|  |  |  |
| --- | --- | --- |
| Grb Univerziteta | Grb PMF-aUniverzitet u Novom Sadu  Prirodno-matematički fakultet  Departman za matematiku i informatiku |  |
|  |  |  |

Robert Sabo

**Skalabilan mikroservis orijentisan sistem namenjen berzi kriptovaluta***C r y p t O f f e r*

-Diplomski rad-

Novi Sad, 2018

**SADRŽAJ**

[PREDGOVOR 5](#_Toc524980949)

[1. UVOD 7](#_Toc524980950)

[Šta su kriptovalute? 7](#_Toc524980951)

[Domen primene i cilj *CryptOffer-*a 8](#_Toc524980952)

[*CryptOffer* kao zatvoren sistem kontrole zajedničkih resursa 9](#_Toc524980953)

[2. KORIŠĆENE TEHNOLOGIJE 11](#_Toc524980954)

[2.1. Klijentski deo aplikacije 12](#_Toc524980955)

[Angular 12](#_Toc524980956)

[Bootstrap 20](#_Toc524980957)

[2.2. Serverski deo aplikacije 21](#_Toc524980958)

[Spring Boot 22](#_Toc524980959)

[Baza podataka 25](#_Toc524980960)

[Komunikacija 27](#_Toc524980961)

[Sigurnost podataka 33](#_Toc524980962)

[3. ARHITEKTURA I SARŽAJ SOFTVERA 36](#_Toc524980963)

[3.1. Angular klijenska aplikacija 38](#_Toc524980964)

[3.2. CryptOffer 38](#_Toc524980965)

[3.3. CryptoAuthService 39](#_Toc524980966)

[3.4. CryptoEurekaServer 40](#_Toc524980967)

[3.5. CryptoAdvertising 41](#_Toc524980968)

[3.6. CryptoWallets 42](#_Toc524980969)

[4. IMPLEMENTACIJA SOFVERA 43](#_Toc524980970)

[4.1. Angular klijentska aplikacija 43](#_Toc524980971)

[4.2. CryptOffer 46](#_Toc524980972)

[4.3. CryptoAuthService 49](#_Toc524980973)

[4.4. CryptoEurekaServer 50](#_Toc524980974)

[4.5. CryptoAdvertising 51](#_Toc524980975)

[4.6. CryptoWallets 54](#_Toc524980976)

[5. UPOTREBA APLIKACIJA 56](#_Toc524980977)

[6. ZAKLJUČAK 64](#_Toc524980978)

[7. LITERATURA 65](#_Toc524980979)

[8. REFERENCE 66](#_Toc524980980)

[BIOGRAFIJA 68](#_Toc524980981)

# PREDGOVOR

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

Mikroservis je tehnika razvoja *software-*a bazirana na servis orijentisanoj arhitekturi (SOA[1]). Osnovna ideja je podela sistema na nezavisne celine. Svaka celina prestavlja po jedan mikroservis koji je precizno definisan svojim interfjesom 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, prednosti ovakve arhitekture je i modularnost u cilju lakšeg razumevanja, razvoja i testiranja.

*SpringBoot* framework[2] trenutno je jedna od najpopularnijih tehnologija za implementaciju serverske strane aplikacija. Ovaj frameworkzasnovnan je na *Java*[3]platformi, što znači da bilo koje aplikacije koje su pisane u programskom jeziku Java mogu da koriste ovaj framework*. SpringBoot* frameworkje platforma otvorenog koda (*open source*), što znači da je njegov kod javno dostupan svima.

*Angular* je JavaScript[4] framework koji služi pravljenju dinamičkih web i moblinih aplikacija. Napravljen je od strane Google-a[5]. *Angular* nam omogućava da pravimo dobro struktuirane, tipizirane aplikacije koje su lake za testiranje i održavanje. Ovaj framework je otvorenog koda, kao i *SpringBoot*.

Diplomski rad sastoji se od sledećih poglavnja:

1. U *uvodnom* delu je opisan cilj i domen primene apikacije.
2. Poglavlje *Korišćene tehnologije* opisuje sve tehnologije korišćene za izradu ovog rada.
3. U poglavlju *Arhitekura i sadržaj mikroservisa* je data celokupna arhitektura mirkoservisa kao i distribucija logike i implementacije
4. Poglavlje *Implementacija softvera* se sastoji od prikaza samog koda aplikacija, uz detaljan opis.
5. *Upotreba sistema* sadrži upustvo za korišćenje aplikacije, iz perspektive korisnika.
6. U *zaključku* se nalazi lični utisak procesa razvoja ovog pojekta.
7. U *literaturi* je spisak svih korišćenih literatura pri izradi projekta.
8. *Reference* gde se nalaze usmerenja za više informisanja o terminima koji su korišćeni a nisu obrađivani u ovom radu.

# UVOD

Drugom polovinom 2017. godine došlo je do ogromne popularizacije **kriptovaluta**. Najveću pažnju je privuklo kretnje cena kritpovaluta, potpuno očekivano, rastom popularnosti raste tražnja pa raste i sama cena, što je pogrešnan pravac razmišljanja u sveri tehnoogije: 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 sve 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 sistemam. Preciznije kriptovalute su decentralizovane, dok je redovna moneta centralizovan sistem. Razlika je što kod centralizovanog sistema jedna, centralna, jedinica je glavna i ona cuva podatke bez sunje u tačnost, dok ko decentralizovanog sistema mnogo jedinica čuva isti podatak i validan je podatak (koncenzus) 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 (koncenzus).

### 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 finansijsko 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 evideniji. Tu se podrazumeva kalkulacija imovine lica, odnosno prikaz rezultata rezultat trgovine kao i odgovori na pitanja kao što su: koji koraci su biti 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 reprezentaija 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 transkcije (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 uloženih 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
* *CryptoWallet*s 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 koristi ko god poželi. U tom slučaju *CryptoAds* sistem služi za stupanje u kontakt s klijenata i razmene valuta na taj način. Problem koji se pojavljuje je da nakon obavljene transakije unutar ovog sistema korisnici moraju i realnu promenu da izvrše: prodavac pošalje kriptovalute u kriptonovcanik kupca, a kupac uplati novac prodavcu.

Međutim, šta ako bi svi korisnici koristili isti kriptonočanik? U tom slučaju, ukupna raspoloživa sredstva u kriptonočanicima 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 kriptonovcanika 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 apsurda i isključuje u potpunosti benefite sistema kritpovaluta. 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 transakije vrši u kriptovalutama. Tada modul *CryptoWallet* pruža stanje. Direktan unos transakcija u *CryptoWallet* su ulazi i izlazi novca iz organizacije (prodaja kritpovalute je isplata , npr.: radnik podiže gotovinski novac; kupovina je uplata kritpovaluta, 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 konrketnom 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.

# KORIŠĆENE TEHNOLOGIJE

*CryptOffer* je, celokupno gledano, web aplikacija. Korisnici joj pristupaju i koriste je putem web-browser-a (istraživač) putem adrese. Sve ostalo potrebno za rad aplikacije treba 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 predstvalja kod koji se izvršavama u samom   
web-browser-u korisnika i on je najčešće zadužen da vizualizaciju, odnosno odgovoarajući prikaz aplikacije. Takođe, pored prikaza služi i za koridiniranje 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 klijnta, gde se kod izvršava u browseru svakog korisnika, ovaj kod se uglavnom izvršava na jednom mestu, na serveru. Osnovna uloga servera je obsuživanje klijenata. Pored toga, najčešće se server bavi celom poslovnom logikom aplikacije.

## Klijentski deo aplikacije

Za izradu klijentskog dela web aplikacije se najčešće koristi kombinacija HTML-a[6], kao deskrpitivnog 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 najzastupljenjija 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] (vrevodi 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. U prvoj verziji, ovaj se je okvir nazivao 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.

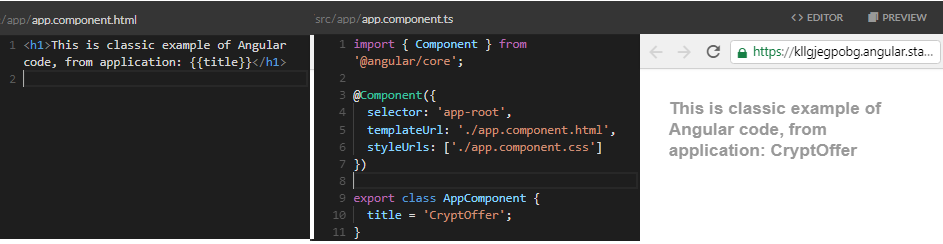
Angular *framework* napisan je programskom jeziku TypeScript. Ovaj programski jezik razvijenom od strane Microsoft-a[7]. Njegova je svrha ispraviti nedostatke JavaScript-a. TypeScript kod nije izvršiv od strane   
web-browser-a pa se on prevodi (eng. compile) u JavaScript kod. TypeScript omogućava strogo 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-aa (kaže se u runtime-u aplikacije), dok se u TypeScriptu tipska greška signalizira prilikom prevođenja (compiletime), za vreme samog kucanja koda. Pored strogo tipiziranje i otkrivanje 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, iliti 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 koji najznačajniju ulogu igraju: servisi i direktive. U okrivu *CryptOffer* a su pravljene komponente i servisi, dok su direktive korišeni predefinisani 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 kompomente čine 4 fajla:

* Šablon (template) – \*.hml – HTML statički kod deskriptivnog tipa koji se dopunjava Angular specificnim delovima kao dinamičko referenciranje za podakte 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 interpetirati 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, svakaka 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 *CryptOffer*a
* Test – \*.spec.ts – fajl namenjen za testiranje komponente. (nije korišćeno niti izučavan za ovaj diplomski rad)

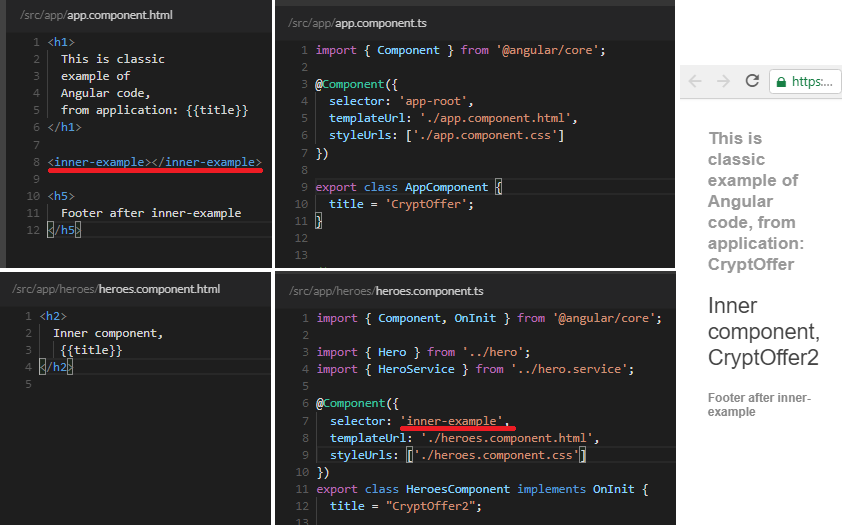


Kod 2.1.1.- Primer komponente u Angular-u. Levi deo je HTML sa Angular specifikacijom. Sredina je TypeScript klasa sa vrednost. Desno je predstavljen rezultat ovog koda

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 ovkviru pasusa su boldovana imena koja referenciraju na prezentaciju sa slike).

Međukomponentsko uvezivanje se definiše unutar šablona. Da bi se unutar jedne komponete 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 poglavnju 4. Implementacija softvera)

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



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

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 Angualar komponente. Ova naredba će izgenerisati komponentu po imenu ‘dashboard-component’ kako je to navedeno u komadni

#### Servisi u Angularu

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

* Klasa – \*.ts – TypeScript klasa koja je zaduzena 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 netrivijalne 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 genersanja servisa može se videte na kod 2.1.4. Pored generisanja fajlovak 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 sam potreban posao generisanja novog servisa (komponente ili drugo).

>ng generate service auth

Kod 2‑1.4 CMD comanda za generisanje Angualar servisa. Ova naredba će izgenerisati servis po imenu ‘auth-service’ koij 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 ekspicitnog instanciranja zahtevanog objekta. U različitim tehnologijama različito se navodi korišćenje ovog paterna, ali je suština upotrebe svigde isti. 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 možemo pogledati u Kod 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: ActivedRoute 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 u Angularu

Direktive predstavljaju izvršni kod u okrviru HTML stranica komponenti. U okviru komponente je da šablon komponente sadrži deskrpitvni deo – HTML i dinamički deo koji se menja realnim vrednostima iz objekta klase komponente. Međutim, ovde nije uzeto u obzir dinamičnost dekriptivnog 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 directive 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 izdradi ovog rada.

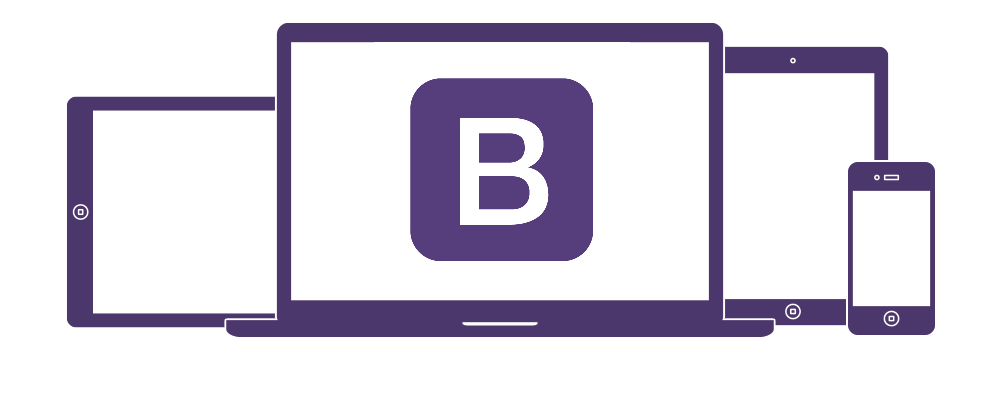
Pored navedenih karakterističnih stvari za Angular, u mogućnosti upotrebe stoje i mnogo drugih softverski dobro poznatih stvari, kao što su (navode se upotrebe u izradi *CryptOffera*):

* 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 stranie, obimnih ugrađenih component i stilova, kao i mogućnosti dodataka zasnovanih na jQuery-u [11] ili JavaScript-u, stilizacija i vizualizacija web aplikacije je u mnogoolakšan.



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:

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

Bootstrap u sebi ima ugrađen set alata i biblioteka za vizuelno sređivanje web aplikacija. Primenjuje 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 visokorezolucionarnih prezentacija do malih telefonskih prezentaija.

## 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 obsluživanje svih klijenata. Server koristi razne tehnologije kako bi najopštije i najuspešnije vršio svoj posao. Većinu tehnologiju korišćenu u *CryptOffer* u će biti objašnjenje u ovom poglavlju.

U prethodnom pasusu je navedeno da najčešće jedna serverska aplikacija obslužuje sve klijente. Međutim, *CryptOffer* je primer web aplikacije gde serverski deo nije jedna aplikacija već skup koji uključuje nekoličinu aplikacija. Serverska logika distribuira se na različite celine kako bi se postigla skalabilnost. To je postignuto mirkoservisnom 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 – mirkoservisi 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. Arhitekutra mikroservisa *CryptOffera*  je opisanu u poglavlj 3-Arhitektura i sadrđaj mirkoservisa. Ovo poglavlje opisuje pomoćne tenhologije koje koriste mikrosevisi.

### Spring Boot

Svaki od mirkoservisa je napisan u programskom jeziku Java koristeći *Spring Boot framework.* Spring Boot predastavlja vrlo uprošćenu osnovu za pokretanje kopleksnih 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 serversko dela web aplikacija. Sve Springove komponente su pod nadzorom kontejnera koji je zadužen za njihovo efikasno korišćenje. Spring kompontente su sledeće:

* Kontroler – predstalja prezentacioni deo aplikacije. Zadužen je za komunikaciju sa klijentima, mapiranje i preusmeravanje klijenskih zahteva ka odgovarajućim servisima
* Servisi – predstavljaju poslovnu logiku servera. Zaduženi su za pružanje različitih usluga svima koji ih pozivaju (najčešće pozivanje od strane kotrolera 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 čin SpringBoot

Pored navedenih komponenti, u *Spring*u 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 aplikaije ne igra ulogu, *Spring Boot* ne konfigurisanje 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 aplikaje.

Pored uprošćavanja konfiguracija, postoji još jedna bitna razlika između Springa i SpringBoota. Spring aplikacija, kao izvršna, postavlja na web server sa konteinerom (Apache Tomcat [12] ili neki drugi), odnosno razvijena aplikacija zahteva podešavanje web servera koji će izvršavati aplikaciju. *Spring Boot*, ponovo principom podrazuevanog podesavanja, 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 ulog. Uvezivanje komponenti radi po već pominjano principu Dependecy Injection (Opisan u ). Svaka komponenta se incijalizuje od strane servera koji pamti instancu i dobavlja je drugim komponentama. U *CryptOffer* sistemu se depencency injection koristio anotiranjem polja komponente (pogledati Kod 2.2.1) Uvezivanje komponenti je moguće kroz konstruktor takođe (Kao i u Angularu).

Kod 2.2.1 Primer Dependecy Injecitona u Springu

@RestController

**public** **class** AdsAndWalletController {

@Autowired  
AdAPIManager adAPIManager;

#### Maven

*Maven* je alat za izgradnju (build) aplikacije. Zasnovan je na XML[13] fajlu koji opisuje sve ključne karakteristike aplikacije, porebne 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 način uvezivanja pomoćnih, već postojećih biblioteka. Naime, među zavisnostima, odnosno unutar <dependencies> taga *pom.xml* fajla, se navode potrebne biblioteke aplikacije. Pri izgradnji, *Maven* sam skinuti 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 okvru 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 bibilioteke Feign iz springframework.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 vrednsot 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 pisu 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* korsti *JPA Persistence* sa implementacijom *Hibernate*-a[16].

#### JPA Persistence

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 manimpulisanje 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, dobaljvanje podataka i sl. Svaki od tih sinonima označava isto, a to je tok razmene podatak između aplikacija. Način razmena 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 novi potpunu platformsku i jezičku nezavisnost. Potpuno je nebitno 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 conteksta (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 funkcinalno 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 okrviru Springa definisanje RESTa 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 reprezntativan primer se vidi je pokazan na Kod 2.2.5 i rezultat na Slici 2.2.2

Kod 2‑2.3 Implementacija REST kontrolera u Springu. Pokazana je implementacija GET metode ciji je link ‘{linkAplikacije}/public’ kako je navedeno u anotaciji.

@RestController

**public** **class** HBController {

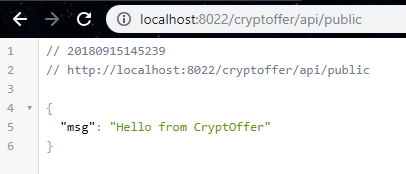
@GetMapping("public")

**public** String hello(){

**return** "{ \"msg\":\"Hello from CryptOffer\"}";

}

}



Slika .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 *CryptOffer*-a postoje 2 mesta čitanja REST APIa 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 bibilioteka

#### Feign klient

Feign predstavlja biblioteku iz Spring Cloud [19] projekta koja se koristi za za čitanje REST APIa. 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 paramtri i povratne informacije moraju da se poklapaju sa ulaznim i izlaznim paramatrima 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: CryptoWallets/api/coins).

@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 APIa u okviru projekta CryptOffera.

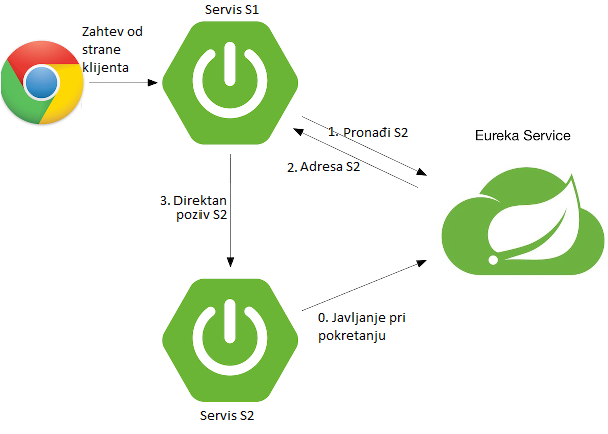
Na primeru Kod 2.2.4 je dat kod za čitanje mikroservisa *CryptoWallets*. Anotacija @FeignClient (“CryptoWallets”) označava da ovaj interfejs čita podatke sa mirkoservisa koji je registrovan po imenu *CryptoWallets* (Više o registrovanim mikroservisima u sekciji o Eureka Serveru). Nakon toga je naveden,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

#### Eureka Server

Eureka Server je biblioteku iz Srping Cloud Netflix [20] projekta. To je, ustvari, biblioteka koja se stavlja u jedan samostalan servis mikroservis. Zadatak ovog Eureka Servera kao mirkoservisa (u daljem tekstu: Eureka Server) je jedinstveno mesto registracija svih drugih mikroservisa.

Eureka Server za cilj ima centralizaciju komunikacije mirkoservisa. Kako bi se mirkoservisi lakoše pronalazili, oni predstavljaju Eureka Klijente, koji se javljaju Eureka Serveru preko svog identifikacionog imena. Kada se mikrosevis javi pod imenim, kažemo da je registrovan na Eureka Server pod tim imenom. Registrovane adrese po imena Eureka Server pruža po zahtevu. Paralelna uloga iz društevnog 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 mirkoservis registruje Eureka Serveru. Ukoliko je servis registrovan, Eureka garantuje da će se svaki zahtev biti korektno usmeren ogovarajućem mikroservisu.

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

Slika 2.2.3 Tok poziva između mikroservisa uz upotrebu Eureka Servra

1. Svaki mirkoservis se po podizanju javlja Eureka Serveru po imenu
2. Servis S1 pita Eureka Server za adresu servisa S2.
3. Eureka Server odgovara sa adresom na kojoj je dostupan servis S2
4. 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 intefejsa 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 finkcionisanje). 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 mirkoservis-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 mirkoservisa 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 podakte 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 authorizacije. U terminologiji ovog protokola, autorizaconi tipovi se nazvivaju Grant tipovi (grant type). Grant tipovi OAuth2 protokola su :

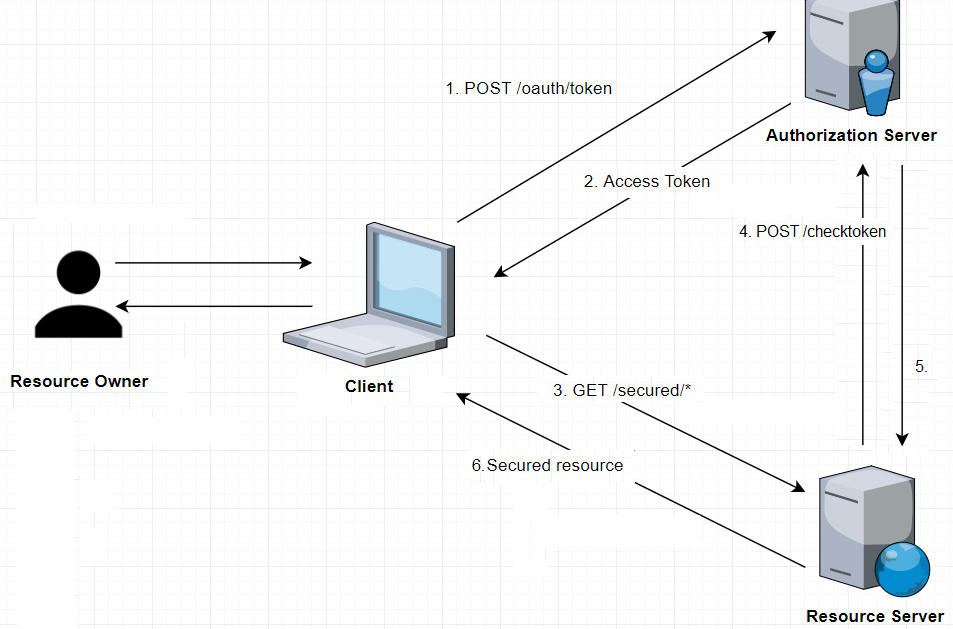
* Autorizacioni kod (eng Authorization Code)
* Lozinka (eng Password)
* Klijentski kredencijali (eng Client Credentials)
* Implicitni (eng Implicit)

*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 logovanje 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 Autorization 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 process se tu neuspešno završava. U suprotnom, ako su kredencijali u redu, autorizacioni server odgovara sa pristpunim tokenom (Access Token). Taj token klijent čuva u sesiji i prize ga svakom zahtevu API Servera.
3. **Klijent** šalje zahtev **API Serveru** za potrebnim podacima kako bi obslužio korisnika. U okrviru 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 zathev sa uključenim tokenom, mora prvo da proveri da li je token zaista validan. Tada se **API server** obrača **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 .2.4 – 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 process 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 uspesne autorizacije (prvi zahtev), svaki sledeći zahtev se sastoji od koraka 3.-6.s

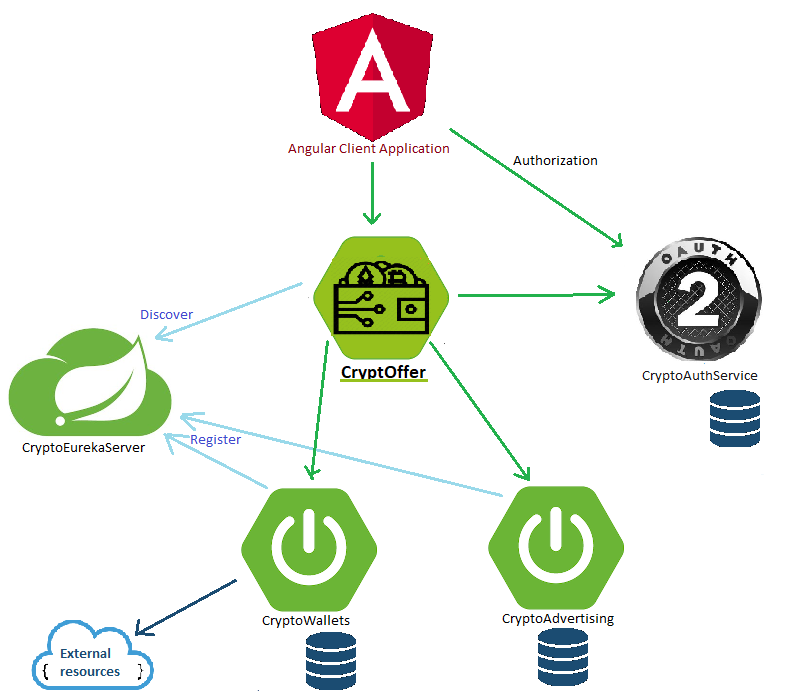
# ARHITEKTURA I SARŽAJ SOFTVERA

U prethodnim poglavljima ovog rada je navedena ideja i cilj celokupnog sistema. Navedene i razjašnjene su tehnologije koje su korštene 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 delina:

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 mirkoservisa aplikacije
5. CryptoAdvertising – servis zadužen za logiku oglasa kupoprodajnih oglasa kritpovaluta
6. CryptoWallets – servis zadužen za evidenciju stanja kriptovaluta korisnika



Slika 3.1. Skica celokupne arhitekture aplikacije. Celokupna aplikaija se sastoji od klijentske aplikacije i serverskih 5 mikroservisa.

## 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 APIa 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, konkrenot *CryptoAuthService*, za proveru kredenijcala i pribavljanje **pristupnog tokena**. Pristupni token se čuva u memoriji web aplikacije i dodaje se uz svaki zahtev kao **API serveru**, konkretno *CryptOffer*-u.

## 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 interfejst 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 mirkoservisima 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 mikroservisu, 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.

## CryptoAuthService

*CryptoAuthService* predstavlja jedini mikroservis, pored osim centralne jedinice (CryptOffer), koji poseduje konekciju sa web klijentom. Svrha komunikacionog kanala sa web klijentskim delom aplikacije se ugleda u ulozi ovog mikroservisa. *CryptoAuthService* predstalja **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 koriscničkih 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 kao odgovor na zahtev. Takođe, isti token biva sačuvan u memoriji zajedno sa podacima o korisniku. Dužina čuvanja tokena se ograničava kroz konfiguraciju.

Druga bitan 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 pristpuni token i svaki put validnost tokena treba da se proveri. Uz proveru validnosti tokena **autorizacioni server** pruža podatek 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.

## CryptoEurekaServer

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

Kada se vrši poziv izneđu mikorservisa, prvo se od ovog mikroservisa zatraži adresa, a tak nakon toga se zahtev usmera na konkrentu instancu mikroserivisa. Tok podataka poziva između mikroservisa detaljno je opisan u opisu same tehnologije Eureka Servera u poglavnju 2.2. podnaslov komunikacija. Slikov prikaz toga podataka je dat na slici 2.2.3.

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

## CryptoAdvertising

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

* Prezentacioni 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 predstalja 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.

## CryptoWallets

*S*lično 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.

Specifičnsot CryptoWallets-a se ogleda u korišćenju eksternih izvora podataka. Naime, on koristi **API berze** koji pruža realne podakte 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 APIa se mogu pronaći u referencama [23].

Korišćenje ekstarnih 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.

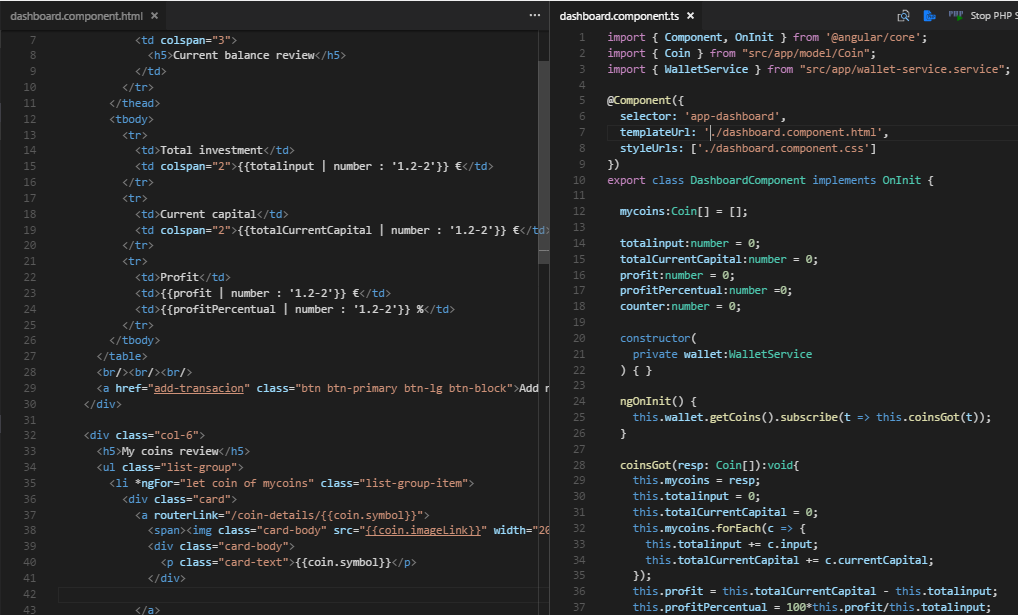
Pored korisnih podataka, integracijom ovog eksternog API servisa obrađen je i prikaz slikovitih simbola valuta. Kako svaka kritpovaluta ima svoj logo koji ga karakteristiše i po kome se prepoznaje, a oni se vrlo često menjaju i dodaju novi, koriste se sličice sa istog servera. Ovakav pristup prebacuje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

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 rezutat.

# IMPLEMENTACIJA SOFVERA

*CryptOffer* kao celokupna aplikacija, samim pogledamo na arhitekturu (Slika 3.1.) deluje poprilično kompleksno. Međutim, kada se stavi fokus na implementaciju, situaciju je dosta prostija. Za sve to su zaslužne dobro uređene tehnologije koje su korišćene tokom razvoja. Nakon detaljne analize svakog dela ovog projekta, u ovom poglavnju će biti prikazani delovi implementacija grupisano po delovima projekta.

## Angular klijentska aplikacija



Slika 4.1.1 Prikaz implementacije komponente u Angularu

Na slici 4.1.1 je prikazan primer impelentacije komponente. U pitanju je komponenta ‘dashboard’ što se vidi u na vrhu slike. Na slici su paraelno prikazana 2 fajla: šablon (HTML)i klasa komponente. Na levom delu vidimo HTML kod sa ugrađenim referenciranjem na promenljive iz klase, što se navodi u dvostruim vitičastim zagradama (na levoj strani linija 15 će pokazati realnu vrednost promenljive objekta klase sa desne strane koja je deklarizovana na liniji 14).

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

Kod 4.1.2 Kod servisa koji čita REST Api u Angularu.

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);

})

);

}

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, Obervable 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 isčekuju podatke. Prilikom poziva REST APIa navedena je defenicija metod: GET; tip u koji se rezultat konvertuje: kolecija objekata tipa Coin (Coin[]); URL adresa i objekat tipa HTTPOptions u kome su definisani pomoćni delovi zahteva poziva RESTa.

@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 preserće svaki izlazni zahtev i doda u opis zahteva autorizacioni kod. Pored toga, vidimo da presreće i svaku dolaznu poruku te u slučaju greškre preusmera na login

Još jedan prezentaciono zaniljiv 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 sa čuvan u memoriji aplikacije. Ukoliko taj uslov važi, u opis 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.

## CryptOffer

*CryptOffer* sadrži nekoliko zanimlijvih delova koji se tiču konfigurisanja. Na Ko 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 mirkoservis 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 Konfigurisanje SpringBoot aplikacije CryptOffer.

Pošto ovaj servis predstavlaj **API Server** ulogu OAuth2 bezbenostnog 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 **autorizacionom serveru**.

@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

Osim konfiguracije CryptOffer sadrži jednu zanimlijvost koja je već prikazana u opisu tehnologija a toje Feign klijent. Podsetimo, Feign predstavlja način čitanja REST API-a između mirkoservisaa.

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

@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();

… }

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

*CryptOffer* kao mikroservis predstavlja cenralnu jedinicu i najčešće samo preusmerava zahteve. Međutim, jedna metoda je zanimlijva jer za dobijen jedan poziv ona upućuje 2 nova poziva. To je metoda za **realizaciju oglasa**, odnosno ostvarenja kupovine ili prodaje. U tom slučaju, *CryptOffer* dobije zahtev realizacije i podakte 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 objeksta 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 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.

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

@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;

}

## 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 nepohodno za ovaj postupak je dodavanje anotacija @EnableAuthorizationServer . Ovom anotacijom se daje do znanja *SpringBoot* da ovaj servis se predstalja autorizacioni server, što je cilj. Pored toga, neophodno je zadati konfiguracije servera, kao što su: kredencijali klijentske aplikacije, ključ (id) autorizacionog servera, dužina držanja pristpunog tokena aktivnim i dr. Navedeno konfigurisanje je prikazano takođe na okviru koda 4.3.1 unutar metode configure(ClientDetailsServiceConfigurer).

## CryptoEurekaServer

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

@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-resurce")

.accessTokenValiditySeconds(3600)

.secret(passwordEncoder.encode("crypto-secret"));  
 }

*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.

Kod 4.4.1. Jedina klasa servisa CryptoEurekaServer

@SpringBootApplication

@EnableEurekaServer

**public** **class** CryptoEurekaServerApplication {

**public** **static** **void** main(String[] a) {

SpringApplication.*run*(CryptoEurekaServerApplication.**clas,**a);

}

}

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 porazumevanoih 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.

## 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 mikroservisa

*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.

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 CrypoAdvertising

@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 porazumevane 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 antirana 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 dosupne osnovne funkcionalnosti kao što su findById(..), findAl(), 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)

## CryptoWallets

*CryptoWallets* pored svoje baze sadrži čitanje ekstarog izvor podataka. U pitanju je već pominjani API za realne cene valuta na tržištu. Za to je zadužen REST Reader čiji je interfejs prikazan u kodu 4.6.1

Kod 4.6.1 Interfejs čitača ekternog izvora podataka u okrviu CryptoWallets servisa

@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);

}

Pošto se kombinuju 2 izvora podataka, u okviru ovog mikroservisa postoji servis (kao SpringBoot komponenta) koji se bavi dobavljanjem svih podataka u i nazvan je CoinManager. Interfejs CoinManager servisa u potpunosti sakriva izvore podataka te ukoliko se posmatra samo interfejs ne moze 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 jj 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 APIa 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.

Grupisanje podataka na nivou upita je definisano u imenovanom upitu (NamedQuery) koji se definiše u entiteu 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 filtriranje podataka čija je ukupna količina 0, u cilju izbacivanja prodatih valuta.

@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 transakija u sumirane podatke jedne valute. Ovaj upit zaista pruža trenutnu realno stanje po valutama.

@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());

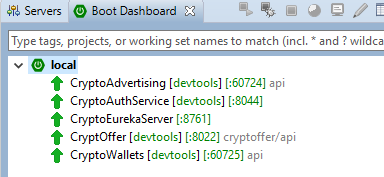
**return** **this**.myCoins;

}

Kod 4.6.2 Metoda CoinManagera koja vraća podatke za Dashboard komponentu u klijentskom delu aplikacije. Zanimlijvost 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.

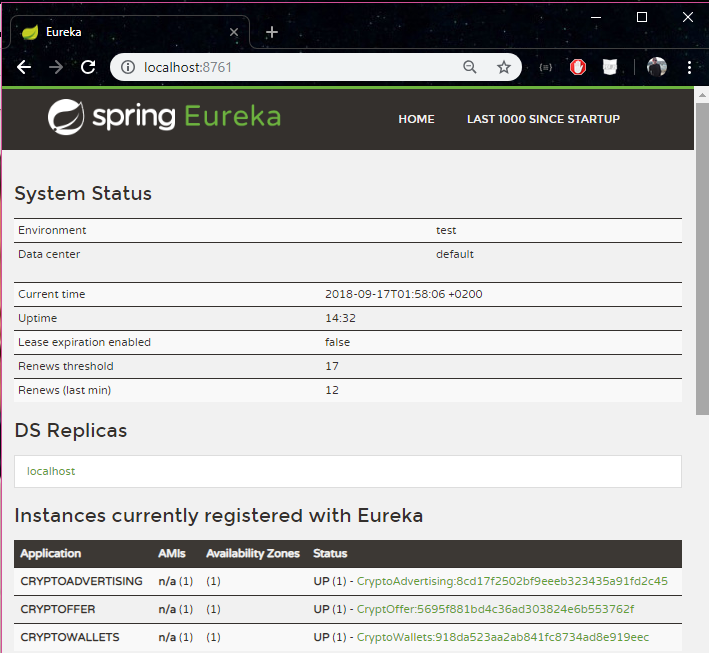
# UPOTREBA APLIKACIJA

Pokretranje 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



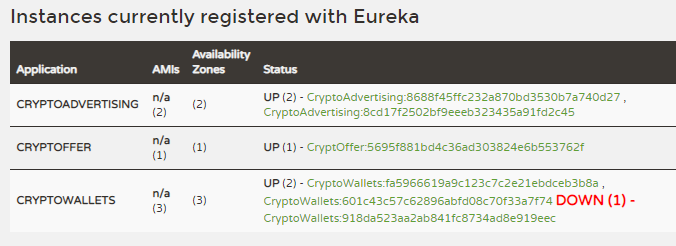
Slika ‑ Slika primera pokrenutih servisa u razvojnom okruženju. U zagradama brojčani zapis predstavlja port na kome aplikacija drži otvoren komunikacioni kanal RESTfull

Preporučivo je najpre aktivirati Eureka Server kako bi ostali mirkoserisi po pokretanju uspešno izvršili registraciju. Kada se Eureka Server podigne, u okriru 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 ovaranjem adrese pokrenute aplikacije u web-browser-u (na primeru ovaj servis je podignut na portu 8761 lokalne adrese).



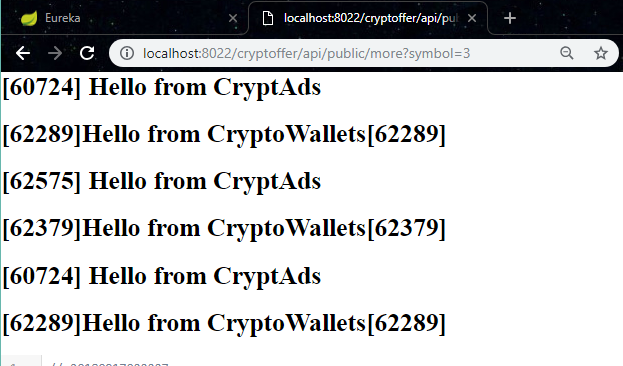
Slika .. Ugrađen web klijent eureka servera koji služi za prikaz stanja.

Slika 5.3 predstavlja tabelu registrovanih mikroservisa u slučaju kada postoji više od jednog aktivno servisa po imenu. Prvi red tabele daje do znanja da su aktivne 2 instance mirkoservisa 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).



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

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



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

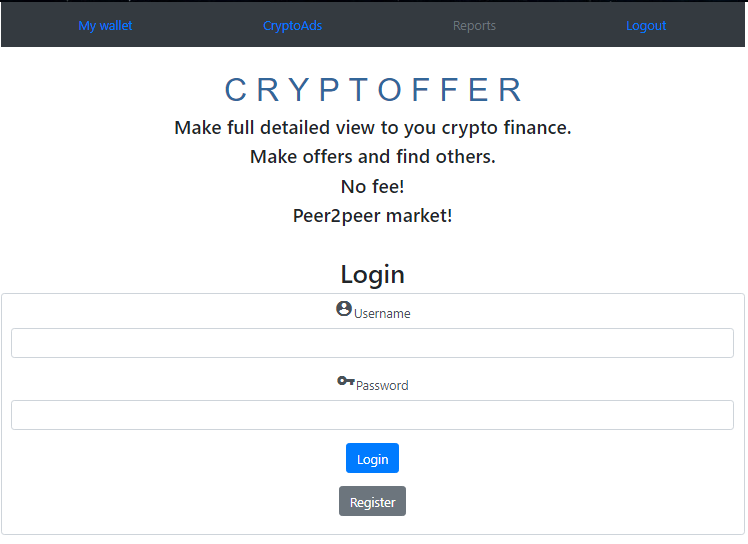
Ovaj primer je napravljen samo za namene testiranja Ribbon balansera zahteva i reporezntativnog je karaktera. Dokaz rada raspodele zahteva se ovde očito vidi. CryptoWallets šalje 3 odgovora: 1.i 3. Odgovori su od mikroservisa koji je aktivan na portu 62289 dok je 2. odgovor sa porta 62379. Situacija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

Nakon uspešnog startovanja serverskih servisa, za rad je neophodno pokrenuti i server koji pruža Klijentsku aplikaciju. Pokretanj servera za Angular aplikaciju se vrši u Command Prompt-u komandom navedenom u kodu 5.1

>ng serve

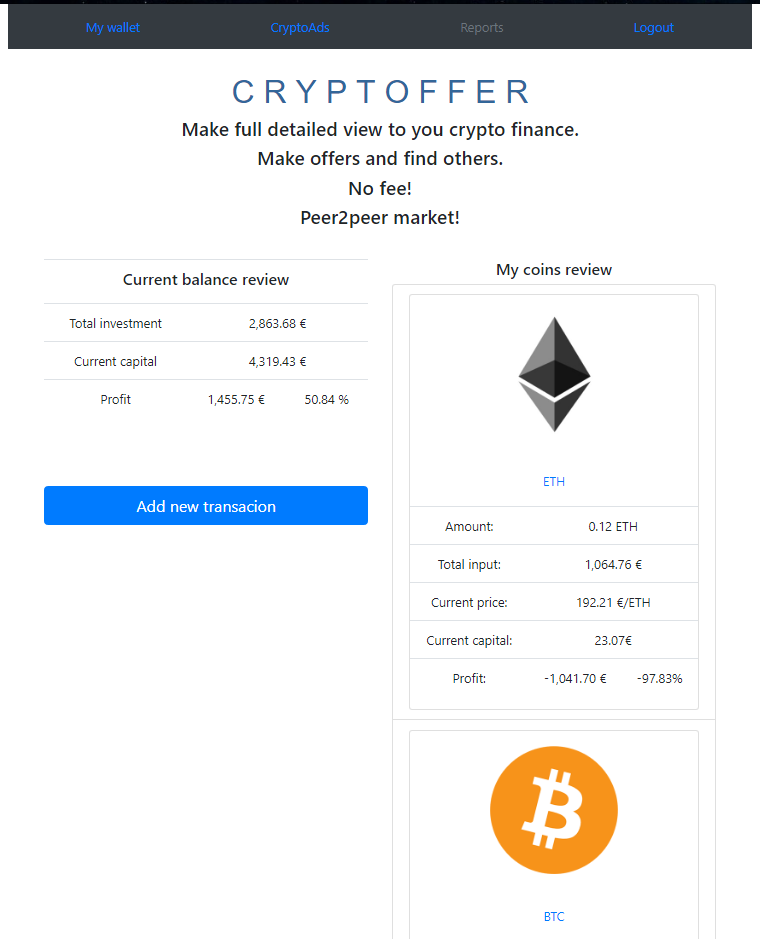
Kod 5.1. CMD comanda za pokretanj Angualar klijentske aplikacije

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



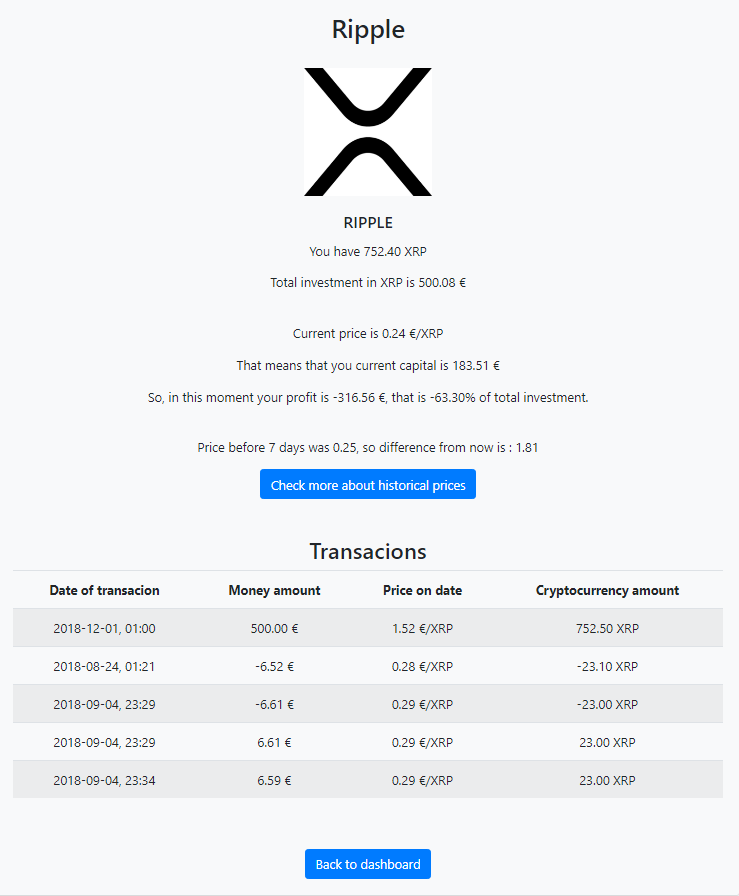
Slika 5.5 Login ekran pokrenute klijentkse aplikacije

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 je predstavlja sveukupni pokazatelj stanja kritpovaluta koje korisnik poseduje.



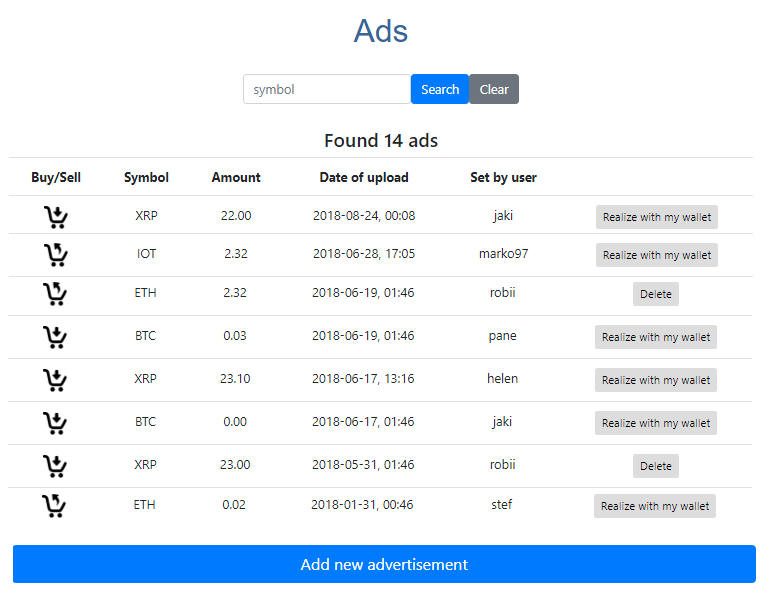
Slika 5.6. Dashboard ekran. Prikaz stanja kritpovaluta ulogovanog korisnika.

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 transakijama ove valute.

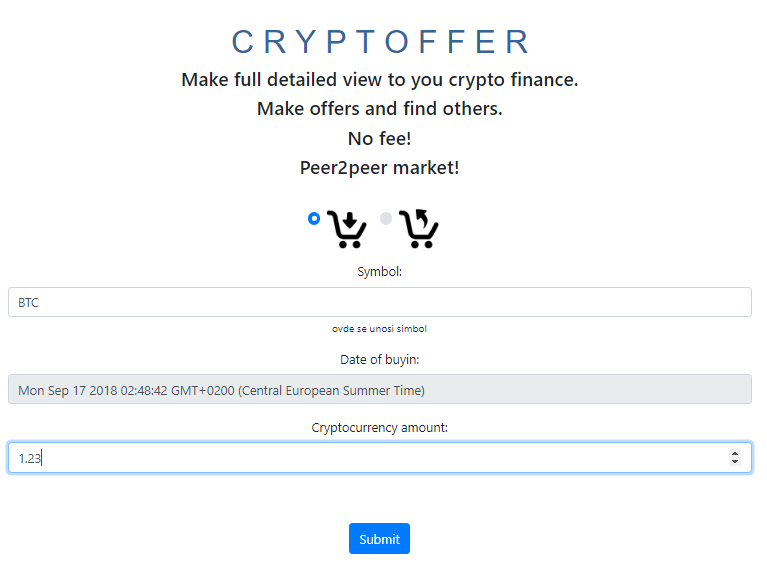


Slika 5.7. Stranica za prikaz detalja jedne kriptovalute – konkretan primer: Ripple

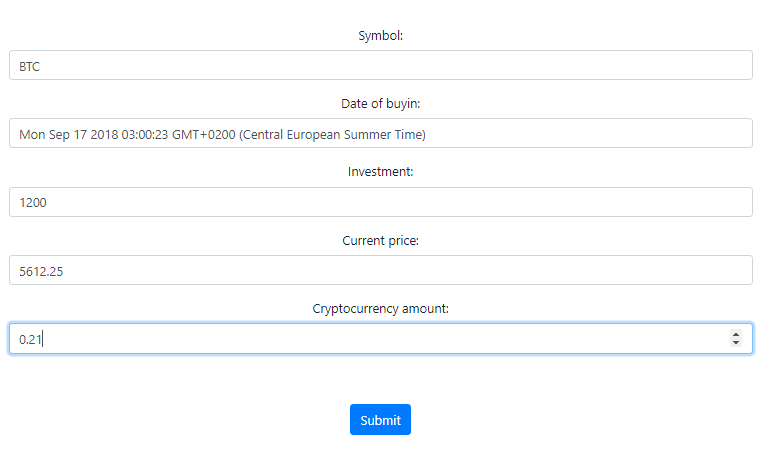
Na svim stranica je prisutan navigacioni bar koji služi za kretanje u okviru aplikacije. U navigacionom baru se nalazi dugme za prelazak na stranicu oglasa. Klikom na *CryptoAds* korisnik će biti prebačen na stranicu koja je prikazana na slici 5.8. Pretraga po valuti je implementirana serverski (ne samo 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 starne drugih korisnika, ponuđena akcija je prihvatanje i realizacija ponude.



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

Na slici 5.9. je prikazan ekran dodavanja novog oglasa. Mogućnost unosa podataka se ograničava na 3 polja: da li je u pitanju kupovina ili prodaja, symbol kriptovalute za koju se vezuje ovaj oglas i količina odabrane valute koja se oglašava. Cena se utvrđuje pri ralizaciji, dakle ta informacije je suvišna u ovoj formi.

Slika 5.9. Forma za odavanje novog oglasa

Osim forme za dodavanje oglasa, postoji i forma za unos transakcije koja je prikazana na slici 5.10.

Slika 5.10. Forma za direkan unos transakcije.

# 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 jednom 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.

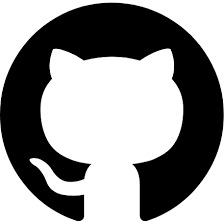
Motivacija mikroservis orijentisanog razvoja je trenutni trend u svetu informacionih tehnologija. Napredak hardverskih komponenti polako gubi na značaju zbog odnosa cene i dostignuća. Napredak softvera ima konstantne finansijkse troškove a svaki korak napredka čini osnovu za sledeći. Na taj način se 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 a radi na jednoj hardverskoj komponenti. Distribuirani (mirkoservisni) sistem je raposređen na više harverskih komponenti te se pojačanje performanski postiže integracijom nove hardervske skomponente.

Osim pravca kretanja IT-a motivacija je i modularan razvoj. Mikroservis orijentisan razvoj zahteva strogu podelu po modulim. Projektni zadatak treba dobro podeliti kako na nezavisne celine. Kada se uspešno uradi, razvoj je mnogo lakši i jednostavniji nego kod monolitnog sistema. Razlog je što se svaka komponenta pojedinačno istestirana i obrađena, 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.

# 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://spring.io/guides/gs/service-registration-and-discovery/>
8. <https://www.w3schools.com/>
9. <https://searchdatamanagement.techtarget.com>

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>

# 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-introtojava1/index.html>
4. JavaScript
   * <https://www.javascript.com/>
5. Google
   * <https://www.google.com/about>
6. HTML
   * <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_whatis.asp>
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>
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/>

1. CryptoCompare API docs
   * <https://min-api.cryptocompare.com/>
2. Reactive programing
   * <https://dzone.com/articles/5-things-to-know-about-reactive-programming>

# BIOGRAFIJA

Robert Sabo

Cara Lazara 34

Riđica 25280, Sombor

email: [robert.sabo0@gmail.com](mailto: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 Riđici zavšio 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**

Indentifikacioni broj:

**IBR**

Tip dokumentacije: Monografska

**TD** dokumentacija

Tip zapisa: Tekstualni

**TZ** štampani materijal

Vrsta rada: Završni rad

**VR**

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

**GО**

Izdavač: Autorski reprint

**IZ**

Mesto i adresa: Prirodno – matematički fakultet, Trg

**MA** Dositeja Obradovića 4, Novi Sad

Fizički opis rada: 8 poglavlja, 79 strana,

**FO** 17 slika, 26koda

Naučna oblast: Informatika

**NO**

Naučna disciplina: Web Aplikacije

**ND**

Predmetna odrednica/ Angular, Sprig Boot

Ključne reči: Microsevice, Skalabilnost

**PO**

**UDK:**

Čuva se: Biblioteka Departmana za matematiku

**ČU** i informatiku, Novi Sad

Važna napomena: Nema

Izvod

**IZ**: Opis aplikacije za rezervaciju restorana

Datum prihvatanja teme:

**DТ**

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**

|  |
| --- |
| **KЕY WORDS DOCUMENTATION** |

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: Microservice oriented system

**AB** 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, menthor

**3.** PhD Mirjana Mikalački, associate professor, Faculty of Sciences, Novi Sad, member