

ELEKTRONSKI FAKULTET NIŠ

STRUČNA PRAKSA

Izveštaj o realizaciji stručneprakse

Nemanja Milošević 16216

Niš, Jun 2020.

Sadržaj

[**Uvod** 3](#_Toc44807646)

[**Baza podataka** 3](#_Toc44807647)

[Entiteti 4](#_Toc44807648)

[**JOOQ** 5](#_Toc44807649)

[**BookMaker Klijent** 6](#_Toc44807650)

[**Serverska strana aplikacije** 7](#_Toc44807651)

[**Arhitektura sistema(dataflow)** 8](#_Toc44807652)

[**Gameloop** 9](#_Toc44807653)

[Kafka consumer-i 9](#_Toc44807654)

[Generisanje podataka 10](#_Toc44807655)

[Simulacija sezone 11](#_Toc44807656)

[**HTTP feedpusher** 13](#_Toc44807657)

[**Spring Security, OAuth2, JWT** 15](#_Toc44807658)

[Spring Security 15](#_Toc44807659)

[OAuth2 15](#_Toc44807660)

[JWT token 16](#_Toc44807661)

[**Angular Klijent** 16](#_Toc44807662)

[**Zaključak** 19](#_Toc44807663)

# **Uvod**

Na početku treba napomenuti da detalji implementacije sa prakse neće biti opisani, zbog potpisanog NDA ugovora. Biće opisani alati, okruženja i osnovna struktura projektovanog softvera i ideja iza nje. Sama implementacija neće biti prikazana. Praksa je odrađena u firmi DualSOFT u Nišu, i to u periodu od 15.03.2020. do 15.06.2020. Firma je osnovana 2003. godine i bavi se pravljenjem i projektovanjem informacionih sistema vezanih za sportsko klađenje. Kada je reč o softveru koristi se programski jezik JAVA, odnosno Spring Framework kao glavna tehnologija na serverskoj strani sistema, Kafka I RabbitMQ kao message brokeri, Cassandra, Redis I PostgreSQL kao tehnologije vezane za skladištenje podataka. Front End koristi Angular, Ionic I Thymeleaf Framework-e za prikaz podataka pored Bootstrap-a I CSS-a za dizajn.

Za programiranje na programskom jeziku JAVA potrebno je instalirati Javu (jre I jdk), a za korišćenje Spring Framework-a moramo instalirati MAVEN na mašini. Zato što svi Spring projekti se generišu kao MAVEN projekti.

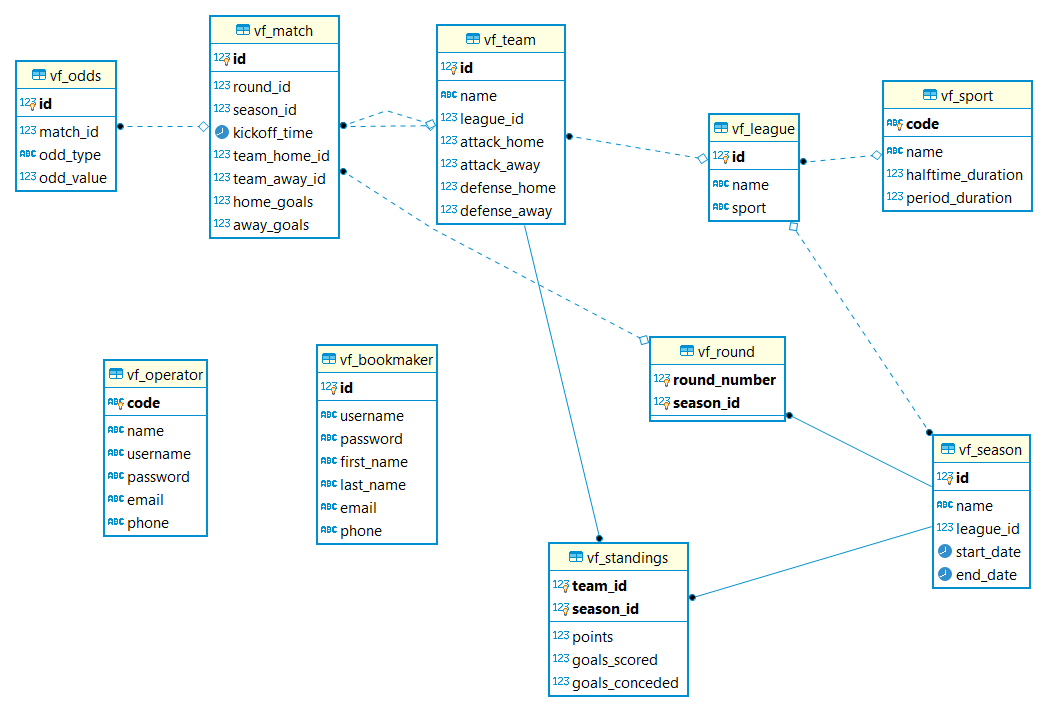
Za korisćenje Kafke I RabbitMQ-a moramo imati instalirane te brokere ili na localhost Ili na remote serveru, kod nas te komponente smo imali na remote serveru. Takodje treba dodati da je za korišćenje Angular-a potrebno instalirati Node na mašini.

Pošto sam objasnio koji stek tehnologija, i nešto osnovno o njegovim komponentama, je korišćen za izradu našeg projekta, treba reći nešto osnovno o samom projektu. Naš zadatak je bio da napravimo platformu zvanu VirtualFeed koja ima zadatak da simulira tok neke sportske lige tako što adminski deo platforme dodaje timove, lige sportove u sistem koji kasnije po samo-generisanom rasporedu stvara sezone, kola, utakmice I rezultate tih utakmica. Uz sve to sistem mora obaveštava zainteresovane klijente za događaje(postignut gol, kraj meča, početak nove runde ...). Takodje platforma šalje samo-generisane kvote koja je izračunala za svaki meč koji uskoro počinje. Detaljnije o samom projektu i implementaciji biće više reči kasnije.

# **Baza podataka**

*(dan 1. – dan 5.)*

Korišćena je PostgreSql baza podataka(verzija dependency-ja je 42.2.11) za skladištenje podataka o virtuelnim sportovima, ligama, sezonama, rundama, timovima, mečevima, kvotama, kao i samoj poziciji tima na tabeli(standings). Takođe uz ovu bazu je korišćen i JOOQ (Java Object Oriented Querying) za automatsko generisanje koda na osnovu šeme baze podataka(generisanje pojo, tables i records klasa adekvatnih za bazu podataka) o kojem će više reči biti u nastavku u narednom poglavlju. ). Sam ER dijagram baze podataka prikazan je na narednoj slici, a u nastavku je dat i detaljniji opis svakog od entiteta u bazi podataka.



*Slika1. ER dijagram baze podataka*

Entiteti:

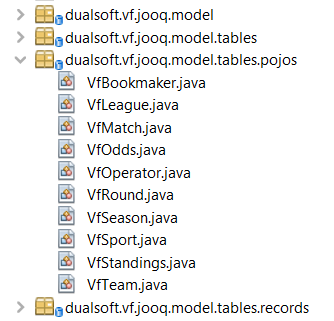
* VF\_SPORT(code, name, halftime\_duration, period\_duration) – za čuvanje podataka o konkretnim virtuelnim sportovima
* VF\_LEAGUE(id, name, sport) – za čuvanje podataka o konkretnim virtuelnim ligama
* VF\_TEAM(id, name, league\_id, attack\_home, attack\_away, defense\_home, defense\_away) – za čuvanje podataka o konkretnim timovima i njihovim napadačkim i odbranbenim koeficijentima korišćenih pri Poasonovoj distribuciji
* VF\_SEASON(id, name, league\_id, start\_date, end\_date) – za čuvanje podataka o konkretnim virtuelnim sezonama
* VF\_ROUND(round\_number, season\_id) – za čuvanje podataka o konkretnim rundama za određenu virtuelnu sezonu
* VF\_MATCH(id, round\_id, season\_id, kickoff\_time, team\_home\_id, team\_away\_id, home\_goals, away\_goals) – za čuvanje podataka o konkretnom meču u okviru određene runde konkretne virtuelne sezone
* VF\_STANDINGS(team\_id, season\_id, points, goals\_scored, goals\_conceded, rank)- za čuvanje podataka o učinku jednog tima za konkretnu virtuelnu sezonu u pogledu postignutih/primljenih golova, osvojenih poena, ranku na tabeli
* VF\_ODDS(id, match\_id, odd\_type, odd\_value)- za čuvanje podataka o kvotama koje su u ponudi za konkretne virtuelne mečeve
* VF\_OPERATOR(code, name, username, password, email, phone) – za čuvanje podataka o konkretnim kladionicama/operatorima
* VF\_BOOKMAKER(id, username, password, first\_name, last\_name, email, phone) – za čuvanje podataka o konkretnim bookmakerima

# **JOOQ**

*(dan 5. – dan 10.)*

Uz navedenu PostgreSql bazu podataka korišćen je i popularni alat JOOQ(Java Object Oriented Querying) za automatsko generisanje koda na osnovu šeme baze podataka(generisanje pojo, tables i records klasa adekvatnih za bazu podataka). Korišćena verzija je JOOQ 3.13.1.

*Slika2. Specifikacija JOOQ code generator plugin-a*

- Na gornjim slikama je prikazana specifikacija jooq code generator plugin-a u pom.xml fajlu projekta korišćena za generisanje paketa vf.jooq.model, vf.jooq.model.tables, vf.jooq.model.tables.pojos, vf.jooq.model.tables.records. Za potrebe projekta prevashodno su korišćene pojo generisane klase(VfBookmaker.java, VfLeague.java, VfMatch.java, VfOdds.java, VfRound.java, VfSeason.java, VfSort.java, VfStandings.java, VfTeam.java) koje predstavljaju klase koju su objektno orjentisano preslikavanje entiteta u PostgreSql bazi podataka. Ove klase su prikazane na slici pored.

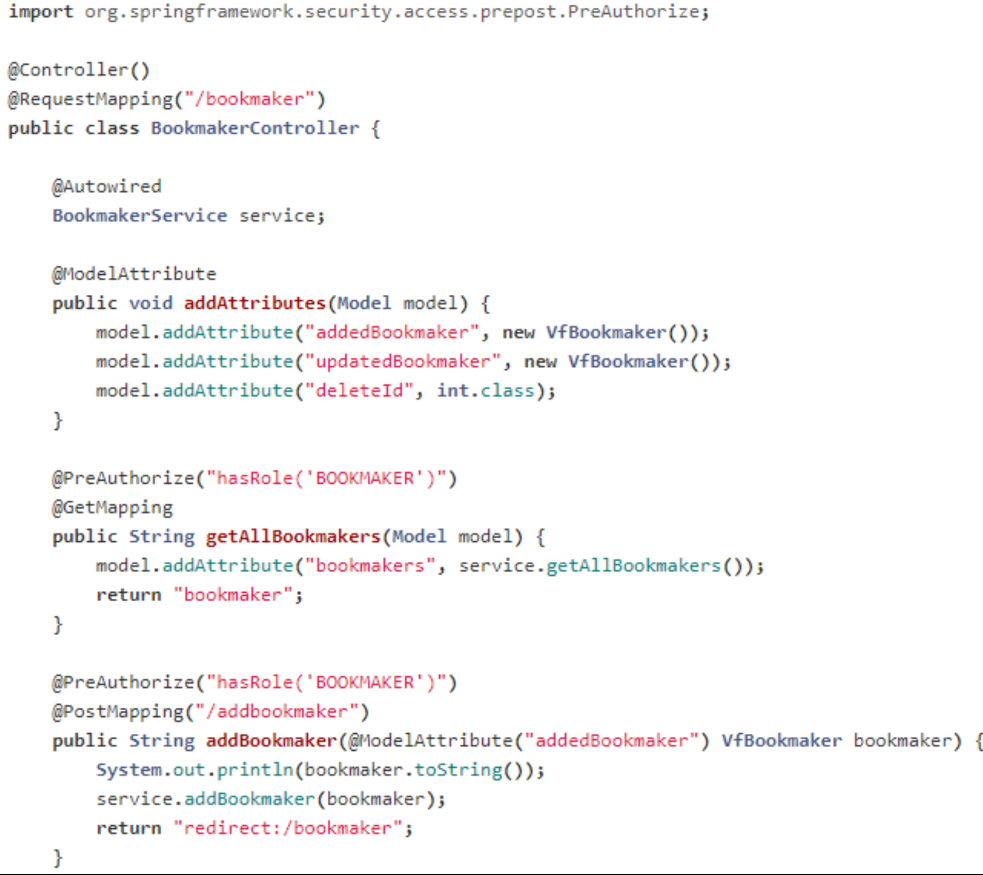
*Slika3. JOOQ generisane pojo klase*

# **BookMaker Klijent**

*(dan 10. – dan 35.)*

Bookmaker mikroservis obavlja funkciju adminskog dela platforme koji nudi mogućnosti dodavanja novog sporta, lige, tima, … Takođe Bookmaker mikorservis poseduje Front End koji služi za unos tih podataka. Ovaj prikaz podataka obezbeđujemo pomoću biblioteke ThymeLeaf. Koncept ovog mikroservisa je jednostavan: Radi na principi klasičnog MVC modela gde se View deo piše u ThymeLeaf-u, a Controller deo predstavljaju Spring mikroservisi kucani u Javi, Model predstavljaju JOOQ autogenerisane klase na osnovu tabela projektovanih u PostgreSQL Bazi.

Detaljnije gledano, svaki url I endpoint se slika u zasebni Spring Controller koji obavlja određenu funkciju i njemu je dodeljena jedan View čije evente on obrađuje. Za malo lepši prikaz od klasičnog HTML-a koristili smo Bootstrap biblioteku i time ulepšali taj deo platforme.



*Slika4. Primer Controller-a(BookmakerController)*

I tako se obezbeđuje da se View u ThymeLeaf-u poveze sa Spring Controller-om I time obezubedi funkcionalnosti koje taj View prikazuje. Vrlo često posao ovih rest kontrolera je da na osnovu unetih podataka generišu neku POJO mapper klasu koju upisujemo u neku bazu I o tome eventualno obavesti ostale mikroservise platforme (Game Loop mikroservis, Operator mikroservis...) bilo preko slanja novog http zahteva prema njima ili češće slanjem poruke u Kafka Topic koji osluškujumikroservisi zainteresovani za tu poruku.

Treba napomenuti da bookmaker mikroservis zajedno sa svojim Front End-om može stopirati ili ponovo pokrenuti neku generisanu sezonu određene lige, kao I da za sve ovo je potrebna odgovarajuća autentifikacija na neki od bookmaker (admin) profila što čemo detaljnije objasniti u nastavku.



*Slika5. Isečak Thymeleaf koda korišćenog za vizuelizaciju u BookMaker klijent mikroservisu*

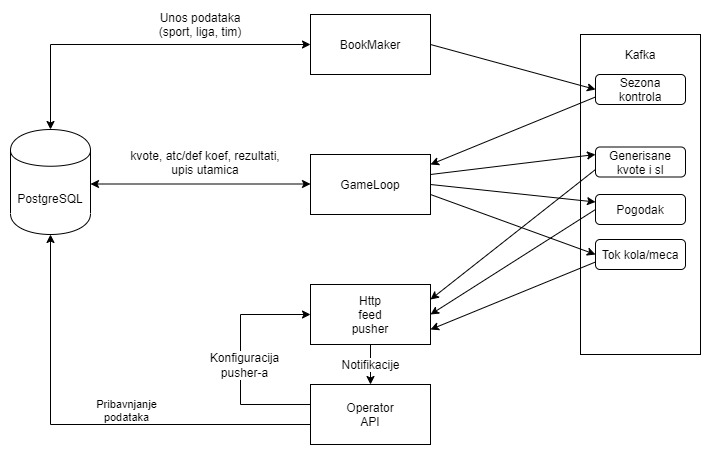
# **Serverska strana aplikacije**

*(dan 35. – dan 70.)*

Serverska strana aplikacije, pored servisa za autorizaciju, sadrži još dva mikroservisa koji obezbeđuju svu potrebnu funkcionalnost aplikacije. Servisi su implementirani korišćenjem spring boot framework-a, dok se komunikacija mikroservisa obezbeđuje korišćenjem više apache kafka topic-a, a veza sa ostatkom sistema se održava preko http feed-a. U nastavku ćemo detaljnije opisati rad i protok podataka kroz sistem. Konkretno kroz dva mikroservisa – Gameloop i Http-feed-pusher, koji čine srce sistema.

# **Arhitektura sistema(dataflow)**

*(dan 35. – dan 40.)*

*Slika6. Arhitektura sistema(dataflow)*

Arhitektura sistema prikazana je na prethodnoj slici. Nakon unosa podataka od strane korisnika, preko interfejsa koji pruža Bookmaker, korisnik ima opciju da pokrene jednu ili više sezona. Izdavanja tražene komande šalje se poruka kojom se inicira početak date sezone. Poruka se šalje na topic u okviru apache kafke koji prima poruke za kontrolu sezone. Na isti topic se, ukoliko korisnik zada takvu komandu šalju i poruke za završetak i pauziranje sezone. Nakon toga na topic-u se nalazi poruka koja sadrži sve podatke potrebne za pokretanje (završetak ili pauziranje) tražene sezone.

U okviru gameloop mikroservisa nalazi se kafka listener koji proverava da li je stigla poruka za početak sezone. Listener se pokrece u posebnoj niti, kako bi se omogućilo da i nakon čitanja poruke ostatak Sistema nastavi da radi. Prijem poruke za početak sezone inicira generisanje mečeva, kvota, kao i tabele za tekuću sezonu (detaljniji opis rada ovog dela sistema biće dat u nastavku). Kada se završi generisanje svih potrebnih podataka na topic u okviru kafke salje se odgovarajuća poruka, čime se ostatak sistema obaveštava da su podaci generisani i da simulacija utakmica u okviru kola može da počne.

Poruka kojom se sistem obaveštava da je generisanje podataka završeno, sa topic-a čita novi kafka listener, koji se takođe, iz razloga navedenih ranije, pokreće u zasebnoj niti. Nakon prijema poruke može se otpočeti sa simulacijom sezone. Samo generisanje rezultata mečeva i kvota odvija se u zasebnoj niti. Ovim se postiže to da gameloop mikroservis može istovremeno da pokreće i simulira više nezavisnih sezona.

Obaveštenja o kvotama, golovima, početku i kraju meča i poluvremenu šalju se na kafka topic, odakle ih čitaju kafka consumer-i koji se nalaze u okviru http-feed-pusher mikroservisa. Zadatak http-feed-pusher servisa je da prima poruke koje generiše gameloop i dalje ih prosleđuje krajnjim korisnicima. U nastavku ćemo detaljnije objasniti rad najvažnijih delova gore navedenih mikroservia i način na koji oni obezbeđuju traženu funkcionalnost sistema.

# **Gameloop**

*(dan 40. – dan 60.)*

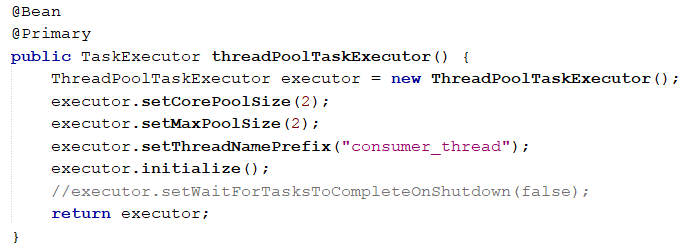
Prilikom komunikacije sa postgresql bazom podataka, koja sadrži sve podatke koje aplikacija koristi tokom rada, koristi se objektno orjentisani mapper Jooq koji značajno olakšava rad sa podacima iz baze. Jooq vrši mapiranje podataka koji se nalaze u okviru tabela baze u odgovarajuće modele koji će se koristiti tokom rada sa podacima. Jooq generiše, kako modele koji se koriste za lakše pisanje sql upita, tako i pojo modele koji se koriste u okviru aplikacije.

## Kafka consumer-i

*(dan 40. – dan 47.)*

U okviru gameloop mikroservisa nalaze se dva kafka consumera, zadužena za prijem poruka kojima se vrši kontrola sezone i poruka kojima se pokreće simulacija date sezone. Oba consumer-a se pokreću u okviru zasebnih niti, čime se obezbeđuje responsivnost sistema u svim situacijama. Ovo je omogućeno kreiranjem thread pool-ova koje obezbeđuje spring boot framework.

Primer koda koji obezbeđuje dva threada za osluškivanje odgovarajućih Kafka topic-a :



*Slika7. Kod koji obezbeđuje dva thread-a za osluškivanje odgovarajućih Kafka topic-a*

Implementacija jednog Kafka consumera prikazana je na sledećoj slici.

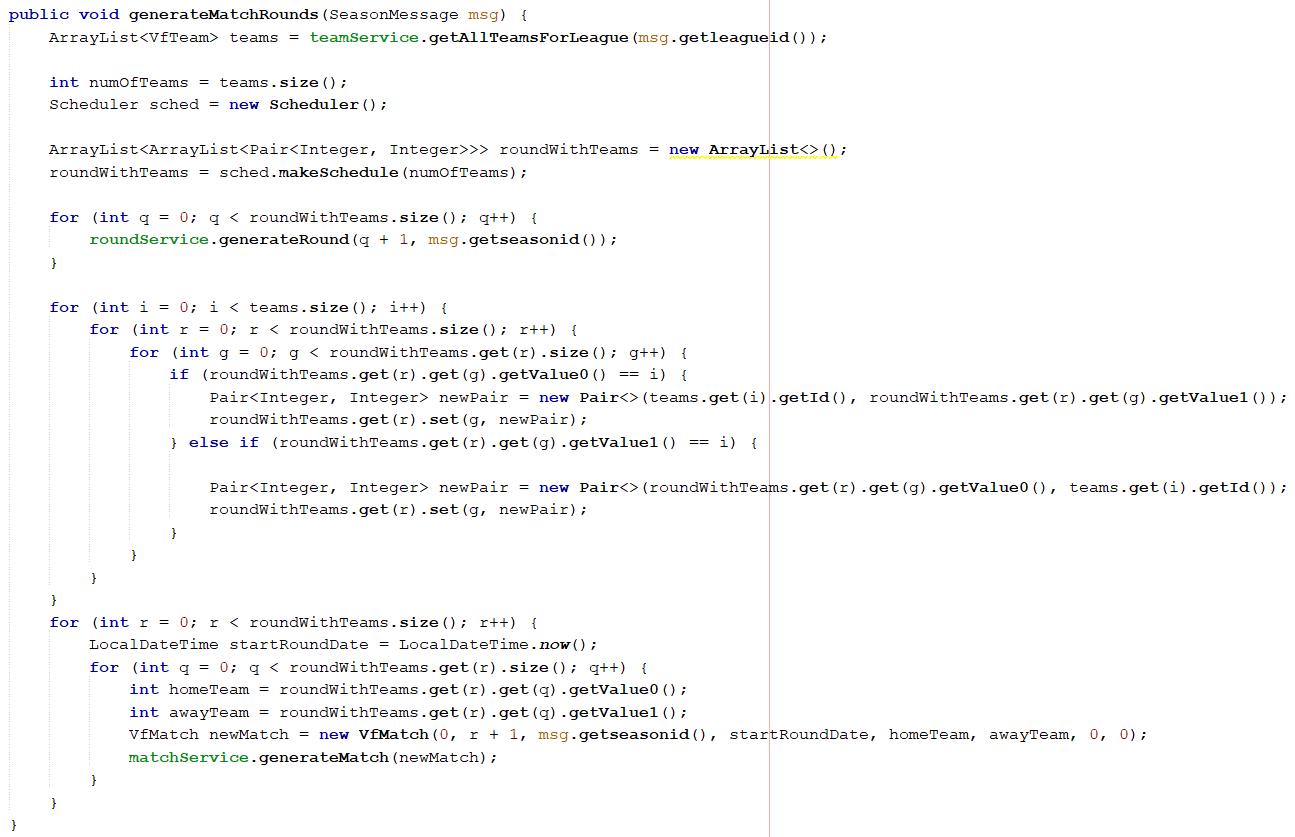


*Slika8. Implementacija Kafka consumera*

## Generisanje podataka

*(dan 47. – dan 53.)*

Kada gameloop servis primi poruku za početak sezone treba da se napravi odgovarajući raspored utakmica po kolima za odgovarajući broj ekipa u ligi. MatchRoundGenerator klasa uzima odgovarajuće podatke iz baze za novozapočetu sezonu i kreira minimalan broj kola potreban da se odigraju sve utakmice (home-away, away-home). Nakon toga kreira se stablo odluke pomoću koga se nalaze odgovarajuće kombinacije utakmica koje će biti kreirane tako da bude najmanje pauziranja ekipa po kolima. Nakon kreiranja rasporeda utakmice se upisuju u bazu podataka bez konačnog rezultata i kvoti. U Kafku se upisuje poruka da je kreiran raspored za sezonu koju treba da pročita drugi servis koji će za svaku od utakmica izračunati kvote i konačne rezultate.

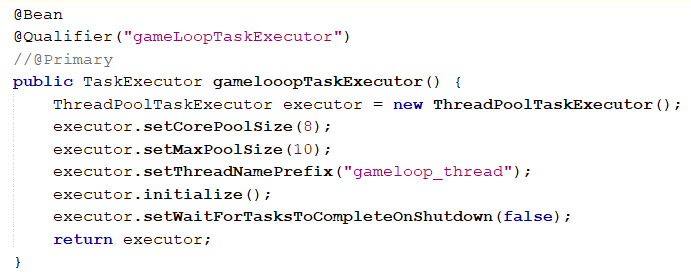


*Slika9. Generisanje podataka u klasi MatchRoundGenerator*

## Simulacija sezone

*(dan 53. – dan 60.)*

Nakon prijema poruke kojom se sistem obaveštava da je generisanje podatak završeno pokreće se simulacija sezone. Korišćenjem thread pool-a simulacija se pokreće u okviru zasebne niti.



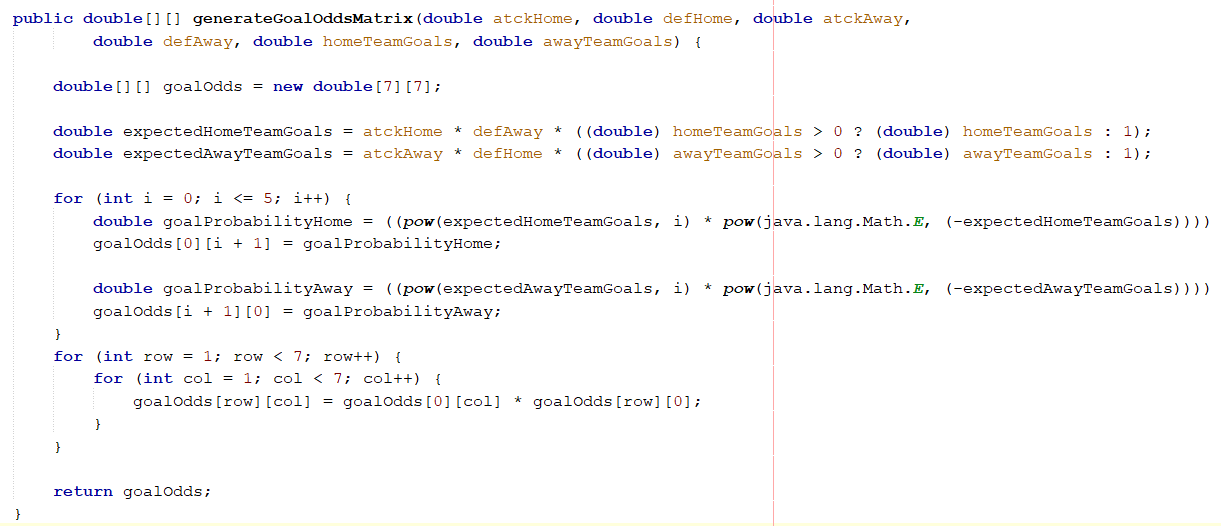
*Slika10. Primer koda koji obezbeđuje 10 threada koji će kreirati sezonu*

Simulacija sezone se vrši u okviru RoundCalculatorService klase. S obzirom da se sezona sastoji iz više kola, i da se za generisanje rezultata i kvota za svako od njih koriste podaci iz prethodnih kola, kvote i rezultati za mečeve se generišu na početku svakog kola. Generisanje se vrši istovremeno za sve mečeve u okviru istog kola, i potom se vrši slanje poruka koje dalje obrađuje http-feed-pusher mikroservis. Na kraju svakog kola nit na kojoj se sezona simulira pauzira se na određeni period pre nego što se krene na simulaciju narednog kola.

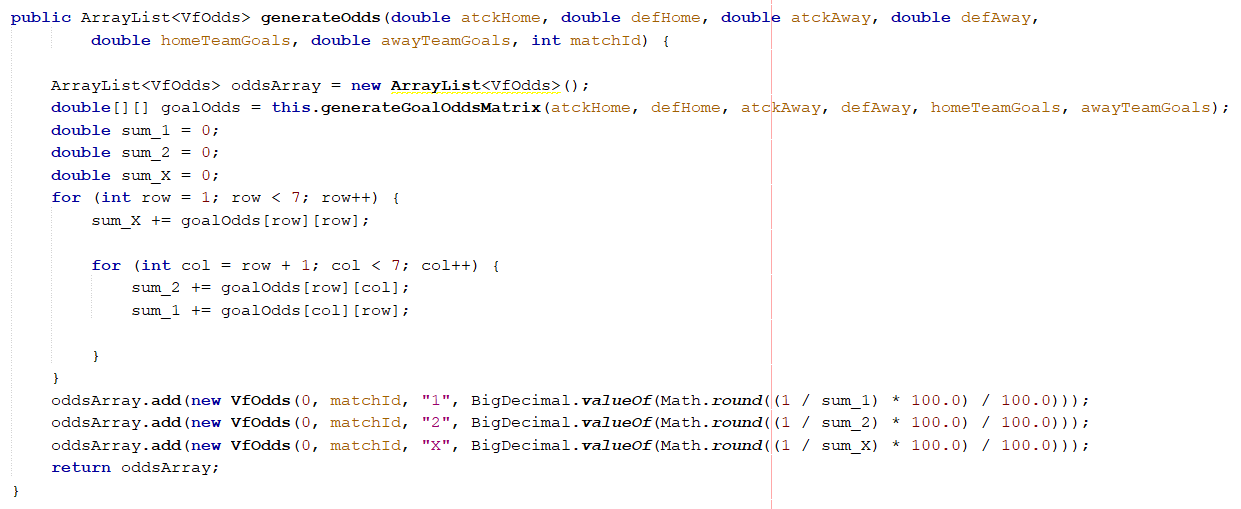
Kvote i rezultati mečeva se određuju na osnovu Puasonove raspodele. Izračunavanja se obavljaju u okviru OddsGenerator klase, koja na osnovu podataka o snazi napada i odbrane za oba tima i na osnovu broja golova koji su postizali određuje verovatnoće, a kasnije i kvote za ishode meča – pobedio domaćin, pobedio gost i nerešeno. Korišćenjem već određenih verovatnoća za različite ishode meča, vrši se odabir jednog od rezultata i on se koristi kao krajnji rezultat meča.

Kada se završi sa generisanjem kvota i rezultata kontrola se vraća RoundCalculatorService klasi, koja inicira slanje poruke na kafka topic na koji se šalju poruke o događajima na meču, kao i poruke koje sadrže kvore za mečeve u okviru datok kola. Samo slanje poruka obavljaju odgovarajući kafka producer-i koji su implementirani u okviru klasa MatchNotificationProducer i OddsProducer.

RoundCalculatorService simulira kolo tako što poruku koja sadrži kvote šalje na početku kola, nakon čega napravi pauzu u trajanju koje određuje administrator sistema. Nakon što pomenuta pauza istekne nastavlja se sa simuliranjem mečeva i golova na svakom od njih tako što se poruke koje sadrže obaveštenje o događaju rasporede u okviru predefinisanog vremenskog perioda (npr 90 sekundi), čime se stvara privid da se meč odigrava uživo. Vreme postignutog gola na utakmici se odredjuje nasumično tako da se na kraju utakmice zadovolji izračunat rezultat koji je upisan u bazi.



*Slika11. Primer generisanja matrice verovatnoća za golove na utakmici*



*Slika12. Primer koda za generisanje kvota*



*Slika13. Primer koda za simulaciju jednog kola*

# **HTTP feedpusher**

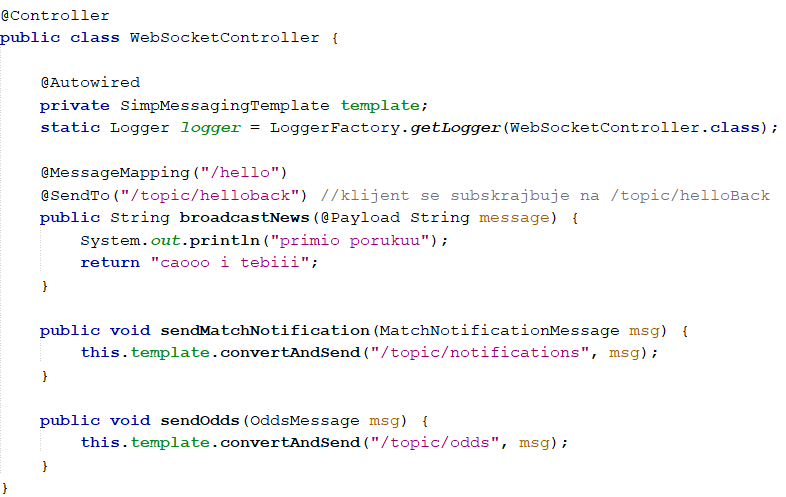
*(dan 60. – dan 70.)*

Http feedpusher je servis koji prima poruke sa Kafke o kvotama za utakmice i događajima na tim utakmicama koje nakon primanja formatira u prikladan oblik za prikaz korisniku. Nakon formatiranja poruke njegov zadatak je da te poruke prosledi svim krajnjim korisnicima koji su subscribe-ovani na odgovarajuće poruke.

Ovaj servis je izvdojen kao poseban zbog potrebe za horizontalnim skaliranjem. Postoje više instanci HTTP feedpushera koje se nalaze na posebnim adresama. Krajnji korisnik prilikom pretplaćivanja na neki topic uspostavlja vezu sa jednim od HTTP feedpushera preko rest kontrolera koji prilikom analize opterećenja svih HTTP feedpushera vraća adresu onog koji je najmanje opterećen. Nakon ovog koraka klijent obavlja handshake protokol sa odgovarajućom instancom feedpushera koji se nalazi na dobijenog adresi. Uspostavlja se websocket komunikacija preko koje feedpusher salje odgovarajuće poruke svim klijentima.



*Slika14. Primer koda za konfiguraciju websocket komunikacije*



*Slika15. Primer koda za uspostavljanje komunikacije klijenta i feedpushera*

# **Spring Security, OAuth2, JWT**

*(dan 70. – dan 80.)*

## Spring Security

*(dan 70. – dan 72.)*

Za Bookmaker mikroservis korišćen je **Spring Security** kao visoko prilagodljiv framework za autentifikaciju i kontrolu pristupa. Kontrola pristupa je dozvoljena samo prijavljenim korisnicima koji imaju @PreAuthorize("hasRole('BOOKMAKER')"), tj. dodeljenu ulogu Bookmakera.

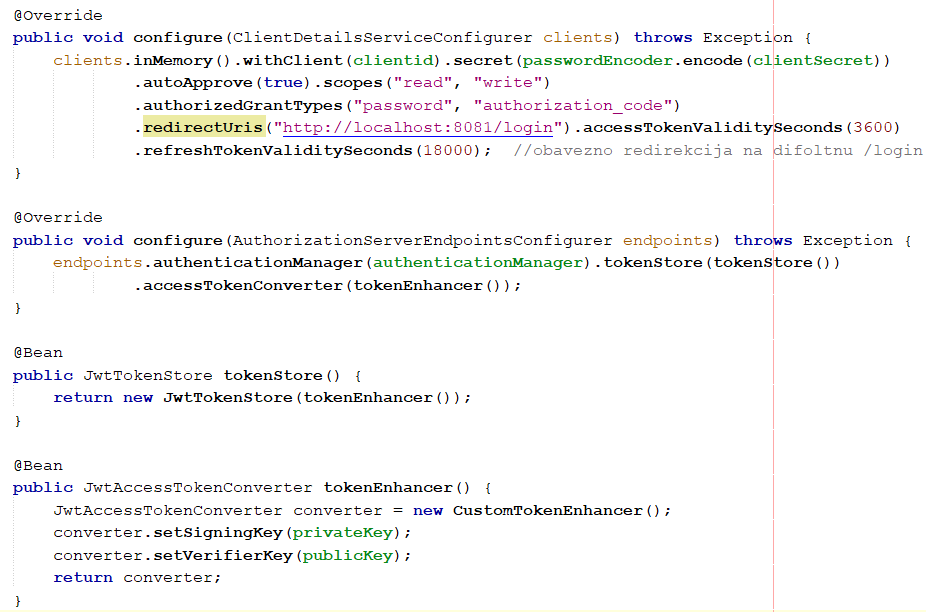


*Slika16. Security konfiguracija na strani autorizacionog servera*

## OAuth2

*(dan 72. – dan 76.)*

Sama autorizacija je rađena preko zasebnog nezavisnog autorizacionog servisa, korišćenjem najnovijeg **OAuth2** autoriizacionog frameworka i posredstvom izdavanja JWT tokena za uspešno autentifikovane korisnike.



*Slika17. OAuth2 konfiguracija na strani autorizacionog servera*

## JWT token

*(dan 76. – dan 80.)*

Home page na strani Bookmaker klijenta je dozvoljen svim korisnicima, dok sve ostale operacije zahtevaju prijavljivanje korisnika, koje se vrši redirekcijom na autorizacioni server gde se obavlja logovanje i za uspešno ulogovane korisnike izdaje se **JWT** **token**(kastomizovan tako da sadrži u sebi i informacije o logovanom korisniku) koji se čuva u HTTP cookie-ju na Bookmaker strani i traje podešenih 60min(expiration time), nakon toga se zahteva ponovno logovanje korisnika. Bookmakerima je u bazi podataka dodeljena uloga ROLE\_BOOKMAKER, pa je u Controller-ima Bookmakera setovan zahtev za autorizaciju kod onih request-a kod kojih je to neophodno (@PreAuthorize("hasRole('BOOKMAKER')")). Za passworde se koristi BCryptPasswordEncoder, kako se passwordi u bazi ne bi čuvali kao plain text.



*Slika18. Kastomizovan JWT token koji autorizacioni server izdaje po logovanju autentifikovanog korisnika*

# **Angular Klijent**

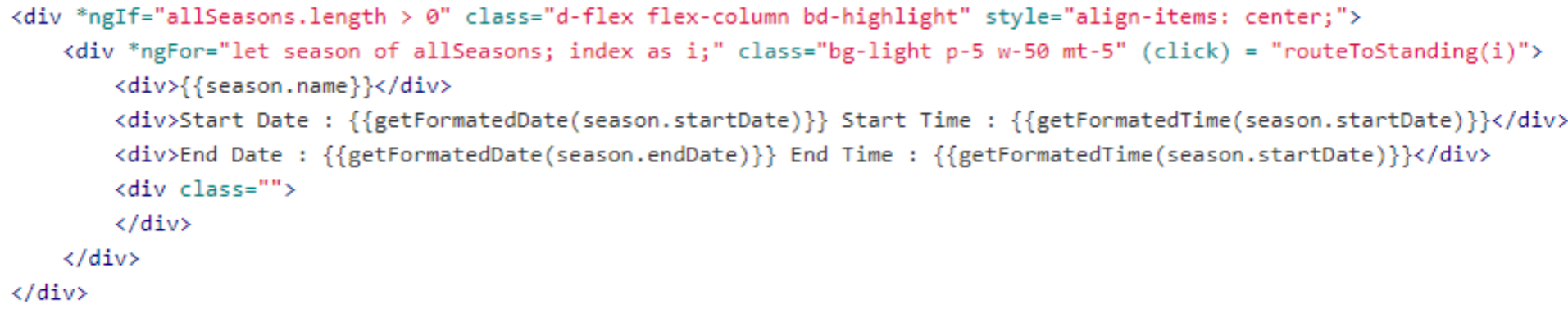
*(dan 80. – dan 90.)*

Naša platforma obezbedjuje I klasičnog WEB Angular Klijenta koji prikazuje trenutna obaveštenja I događaje vezane za naše virtulene lige. Isto tako nudi I tabelarni prikaz kvota za trenutno aktivne utakmice I sezone.

Kao I svaka Angular staranica I naša se sastoji od komponenti, klijentskih modela I servisa za komunikaciju I slanje/primanje podataka. Ovaj Framework je MVC orijentisan gde svaka komponenta ima svoj Controller fajl koji kontroliše evente I podatke koji se prikazuju na View fajlu. Tip podataka koji se prikazuju su često neke vrste klijentskih klasa pisane u jeziku TypeScript. Controlleri za svaku komponentu su zasebna klasa koja kontrolise HTML fajl koji odredjuje kako ce se ta komponenta I prikazati korisniku odnosno sadrzi HTML kostur I vrlo često kod za dizajn.



*Slika19. Primer kontrolora angular komponente*



*Slika20. Primer HTML view dela komponente*

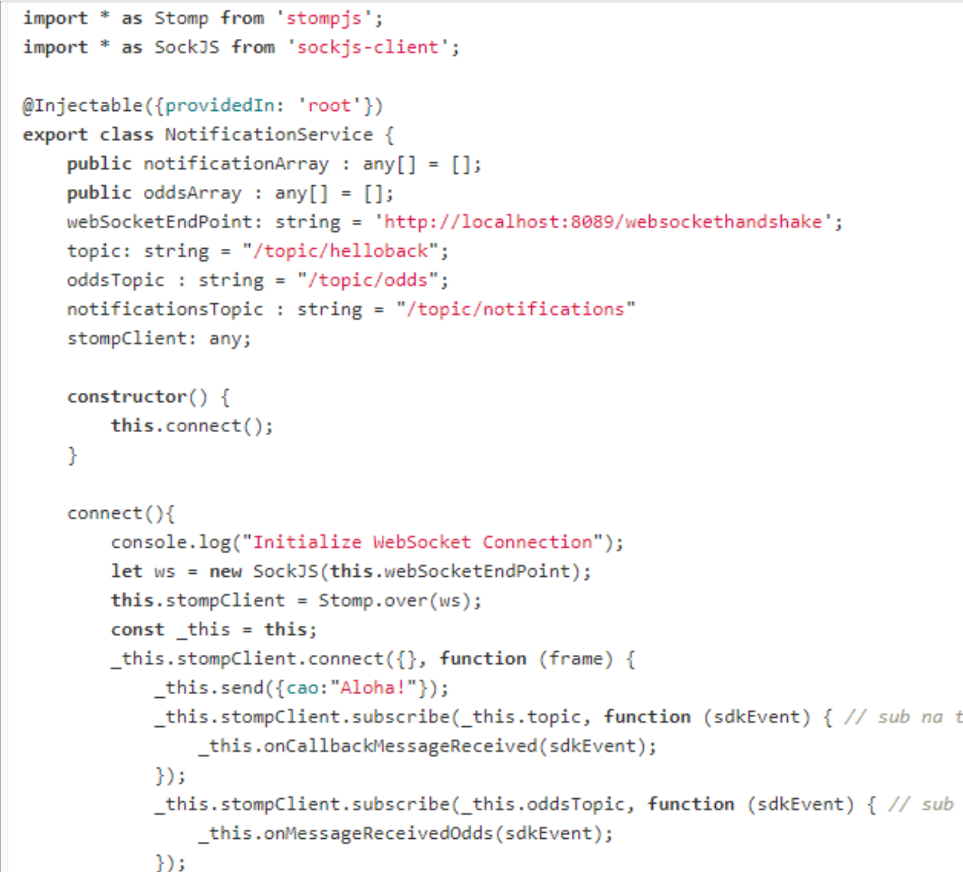
Naš klijent poseduje I servise koji služe da pribave podatke koji su aktuelni da bi ti podaci I bili prikazani. VirtualFeed klijent može podatke pribaviti sinhrono, odnosno putem request/response mehanizma takozvane klijent-server komunikacije I asinhrono osluškujući na Web Socket notifikacije od odgovarajućeg mikroservisa. Oba mehanizma se postižu stvaranjem Angular Servisa koji upućuju HTTP GET/POST zahtev za podacima na neki REST Spring mikroservis I time sinhrono dobavljaju podatke. Da REST Spring mikroservis koji opslužuje klijentske zahteve ne bi postao usko grlo sistema implementovali smo Load Balancer gde svaki klijent pre nego da se ucita pita za gateway mikroservis za IP adresu najmanje opterećenog REST mikroservisa koji ce biti zadužen da opslužuje taj WEB klijent.

Dok Klijent radi, on drži otvorenu konekciju sa Web Socket servisom koji mu u bilo kom trenutku može poslati novu asinhronu notifikaciju, a isto tako na zahtev korisnika može se pokrenuti zahtev za određenim podacima koji ce Angular servis da dispečuje preko HTTP zahteva odgovarujećem REST data mikroservisu.

Dizajn našeg Web klijenta radjen je pomoću nativnog CSS-a I Bootstrap biblioteke.



*Slika21. Primer Angular Servisa za komunikaciju sa REST mikroservisom*



*Slika22. Primer Angular Servisa za komunikaciju preko Web Socket-a*

# **Zaključak**

Kroz praktičnu primenu postojećeg znanja i sticanjem novih, stekao sam vredno iskustvo vezano za kompletan razvoj i projektovanje informacionih sistema I softvera, od ideje I zahteva pa do gotovog proizvoda. Takođe sam stekao iskustvo rada u timu i radnu naviku, odnosno navikavanje na radno vreme I rad u firmi. Što se tiče firme DualSOFT mogu reći sve najbolje, jer je atmosfera za rad odlična a kolege I kolektiv interesantan I prijatan. U firmi je bila obezbeđena odlična oprema za rad u vidu dobrog hardvera, a tempo rada je bio usklađivan sa našim obavezama na fakultetu što nam se jako dopalo i uvek je bilo dovoljno vremena da se završe svi poslovi. Mentori su bili strpljivi i dostupni, I motivisali su nas na samostalno učenje i rad. Nakon svega ovoga, moram reći da je praksa bila vrlo edukativna I da smo mnogo toga naučili I iskusili, a isto tako upoznali jedan dobar radni kolektiv koji nas je lepo prihvatio i sa kojim je bilo I zabavno raditi. Tako da bih praksu u DualSOFT-u preporučio I drugim studentima.