opis skalabinlog mikroservis orijentisanog
sistema namenjenog berzi
kripto valuta

Datum prihvatanja teme:

DT

Datum odbrane:

DO

Članovi komisiji:

1. dr Vladimir Kurbalija, vanredni profesor PMF-a, predsednik
2. dr Srđan Škrbić, vanredni profesor PMF-a, mentor
3. dr Mirjana Mikalački, docent PMF-a, član

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCES
DEPARTMENT OF MATHEMATICS AND
INFORMATICS

<p>KEY WORDS DOCUMENTATION</p>

Accession number:

ANO

Identification number:

INO

Document type:

Monograph type

DT

Type of record:

Printed text

TR

Contents code:

Diploma's thesis

CC

Autor:	Robert Sabo
AU	
Mentor:	PhD Srđan Škrbić
MN	
Title:	Scalable microservice oriented for
XI	cryptocurrency stock exchange
Language of text:	Serbian/Latinica
LT	
Language of abstract:	Serbian
LA	
Country of publication:	Serbia
CP	
Locality of publication:	Vojvodina
LP	

Publication year:	2018.
PY	
Publisher:	Author's reprint
PU	
Publication place:	PMF, Trg Dositeja
PP	Obradovića 4, Novi Sad
Physical description:	8 chapters, 79 pages,
PD	17 photos, 26 codes
Scientific field:	Informatics
SF	
Scientific discipline:	Web applications
SD	
Key words:	Angular, Spring Boot Cloud,
KW	Microservice,
Scalability	
Holding data:	Library of Department of mathematics
SD	and informatics, Novi Sad

Note:

None

Abstract:

AB

Microservice oriented system
implemented by using Angular
and SpringBoot Cloud technologies

Accepted by the Scientific Board on:

Defended on:

Thesys defend board:

1. PhD Vladimir Kurbalija, associate professor, Faculty of Sciences,
Novi Sad, president
2. PhD Srđan Škrbić, associate professor, Faculty of Sciences, Novi
Sad, mentor
3. PhD Mirjana Mikalački, associate professor, Faculty of
Sciences, Novi Sad, member