

UNIVERZITET SINGIDUNUM
FAKULTET ZA SAVREMENE INFORMACIONE TEHNOLOGIJE



**Veb aplikacija “Individualna statistika košarkaša
Evrolige“**

- master rad -

Mentor:
prof. dr. *Milan Čabarkapa*

Kandidat:
Dušan Pavlović
Br. Indeksa 2015/410028

Beograd, 2017.

Sadržaj

1. Uvod.....	3 -
2. Dokumentacija veb aplikacije.....	4 -
2.1. Okruženja koja su korišćena za izradu veb aplikacije.....	6 -
3. SQL.....	6 -
4. Hiberante - ORM.....	8 -
5. Bootstrap.....	12 -
5.1. Grid opcije.....	14 -
5.2. Bootstrap - elementi.....	17 -
6. Servlet i JSP tehnologije.....	26 -
6.1. Servleti.....	26 -
6.2. Java server pages (JSP).....	37 -
7. Opis aplikacije - struktura koda.....	42 -
7.1. Backend aplikacije.....	48 -
7.2. Frontend aplikacije.....	66 -
8. Funkcionalnost aplikacije.....	81 -
8.1. Izgled aplikacije.....	82 -
9. Zaključak.....	94 -
Spisak referenci.....	95 -

1. Uvod

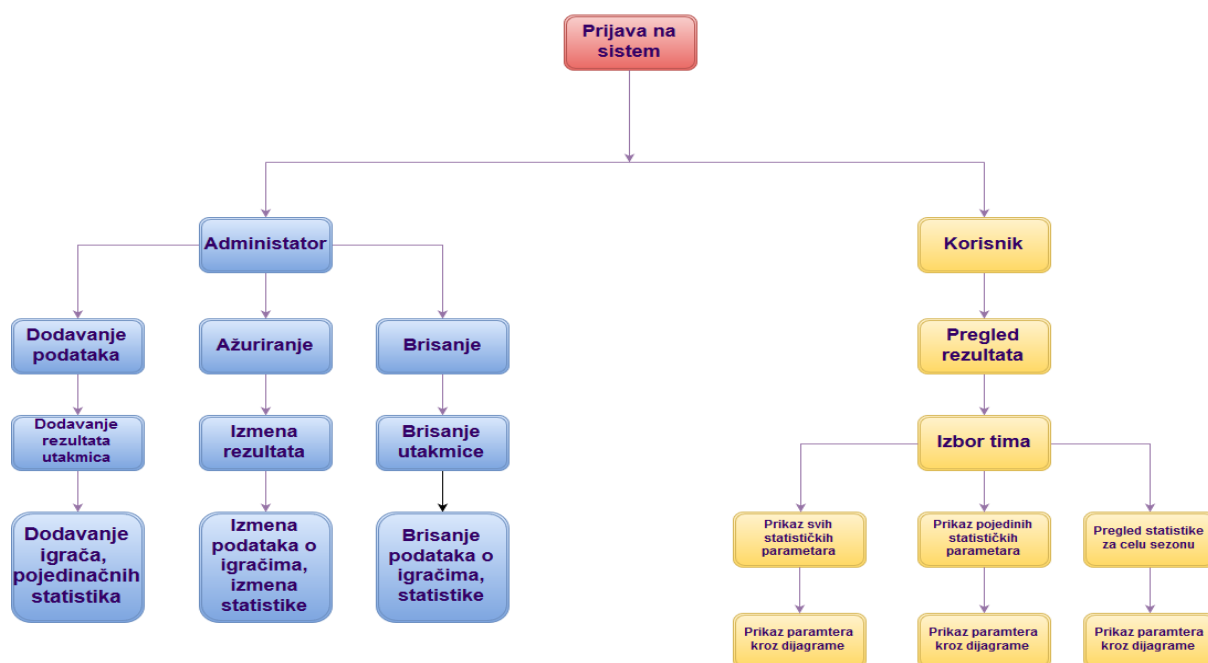
U skladu sa trendom povezivanja svih životnih segmenata sa internetom, imamo situaciju da se u svim branšama na neki način koristi internet. Sve je više različitih aplikacija sa različitim namenama. Kako velike multinacionalne kompanije, koje zapošljavaju veliki broj ljudi, tako i mali preduzetnici, svi imaju potrebu za određenim aplikacijama i softvetima koji će im olakšati kako komunikaciju sa klijentima (mail-ovi), tako i druge aktivnosti vezane za poslove kojim se bave. Upravo iz tog razloga sve je veći broj različitih aplikacija, koje se prave, razvijaju i unapređuju iz dana u dan. Kako je, razvijanjem tehnologija internet postao dostupan na svakom koraku, a većini aplikacija je potreban, vremenom su "offline" aplikacije zamenjene portalima i veb aplikacijama. Kako u svim oblastima, ista potreba se javila i u sportu. Premda je savremeni sport baziran na brzom razmeni informacija, logično je da je internet jedan od načina komunikacije. Kao jedan od najpopularnijih sportova **košarka** je bila povod za pravljenje dosta portala, veb strana i aplikacija sa raznim namenama. Najčešće prevladavaju portali sa vestima, na kojima se mogu naći tekstovi o određenim ligama i utakmicama. Retko se mogu naći neki specijalizovani sistemi za stručnjake iz tog sporta, gde postoje informacije o igračima sa kojima se oni mogu služiti. U pripremi svake utakmice, ozbiljan tim ima svoju skauting službu, koja se bavi analizom predstojeće utakmice. Analiza je posao za skaute i pomoćne trenere, koji pregledom utakmica (video zapisa) izvlače statistike svih igrača, prave analize akcija protivnika i sačinjavaju pisani izveštaj u formi analize igre i analize pojedinačnih karakteristika igrača. Pojedinačne karakteristike se izvlače delom iz statistika prethodnih utakmica, a delom iz analize video zapisa. Otuda se javila ideja da se napravi veb aplikacija (servis) koji će služiti za analizu i upoređivanje pojedinačnih statistika igrača u toku sezone. Mesto na kome bi skauti mogli da pronađu parsirane podatke i grafikone na kojima mogu da prate kretanje određenih parametara kroz sezonu. Takav pristup podacima može da napravi razliku između pobede i poraza. Obični statistički podaci uglavnom pokazuju globalnu statistiku (na nivou sezone) ili trenutnu (utakmica), ono što bi ovaj veb servis pružao je mogućnost da se analizira određen parametar kroz određen period ili celu sezonu.

Korišćene tehnologije za izradu aplikacije su MySQL za bazu, Java za backend i HTML5, CSS3, Javascript i Bootstrap za frontend. Pristup servisu se odvija preko prijave na sistem. Postoje dva tipa naloga: administrator i korisnik. Administrator ima pristup svim podacima i ovlašćenja da ih dodaje, menja ili briše. Korisnik ima mogućnosti izbora tima, posle koga ga sistem prebacuje na stranu izabranog tima, gde se nalaze grafikoni i tabela sa isparsiranim podacima. Kroz pristup grafikonima, korisnik ima uvid u poređenje određenih individualnih statističkih parametara. Parametri koji se porede indirektno opisuju igrača ili igrače koji se porede. Od izbora administratora, moguća su različita poređenja, pa tako može da se uporede podaci jednog igrača na određenom vremenskom intervalu ili više igrača, kako kroz sezonu, tako i na samo jednoj utakmici.

2. Dokumentacija web servisa

Zahtevi aplikacije:

- Namena aplikacije je da omogući korisnicima uvid u individualne statističke podatke košarkaša, sa utakmica Euroleague, na jednom mestu
- Sistem treba da omogući posetiocu da brzo i lako pristupi podacima igrača za koje je zainteresovan
- Servis treba da bude realizovan u Veb varijanti, gde smo “frontend”¹ realizovali uz pomoć tehnologija kao što su HTML5, CSS3, JavaScript i Bootstrap a za “backend”² odnosno ono što će se izvršavati u pozadini aplikacije koristiće se Java tehnologija i MySQL baza podataka
- Treba da postoje dva različita tipa korisničkih naloga administrator i korisnik
- Na početnoj strani postoji uputstvo za prijavljivanje na sistem i tekst dobrodošlice. Pored toga tu je i „login“ dugme kojim se otvara prozor za prijavljivanje na sistem nakon čega su dostupne funkcionalnosti u zavisnosti od tipa korisničkih naloga
- Administrator je zadužen za održavanje i ažuriranje stranice, kao i unošenje novih podataka posle odigranih utakmica
- Korisnik dobija mogućnost izbora tima koji ga interesuje i potom uvid u individualne statističke parametre uneth podataka za igrače



Slika 1. – Prikaz Hijerarhijskog dijagrama funkcionalnosti sistema

¹ frontend – deo veb aplikacije koji vidi korisnik na svom internet pregledaču, sastoji se iz HTML statičkih strana i JSP dinamičkih strana

² backend – deo veb aplikacije koji nije dostupan korisnicima, na back end-u se nalazi arhitektura celog sajta, veze sa bazom, sa front end-om i relacije unutar aplikacije

Funkcionalnosti za svaki od tipova naloga:

1. Administrator:

Uloguje se na sistem uz pomoć korisničkog imena i lozinke. Nakon logovanja pristupa administratorskom panelu na kom ima mogućnosti:

- Unos rezultata
- Unos igrača
- Unos pojedinačnih statističkih parametara
- Ažuriranje svih informacija, ukoliko je došlo do nekih grešaka
- Brisanje postojećih parametara iz evidencije

Nakon logovanja administrator ima sledeće operacije na raspolaganju:

- 1) Dodavanje rezultata - Posle odigrane utakmice, administrator unosi ime ekipa koje su igrale i rezultat koji je postignut
- 2) Dodavanje igrača - Administrator ima mogućnost da dodaje igrače svih timova, ukoliko se dogode neki transferi iz tima u tim
- 3) Dodavanje individualnih statističkih podataka - Posle unosa rezultata određene utakmice, sledeći korak administratora je da unese pojedinačne učinke igrača
- 4) Ažuriranje postojećih podataka - Administratoru je omogućeno da pristupa svim ranije unetim podacima i da ih menja
- 5) Brisanje postojećih podataka - Ukoliko se desi da je neka utakmica poništena ili iz nekog drugog razloga treba da bude uklonjena, administratoru je omogućeno da ukloni celu utakmicu, igrača iz ekipe ili pojedinačne statistike igrača.

2. Korisnik:

Uloguje se na sistem pomoću korisničkog imena i lozinke. Nakon logovanja u mogućnosti je da vidi sve rezultate utakmica koje su na raspolaganju za analizu. Na strani sa rezultatima ima mogućnost izbora tima koji ga interesuje. Nakon izbora tima otvara se pojedinačna strana za tu ekipu tj. statistika i pregled učinka. Od opcija ponuđeni su grafikoni koji upoređuju određene statističke parametre. Grafikoni šuta, skokova, odnosa minuta i poena, minuta i indeksnih poena i na dnu strane tabele sa rezultatima. Korisnik u svakom trenutku može da se vrati na stranu gde su mu timovi i da izabere drugi tim čiju statistiku će da pogleda.

2.1. Okruženja koja su korišćena za izradu veb aplikacije

Korišćen je NetBeans IDE 8.1 kao JAVA razvojno okruženje. MySQLWorkbench kao okruženje u kom je kreirana baza podataka. Korišćena je XAMPP aplikacija koja otvara portove za komunikaciju sa serverima, konkretno ka Apache serveru (port ovi 80 i 443) i MySQL serveru (port 3306) koje aplikacija koristi.

3. SQL

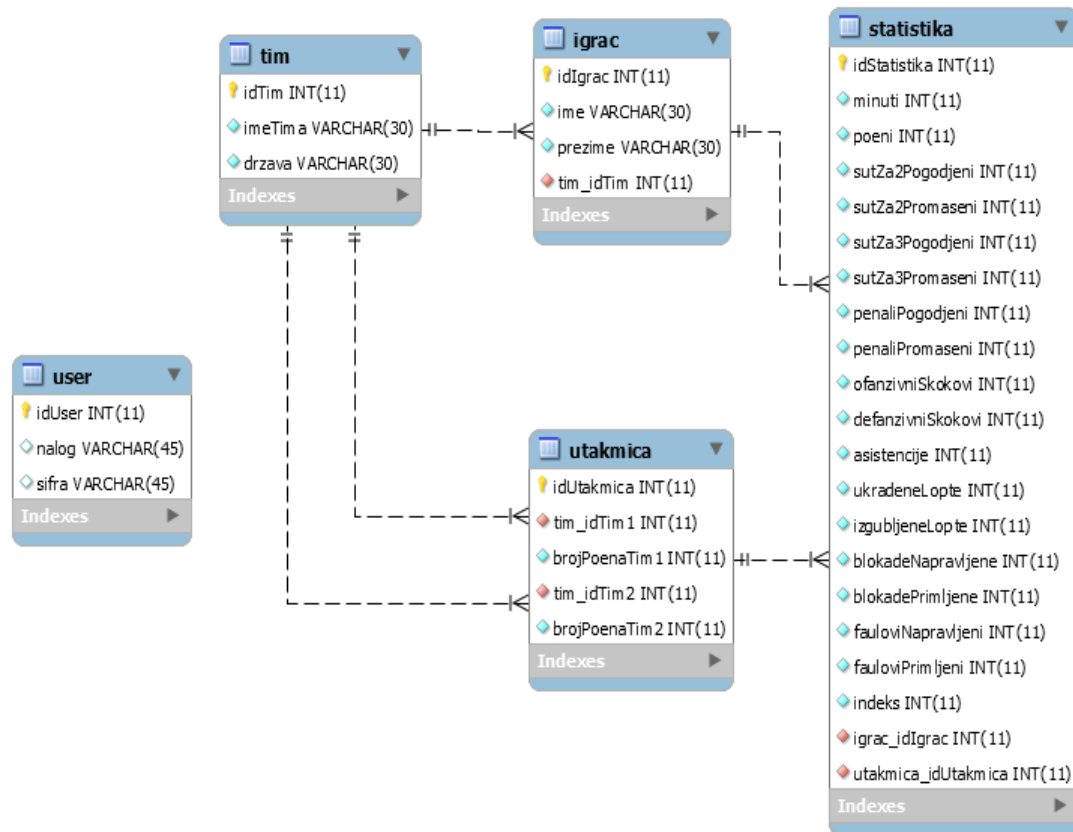
SQL - Structured Query Language, predstavlja relacioni upitni jezik. Relacije se kreiraju jednom naredbom (<http://searchsqlserver.techtarget.com/definition/SQL>, 2016). Jedinstven je, jer se svi podaci i rezultati operacija prikazuju u vidu tabele. Iako su prve verzije SQL-a bile bliske korisniku, za SQL(1999) se može reći da je kompleksan jezik. U njega su uključeni koncepti objektno tehnologije. Da bi se povećala funkcionalnost jezika uvedene su upravljačke strukture slične upravljačkim strukturama klasičnih programskih jezika. Tako postoje rezervisane reči (komande), kojima kreiramo, menjamo i brišemo kako tabele, tako i cele baze podataka. Pored osnovnih tabela u koje smeštamo vrednosti parametara koje skupljamo, SQL je i poznat po relacijama koje između datih tabela kreiramo ukoliko je to potrebno. Komande postoje i za kreiranje relacija.

Prilikom kreiranja Web servisa Individualna košarkška statistika, kreirali smo SQL bazu. Okruženje koje je korišćeno je MySQL Workbench, okruženje specijalizovano za kreiranje i rad sa bazama. U samom imenu okruženja se pojavljuje skraćenica MySQL, to je višekorisnički i višenitni SQL sistem za upravljanje bazama. Sistem radi kao server, obezbeđujući interfejs za pristup bazi podataka. MySQL je jako popularan u izradi web aplikacija i delimično slobodan softer, pa je zato i interesantan velikom broju korisnika, kojima trebaju relacione baze. MySQL pod Gnu-ovom opštom lincencom je slobodan, a ukoliko se koristi neka druga licenca, MySQL je prilično dostupan i uslovi plaćanja licence su povoljni.

Za servis koji kreiramo, napravili smo MySQL bazu čiji EER dijagram³ se nalazi na slici 2. Postoji pet različitih šema (korisnik, utakmica, tim, igrač i statistika). Šema korisnik je potpuno nezavisna i koristiće za odvajanje korisnika od administratora, oba naloga će imati svoj nalog i šifru. Svaka šema ima primarni ključ, koji je obavezan i određuje datu šemu. Pored primarnog postoje i strani ključevi (foreign keys), koji služe za povezivanje šema po funkcionalnosti. Postoje različite vrste veza između šema. 1 prema 1 su veze u kojima se jedan element iz jedne šeme vezuje za jedan element iz druge šeme. 1 prema više je veza gde se jedan element iz jedne šeme vezuje za više elemenata druge šeme. Više prema više je veza kojom se vezuje više elemenata jedne šeme za više elemenata druge šeme. U aplikaciji postoje veze 1 prema više, tim – igrač (u svakom timu može biti više igrača), tim – utakmica (svaki tim može igrati više utakmica). 1 prema 1, utakmica - statistika (sa jedne utakmice može biti samo jedna statistika), igrač – statistika (svaki igrač može imati samo jednu statistiku sa date utakmice).

³ EER dijagram - enhanced entity relationship diagram, predstavlja sliku šema za jednu bazu podataka sa svim njenim relacijama

Svaka od šema ima svoje kolone koje se popunjavaju i svaka kolona je definisana sa tipom podataka kojim se popunjava. Tako šema user ima idUser (ujedno i primarni ključ) koji je tipa INT (brojevni), nalog, sifra VARCHAR (string – niz karaktera). Isto su i ostale šeme definisane sa svojim kolonama pod čije okrilje se upisuju podaci koje čuvamo i za koje nam šeme trebaju. Skup šema koje se nalaze na slici ispod nazivamo baza podataka (relaciona baza – MySQL).



Slika 2. MySQL baza podataka veb servisa Individualna košarkška statistika

Kada se kreira baza podataka, njene šeme (tabele) su prazne i potrebno je uneti podatke da bi baza imala smisla. Unos podataka može da se radi na više načina. Može se unositi u okruženju MySQL Workbench, putem koda ili prilikom pregleda šema ručno. Alternativni način je prilikom konekcije sa nekim od kreiranih programa, recimo u Java-i. Da bi se šeme popunjavale preko neke aplikacije, potrebna je konekcija aplikacije sa serverom na kome se nalazi baza i niz komandi koje aplikacija šalje. Upravo za tu vezu između veb servisa koji kreiramo i baze podataka koju smo napravili potrebni su nam JDBC i Hibernate.

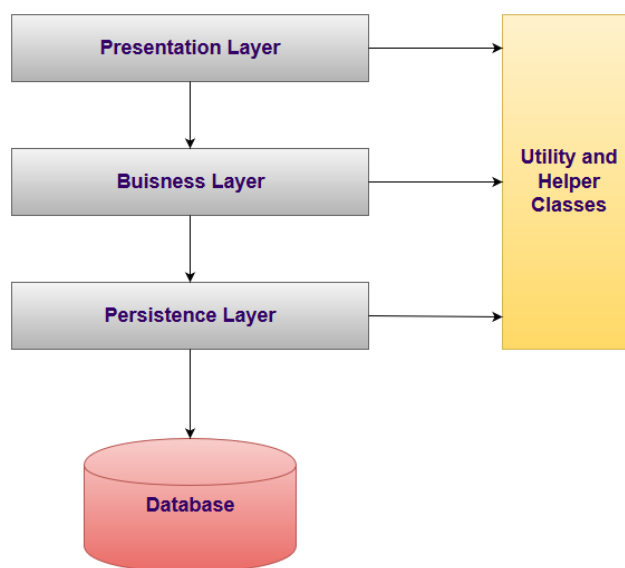
4. Hibernate – ORM

Hibernate Core – Hibernate za Javu sa osnovnim API-jem i anotacijama. Cilj razvoja Hibernate-a je da se programeri oslobode velikog dela posla koji se odnosi na perzistenciju podataka (čuvanje podataka u bazi pomoću JDBC-a i SQL-a). Hibernate Core generiše SQL kod u toku izvršavanja programa. Omogućava da aplikacija može da radi na bilo kom sistemu za upravljanje bazama podataka. Definiše sopstveni objektno-orijentisani upitni jezik – HQL (Hibernate Query Language). Ako se koristi Maven samo dodamo zavisnosti za Hibernate u pom.xml i Maven sam dovuče sve potrebne biblioteke (<https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/tutorial.html#tutorial-firstapp-setup>, 2004.).

Perzistencija podataka

Perzistencija podataka je Jedan od osnovnih koncepata razvoja aplikacija. Pod tim terminom podrazumevamo čuvanje podataka i posle gašenja računara. U Java-i perzistencija je skladištenje Java objekata u relacionu bazu podataka. Objekti se mogu skladištiti i u fajlove, objektno-orijentisane baze podataka, XML baze podataka i td. ORM je srkaćenica od Object Relational Mapping (Objektno Relaciono Mapiranje) (<https://coderanch.com/t/218866/databases/Persistence-Hibernate>, 2017).

Na Slici 3. vidimo strukturu jedne aplikacije i njene slojeve. Na vrhu aplikacije se nalazi prezentacioni sloj (front end), koji je dostupan korisnicima aplikacije. Ispod njega se nalazi biznis sloj u kome se smešta biznis logika i suština kako aplikacija funkcioniše, koje su zavisnosti između elemenata u tom sloju i šta aplikacija treba korisniku da donese. Ispod biznis sloja nalazi se Persistence sloj, koji služi za vezu sa bazom. Svaki od slojeva u svakom trenutku ima vezu sa pomoćnim klasama.



Slika 3. Slojevit struktura aplikacija

Tehnike za implementaciju Persistent sloja u Javi:

- SQL i JDBC predstavljaju ručno programiranje, koje zahteva mukotrpan rad. Prilikom promene platforme zahtevaju se promene u kodu, što iziskuje više posla, samim tim i problema
- Jedan od standarda za persistenciju je ORM, JPA - Java Persistence Architecture API. Implementacije JPA su Hibernate i EclipseLink. JPA se zasniva na POJO konceptu (Plain Old Java Object) (<http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>, 2014) . Anotacijama se komanduje OR mapperu na koji način će perzistirati objekte te klase. SQL naredbe mogu biti ugrađene u kodu ili generisane u toku izvršavanja programa. Neke od funkcija JDBC API-ja:
 - dodela vrednosti parametrima u SQL upitu
 - izvršavanje upita
 - iteracija kroz listu pogodaka
 - preuzimanje vrednosti rezultata upita

Sve navedene funkcije za pristup podacima su “niskog nivoa” i zahtevaju mnogo mehaničkog i napornog programiranja. POJO klasa je klasa sa uvedenim objektima, njihovim deklaracijama, konstruktorima, geterima i seterima. Postala je standard prilikom korišćenja Hibernate-a.

Perzistencija u objektno-orijentisanim aplikacijama omogućava objektima da “prežive” i po završetku procesa koji ih je kreirao. Stanje objekta se čuva na disku i moguće ga je u budućnosti ponovo kreirati sa istim stanjem. Može se sačuvati i više objekata zajedno sa vezama. Svaki informacioni sistem sadrži perzistentne objekte. Java klasa služi za definisanje šeme baze. Prilikom snimanja, podaci iz objekata se snimaju u tabele. Prilikom učitavanja, podaci se iz tabela stavljaju u objekte koji su za to napravljeni.

ORM je preslikavanje Java objekata na tabele relacione baze podataka korišćenjem metapodataka (anotacije) koji opisuju pravila za preslikavanje. ORM transformiše podatke iz jednog oblika u drugi. ORM tehnologija treba da podrži sledeće funkcionalnosti:

- API za standardne operacije nad bazom (CRUD – create, read, update, delete) koje bi se izvršavale nad objektima
- Poseban jezik ili API za postavljanje upita nad klasama i atributima klasa
- Tehniku za specificiranje metapodataka za mapiranje (preslikavanje)
- Tehniku za optimizaciju funkcija nad bazom podataka.

Korišćenjem ORM-a aplikacije rade sa ORM API-jem, a cela komunikacija preko SQL-a i JDBC-a je “skrivena”.

Prednosti korišćenja ORM-a:

- Produktivnost, prilikom razvoja aplikacija možemo de se koncentrišemo samo na poslovnu logiku aplikacije
- Jednostavno održavanje programa, manja povezanost između objekata i tabela u odnosu na ručno programiranu perzistenciju
- Performanse, moguće su različite optimizacije, specijalizovani alati imaju bolje performanse od onih koji se ručno programiraju
- Nezavisnost od platforme

Osnovne karakteristike Hibernate Core-a su prirodni model programiranja (podržava sve koncepte OO programiranja: nasleđivanje, polimorfizam, kolekcije). Pored toga podržava sofisticiranije modele podataka (različite vrste mapiranja za kolekcije i objekte koji zavise od drugih objekata). Definiše sopstveni upitni jezik. Besplatan je (open source).

Entiteti (Entities) su namenjeni za predstavljanje podataka. POJO (Plain Old Java Object) klase. Anotacijama naglašeno šta je primarni ključ, kako se formira, itd. Može i ne mora da implementira Serializable interfejs. Ako se implementira, može da se koristi kao parametar metode u distribuiranim sistemima (EJB). Ime entity klase ne sme da bude iz skupa rezervisanih reči (npr. order, date i sl.)

Anotacijom **@Entity** se označi klasa da je entity. Anotacija **@Id** je obavezna i označava koji atribut se mapira na primarni ključ. Na sledećem primeru je prikazana primena anotacija. Anotacijom **@GeneratedValue** se može definisati generator primarnog ključa.

```
@Entity
public class Book implements Serializable {
    @Id    // anotacija za primarni ključ
    private Integer id;
    private String title;
    public Book(){}
    public Book(Integer id, String title)
    {
        this.id = id;
        this.title = title;
    }
    public Integer getId()
    {
        return id;
    }
    public void setId(Integer id)
    {
        this.id = id;
    }
    public String getTitle()
    {
        return title;
    }
    public void setTitle(String title)
    {
        this.title = title;
    }
}
```

Stanja Entity-ja

- New - entity kreiran, ali još nije perzistiran u bazu.
- Managed - sve promene nad atributima se sinhronizuju sa bazom dok je "aktuelan" EntityManager (transakcije, završetak metode), metode persist(), merge() i find() prebacuje entity u ovo stanje.
- Detached - stanje entity-ja nije sinhronizovano sa bazom, po završetku transakcije i prilikom prosleđivanja entity-ja drugoj metodi.
- Removed - nakon poziva remove() metode uklanja se entity.

(<https://vladmihalcea.com/2014/07/30/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/> , 2014)

Metode EntityManager-a

- clear (svi managed entity-ji se detach-uju)
- close (prekida rad entity manager-a)
- find (vraća managed entity, na osnovu primarnog ključa)
- createQuery, createNamedQuery, createNativeQuery
- flush – sinhronizuje memoriju sa bazom (unutar transakcije)
- persist (insert – dodaje u bazu podatke)
- merge (update – menja postojeće odatke)
- refresh (ponovo čitanje iz baze u Java objekte)
- remove (delete – briše podatke)

Od operacija sa entity-ima postoje kreiranje novog, update postojećeg i brisanje postojećeg. Kreiramo novi tako što pozovemo `EntityManager.persist(obj)`. Merge koristimo kada se ažurira objekat. poziva se `em.merge(obj)`. Menjamo attribute objekta. Povratna vrednost metode merge je referenca na managed entity. `x = em.merge(x);` Za brisanje koristimo `EntityManager.remove(obj)`. Java objekat mora da bude u managed stanju da bi se obrisao pozivom `em.remove(obj)`. Ako imamo detach-ovan entity, prvo ga merge-ujemo, pa ga onda obrišemo. Prvo komanda `x = em.merge(x);` zatim `em.remove(x);`

Atributom strategy se podešava kako se generiše primarni ključ:

- GenerationType.IDENTITY
- GenerationType.SEQUENCE
- GenerationType.TABLE

5. Bootstrap

Od pojave interneta i prvih HTML stranica u CERN-u do danas desile su se mnoge promene. U međuvremenu HTML-u su se pridružili CSS i JavaScript, a razvojem ovih tehnologija razvijali su se i veb-pregledači Netscape, Opera, IE, Firefox, Chrome, i drugi. U poslednjih nekoliko godina, pojavili su se i različiti uređaji na kojima se koriste pomenuti pregledači pa su računari posatali "vebcentrični", veb-pregledač (web browser) postao je program koji najviše koristimo (<http://it-konekt.com/sr/comment/reply/203>, 2015). Pored toga dimenzije uređaja su različite, pa je i to postao problem kod kreiranja veb strana. Spektar dijagonale monitora na kojima se prikazuje ovaj sadržaj trenutno se nalazi u opsegu od 3 do 27 inča (7,62cm - 68,58cm), pa i više. Takav razvoj događaja promenio je način rada programera koji se bave krajnjim proizvodom veb sadržaja. Jedan od glavnih zahteva danas je da sajt bude dobro prikazan na svim uređajima i da sadržaj ima odgovarajući vizuelni izgled. Za opis tehnologije između ostalih, korišćeni su i primeri sa oficijalnog sajta (<http://getbootstrap.com>, 2016).

Bootstrap je jedan od framework-a za jednostavnije pravljenje veb sajtva. Dva programera (Mark Otto i Jacob Thornton) koja su radila za kompaniju Twitter (socijalnu mrežu) kreirala su potpuno nov framework i besplatno ga podelili sa svima. U početku framework je koristio unutar firme za izradu frontend-a, ali je kasnije njihova kreacija postala potpuno besplatna, a pored toga je i open source (svaki korisnik može da ga menja, proširuje...). Prednost korišćenja Bootstrapa, u odnosu na druge alate za izradu veb sajtova, je u tome što biblioteke koje sadži imaju veliku količinu gotovog koda koji treba samo da se iskoristi. U prevodu Bootstrap ima gotove obrasce, koji se odredjenim komandama samo koriste i pritom prave odličan izgleda sajta.

Potrebno je napraviti „responsive“ sajt, sajt koji treba da se prilagođava uređejima (mobilnom telefonu, tabletu, desktop računaru...), kreiranje takvog sajta bi oduzelo puno vremena, pošto bi moralo da se odvojeno piše kod za svaku opciju (veličinu ekrana). Bootstrap ima svoje stilove (obrasce) kojima je potrebno samo da se dodele elementi koje koristimo i sajt postaje „responsive“. Na samom sajtu Bootstrap-a može se pronaći dosta primera, tako da ni korišćenje nije preterano teško. Pored toga na internetu postoji dosta primera i tutorijala za korišćenje ovog framework-a. Bootstrap je trenutno najpopularniji projekat na Github-u. Najvažnija karakteristika Bootstrap-a je postojanje podrške za sve veće veb-pregledače, kako na računarima tako i na mobilnim telefonima. Sledeća važna karakteristika je prilagodljiv dizajn (responsive design). Od verzije 3.0 Bootstrapa podržavaju "Mobile first" princip razvoja. U prevodu, prvo se radi razvoj za mobilne uređaje, a zatim i za veće ekrane.

Zbog specifičnosti korišćenja već gotovih framework-a, postoji podela među programerima da li ih je dobro koristiti ili nije. Ista stvar je i sa Bootstrap-om. Bootstrap nudi način da se uštedi vreme i ubrza razvoj aplikacije, ali korišćenjem njegovih gotovih elemenata, pravimo nešto što bilo ko, ko koristi Bootstrap može da napravi, znači nema originalnosti. Opet korišćenjem framework-a pravi se proizvod zavidnog kvaliteta, jer je sve stabilno i istestirano mnogo puta, pa nema bojazni. U svakom slučaju treba odrediti šta nam je potrebno i na osnovu toga odlučiti da li koristiti ili ne Bootstrap-a.

Na slici 4. se nalazi pregled zastupljenosti Bootstrap-a na internet pregledačima i na operativnim sistemima. Možemo zaključiti da je zastupljenost totalna, tj. da prilikom izbora

pregledača koji se najčešće i koriste aplikacije u kojima je korišćen Bootstrap, biće prikazane bez ikakvih problema.

	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	✓ Supported	✓ Supported	N/A	✗ Not Supported	N/A
iOS	✓ Supported	N/A		✗ Not Supported	✓ Supported
Mac OS X	✓ Supported	✓ Supported		✓ Supported	✓ Supported
Windows	✓ Supported	✓ Supported	✓ Supported	✓ Supported	✗ Not Supported

Slika 4. Zastupljenost Bootstrap-a na internet pregledačima

Framework sadrži CSS, JavaScript i Font-ove. Na slici 5. ispod, vidi se sadržaj biblioteka, tačnije podfolderi u kojima se nalaze datoteke sadržine sa slike (biblioteke).

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   ├── bootstrap-theme.min.css
│   └── bootstrap-theme.min.css.map
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Slika 5. Sadržaj Bootstrap-a

5.1. Grid opcije

Grid – rešetka. Sistem kojim se služi framework, pod nazivom Grid, je sistem koji se bavi „responsive“ (primeniti veličinu prema potrebi) oblikovanjem aplikacija. Prepoznavanjem veličine ekrana, aplikacija može da menja svoj raspored elemenata. tj. da premesti elemente tako da budu pregledniji i da nije potrebno uveličavati sliku da bi se pristupilo ili videli određeni delovi sajta. Od trenutka kada su responsive sajtovi posatli deo interneta, korišćenje malih ekrana je postalo znatno olakšano i upotreba telefona u korišćenju interneta povećala. Na slici 6. se nalazi tabela na kojoj je podela po veličini ekrana i osobine Bootstrap-a koje se menjaju u zavisnosti od uređaja (<http://getbootstrap.com/css>, 2016).

	Extra small devices Phones (<768px)	Small devices Tablets (>=768px)	Medium devices Desktops (>=992px)	Large devices Desktops (>=1200px)
Grid behaviour	Horizontal at all times	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints	Collapsed to start, horizontal above breakpoints
Container width	None (auto)	750px	970px	1170px
Class prefix	.col-xs-	.col-sm-	.col-md-	.col-lg-
Number of columns	12	12	12	12
Column width	Auto	~62px	~81px	~97px
Gutter width	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)	30px (15px on each side of a column)
Nestable	Yes	Yes	Yes	Yes
Offsets	Yes	Yes	Yes	Yes
Column ordering	Yes	Yes	Yes	Yes

Slika 6. Grid opcije

Grid tj. responsive strana sadrži kontejnere, redove i kolone. Kontejneri su osnovni elementi Grid-a koji su podeljeni u redove. Redova može biti jedan ili više, a kolona u zbiru 12.

`<div class="container">` - otvorena klasa kontejner

`<div class="row">` - otvorena klasa red

`<div class="col-md-12">` - otvorena klasa kolona (sve klase se zatvaraju sa `</div>`)

Različita je naravno i pikselaza (pixel – merna jedinica digitalne slike, najmanja vrednost) u zavisnosti od veličine ekrana. U tabeli je napisana pikselaza za kontejnete i kolone. Napisane su i prefiks klase u zavisnosti od veličine slike.

Na slici 7. prikazan je izgled grid-a koji je pravljen za prikaz sajta na desktop računaru. Prikazane su različite kolone spakovane u širinu ekrana.

.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1	.col-md-1
.col-md-8								.col-md-4				
.col-md-4				.col-md-4				.col-md-4				
.col-md-6						.col-md-6						

Slika 7. Izgled bootstrap sajta na desktop računaru

Ispod se nalazi kod za grid sajta čiji prikaz se radi za desktp računar, čija je slika 5. prikazana iznad. Uočljivo je da `col-md-1` predstavlja klasu kolone sa najmanjom vrednošći i u širinu strane staje 12 takvih kolona, svaki veći broj na kraju, povećava veličinu kolone.

```

<div class="row">
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
  <div class="col-md-1">.col-md-1</div>
</div>
<div class="row">
  <div class="col-md-8">.col-md-8</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>

```

Na slici 8. prikazan je izgled grid-a koji je pravljen za prikaz sajta na destop računaru i mobilnom tefonu, a na slici 9. izgled grid-a za mobilni, tablet ili desktop.

.col-xs-12 .col-md-8		.col-xs-6 .col-md-4
.col-xs-6 .col-md-4	.col-xs-6 .col-md-4	.col-xs-6 .col-md-4
.col-xs-6		.col-xs-6

Slika 8. Grid za destop i mobilni

.col-xs-12 .col-sm-6 .col-md-8		.col-xs-6 .col-md-4
.col-xs-6 .col-sm-4	.col-xs-6 .col-sm-4	.col-xs-6 .col-sm-4

Slika 9. Grid za mobilni, tablet i desktop

Kod za izled grid-a za mobilni i desktop.

```
<div class="row">
  <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<div class="row">
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<div class="col-xs-6">.col-xs-6</div>
<div class="col-xs-6">.col-xs-6</div>
</div>
```

Kod za izled grid-a za mobilni,tablet i desktop.

```
<div class="row">
  <div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12.col-sm-6.col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6.col-md-4</div>
</div>
<div class="row">
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
  <div class="clearfix visible-xs-block"></div>
  <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4</div>
</div>
```

Pored ovih izgleda grid-a, bootstrap ima još različitih izgleda u zavisnosti kakve su potrebe programera. Postoje opcije “Column wrapping” – različite širine kolona, “Responsive column resets”- mogućnost da se određena kolona resetuje – menja po veličini, “Offsetting columns”- varijanta kada kolone nisu poredane jedna do druge, nego su nabacane po potrebi aplikacije, “Nesting columns”- kada su kolone ugnježdene jedna u drugu i td.

5.2. Bootstrap – elementi

Elementi koje koristi Bootstrap i koje ćemo opisati u ovom radu, su elementi korišćeni na projektu. Konkretno elemenata je mnogo i neki specifični se koriste u retkim situacijama, zatko ćemo se ovde fokusirati na najpotrebnije za izradu servisa. Neki od elemenata se poklapaju sa HTML elementima (<http://getbootstrap.com/components/>, 2016).

Naslov (heading)

Kao i kod HTML strana i ovde postoji 6 različitih veličina naspova, pa će nam tako sledeći kod ispisati “Naslov” u poretku od najvećeg ka najmanjem. Najveći je h1, a najmanji h6. Svaki naslov će biti ispisan u “semibold” varijanti slova, tj. u delimično boldovanoj (široj i tamnijoj) varijanti. U veličini 36, 30, 24, 18, 14, 12 piksela. Dok bi tagom `<small>` pre zatvaranja tagova naslova (heading) tekst ispisan posle, bio manji bez boldovanja, ali takođe u okviru naslova.

```
<h1>h1. Naslov</h1>
<h2>h2. Naslov</h2>
<h3>h3. Naslov</h3>
<h4>h4. Naslov</h4>
<h5>h5. Naslov</h5>
<h6>h6. Naslov</h6>
```

Telo (Body)

U okviru `<body>` taga se u okviru strane piše kod tela strane. Veličina slova je 14 piksela, a dužina linije je 1428 karaktera. Paragrafe ozančavamo sa `<p>` i u okviru paragrafa postoji veliki broj komandi koji nam oblikuju tekst unutar paragrafa.

Poravnanja (Alignment)

Poravnanja su kao i u svim editorima bitna, jer nam je potrebno da početak teksta kreće iz različitih pozicija, pa tako i Bootstrap ima skup komandi koji smešta tekst u različite pozicije ekrana, u zavisnosti od toga šta nam je potrebno. Sledeći kod raspoređuje tekst u levo, srednje i desno.

```
<p class="text-left">Left aligned text.</p>
<p class="text-center">Center aligned text.</p>
<p class="text-right">Right aligned text.</p>
```

Pored ovih postoje i komande za smanjivanje slova (lowercased text), povećavanje slova(Uppercased text), veliko slovo na početku rečenice (Capitalized text). Postoje opcije za podvlačenje teksta, za inicijale kao na primer u sledećem kodu:

```
<abbr title="I tako dalje" class="initialism">ITD</abbr>
```

i mnoge druge za oblikovanje teksta.

Tabele (tables)

Kod tabela postoje nekoliko osnovnih razlika. Počevši od standardne tabele koju pišemo unutar tagova `<table class="table">` i `</table>`, naizmenično sivih redova (Striped rows) gde u početni tag dodajemo `<table class="table table-striped">`, tabele sa borderima – okvirima (Bordered table), gde u početni tag dodajemo `<table class="table table-bordered">`, redovi koji lebde (Hover rows) su tabele kod kojih su linijama odvojene vrste, ne i kolone. Kod takvih tabela u početni tag pišemo `<table class="table table-hover">`. Postoje i kod kojih se smanjuju prostori između kolona i vrsta (Condensed table), dodaćemo `<table class="table table-condensed">` umesto početnog taga.

Interesantan dodatak bootstrap-u su Responsive tables, o kojima bi mogli da pišemo kao o prilagodljivim tabelama. Prilagodljivim, zato što su u zavisnosti od uređaja na kojem se prikazuju, te tabele različito prikazane, tačnije kroz njih se prolazi drugačije. Ukoliko je širina ekrana veća od 768 piksela, tabele nemaju razliku, međutim ukoliko su ekrani manji, pomeranja se vrše po horizontali. Razlika u pisanju koda za Responsive tabelu je u tome što se pre pisanja tabele u “kontejneru” te tabele napiše `<div class="table-responsive">`, a kasnije tabela normalno nastavi: `<table class="table">...</table>` i na kraju zatvori tag kontejnera `</div>`. Na slici 10. se nalazi primer jedne uokvirene (Border) tabele.

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

Slika 10. Primer tabele

Forme (Forms)

Forma je deo internet strane koji služi za komunikaciju sa korisnikom. Konkretno korisnik na mestu Forme ima ponuđena polja na kojima je potrebno da unese neki tekst ili da upload-uje neku sliku, napiše mail, neke podatke o sebi, izabere ponuđene opcije iz padajućeg menije, čeka prostor predviđen za to i td. Na slici 11. nalazi se osnovni primer Forme na kojem imamo potrebu da umesemo svoj mail, lozinku, okačimo neki fajl i čekiramo je za izlazak.

Kod koji nam je potreban za ispis ovakve strane se nalazi ispod, s’tim što treba napomenuti da je ovo samo frontend, odnosno deo koji nam služi za oblikovanje Veb strane, a da bi ova forma mogla da se koristi treba pored ovog Bootstrap koda napraviti i backend i „oživeti“ aplikaciju koja bi koristila ovu formu. Postoji razlika između Formi i grupa za unošenje (Input group), grupe za unošenje su elementi Formi i nalaze se unutar Formi. Jedna od razlika u odnosu na standardne Forme su varijante kada imamo horizontalni izgled forme, to je situacija kada u kodu na početak Forme unesemo `<form class="form-horizontal">`. Što se u nosa tiče, Bootstrap podržava sve unse kao i HTML5. Tekst, lozinke, vreme i datum, mesec, sat, nedelju, broj, imejl, url, pretragu, telefon i boje.

Email address**Password****File input** No file selected.

Example block-level help text here.

☐ Check me out

Slika 11. Primer Forme

Kod za primer Forme sa slike 11.

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1"
placeholder="Email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
  </div>
  <div class="form-group">
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p class="help-block">Example block-level help text here.</p>
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

Dugmad (Buttons)

Sledeći kod predstavlja izbor standardnih dugmadi. Razlikuju se po boji, a funkciju bi im kasnije određivali u aplikaciji. Na slici 12. se nalazi redom dugmad, kako su u kodu kucana. Primetićemo da svaka od ovih reči koje stoje na dugmićima ima svoje značenje u kontekstu boja, isto tako ima značenje i u kontekstu koje dugme kada treba upotrebiti.

```
<button type="button" class="btn btn-default">Default</button>
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-link">Link</button>
```



Slika 12. Dugmad

Pored boje dugmad se mogu klasifikovati i po veličini. Sledećim otvorenim tagovima kreiramo “default” belo dugmad različite veličine:

- Komandom `<button type="button" class="btn btn-default btn-lg">` kreiramo veliko dugme.
- Komandom `<button type="button" class="btn btn-default">` kreiramo dugme standardne veličine
- Komandom `<button type="button" class="btn btn-default btn-sm">` kreiramo malo dugme
- Komandom `<button type="button" class="btn btn-default btn-xs">` kreiramo izuzetno malo dugme
- Komandom `<button type="button" class="btn btn-default btn-lg btn-block">` kreiramo blok šire dugme

Pored ovih postoje i komande kao sto su `active`, koje sa koriste da bi dugme dobilo tamnije nijanse i da bi bilo upadljivije. Postoji komanda `disabled`, koja će nam prikazano dugme posvetleti i neće dozvoliti mogućnost pritiska na isto.

Slike (Images)

Od Bootstrap-a 3 uvođenjem klase `.img-responsive` slike su postale skalabilne, tj. od veličine ekrana na kome su se prikazivale bi zavisio način i veličina prikazivanja. Koristeći neke od sledećih komandi moguće je menjati osobine slike koju koristimo:

- `max-width: 100%` - maksimalna širina 100%
- `height: auto` - automatska visina slike
- `display: block` - prikaz bloka
- `img-rounded` - zaobljen oblik slike
- `img-circle` - kružni oblik slike
- `img-thumbnail` - kvadratni oblik sa ramom

Prilikom korišćenja slike potrebno je navesti izvor odakle se slika poziva, internet ili neke lokalna mašina. To se rešava u otvorenom tagu na početku `` - primer koda za sliku zaobljenog oblika.

U okviru Pomoćnih klasa (Helper classes) postoje komande kojima možemo menjati boju teksta, pa tako recimo tagovima `<p class="text-danger">Neki tekst.</p>`, rečenicu “Neki tekst.” ćemo ofarbati u crveno. Boje se isto pozivaju sa rezevisanim rečima, primary, danger, warning, success. Poput teksta i pozadinu kontejnera moguće je obojiti određenm bojom, s'tim što treba biti oprezan posle toga sa bojom teksta, ukoliko postoji u tom polju, jer postoji mogućnost da se poklope boje, pa da se tekst ne vidi ili da boje budu kompatibilne, pa opet budu teške za čitanje. Pomoću komandi `<p class="bg-primary">...</p>` ćemo pozadinu polja ofarbati u plavo.

Za brže korišćenje aplikacija koje se prave, Bootstrap nudi određene klase, kojima će aplikacija postati responsive i kojim će se ubrzati rad na svakom uređaju. Na Slici 13. nalaze se klase koje treba koristiti da bi naša aplikacija bila vidljiva (ili nevidljiva) za uređaje određenih dimenzija.

	Extra small devices Phones (<768px)	Small devices Tablets (≥768px)	Medium devices Desktops (≥992px)	Large devices Desktops (≥1200px)
<code>.visible-xs-*</code>	Visible	Hidden	Hidden	Hidden
<code>.visible-sm-*</code>	Hidden	Visible	Hidden	Hidden
<code>.visible-md-*</code>	Hidden	Hidden	Visible	Hidden
<code>.visible-lg-*</code>	Hidden	Hidden	Hidden	Visible
<code>.hidden-xs</code>	Hidden	Visible	Visible	Visible
<code>.hidden-sm</code>	Visible	Hidden	Visible	Visible
<code>.hidden-md</code>	Visible	Visible	Hidden	Visible
<code>.hidden-lg</code>	Visible	Visible	Visible	Hidden

Slika 13. Klase kojima odobravamo vidljivost u zavisnosti od veličine ekrana

Grupisanje formulara (Input groups)

Napredni oblik forme, po principu elektronskog formulara, postoje elementi grupisani u kontejnere koji služe za interakciju sa korisnikom. Prilikom konstrukcije ovakvih formulara koriste se razne vrste već pomenutih elemenata, samo su ovde grupisani na jednom mestu. Osnovni primer se nalazi na slici 14. Klase koje koristimo su `.input-group`, `.input-group-addon`, `.input-group-btn`, `.form-control`. Kao kod ostalih elemenata postoji Sizing – menjanje veličine elemenata, koriste se checkbox i radio elementi u okviru Input groups-a, Button addons je dugme koje se nalazi ispred polja za unos texta, Button with dropdown – dugme koje se nalazi ispred polja za unos teksta i ima padajuću listu, Segmented buttons – Isto kao i Button with dropdown, samo je odvojeno polje sa trougličem koji pokazuje gde se otvara padajuća lista i Multiple buttons – opcija sa više dugmića pre polja za unos teksta.

The image shows four examples of input groups in a form:

- Example 1: A text input field with a placeholder "Username" and an "@" icon on the left.
- Example 2: A text input field with a placeholder "Recipient's username" and "@example.com" on the right.
- Example 3: A text input field with a "\$" icon on the left and ".00" on the right.
- Example 4: A label "Your vanity URL" above a text input field with "https://example.com/users/" on the left.

Slika 14. Primer Grupisanja formulara

Kod za kreiranje Grupisanog formulara:

```
<div class="input-group">
  <span class="input-group-addon" id="basic-addon1">@</span>
  <input type="text" class="form-control" placeholder="Username" aria-
describedby="basic-addon1">
</div>
<div class="input-group">
  <input type="text" class="form-control" placeholder="Recipient's
username" aria-describedby="basic-addon2">
  <span class="input-group-addon" id="basic-addon2">@example.com</span>
</div>
<div class="input-group">
  <span class="input-group-addon">$</span>
  <input type="text" class="form-control" aria-label="Amount (to the
nearest dollar)">
  <span class="input-group-addon">.00</span>
</div>
<label for="basic-url">Your vanity URL</label>
<div class="input-group">
  <span class="input-group-addon" id="basic-
addon3">https://example.com/users/</span>
  <input type="text" class="form-control" id="basic-url" aria-
describedby="basic-addon3">
</div>
```

Navigacija (Nav, Navbar)

Navs ili tulbarovi nalaze se u bibliotekama `.navs`. Koriste se najčešće na vrhu Veb strane i najčešće imaju oblik „trake“ na kojima se nalaze dugmići koji predstavljaju ili određene komande ili linkove ka drugim stranama. Navs-i su često korišćeni dodaci internet stranama zbog svoje preglednosti, lakog upravljanja i zato što su veoma prilagodljive.

Postoji nekoliko osnovnih izgleda tulbarova, pa tako razlikujemo tabove (Tabs) i tulbarove sa standardnim dugmićima (Pills), kod kojih dugmići mogu biti u horizontalnom ili vertikalnom poretku. Razlika između ova dva Nav-a je u pozvu biblioteke (nav-tabs, nav-pills) Na slici 15. nalazi se primer Navs – Tab i Nav - Pill, a kod iznad slike za Nav-Tab bi izgledao:

```
<ul class="nav nav-tabs">
  <li role="presentation" class="active"><a href="#">Home</a></li>
  <li role="presentation"><a href="#">Profile</a></li>
  <li role="presentation"><a href="#">Messages</a></li>
</ul>
```



Slika 15. Primer Navs-a

Moguće je korišćenje Dropdown menija umesto dugmića. Kao što je već ranije opisano, dropdown bi se u nav-pill uveo:

```
<ul class="nav nav-pills">
  ...
  <li role="presentation" class="dropdown">
    <a class="dropdown-toggle" data-toggle="dropdown" href="#"
    role="button" aria-haspopup="true" aria-expanded="false">
      Dropdown <span class="caret"></span>
    </a>
    <ul class="dropdown-menu">
      ...
    </ul>
  </li>
  ...
</ul>
```

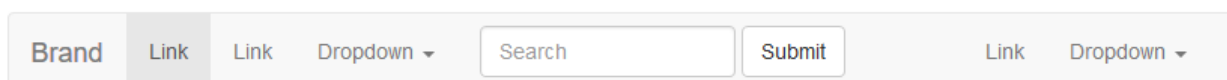
Navbar, kao proširenje Nav-a. Tulbar kojem osim dugmića i dropdown opcija dodajemo i prostor u kome korisnik može uneti neki text po kome će se obaviti pretraga na sajtu. Generalno Navs-i i Navbarovi su osmišljeni kao navigacioni dodaci, pomoću kojih bi korisniku bilo lakše da se kreće kroz sajt. Da ima lakši pristup, pošto su navigacioni tulbarovi, kako ih možemo zvati, prilično jednostani za upravljanje. U primeru ispod nalaze se kod i slika Navbar-a sa najčešćim elementima.

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-
      toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-
      expanded="false">
        <span class="sr-only">Toggle navigation</span>
```

```

        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
    </button>
    <a class="navbar-brand" href="#">Brand</a>
</div>
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-
1">
    <ul class="nav navbar-nav">
        <li class="active"><a href="#">Link <span class="sr-
only">(current)</span></a></li>
        <li><a href="#">Link</a></li>
        <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-haspopup="true" aria-expanded="false">Dropdown <span
class="caret"></span></a>
            <ul class="dropdown-menu">
                <li><a href="#">Action</a></li>
                <li><a href="#">Another action</a></li>
                <li><a href="#">Something else here</a></li>
                <li role="separator" class="divider"></li>
                <li><a href="#">Separated link</a></li>
                <li role="separator" class="divider"></li>
                <li><a href="#">One more separated link</a></li>
            </ul>
        </li>
    </ul>
    <form class="navbar-form navbar-left">
        <div class="form-group">
            <input type="text" class="form-control" placeholder="Search">
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
    </form>
    <ul class="nav navbar-nav navbar-right">
        <li><a href="#">Link</a></li>
        <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown"
role="button" aria-haspopup="true" aria-expanded="false">Dropdown <span
class="caret"></span></a>
            <ul class="dropdown-menu">
                <li><a href="#">Action</a></li>
                <li><a href="#">Another action</a></li>
                <li><a href="#">Something else here</a></li>
                <li role="separator" class="divider"></li>
                <li><a href="#">Separated link</a></li>
            </ul>
        </li>
    </ul>
</div>
</div>
</nav>

```



Slika 16. Navbar

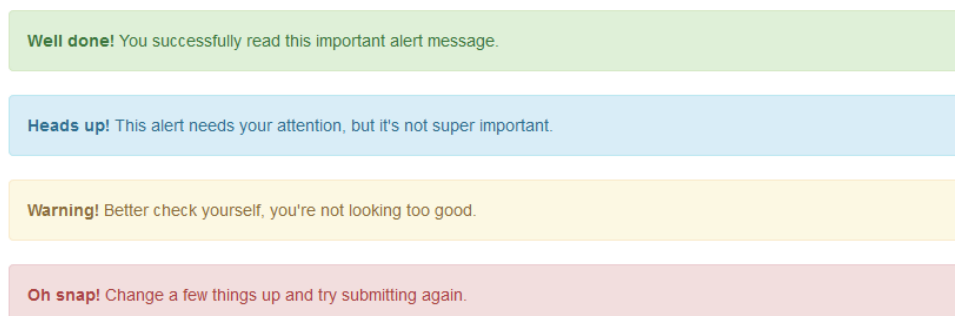
Veliko polje (Jumbotron)

Jedna od elemenata koji se koriste u specifičnim situacijama, nije standardna, ali nalazi čestu primenu je Jumbotron. To je specifično polje, koje se postavlja na vidnom delu sajta i koje je odvojeno od ostalog dela sajta (kontejner). Ukoliko se koristi `.container` klasa, polje je pravougaonog oblika sa oštrim uglovima, dok ukoliko se koristi `.jumbotron` polje je zaobljenih ćoškova. Polje može sadržati komponenti koliko želimo i koliko nam je potrebno. Suština je u uočljivosti, pošto je ono toliko veliko da odmah privuče pažnju korisniku. Zato se i koristi u situaciji kada je potrebno naglasiti neku funkcionalnost.

Uzbunjujuće poruke (Alerts)

Na slici 17. nalazi se primer uzbunjujućih poruka. One su pisane i uokvirene specifičnim bojama, pa tako imamo zelene poruke, koje su najčešće poruke potvrde dobro ispunjenih zadataka korisnika. Plave - obavestavajuće poruke. Žute - blago uzbunjujuće. Crvene – jako uzbunjujuće poruke, koje najčešće opominju na neke greške. U kodu ispod imamo primer ispisa svih poruka, s'tim što nam na mestima "...“ stoji tekst poruke.

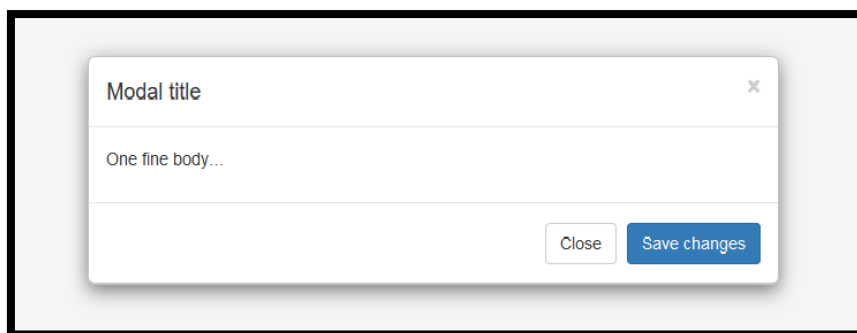
```
<div class="alert alert-success" role="alert">...</div>
<div class="alert alert-info" role="alert">...</div>
<div class="alert alert-warning" role="alert">...</div>
<div class="alert alert-danger" role="alert">...</div>
```



Slika 17. Alerts

Modalni prozori (Modals)

Modeli su prozori koji se pojavljuju, najčešće pritiskom na neko dugme. Koriste se dosta kao „login“ opcije, gde korisnik pritisne na dugme za registraciju ili logovanje i tom prilikom mu na datoj strani „iskoči“ prozor koji ima u sebi opcije koje je potrebno korisnik da ispuni i dugme za nastavak procesa. Nije moguće imati dva otvorena modalna prozora u isto vreme. Ostali deo strane najčešće potamni ili pobledi i nedostupan je korisniku dok ne popuni zadate opcije ili ne izađe iz prozora. Primer modalnog prozora se nalazi na slici 18. Preko JavaScript-a pozivamo modalni prozor sa komandom: `$('#myModal').modal(options)`



Slika 18. Modalni prozor

6. Servlet i JSP tehnologije

Zbog pomeranja celokupnog „kompjuterskog sveta“ ka internetu i ka veb pretraživačima kao osnovnim načinom za komunikaciju korisnika i sadržaja sa interneta koji ga interesuje, pojavljuje se kompjuterska kategorija veb aplikacija. Veb aplikacija je aplikacija, ili bolje rečeno program, vezan za internet koji ima određenu komunikaciju sa korisnikom. Početak interneta se vezuje za statičke veb stranice, koje bi suštinski predstavljale veb strane sa (najčešće) tekstualnim sadržajem, koji bi korisnik mogao samo da čita. Ono što bi bila svojevrsna evolucija statičke veb strane je „dinamička“ veb strana. Osnovna razlika je u komunikaciji sa korisnikom. Pojavila se potreba, ali i mogućnost da se, korišćenjem pretraživača, omogući korisniku, da na osnovu nekih svojih izbora kreira ili dobije podatke koje traži. Tako su nastale dinamičke strane. Pomoću Java programskog jezika moguće je napraviti dinamičke strane, tačnije šire dinamičke veb aplikacije. Izgled veb strane zavisi od podataka, tj. koraka koje je izabrao (uneo) korisnik. Takve strane se često menjaju, pošto zavise od unetih podataka korisnika. Kada korisnik napravi određene izbore, tj. unese podatke, aplikacija može odgovoriti sa nekim svojim funkcionalnostima, koristeći podatke sa servera ili baza podataka. Java podržava ove tehnologije kroz Servlete i Java Server Pages (JSP strane). Tipičan primer za korišćenje servleta je veb aplikacija koja pretražuje neku bazu podataka. Za komunikaciju baze podataka i aplikacije se koriste servleti. Princip rada servleta je da pročita podatke koje šalje veb pregledač u svom zahtevu, zatim izvrši procesiranje koje se od njega očekuje i formira rezultat. Na kraju šalje rezultat klijentu u obliku HTML dokumenta.

6.1. Servleti

Veb serveri i klijenti

Veb server je program koji se izvršava na nekoj fizičkoj mašini. Server obrađuje zahteve klijenata i vraća im odgovor. Svi zahtevi se obrađuju na određenom portu (ulazu, najčešće portu 80). Neki od servera su: Jakarta Apache, Microsoft IIS... Veb klijent je internet pretraživač (Internet explorer, Mozilla Firefox, Chrome, Microsoft Edge browser...) koji korisnik koristi prilikom upotrebe interneta. Veb serveri mogu da koriste Java servlete pomoću ekstenzija. Ekstenzije servera nazivamo servlet kontejneri (<https://www.tutorialspoint.com/servlets/>, 2017).

HTML i HTTP

HTML (HyperText Markup Language) jezik koji korisnici preko pretraživača (čitača) koriste za prikaz sadržaja. Serveri i klijenti imaju zajednički način komunikacije (protokol) - HTTP (Hyper Text Transfer Protocol). HTTP je protokol čiji je sloj iznad protokola TCP/IP. Klijent pošalje zahtev -HTTP **request**, a server odgovori sa HTTP **response**. Svaki par [request/response](#) je nezavisan tj. “stateless”. Nema načina da se prate ili spreče zahtevi upućenih od klijenta. Niz zahteva nazivamo sesija.

Delovi HTTP request-a su:

- HTTP Method - akcija, najčešće GET, POST, PUT, DELETE
- URL – fizička mašina kojoj se pristupa
- Parametri forme - npr. korisničko ime i lozinka koji stižu iz polja HTML od strane klijenta

URL

URL (Universal Resource Locator) jedinstvena adresa fizičke mašine (resursa) na internetu. Primer je <http://localhost:8080/FirstProject/project.jsp>

- [http://](#) predstavlja protokol
- [localhost](#) je jedinstvena adresa servera (najčešće se ime servera preslikava na IP adresu)
- [8080](#) je port na kome server osluškuje zahteve, ako se ne navede, zahtev odlazi na default port za određeni protokol (80 za HTTP, 443 za HTTPS, 21 za FTP itd.)
- [FirstProject/project.jsp](#) je resurs koji se zahteva od servera, to može biti statička Html strana, pdf, JSP, servlet, PHP fajl i sl.

GET metoda

U kodu servlet odgovara samo na GET zahtev. Prednost metode GET je jednostavnost, ne moraju se čak ni praviti polja za unos podataka, već se oni mogu dodati na kraj URL-a. Sa druge strane, nedostaci su ograničena veličina stringa koji se može poslati (do 2000 karaktera). Pored toga u čitaču se prikazuju svi uneti podaci, što nije pogodno ako se šalju osetljive informacije, jer su time faktički dostupne svima. Dok sa druge strane, metod POST nema ograničenja broja i dužine parametara (https://www.w3schools.com/tags/ref_httpmethods.asp, 2017).

HTML obrasci

HTML obrasci služe za pravljenje raznih **kontrola**, kojima se prikupljaju podaci od korisnika, slično formularima za popunjavanje. Svaka kontrola u obrascu najčešće ima ime i vrednost, ime je zadato u HTML kodu, a vrednost unosi korisnik. Ceo obrazac (HTML tag <FORM>) vezan je sa URL adresom programa koji obrađuje podatke. Obavezan atribut taga <FORM> je tip metoda koji se koristi (GET ili POST). HTTP GET dodaje string na kraj zadate URL adrese, nakon znaka pitanja. Kao npr:

```
Ime1=Vred1&Ime2=Vred2...ImeN=VredN.
```

Dok POST metoda, serveru prvo šalje URL zaglavlja HTTP zahteva i prazan red, a u sledećem redu se šalje string sa parametrima. Kao npr:

```
<form method="get" action="FormServlet">
<input type="text" name="tekst">
<input type="submit" value="Poslati">
</form>
```

HTTP response

HTTP response ili odgovor servleta. Najvažniji delovi HTTP odgovora su:

- Status Code – ceo broj koji označava uspešnost zahteva. Česti statusni kodovi su 200 (uspešno), 404 (resurs nije pronađen) i 403 (pristup zabranjen)
- Content Type - text, html, image, pdf itd. Poznat i kao MIME tip (Multipurpose Internet Mail Extensions).
- Content - stvarni podaci koje klijent prikazuje korisniku

Najčešće korišćeni Status kodovi i njihova značenja:

```
"200" - OK,
"201" - Created (kreiran),
"202" - Accepted (prihvaćen),
"204" - No Content (bez sadržaja),
"301" - Moved Permanently (pomeren za stalno),
"302" - Moved Temporarily (pomeren privremeno),
"304" - Not Modified (nije promenjen),
"400" - Bad Request (loš zahtev),
"401" - Unauthorized (neodobreno/bez ovlašćenja) ,
"403" - Forbidden (zabranjeno),
"404" - Not Found (nije pronađen),
"500" - Internal Server Error (unutrašnja greška servera),
"501" - Not Implemented (nije izvršen),
"502" - Bad Gateway (loš izlaz),
"503" - Service Unavailable (Servis je nedostupan).
```

Server pošalje MIME tip klijentu da bi mu pomogao da prikaže podatke korisniku. Najčešće se koriste tipovi text/html, text/xml, application/xml. Primer HTTP response-a :

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2016 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2016 19:15:56 GMT
Content-Length: 88
Content-Type: text/html - OVO JE MIME
Connection: Closed

<html>
<head>
<meta http equiv="Content-Type" content="text/html">
</head>
<body>
<h1>Dobar dan!</h1>
</body>
</html>
```

Servleti su Java programi. Izvršavaju se na veb serveru i rade kao sloj između zahteva koji stižu iz čitača veba (browser) ili nekog drugog HTTP klijenta, i baza podataka ili aplikacija na HTTP serveru. Servleti se izvršavaju u veb serveru tj. servlet kontejneru. Koriste kao kontroleri u MVC⁴ arhitekturi. Trenutno aktuelna verzija je Servlet API 3.1 (i preazak na 4).

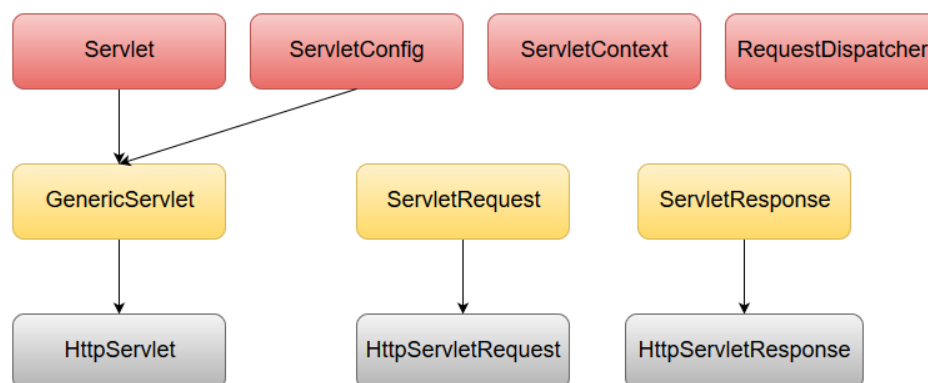
Servlet kontejneri

Servlet kontejnere drugačije zovemo i servlet engines ili application servers. Neki od server kontejnera su Glassfish, Tomcat, Resin, JBoss... Servlet kontejner može da bude proširenje veb servera tj. da radi i kao HTTP server i kao server aplikacija, ali i da bude server aplikacija.

U trenutku kada servlet kontejner dobije zahtev, kreira dva objekta tipa `HttpServletRequest` i `HttpServletResponse`. Potom pronalazi odgovarajući servlet na osnovu URL-a i kreira **nit** za zahtev. Nakon toga poziva metod `service()` servleta, koji u zavisnosti od HTTP metode koja je korišćena poziva `doGet()` ili `doPost()`. Metode servleta generišu dinamičku stranu i upisuju je u odgovor. Kada se **nit** servleta izvrši, kontejner konvertuje odgovor u HTTP response i šalje ga klijentu (<https://dzone.com/articles/what-servlet-container>, 2016).

Zadaci Servlet kontejnera:

- Podrška za komunikaciju između veb servera i servleta/JSP
- Upravljanje životnim ciklusom servleta (kreiranje, inicijalizacija, pozivanje metoda i uništavanje)
- Podrška za rad sa više niti
- Prevođenje JSP strana u servlete
- Upravljanje resursima mašine na kojoj se nalazi, optimizacija memorije, bezbednosna podešavanja, podrška za izvršavanje više aplikacija itd.



Slika 19. Servlet API

⁴ MVC arhitektura – softverski patern koji razdvaja prikaz informacija od korelacije korisnika sa tim informacijama (model, view, control). **View** - obezbeđuje korisniku interfejs pomoću koga korisnik unosi podatke i poziva odgovarajuće operacije koje treba da se izvrše nad model-om, **Controller** - osluškuje i prihvata zahtev od klijenta za izvršenje operacije, **Model** - predstavlja stanje sistema koje mogu promeniti operacije model-a.

Osnovna struktura servleta

Da bi bila servlet, potrebno je da Java klasa implementira interfejs `Servlet` iz paketa `javax.servlet`, ili da bude izvedena iz klase `HttpServlet`, koja već implementira taj interfejs. Neophodno je “nadjačati” metode `doGet()` ili `doPost()`, u zavisnosti od toga kako se podaci šalju, GET ili POST metodom. Ako isti servlet treba da obrađuje i GET i POST metode, onda `doGet` treba da poziva `doPost` (ili obrnuto) ili da oba metoda pozivaju `processRequest()` (Gary Cornell, Cay S. Horstmann - Core Java™ (Volume I, Fundamentals, Ninth Edition), Michigan, 2012.).

Osnovni zadaci servleta su redom: čitanje podataka koje je poslao klijent, traženje svih informacija o zahtevu koje su ugrađene u HTTP request, generisanje rezultata, formatiranje rezultata unutar dokumenta, podešavanje odgovarajućih parametara za HTTP response, slanje povratnog dokumenta klijentu.

Kada se servlet prvi put kreira, poziva se njegov metod `init()`. On treba da sadrži kod koji se izvršava samo jednom. Ovaj metod je potrebno “nadjačavati” jedino ako je za servlet potrebna još neka inicijalizacija. Posle toga, svaki zahtev od klijenta stvara novu nit koja poziva metod `service()`, ranije kreirane instance servleta. Metod `service()` ne treba nadjačavati. Više konkurentnih zahteva obično kreiraju više niti koje istovremeno pozivaju metod `service()`. Metod `service()` potom poziva metode `doGet()` ili `doPost()`, u zavisnosti od tipa primljenog HTTP zahteva. Nadjačavanjem metoda `doGet()` ili `doPost()` postiže se željeno ponašanje servleta. Kada bude potrebno da server obriše instancu servleta, poziva se njegov metod `destroy()`. Ovaj metod obično obavlja oslobađanje resursa koje je servlet zauzimao (zatvaranje datoteka, prekid konekcije sa bazom i sl.)

ServletConfig

Metod `getServletConfig()` interfejsa `Servlet` poseduje objekat `ServletConfig`. Svaki servlet ima sovoj objekat `ServletConfig` koji instancira servlet kontejner. Pomoću metode `ServletConfig.getInitParameter()` čitaju se početni parametri iz deployment descriptor-a (fajla `web.xml`). Metod `getInitParameterNames()` vraća imena početnih parametara definisanih za servlet (<http://www.javatpoint.com/servletconfig>, 2017).

ServletContext

Interfejs `javax.servlet.ServletContext` poseduje metode koje servlet koristi za komunikaciju sa veb serverom. Objekat `ServletContext` je sadržan u objektu `javax.servlet.ServletConfig`. Dobija pozivom metode `getServletContext()`. `ServletContext` je unikatni objekat dostupan svim servletima unutar veb aplikacije. Pomoću objekta `ServletContext` servlet može da :

- Postavlja i čuva attribute kojima će pristupati drugi servleti iz istog konteksta
- Vodi log (dnevnik događaja)
- Dobija URL reference na resurse
- Dobija attribute za inicijalizaciju - to su parametri povezani sa servletom, a ne sa pojedinačnim zahtevom
- Dobija MIME - tip fajlova
- Dobija informacije o servlet kontejneru, kao što su njegovo ime i verzija.

Pravo ime ovog interfejsa trebalo bi zapravo da bude **ApplicationContext**, jer je on zajednički za aplikaciju tj. nije specifičan ni za jedan servlet. Ne treba ga mešati sa servlet kontekstom koji se prosleđuje u URL-u da bi se pristupilo veb aplikaciji (<http://www.javatpoint.com/servletcontext>, 2017).

Kada servlet primi zahtev od klijenta, on tada prihvata dva objekta. **ServletRequest**, koji održava komunikaciju od klijenta do servera i **ServletResponse**, koji održava komunikaciju od servera ponovo ka klijentu. **ServletRequest** and **ServletResponse** su interfejsi definisani u paketu `javax.servlet`.

HttpServletRequest

Klasa **HttpServletRequest** ugrađena u interfejs `ServletRequest`. Ona omogućuje servletu da pristupi parametrima koje pošalje klijent, protokolu koji koristi klijent, nazivu udaljenog računara sa koga je stigao zahtev koji je server primio, zatim broju porta na kome sluša server i informacijama u zaglavlju HTTP zahteva.

Klasa prihvata argumente koje je klijent poslao, kao deo svog zahteva preko metoda `getParameter()`, koji vraća vrednost imenovanog parametra. Ako parametar ima više vrednosti, koristi se metod `getParameterValues()`, `request.getParameterNames()` ili `request.getParameterMap()`, koji daju Enumeration ili Map objekata od poslanih elemenata. Najčešće se koriste za debugovanje.

Čitanje parametara iz zahteva

Komandom `request.getParameter("imeParam")` dobija se vrednost prvog elementa koji se zove imeParam. Ponaša se isto za GET i za POST zahteve. Rezultat je null ako ne postoji takav parametar u elementima HTML obrasca.

Komandom `request.getParameterValues("imeParam")` dobija se niz vrednosti za sve elemente koji se zovu imeParam. Dobija se niz sa jednim elementom, ako se ime pojavljuje samo jednom. Rezultat je null ako ne postoji takav parametar u elementima HTML obrasca.

Čitanje HTTP zaglavlja

HTTP zaglavlje se čita pomoću metode `getHeader()` klase `HttpServletRequest`. Metoda vraća String ako zahtev sadrži traženo zaglavlje, ili null u suprotnom. Imena zaglavlja nisu case sensitive⁵, svedeno je da li se koriste `request.getHeader("Connection")` ili `request.getHeader("connection")`.

⁵ case sensitive – osetljivost na velika i mala slova, za neki unos teksta kažemo da je case sensitive ako postoji razlika ukoliko unesete „A“ ili „a“. Za one koji to nisu, svedeno je da li unosite velika ili mala slova.

Čitanje HTTP zahteva

Za najčešće korišćena zaglavlja objekat `HttpServletRequest` ima posebne metode:

- `getCookies()` – vraća sadržaj zaglavlja `Cookie`, parsiran kao niz objekata `Cookie`
- `getAuthType()` i `getRemoteUser()` – daju sastavne delove zaglavlja `Authentication`
- `getContentLength()` – vraća `int` dužinu zaglavlja `Content-Length`
- `getContentType()` – vraća `String` vrednost zaglavlja `Content-Type`
- `getMethod()` – vraća tip metoda (`GET` ili `POST`)

HttpServletResponse

`HttpServletResponse` omogućuje servletima da generišu odgovor na zahtev. Pritom i nudi metode za definisanje sadržaja odgovora i tipa podataka (`MIME`) i obezbeđuje izlazni tok tj. objekte. `ServletOutputStream` i `Writer` preko kojih servlet šalje podatke klijentu. Od HTTP response metoda postoje:

- `String encodeURL(java.lang.String url)` - kodira zadati URL dodavanjem ID sesije, ako je potrebno
- `void sendRedirect(String location)` - privremeno preusmerava klijenta na URL lokaciju zadatu kao parametar
- `void setStatus(int sc)` - postavlja statusni kod za response

Postoje dve mogućnosti za vraćanje podataka klijentu: `response.getWriter()` - vraća `Writer`, upotrebljava se kada su podaci koji se vraćaju klijentu tekstualnog tipa i `response.getOutputStream()` - vraća `ServletOutputStream`, koristi se za binarne podatke. Pozivanjem metode `close()` ovih objekata nakon slanja odgovora omogućava serveru da zna kada je odgovor kompletn i zatvoren.

Da bi izgenerisali HTML kod potrebno je obavestiti čitač Veb-a da se šalju podaci koji su tipa HTML. To radimo sa `response.setContentType("text/html")`. Menja se naredba **println** da bi se dobila korektna Veb stranica. Naredbe **print** koriste HTML tagove. Za primer servleta koji generiše HTML koristićemo isti primer kao ranije.

Preusmeravanje

Prilikom izvršenja određenih programa servleti po nekad moraju da preusmere obradu klijentskog zahteva na neki drugi URL, servlet ili JSP stranu. To se može uraditi pomoću dve metode:

- `HttpServletResponse.sendRedirect()`
- `RequestDispatcher.forward()`

JavaScript takođe može omogućiti redirekciju na strani klijenta.

RequestDispatcher

RequestDispatcher je interfejs koji se koristi za preusmeravanje zahteva na drugi resurs (HTML, JSP ili drugi servlet unutar istog konteksta). Referenca na objekat RequestDispatcher može se dobiti preko metode `ServletContext.getRequestDispatcher(...)`. U zagradi se nalazi string putanja. Putanja mora da počinje sa "/" i tumači se relativno u odnosu na koren konteksta. RequestDispatcher metode koje koristimo:

- `void forward ( ServletRequest request, ServletResponse response)` - prosleđuje zahtev od servleta ka drugom resursu (servletu, JSP ili HTML strani) na serveru
- `void include (ServletRequest request, ServletResponse response)` - uključuje sadržaj resursa (servleta, JSP ili HTML strane) u response

Kod od metode `forward()` RequestDispatcher-a preusmeravamo na stranu servera preko originalnih objekata request i response, dok metoda `response.sendRedirect()` šalje request sa novom URL adresom nazad klijentu, koji se zatim povezuje sa tim URL-om, što znači da se pravi novi par objekata request/response. `SendRedirect()` je znatno sporiji od metode `forward()`, ali omogućuje preusmeravanje na bilo koji URL, dok `forward()` radi samo unutar iste Veb aplikacije (<http://docs.oracle.com/javaee/6/api/javax/servlet/RequestDispatcher.html>, 2011).

Atributi servleta se koriste za komunikaciju između servleta tj. na nivou cele aplikacije. Postoje tri oblasti važenja: request scope, session scope i application scope. Interfejsi ServletRequest, HttpSession i ServletContext imaju metode get/set/remove atributa iz zahteva, sesije i aplikacije. Atributi servleta su različiti od početnih parametara definisanih u web.xml za objekte ServletConfig ili ServletContext.

Servlet deployment

Servlet nije kompletna Java aplikacija, već mora da bude deo veb aplikacije kojom upravlja servlet kontejner. Korišćenje servleta za generisanje dinamičkih odziva obuhvata programiranje servleta i njihovo postavljanje u veb aplikaciju. Taj postupak nazivamo deploy.

Korenski direktorijum (/) je polazna tačka Veb aplikacije, zove se kontekst. On sadrži sve resurse veb aplikacije (statičke kao što su html i jsp strane, slike i slčno, ali i dinamičke). /WEB-INF/ sadrži Web Application Deployment Descriptor (tj. konfiguracioni fajl web.xml) koji opisuje deploy. Za razliku od statičkih resursa, servlet se ne postavlja direktno u korenski direktorijum, već unutar direktorijuma /WEB-INF/classes, zajedno sa svim drugim kompajliranim Java klasama.

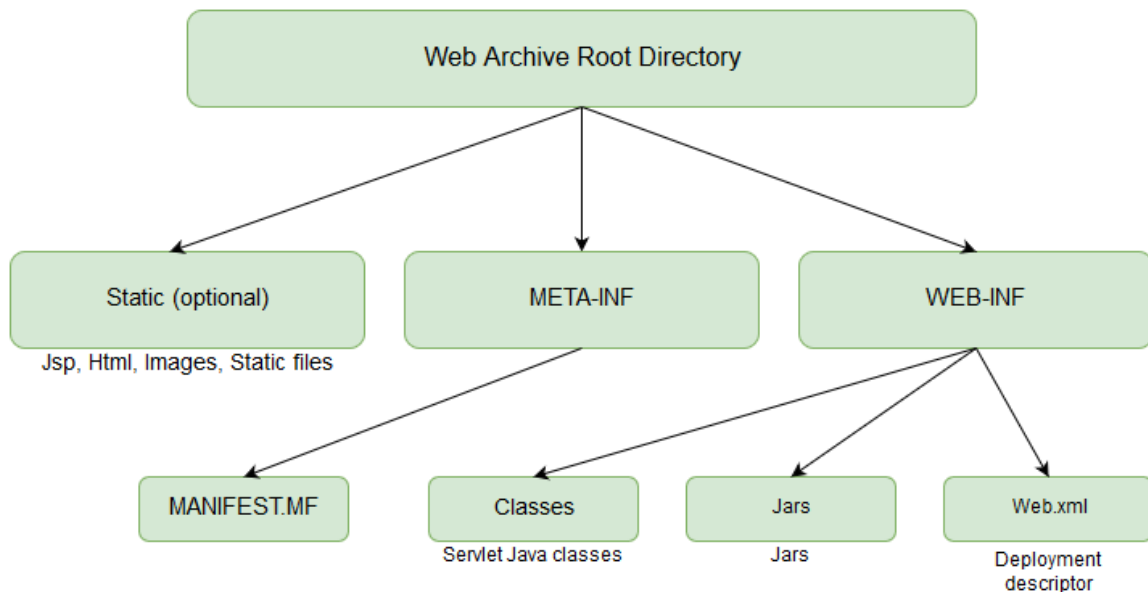
Pozivanje servleta

Servlete možemo pozivati direktno upisom URL putanje unutar čitača. Format URL putanje zavisi od servera koji se koristi. Pristup servletu - opšti šablon `http://[domen]/[context]/[servlet]/[ServletskaKlasa]`. Npr. za Veb server Jakarta Tomcat kod koga je u direktorijumu TOMCAT_HOME/webapps instalirana aplikacija (kontekst) servlet intro, poziv servleta HelloWorldServlet bi izgledao ovako:

`http://localhost:8080/servletintro/servlet/HelloWordServlet.`

Struktura Java Veb aplikacije

Na Slici 20. prikazana je struktura Veb aplikacije. Njena sadržina je opisana ranije.



Slika 20. struktura Veb aplikacije

Deployment descriptor

Umesto direktnog pozivanja servleta, u konfiguracionom fajlu web.xml zadaje se preslikavanje servleta u URL koji definiše administrator

Primer:

- Servletska klasa: HelloServlet.class
- Kontekst aplikacije <http://localhost:8988/servletintro/>
- Direktno pozivanje <http://localhost:8988/servletintro/servlet/HelloWorld>
- Preslikavanje putem fajla web.xml <http://localhost:8988/servletintro/hello>

WAR (Web Application Archive)

Java Enterprise Edition je definisala standardane principe za razijanje veb aplikacija njihovim smeštanjem u WAR archive. Potrebno je da sve komponente veb aplikacije budu smeštene u fajl sa ekstenzijom .war. On ima isti format kao .jar fajlovi. WAR archive se mogu napraviti programom jar, kao i obične archive koje sadrže mnogo fajlova u komprimovanom obliku kompatibilnom sa zip fajlovima. Struktura foldera u WAR arhivi treba da bude sledeća:

- Prvi nivo sadrži resurse koji se i inače stavljaju u korenski direktorijum aplikacije (HTML i JSP strane)
- Poddirektorijum META-INF sadrži fajl manifest.mf sa informacijama o fajlovima u arhivi
- Direktorijum WEB-INF sadrži deployment descriptor (web.xml)
- Poddirektorijum classes sadrži kompajlirane servlete i ostale Java klase
- U poddirektorijum lib smeštaju se dodatne jar archive

Upravljanje sesijom

Veb serveri i Protokol HTTP su oblika stateless, nisu u stanju da prepoznaju da dva uzastopna zahteva stižu od istog klijenta. Potrebno je u aplikaciji da prepoznamo da zahtevi stižu od istog klijenta u određenom vremenskom intervalu, to se naziva **sesija**. Jedini način da očuvamo sesiju je da se između klijenta i servera pri svakoj request/response komunikaciji razmenjuje jedinstvena informacija o sesiji (id sesije). Postoji više načina da se upravlja sesijom:

- Autentifikacija korisnika - na strani za logovanje tražimo od korisnika autentifikaciju i onda informacije o korisničkom imenu prenosimo između klijenta i servera da bi održali sesiju. Ovaj metod nije pogodan ako se isti korisnik prijavi istovremeno sa različitih čitača.
- Skriveno HTML polje - kada korisnik počne da koristi aplikaciju, napravimo jedinstvenu vrednost skrivenog polja i šaljemo je. Ovaj metod ne može da se koristi sa linkovima jer **form** mora da bude poslat sa skrivenim poljem kad god se generiše zahtev od klijenta. Nije bezbedno, jer se vrednost skrivenog polja može pročitati iz HTML-a.
- URL Rewriting - parametar ID sesije se dodaje svakom paru request/response. Prilično je komplikovano jer moramo stalno da pratimo njegovu vrednost.
- Cookies - kolačić je informacija koju Veb server šalje u zaglavlju odgovora, a čuva se kod klijenta. Kada klijent ponovo pošalje zahtev, zaglavlju zahteva se dodaje kolačić i on se koristi za praćenje sesije. Klijent može i da isključi podršku za kolačiće.
- Session Management API - podržava dva prethodna rešenja

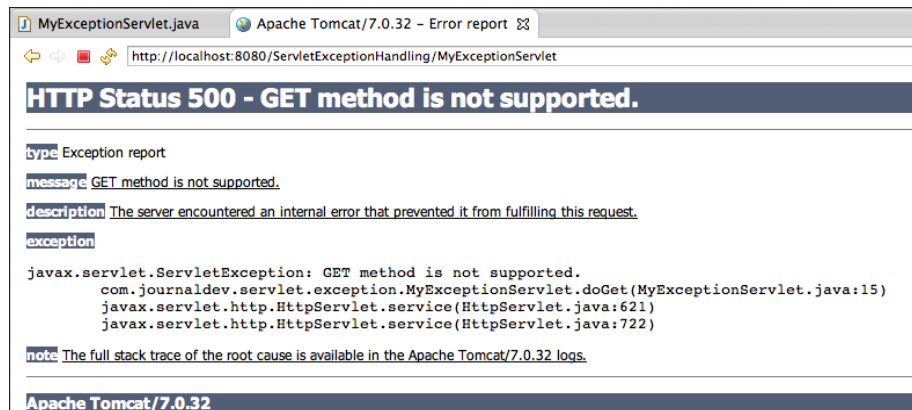
Sesija je zajednička za sve servlete u aplikaciji kojoj klijent pristupa. Objektu klase `javax.servlet.http.HttpSession` koja identifikuje sesiju pristupa se preko metoda `request.getSession()`.

Upravljanje greškama u servletima

Čitač razume samo HTML, kada aplikacija napravi izuzetak, servlet kontejner ga obrađuje i generiše HTML odgovor. Logika obrade je specifična za servlet kontejner. Problem koji se javlja je zato što prikaz greške korisniku ne znači ništa. Pored toga prikazuju se klase i detalji o serveru što nije dobro sa tačke gledišta bezbednosti. Izuzetke generisane u servletima treba “uhvatiti” i obraditi, jer će u suprotnom klijent videti stack trace⁶ u prozoru čitača. Servlet API podržava servlete Exception i Error Handler, koje treba definisati u fajlu web.xml. Svrha ovih servleta je da obrade izuzetak i prikažu HTML response koji je koristan. Treba napraviti sistem koji će prikazati stranu sa linkom na početnu stranu ili neke detalje na osnovu kojih će korisnik zaključiti da nešto nije u redu. Error-page treba zadati u web.xml-u. Svaki element error-page treba da ima ili error-code ili exception-type. Servlet koji obrađuje grešku zadaje se u elementu location. JSP strana takođe može da bude exception handler, samo treba navesti lokaciju jsp fajla umesto servleta.

⁶ Stack trace – spisak metoda koje su pozvane u trenutku kada je nastao prekid programa izazvan nekim izuzetkom

Ako sve moguće izuzetke treba obraditi u jednom exception handleru, tip izuzetka treba da bude Throwable. Ako ima više error-page elemenata, npr. jedan za Throwable i drugi za IOException, a desi se izuzetak tipa IOException, on će biti obrađen error handlerom za IOException. Na Slici 21. nalazi se primer izuzetka koji Veb čitač dobija – Error page.



Slika 21. Error page

Servlet filteri

Filter je vrsta servleta koja dinamički “presreće” zahteve pre nego što stignu do servera, i odgovore pre nego što stignu do klijenta. Zatim transformiše informacije u njima. Velika prednost filtera je to što se mogu dodati u postojeće aplikacije, a da se one ne moraju ponovo kompajlirati (pluggable). Implementira se preko interfejsa javax.servlet.Filter. Umesto u metodama doGet() ili doPost(), filter svoje zadatke obavlja u metodi doFilter(). Filteri se mogu koristiti za nekoliko važnih uloga:

- Pakovanje zadataka koji se često obavljaju u klasu koja se može uvek pozvati
- Formatiranje podataka koji se šalju klijentu
- Autorizacija i blokiranje zahteva
- Logging i auditing (npr. beleženje request parametara u log fajl)
- Komprimovanje⁷ response podataka koji se šalju klijentu

Filteri i servleti su nezavisni, filteri se dodaju i uklanjaju samo iz fajla web.xml. Filteri se mogu povezivati u FilterChain-u, a servlet kontejner konstruiše lanac filtera prema redosledu kojim se filteri pojavljuju u deployment descriptoru (fajlu web.xml). Da bi filter mogao da pristupi inicijalizacionim parametrima, prosleđuje mu se objekat FilterConfig u njegovoj metodi init(). Filter pristupa objektu ServletContext preko objekta FilterConfig i tako učitava sve resurse potrebne za zadatke filtriranja. Kada filter obavi svoj zadatak, poziva metod doFilter() sledećeg filtera u lancu. Ako se radi o poslednjem filteru u lancu, servlet kontejner će pozvati resurs na kraju lanca, tj. običan servlet. Sledi primer funkcije doFilter sa izuzecima.

```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws ServletException, IOException {
```

⁷ Komprimovanje - uklanjanja nepotrebnih podataka

```

HttpServletRequest request = (HttpServletRequest) req;
String requestURI = request.getRequestURI();

if (requestURI.startsWith("/Check_License/Dir_My_App/")) {
    String toReplace =
requestURI.substring(requestURI.indexOf("/Dir_My_App"),
requestURI.lastIndexOf("/") + 1);
    String newURI = requestURI.replace(toReplace, "?Contact_Id=");
    req.getRequestDispatcher(newURI).forward(req, res);
} else {
    chain.doFilter(req, res); //pozovi sledeći filter u lancu
}
}

```

Filteri dodaju aplikaciji u razvojnoj fazi, jedan filter je moguće primeniti na jedan servlet ili više njih. U web.xml moraju se dodati odgovarajući tagovi za mapiranje filtera.

6.2. Java Server Pages (JSP)

Java Server Pages razdvajaju obrade zahteva i poslovne logike od prezentacije. Posao se tako deli, što ima više prednosti. Odvajaju se poslovi, pa se tako java programeri bave kontrolom aplikacije, a veb dizajneri prave interfejs. Različiti delovi aplikacije razvijaju se nezavisno, pa je moguće promeniti poslovnu logiku bez promene korisničkog interfejsa. U odnosu na druge jezike, Java je moćniji za dinamički deo aplikacije. Java ima mogućnost izvršavanja na većem broju servera i operativnih sistema i bolju podrška alata. Nedostaci su mu slaba podrška kod hosting provajdera u Srbiji. Kod Java-e je složenija tehnologija i zahteva više vremena za savladavanje.

JSP strana je isto što i HTML strana uz dodatak JSP elementima koji generišu delove strane različite za svaki request. Servlet kontejner prevodi u pozadini svaku JSP stranu u servlet kada se ona prvi put pozove, nakon toga kompajlira se samo ako je promenjena. Servleti i JSP se dopunjavaju. Servleti se koriste kao kontroleri na serverskoj strani i koriste se za komunikaciju sa klasama modela, a JSP strane će biti sloj prezentacioni sloj. Svi bitni JSP interfejsi i klase definisani su u paketu `javax.servlet.jsp`. Expression Language API nalazi se u paketu `javax.servlet.jsp.el`, dok su JSP biblioteke tagova definisane u paketu `javax.servlet.jsp.tagext` (<http://www.oracle.com/technetwork/articles/java/servlets-jsp-140445.html>, 2003).

Životni tok JSP strane kontroliše servlet kontejner. Organizacija JSP koda ima na dve opcije, upisati veliku količinu Java koda direktno u JSP stranice ili upisati taj kod u odvojenu Java klasu i jedan red koda u JSP stranicu koja ga poziva. Druga opcija mnogo bolja iz tri razloga:

- Razvoj - odvojena klasa se piše u Java okruženju tj. editor-u ili IDE-u
- Debugovanje – ako se pojave sintaksne greške, primetiće se odmah tokom procesa kompajliranja i usput i rešavati
- Višestruko korišćenje - može se koristiti ista klasa za više stranica kao template

U strukturi Veb aplikacije JSP fajlovi se mogu postaviti bilo gde u WAR arhivu, ali ako ih postavimo u folderu WEB-INF, ne može da im se pristupa direktno iz klijenta. Obično se stavljaju direktno u korenski direktorijum WAR arhive.

HTML kod:

- `<H1>Neki naslov</H1>` - Šalje se dalje klijentu. Prevodi se u sledeći servlet kod
`out.print("<H1>Neki naslov</H1>")`

HTML Komentari:

- `<!-- Komentar -->` - Isto kao u HTML-u: šalje se dalje klijentu

JSP komentar:

- `<%-- Komentar --%>` - Ne šalje se klijentu

Vrste dinamičkih elemenata

Izrazi (expressions):

- `<%=java_izraz%>`
- `<%= new java.util.Date() %>`

Skriptleti (scriptlets):

- `<%java_kod%>`
- `<% for (int i = 0; i < 10; i++) ... %>`

Deklaracije (declarations):

- `<%!java_deklaracija%>`
- `<%! int a; %>`

Direktive (directives):

- `<%@ direktiva atribut="..."%>`
- `<%@ page contentType="text/plain" %>`

JSP izrazi

Prečica za naredbu `out.print()`, gde je `out` objekat klase `JspWriter`, slično `PrintWriteru` u servletima.

```
<html>
...
<h3>Hello, <%=username%></h3>
Today is <%=new java.util.Date()%>
...
</html>
```

JSP direktive

Direktive se koriste za zadavanje specijalnih instrukcija servlet kontejneru dok se JSP strana prevodi u servlet.

- `page:`
 - `import` – za unos paketa
 - `contentType` – podešava `ContentType` odgovora
- `include:`
 - uključuje zadatu stranicu u postojeću
- `taglib:`
 - deklariše biblioteku tagova (`custom tag library`) koja se koristi na strani

Primer JSP direktive:

```
<%@ page attribute1="value1" attribute2="value2" attribute3=... %>
```

Direktiva include

Omogućuje programerima da uključuju sadržaj jednog fajla (JSP, HTML) u drugi. Primer: `<%@ include file="localURL" %>`. Nema ograničenja u broju strana koje se mogu uključivati, dozvoljeno je i ugnježđivanje tj. više strana jedne u druge. U atributu **file** mogu se koristiti i relativne i apsolutne putanje, obično se zadaje relativna putanja u odnosu na stranu gde se koristi. Preporučljivo je da se strane (fragmenti) koje nisu samostalne, već se uključuju u druge strane, čuvaju sa ekstenzijom .jspx ili .jsf. Kada se koristi komanda include, sadržaj uključene strane se direktno umeće (kopira) na mestu direktive. Ako se strane uključene pomoću direktive include izmene, to neće prouzrokovati ponovno kompajliranje strane u koju su uključeni, jer servlet kontejner ne prati zavisnosti između fajlova (<http://www.javatpoint.com/jsp-include-action>, 2017).

Direktiva taglib

Proizvodi obaveštenja za servlet kontejner da se strana oslanja na posebnu biblioteku tagova. Primer direktive taglib: `<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>`

Implicitni JSP objekti

JSP kontejner poseduje određeni broj implicitnih objekata koje nudi autorima JSP strana, ovi objekti su automatski dostupni. Devet implicitnih objekata mogu se podeliti u četiri kategorije:

- objekti povezani sa servletom JSP strane
- objekti povezani sa ulazom i izlazom strane
- objekti koji nude informacije o kontekstu u kome se izvršava JSP strana
- objekti koji su rezultat grešaka

Upravljanje greškama na JSP stranama

Da bismo mogli da upravljamo greškama koje baca JSP strana, potrebno je da napravimo error page. Zatim da postavimo njen atribut **isErrorPage** na true u direktivi page. Najčešće se koristi ista error page za sve JSP strane, umesto da se konfigurišu sve JSP strane pojedinačno, bolje je definisati error page u fajlu web.xml pomoću elementa error-page. Na sledećem primeru koda je napisan web.xml file.

JSP akcije

JSP akcije se koriste za prenos kontrole između strana. Podržavaju specifikaciju Java apleta koji je potpuno nezavisan od Web browser-a. Akcije omogućavaju JSP stranama da komuniciraju sa Java Beans komponentama koje se nalaze na serveru. Akcija `<jsp:include>` omogućuje ugrađivanje sadržaja koji je generisan u drugom dokumentu u izlaz tekuće strane.

Primer koda: `<jsp:include page="localURL" flush="flushFlag" />`
Atribut strana može biti statički dokument, servlet ili neka druga JSP strana.

Uključena strana dobija Request. Zatim JSP kontejner izlaz te strane u originalnu stranu i nastavlja sa obradom strane na kojoj se nalazi. To se dešava tokom obrade zahteva. Za razliku od direktive `include` koja deluje u trenutku kada se strana prevodi u servlet, akcija `<jsp:include>` umeće izlaz uključene strane, a ne njen sadržaj.

Akcija `<jsp:forward>` prenosi za stalno kontrolu iz JSP strane na drugu lokaciju unutar iste Veb aplikacije. Primer koda: `<jsp:forward page="localURL"/>`. Sav sadržaj koji je izgenerisan na datoj strani se odbacuje, a obrada zahteva počinje od početka na novoj lokaciji zadatoj u atributu strani. Pošto je objekat request zajednički za originalnu stranu i onu na koju se prosleđuje, svi request parametri biće dostupni i na strani na koju se prosleđuje. Na slici 40. nalazi se primer akcije `<jsp:forward>`.

Java Beans

Java Bean je klasa-objekat koji izlaže podatke preko properties-a. JSP kontejner pregleda bean i otkriva njegova svojstva preko mehanizma koji se zove introspekcija. Da bi neka klasa bila bean potrebno je da ispuni nekoliko uslova. Mora imati default konstruktor (bez argumenata). Ne sme da ima public properties (podatke članove), već se njima pristupa preko getera i setera.

Bean tagovi

Tag `<jsp:useBean>` pronalazi i instancira Java Bean tj. kreira objekat bean ako već ne postoji. Bean se može pozivati na strani pomoću vrednosti dodeljene atributu. Akcija `<jsp:setProperty>` postavlja vrednost za bean property pomoću njegovog metoda set. Akcija `<jsp:getProperty>` čita vrednost za bean property pomoću njegovog metoda get. Osim getera i setera, bean može da sadrži bilo koji drugi metod, bitno je da on poštuje sintaksu getera da bi mu se pristupilo iz strane kao da je property.

Expression Language (EL)

JSP od verzije 2.0 nudi sintaksu za ugrađivanje rezultata jednostavnih izraza u stranu i naziva ga Expression Language (EL). Koristi se sledeća sintaksa `${izraz}`. Pre verzije JSP 2.0, da bi se očitala vrednost iz mape na JSP strani, pisalo se:

```
<%= (Map) pageContext.findAttribute("mojaMapa")).get("nekiKljuc") %> ,
```

EL ekvivalent prethodnog primera je `${mojaMapa.nekiKljuc}`.

Za pristup objektu koristi se sintaksa `${objekat}`. Operator tačka se koristi za pristup vrednostima atributa: `${prviObjekat.drugiObjekat}`, na primer: `${requestScope.zaposleni.adresa}`. Osim poslednjeg dela EL, svi objekti moraju da budu ili Map ili Java Bean, u prethodnom primeru requestScope je Map, a zaposleni je Java Bean ili Map. Ako scope nije naveden, JSP EL će tražiti navedeni atribut u page, request, session i application scope. Operator `[]` je snažniji od tačke i može se koristiti za pristup listama i nizovima. Na primer: `${nizIliLista[0]}`. `${mojaMapa[izraz]}` ako izraz u `[]` nije String, tumači se kao EL. Aritmetički operatori `+`, `-`, `*`, `/` ili `div`, `%` ili `mod` za jednostavna izračunavanja su podržani. Logički operatori su `&&` (and), `||` (or) i `!` (not). Relacioni operatori su `==`, `!=`, `<`, `>`, `<=` i `>=`. EL izrazi u JSP strani se mogu isključiti postavljanjem atributa `isELIgnored` na true u

direktivi stani. EL se može koristiti za čitanje atributa, parametara, kolačića i sl., ali ne i za njihovo postavljanje. Osim objekta `pageContext`, implicitni EL objekti nisu isti kao JSP implicitni objekti. JSP EL je NULL friendly, tj ako traženi atribut nije nađen ili izraz vraća null, ne baca se izuzetak. U aritmetičkim operacijama, EL tretira null kao 0, a u logičkim kao false (<http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html>, 2013).

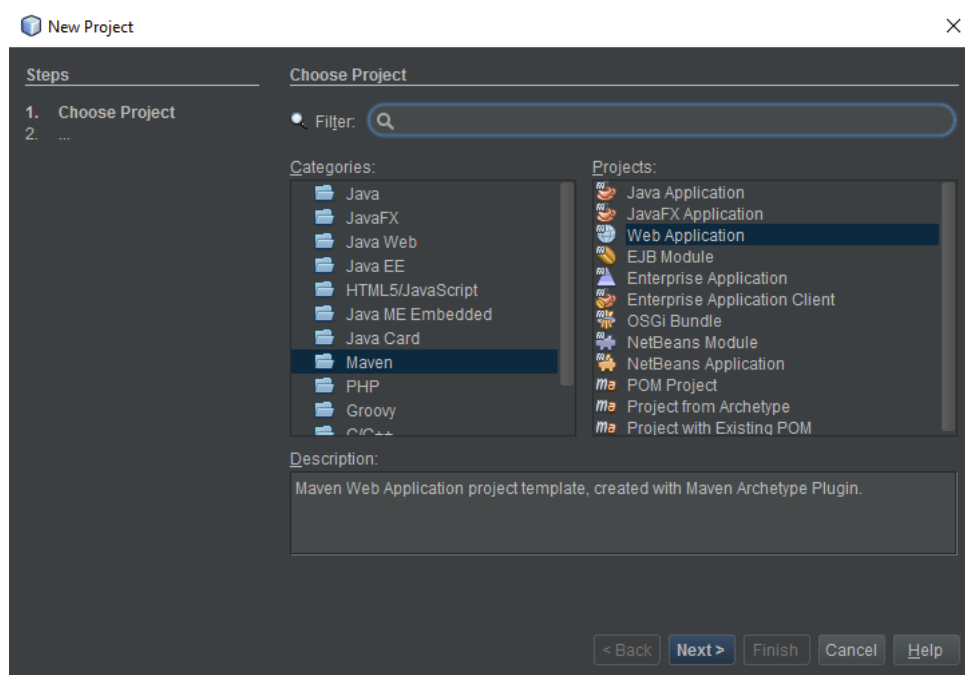
JSTL

EL i akcije su ograničeni, pomoću njih se ne može npr. iterirati kroz kolekciju. JSP Standard Tag Library (JSTL) je standardna biblioteka tagova za kontrolu ponašanja JSP strane, iteraciju i kontrolu, XML i SQL. JSTL je deo JavaEE API-ja i uključen je u većinu servlet kontejnera, ali da bismo koristili JSTL u stranama, moramo da preuzmemo jar specifičan za servlet kontejner koji koristimo. Jar se smešta u direktorijum projekta WEB-INF/lib (https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm, 2017).

Treba izbegavati skriptlete u JSP stranama i umesto njih koristiti EL, akcije i JSTL. U JSP stranama izbegavati poslovnu logiku, one treba da služe samo za generisanje odgovora klijentu. Zbog boljih performansi isključiti sesiju u JSP stranama gde nije potrebna. Direktive `page` i `taglib` koristiti na početku strane zbog čitljivosti. Koristiti direktivu `jsp include` ili akciju `include` u skladu sa potrebama, direktiva `include` je dobra za statički sadržaj, a akcija `include` za dinamički i sadržaj koji se umeće za vreme izvršavanja. Upravlјati greškama pomoću `error page`. Ako strane sadrže CSS i JavaScript, staviti ih u posebne fajlove i uključivati u JSP strane. Ako JSTL nije dovoljan, proveriti dizajn aplikacije: bolje je premestiti logiku u servlet koji će zatim postaviti attribute za prikaz na JSP strani.

7. Opis aplikacije – struktura koda

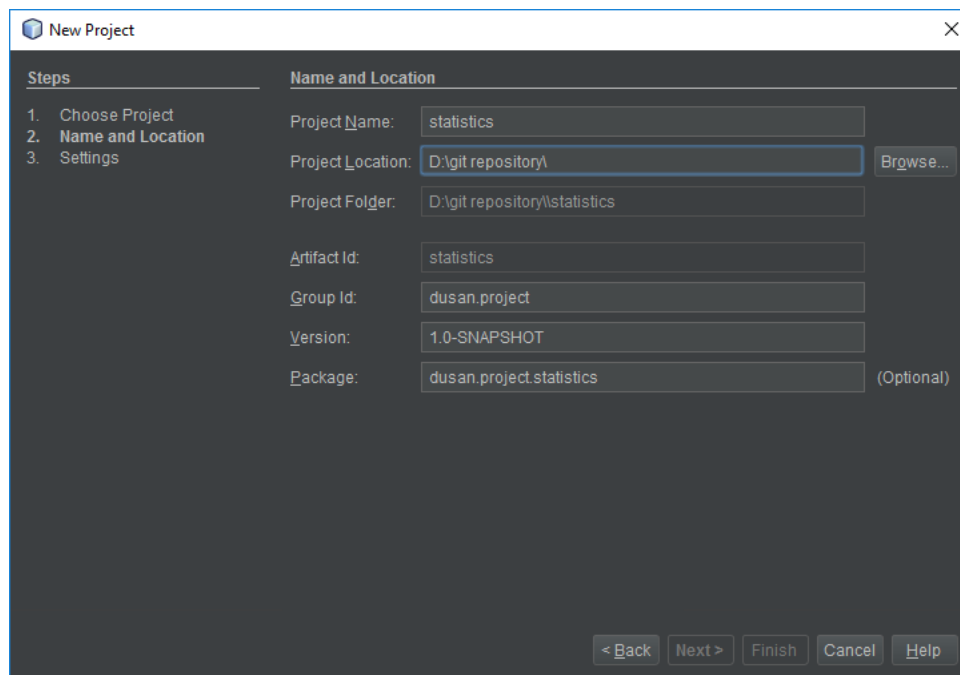
Početak kreiranja aplikacije u NetBeans okruženju kreće izborom File/New Project, pri kom se otvara prozor za izbor daljih opcija. Za kreiranje aplikacije sa tehnologijama koje smo ranije u radu opisali, potrebno je izabrati Maven⁸/Web Application. Na slici 22. se vidi prozor koji smo opisali (Choose Project).



Slika 22. Novi projekat

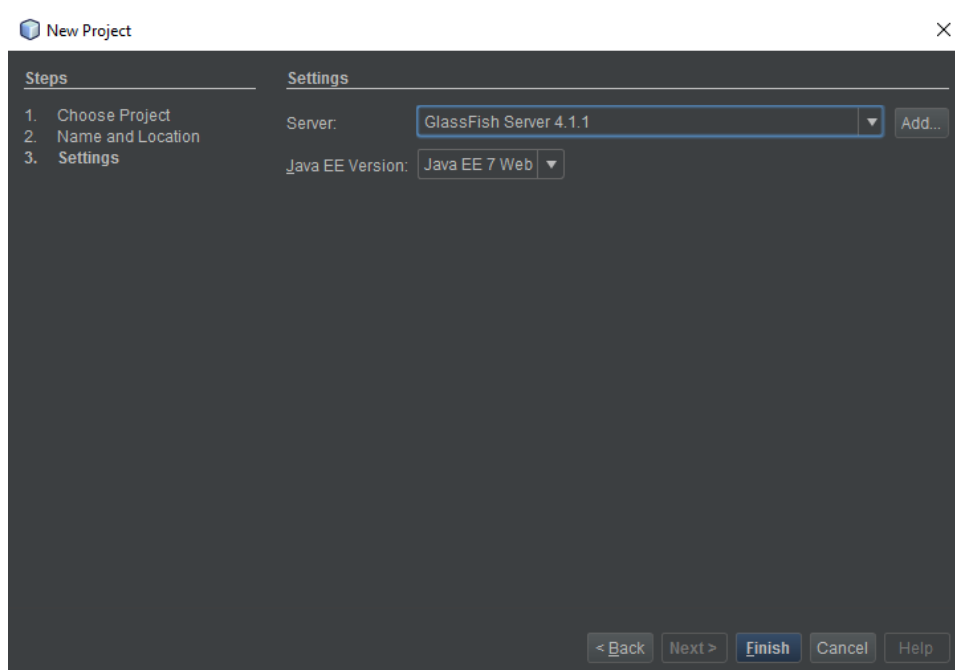
Pritiskom na next dugme prelazi se na sledeći prozor na kome upisujemo ime aplikacije u prostor posle Project Name. Project Location i Project Folder su polja za postavljanje datoteke na lokalnom sistemu, a polja Artifact Id, Grup Id, Version i Package se vezuju redom za ime projekta, zatim, firmu koja radi na projektu (u ovom slučaju je master rad, pa sam stavio svoje ime i paket), verziju na kojoj se trenutno radi (uvek se polazi od 1.0-SNAPSHOT) i paketa tj. osnovne putanje gde će se raspoređivati kod. Na slici 23. se nalazi primer drugog prozora (Name and Location).

⁸ Maven – (ili Apache Maven) je alat, kojim se kompajlira (gradi) kod. Koristi se za Java aplikacije i sam je napisan u Javi. Alat ima centralni repozitorijum na internetu sa koga dovlači biblioteke koje se koriste na projektu u lokalni repozitorijum i kasnije kada god su potrebne za određene komande, biblioteke se nalaze na lokalnom sistemu, što puno olakšava i ubrzava rad.



Slika 23. Ime i lokacija projekta

Klikom na dugme Next pojavljuje se poslednji prozor (Settings), gde se biraju server koji će se koristiti i Java EE verzija. Server koji smo koristili je GlassFish Server 4.1.1, a Java EE verzija je Java EE 7 Web. Na slici 24. se nalazi poslednji prozor kojim se završava kreiranje novog projekta. Posle pritiska na Finish dugme, NetBeans sam izgeneriše projekat. Napravi na lokalnom sistemu datoteke, prema uputstvima iz drugog prozora i smesti odgovarajuće fajlove u odgovarajuće datoteke. Kasnijim dodavanjima datoteka ili fajlova, sve se smešta na već formirane lokacije.



Slika 24. Opcije novog projekta

GlassFish Server 4.1.1

GlassFish Server 4.1.1 je open-source (javno dostupan i besplatan za korišćenje) aplikativni softver, koji se koristi za Java EE platforme. Kompanija Oracle Corporation, u čijem je i vlasništvu, održava server. Server podržava Enterprise JavaBeans-e, JPA, JavaServer Faces, JMS, RMI, JSP strane, servlete itd.

Java EE 7 Web

Java Platform, Enterprise Edition ili Java EE je platforma za razvijanje softvera. Platforma koristi Java objektno orjentisani jezik. Platforma pruža podršku za API-je objektno relacionog mapiranja, distribuciju i veb servise. Može se koristiti XML za override-ovanje (prenamenu) anotacija tj. odstupanje od osnovnih komandi na platformi.

pom.xml

Kada smo kreirali projekat, potrebno je da pregledamo i doradimo određene fajlove kojima se setuju (podešavaju) veze se serverima. Postoje takozvani settings fajlovi, kojima konfigurišemo određene veze. Jedan od takvih je pom.xml ili Project Object Model. To je Maven-ov fajl koji se automatski napravi prilikom kreiranja Maven projekta. To je fajl sa XML strukturom u kome se nalaze podaci o projektu i konfiguraciji. Fajl se nalazi u osnovnom direktorijumu, u ovom slučaju u folderu „statistics“. Kod ispod prikazaćemo redom

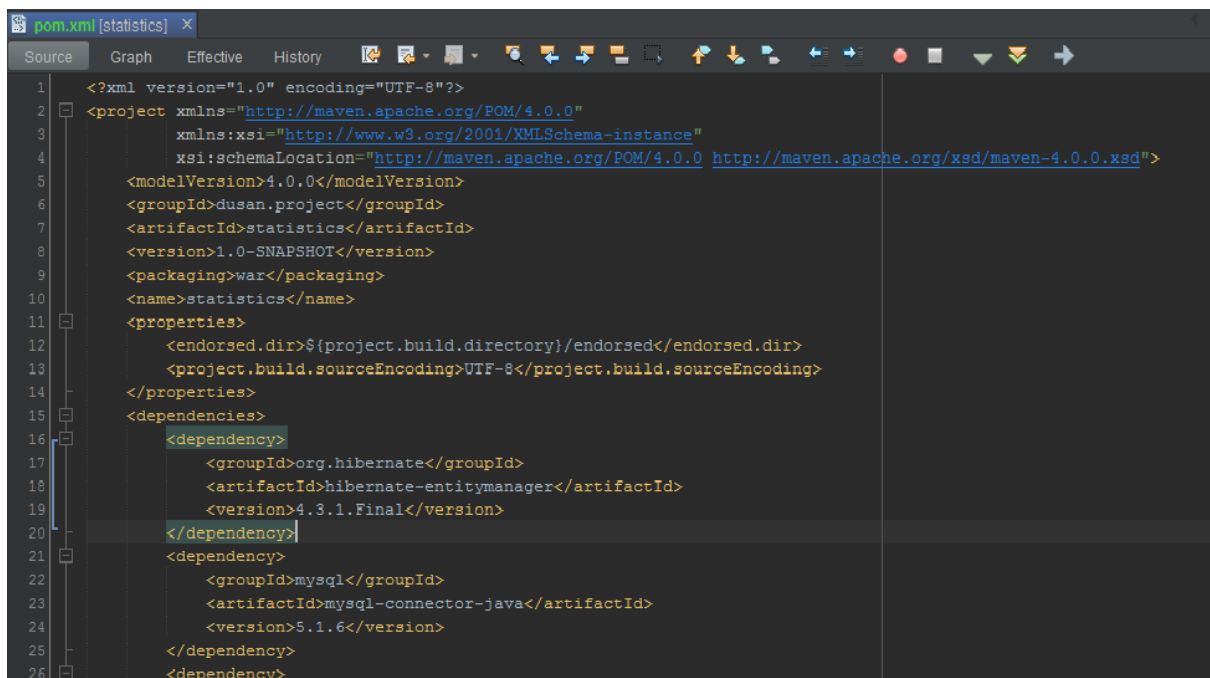
```
<?xml version="1.0" encoding="UTF-8"?>
  <modelVersion>4.0.0</modelVersion>    
  <groupId>dusan.project</groupId>
  <artifactId>statistics</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>statistics</name>    
  <properties>
    <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>4.3.1.Final</version>
    </dependency>
    
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.6</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.persistence</groupId>
      <artifactId>eclipselink</artifactId>
      <version>2.5.2</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
```

```

    <dependency>
      <groupId>org.eclipse.persistence</groupId>
    <artifactId>org.eclipse.persistence.jpa.modelgen.processor</artifactId>
      <version>2.5.2</version>
      <scope>provided</scope>
    </dependency>
    //veza za Java web api
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>7.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
  </dependencies>
  // ispod su plugin-ovi za maven server koji se automatski setuju
<build>
  <plugins>
    <plugin>...

```

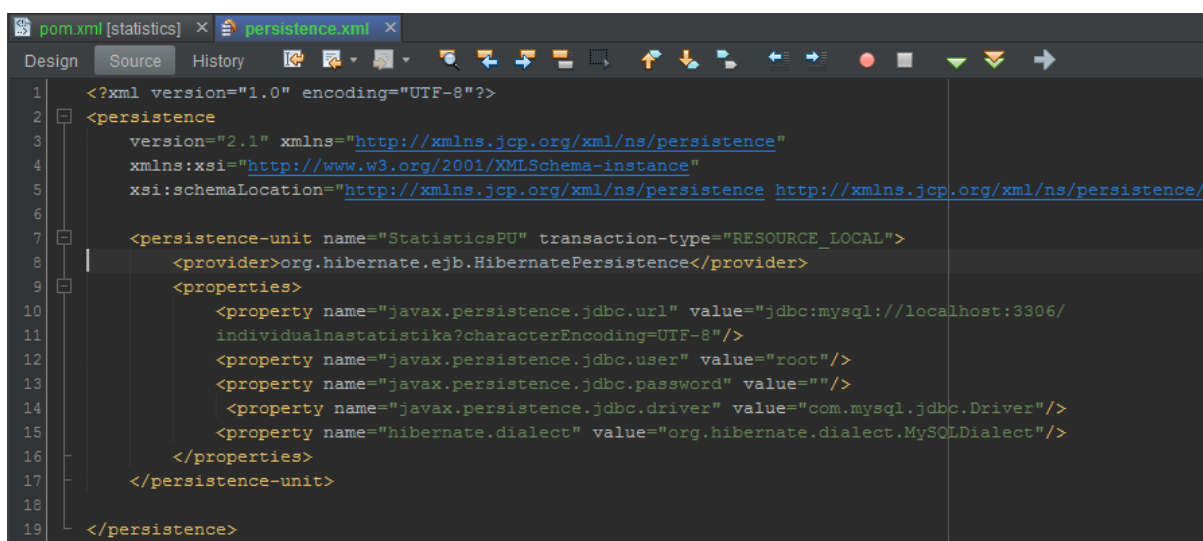
Na slici 25. prikazan je izgled pom.xml fajla u NetBeans okruženju.



Slika 25. pom.xml

persistence.xml

Persistence je konfiguracioni fajl koji pravi vezu sa MySQL bazom. Na slici 26. prikazan je fajl u master projektu. Prvi tag kao i kod pom.xml je vezan za verziju i pismo koje se koristi. Drugi tag je veza sa serverom. Persistence Unit(PU) sadrzi podatke potrebne za povezivanje na bazu podataka. Naziv PU-a (name="StatisticsPU") koristimo u kodu kako bismo kreirali EntityManagerFactory. Ispod PU nalazi se veza sa Hiberante-om, a ispod JDBC veza sa bazom na localhost-u. Port 3306 je veza za MySQL bazu na šemu individualnastatistika, posle toga je pismo koje se koristi u šemi. Ispod toga su nalog i šifra prilikom logovanja na MySQL bazu, root za nalog i "" znači da ne postoji šifra. Ispod je JDBC driver, pa Dijalekt property koji omogućava da Hibernate generiše odgovarajući SQL u zavisnosti od baze podataka na koju se povezuje. Persistence fajl se nalazi u resources/META-INF, recourses se nalazi u glavnoj datoteci.



Slika 26. persistence.xml

web.xml

Od konfiguracionih fajlova postoje još i nb-configuration.xml koji se automatski generiše kada se napravi projekat. On setuje (podešava) određene parametre vezane za NetBeans. Pored nb-configuration.xml-a postoji i web.xml. Web.xml ili Deployment Descriptor je konfiguracioni fajl, koji kao i prethodni koristi XML šemu da napravi određene veze i mapiranja. Java veb aplikacije koriste ovaj fajl da odrede URL-servlet mapiranja. Odnosno da povežu određene veb stranice projekta sa servetima i tako omoguće aplikaciji da bude interaktivna sa korisnicima. Fajl se nalazi na lokaciji main/webapp/WEB-INF. Potrebno je tokom kreiranja koda menjati fajl tačnije dodavati mapiranja svaki put kada se kreira određeni servlet i JSP stranice za koje je taj servlet “zadužen”. Fajl počinje sa verzijom xml-a i pismom (primetićemo kao i ostali xml fajlovi), apotom sa vezom sa xml serverom. Jedna od vrsta mapiranja je filter, u kodu ispod se nalazi mapiranje na projektu.

```

<filter>
  <filter-name>AccessFilter</filter-name>
  <filter-class>dusan.project.statistics.filter.AccessFilter</filter-
class>
</filter>
<filter-mapping>
  <filter-name>AccessFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

```

AccessFilter je filter fajl koji nam odvaja URL-ove za koje nam je potrebna, i za koje nije potrebna autentifikacija tj. logovanje na sistem. Prvi tag <filter> nam govori o kom filteru je reč i gde se on nalazi na projektu (Koristi se drugačija putanja, ne lokacijski neko preko Project principa). Drugi tag <filter-mapping> podešava šta će na URL-u da se prokaže ukoliko filter ili servlet bude komitovao svoj sadržaj. Slično kao i kod filtera na kodu ispod nalazi se primer za servlet sa oba taga.

```

<servlet>
  <servlet-name>UtakmicaServlet</servlet-name>
  <servlet-
class>dusan.project.statistics.controller.UtakmicaServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>UtakmicaServlet</servlet-name>
  <url-pattern>/utakmica</url-pattern>
</servlet-mapping>

```

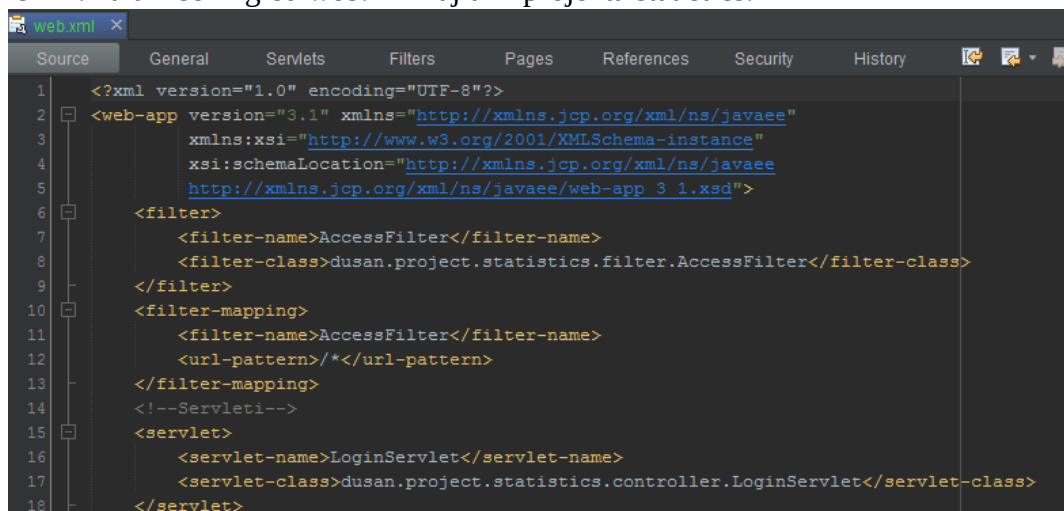
Primetićemo isti princip i za servlete na primeru UtakmicaServlet. Postoji prvo tag za poziciju servleta, a potom i za URL mapiranje. Na kraju web.xml fajla postoji kod koji konfiguriše dužinu trajanja sesije. Na ovo projektu sesija traje 30 sekundi. Kod izgleda ovako:

```

<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>

```

Na slici 27. nalazi se izgled web.xml fajla iz projekta statistics.

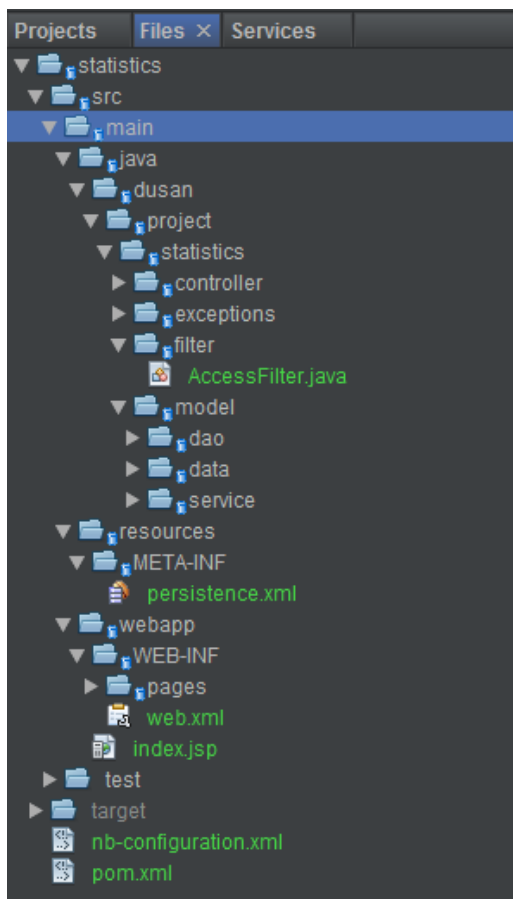


Slika 27. web.xml

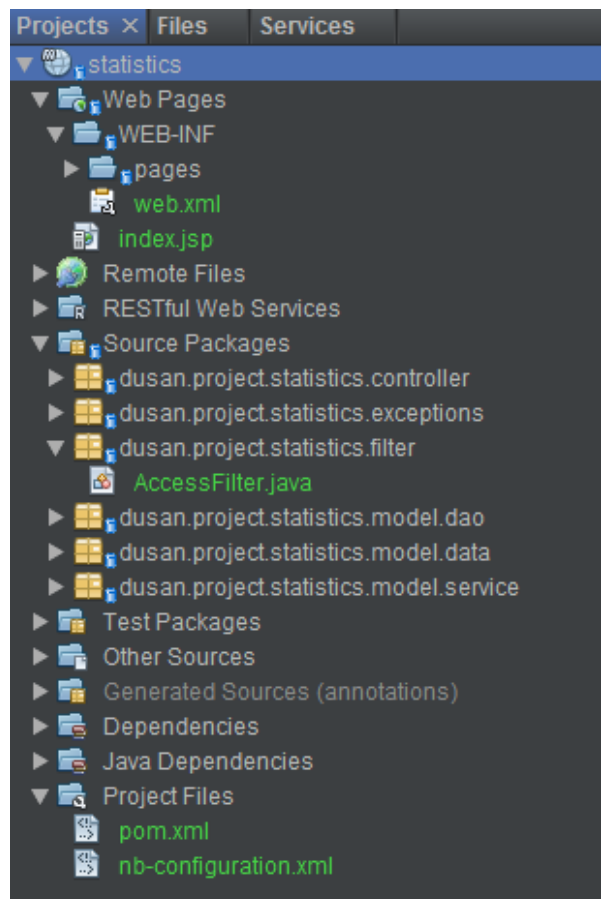
7.1. Backend aplikacije

Postoji takozvana **MVC** arhitektura. To je pristup pisanju koda u kome se odvajaju tri različita nivoa. Model, deo backend-a koji služi za kreiranje objekata i operacija nad njima. Contrller, koji je zadužen za osluškivanje poziva od klijenata za izvršenje operacije nakon koga poziva operacije definisane u modelu i vraća stanje na View ukoliko je bilo promena. View obezbeđuje korisniku interfejs pomoću koga korisnik unosi podatke i poziva odgovarajuće operacije i pokazuje stanje Model-a. Model, Controller, View otuda MVC.

NetBeans okruženje pruža nekoliko načina da se pristupi fajlovima projekta. Prvi način smo spominjali ranije i to je preko meniji koji se zove Files. To je klasičan pristup kroz datoteke, on je dobar zato što pruža uvid u tačnu lokaciju fajla, međutim mana mu je to što u situacijama kada ima puno datoteka i fajlova dosta se usporava rad, jer kretanje kroz iste ume da bude komplikovano. Na slici 28. prikazan je meni Files. Pored klasicnog prolaska kroz datoteke postoji drugi prisup fajlovima. To je meni Projects, gde se nalazi sličan hijerarhijski pristup fajlovima, ali su datoteke drugačije predstavljene, pa nema prolaska kroz svaku nego se jednostavnije i brže pristupa fajlovima i datotekama u kojima se isti kreiraju. Naslici 29. nalazi se meni Projects.



Slika 28. Files meni



Slika 29. Projects meni

U Projects meniju postoji folder Source Packages u koji smeštamo Java kod na backend-u za aplikaciju. Pošto se sav backend kod sastoji od Java klasa. Da bi projekat ispunjavao određene norme, smešćaćemo klase u u datoteke koje ćemo prethodno napraviti. Na slici 31. (ranije) nalazi se meni Projects i pomenuti Source Packages u kome smo napravili sledeće datoteke: model.data, model.dao, model.service, filter, exceptions i controller. Prve tri pomenute datoteke imaju „model.“ zajedničko, jer svi imaju naddatoteku model. U te tri datoteke se smeštaju klase koje imaju zajednički funkciju da naprave osnovu za biznis logiku aplikacije.

Model Data – JavaBeans klase

U model.data se smeštaju takozvane JavaBeans klase. To su klase u kojima formiramo promenljive sličnih ili istih imena kao promenljive iz pojedinačnih tabela, u šemi sa kojom se povezujemo na MySQL bazu. Pošto naša šema „individualnastatistika“ u bazi ima pet tabela (Igrac, User, Tim, Utakmica i Statistika) potrebno je da kreiramo pet klasa sa istim imenima. Pa tako postoje Igrac.java, User.java, Tim.java, Utakmica.java i Statistika.java. Svaka od klasa je pravljen po istom principu, pa ćemo dati primer jedne. Uzećemo za primer klasu Tim.java. u tabeli Tim postoje tri promenljive čije su vrednosti ime kolone idTim, imeTima i država. Na slici 30. prikazan je deo tabele iz MySQL workbench okruženja. U klasi Tim.java potrebno je napraviti privatne promenljive koje će biti mapirane tj. povezane sa promenljivim iz baze.

	idTim	imeTima	drzava
►	1	Anadolu Efes	Turska
	2	Baskonia Vitoria Gasteiz	Spanija
	3	Brose Bamberg	Nemacka
	4	Crvena Zvezda	Srbija
	5	CSKA Moscow	Rusija
	6	Darussafaka Dogus Istambul	Turska
	7	EA7 Emporio Armani Milan	Italija
	8	FC Barcelona Lassa	Spanija

Imena ne moraju biti ista, ali je zbog praktičnosti dobro tako napraviti. Iznad tih promenljivih postoje anotacije koje prave vezu. Anotacije iznad idTim postavljaju promenljivu na primary key (primarni ključ tabele) i vezuju promenljive za ime kolone, a ostale promenljive su samo samo sa imenima kolona. Na

Slika 30. MySQL tabela Tim, šeme individualnastatistika
slici 31. prikazan je vrh klase Tim.java.

```
@Entity
@Table(name = "Tim")
public class Tim implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "idTim")
    private Integer idTim;

    @Column(name = "imeTima")
    private String imeTima;

    @Column(name = "drzava")
    private String drzava;
```

Slika 31. Promenljive i anotacije promenljivih u klasi Tim.java

Pored promenljivih u JavaBeans klasi potrebno je napraviti konstruktore preko kojih ćemo kasnije u klasama kreirati Tim. Kreiramo jedan privatni prazan konstruktor, jedan samo sa idTim promenljivom i ostale kako nam treba u bizns logici, na kraju uvek konstruktor koji sadrži sve promenljive. Posle konstruktora kreiraju se takozvani seteri i geteri. Funkcije koje setuju (postavljaju) i getuju (uzimaju) vrednosti promenljive, one vraćaju String, Integer, Boolean ili neku drugu vrednost u zavisnosti kako smo kreirali promenljivu na početku klase. Na kodu ispod postoji seter i geter za idTim. Posle setera i getera potrebno je da se napravi toString funkcija koja vraća vrednost String redom ime promenljive, pa njenu vrednost u datom trenutku kada se pozove. Na kodu ispod postoji toString funkcija klase Tim.java. NetBeans automatski nudi **include** opciju za biblioteke koje treba da se „uvezu“ na početak klase, da bi moglo da se koriste određene komande. Tako da je mnogo olakšano kucanje koda.

```
public Integer getIdTim() {                //geter za idTim
    return idTim;
}
public void setIdTim(Integer idTim) {      // seter za idTim
    this.idTim = idTim;
}
@Override                                // toString funkcija
public String toString() {
    return "Team{" + "idTim=" + idTim + ", imeTima=" + imeTima + ",
    drzava=" + drzava + '}';
}
```

Sve ostale JavaBeans klase imaju veze sa imenima kolona tabela odnosno sa promenljivim iz baza. Veza sa samom tabelom se pravi preko anotacije iznad imena klase, pa je tako:

```
@Entity
@Table(name = "Tim")
```

dodatak iznad imena klase Tim čime se mapira java klasa sa MySQL tabelom, šeme individualna statistika. Sve ostale klase su mapirane na ista imena tabela iz šeme. I kreirane su promenljive kao i u bazi, napravljene anotacije po istom principu, pa tako postoje Igrac tabelu i Igrac.java klasu mapirane i vezane, i promenljive iz tabele, idIgrac, ime, prezime, kreirane kao promenljive u klasi i anotacijama vezane za promenljive iz tabele. Dodati su konstruktori, seteri, geteri i toString funkcija.

Jedina razlika je u dve funkcije u User.java klasi. U kodu ispod se nalaze te dve funkcije:

```
@Override
public int hashCode() {
    int hash = 7;
    hash = 89 * hash + Objects.hashCode(this.idUser);
    return hash;
}
@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final User other = (User) obj;
    return Objects.equals(this.idUser, other.idUser);
}
```

Funkcija **hashCode**, vraća hash-ovan nalog i šifru user-a. To je potrebno da bi se zaštitila aplikacija i da bi se u bazi držala šifra u obliku u kom je teško doći do iste. Funkcija **equals** upoređuje dva objekta user, jedan iz baze i drugi unet od korisnika, ukoliko ona vrati true obavlja se logovanje na sistem, ukoliko vrati false, logovanje je neuspešno.

Model Dao

U dao delu modela, potrebno je da kreiramo veze sa bazom i funkcije kojima ćemo kreirati objekte JavaBeans klasa. Tako kreiramo IgracDAO.java, StatistikaDAO.java, TimDAO.java, UserDAO.java i UtakmicaDAO.java klase kao nastavak data modela i imamo dodatak EntityManagerWrapper.java klasu. Prvih pet klasa su iste kodom, sa izuzetkom dodatka u UserDAO klasi. Uzećemo primer TimDAO klase, a pre toga EntityManagerWrapper klasa, koja je klasa "pomoćnik" za dohvaćanje instanci EntityManager-a. Na slici 32. se nalazi EntityManagerWrapper klasa.

```
1 package dusan.project.statistics.model.dao;
2
3 import javax.persistence.EntityManager;
4 import javax.persistence.EntityManagerFactory;
5 import javax.persistence.Persistence;
6
7
8 public class EntityManagerWrapper {
9
10     private static final EntityManagerFactory emf = Persistence.createEntityManagerFactory("StatisticsPU");
11
12     public static EntityManager createEntityManager() {
13         return emf.createEntityManager();
14     }
15
16     public static EntityManager getEntityManager() {
17         return emf.createEntityManager();
18     }
19
20 }
```

Slika 32. EntityManagerWrapper klasa

Fabrika EntityManagerFactory služi za kreiranje instanci EntityManager-a. Kao što se vidi na slici EntityManagerFactory kreira entitet ManagerFactory „StatistikaPU“ (Persistence Unit) koji smo setovali u persistence.xml-u i tako napravili vezu sa šemom individualnastatistika u MySQL bazi. U klasi smo napravili još dve funkcije koje kreiraju i uzimaju EntityManager. Svrha klase je da se preko nje povežemo na šemu. Objekat ove klase ćemo praviti u svim DAO klasama. Tako u klasi TimDAO, koju ćemo objasniti kao primer ima sve funkcije koje kao argument prihvataju EntityManager objekat. U Klasi TimDAO prvo kreiramo instancu TimDAO, prazan konstruktor i funkciju getInstance koja samo vraća instancu objekta. U kodu ispod vidimo kako:

```
private static TimDAO instance = new TimDAO();
public TimDAO() {
}
public static TimDAO getInstance() {
    return instance;
}
```

Na slici 33. nalaze se ostale funkcije.

```
public List<Tim> findAll(EntityManager em) {
    Query query = em.createQuery("SELECT c FROM Tim c");
    return query.getResultList();
}

public void add(EntityManager em, Tim tim) {
    em.persist(tim);
}

public void edit(EntityManager em, Tim tim) {
    em.merge(tim);
}

public void delete(EntityManager em, Tim tim) {
    Tim managedTeam = em.merge(tim);
    em.remove(managedTeam);
}
```

Slika 33. Funkcije klase TimDAO.java

Funkcija `findAll` vraća listu podataka tipa `Tim`. Svi objekti sa svim podacima (vrednostima promenljivih) iz tabele `Tim` iz baze. Upit „`SELECT c FROM Tim c`“ je upit koji se u tom obliku šalje na bazu i odgovor je lista svih podataka te tabele. Ostale funkcije imaju jasne rezultate. `Add` dodaje objekat tipa `Tim`, `edit` menja, a `delete` briše objekat `Tim`. Sve tri funkcije prihvataju kao argument `EntityManager` pored objekta `Tim`.

Sve DAO klase su pisane po istom principu, s’ tim što u `StatistikaDAO` postoje dodaci funkcija sa upitima na bazu i `UserDAO` je kompletno drugačija. `StatistikaDAO` klasa pored ovih funkcija i konstruktora ima i funkcije `findAll1`, `findAll2` i tako redom do `findAll16`. Funkcije izgledaju kao kod ispod:

```
public List<Statistika> findAll1(EntityManager em) {
    Query query = em.createQuery("SELECT c FROM Statistika c WHERE igrac_idIgrac.tim_idTim='1'");
    return query.getResultList();
}
```

Razlog kreiranja ovih funkcija je kasnija biznis logika, gde nam trebaju statistike igrača koji igraju za tim sa promenljivom `idTim=1`. Otuda upit „`SELECT c FROM Statistika c WHERE igrac_idIgrac.tim_idTim='1'`“. Kao rezultat se vraća lista svih statistika igrača koji nastupaju za tim sa identifikacionim brojem 1. Sve ostale funkcije imaju isti princip, samo su pisani za ostale timove i njihove `idTim`-ove. Posotji 16 timova, pa otuda poslednja funkcija `findAll16`.

Klasa UserDao ima drugačiji kod. U klasi smo kreirali finalnu instancu, prazan konstruktor i funkciju koja vraća instancu i potom dve funkcije koje se vide na slici 34.

```
public boolean checkLogin(EntityManager em, User user) throws StatisticsException {
    try {
        Query query = em.createQuery("SELECT b FROM User b WHERE b.nalog=:nalog AND b.sifra=:sifra");
        query.setParameter("nalog", user.getNalog());
        query.setParameter("sifra", calculateHash(user));
        query.getSingleResult();
        return true;
    } catch (NoResultException ex) {
        return false;
    } catch (UnsupportedEncodingException | NoSuchAlgorithmException ex) {
        throw new StatisticsException("Neuspešna provera korisničkog naloga.", ex);
    }
}

private String calculateHash(User user) throws UnsupportedEncodingException, NoSuchAlgorithmException {
    String nalogAndSifra = user.getNalog() + user.getSifra();

    MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
    byte hashBytes[] = messageDigest.digest(nalogAndSifra.getBytes("UTF-8"));

    return DatatypeConverter.printBase64Binary(hashBytes);
}
```

Slika 34. funkcije klase UserDao.java

Funkcija checkLogin proverava logovanje na sistem. Prvim query upitom traže se nalog i šifra iz baze. Potom se setuju parametri, s'tim što se nalog samo uzima, a kod šifre funkcijom calculateHash, prvo računa Hash user-a. Hvataju se dva izuzetka. NoResultException, za situaciju kada ni nema naloga ili sifre ili oba. I drugi izuzetak ako je neuspešna provera. Funkcija calculateHash forima argument User. Kreira String nalogAndSifra spajanjem stringova nalog i šifra, potom poziva funkciju MessageDigest koja postavlja instancu uz „SHA-256“ kriptografsku formulu. Funkcija uzima ulazni string i proizvodi niz karaktera kratke dužine (npr. 128 ili 160 bita). Potom se taj objekat poziva u kreiranju hashBytes, gde se enkriptovani objekat prevod u bajtove i poslednjom linijom vraća rezultat kao string.

Model - Service

Poslednji deo modela je service. U service modelu imamo pet klasa i svaka odgovara klasama iz prethodnih delova model hijerarhije. Tako postoji IgracService.java, StatistikaService.java, TimService.java, UserService.java i UtakmicaService.java. Klase su slične, pa ćemo uzeti primer IgracService i nju opisati, a potom razlike iz drugih klasa. Na početku klase kreiramo finalnu instancu klase, prazan javni konstruktor i funkciju getInstance koja kao rezultat vraća instancu klase. Kao u kodu ispod:

```
private static final IgracService instance = new IgracService();
public IgracService() {
}
public static IgracService getInstance() {
    return instance;
}
```

Potom su kreirane određene funkcije koje odgovaraju funkcijama iz DAO klase. Tako postoji funkcija add, za dodavanje igrača, edit za promenu parametara, delete za brisanje, validate za validaciju tj. proveru dali je objekat postojeći i findAll funkciju kojom kreiramo listu objekata sa svim parametrima. Na kodu ispod nalazi se add funkcija:

```

public void add(Igrac igrac) throws StatisticsException {
    validate(igrac);
    EntityManager em = EntityManagerWrapper.createEntityManager();
    try {
        em.getTransaction().begin();
        IgracDAO.getInstance().add(em, igrac);
        em.getTransaction().commit();
    } finally {
        em.close();
    }
}

```

Javna funkcija, koja se može pozivati i iz ostalih klasa, izbacuje `StatisticsException` grešku (kasnije će biti reči), prihvata argument objekta `Igrac` i prvo poziva spoljnu funkciju `validate`, da validira objekat. Potom kreiramo `EntityManager` i otvaramo `try` blok u kome kreiramo transakciju, potom dodajemo instancu iz `IgracDAO`, kojoj dodajemo nove parametre, potom komitujemo podatke i u svakom slučaju, bila transakcija uspešna ili ne zatvaramo transakciju `EntityManager`-a. Vrlo slična je i `edit` funkcija:

```

public void edit(Igrac igrac) throws StatisticsException {
    validate(igrac);
    EntityManager em = EntityManagerWrapper.createEntityManager();
    try {
        em.getTransaction().begin();
        IgracDAO.getInstance().edit(em, igrac); //razlika - edit
        em.getTransaction().commit();
    } finally {
        em.close();
    }
}

```

Razlika je u `edit` funkciji koju pozivamo na instancu `IgracDAO` objekat sa argumentima `EntityManager`-a `em` i `igrac`. Tako da umesto `add` funkcije koristimo `edit`, koja za ulogu ima da već postojeće podatke kojima transakcijom pristupamo promeniti. Funkcija `delete`, briše podatke kojima se transakcijom pristupa.

```

public void delete(Igrac igrac) throws StatisticsException {
    if (igrac.getIdIgrac() == null) {
        throw new StatisticsException("Nepoznat ID igrača.");
    }
    EntityManager em = EntityManagerWrapper.createEntityManager();
    try {
        em.getTransaction().begin();
        IgracDAO.getInstance().delete(em, igrac);
        em.getTransaction().commit();
    } finally {
        em.close();
    }
}

```

Kod `delete` funkcije imam prvo `if` iskaz kojim proveravamo preko `idIgrac`-a da li uopšte postoji objekat igrač kojem pristupamo, ako postoji na isti način pristupamo objektu, kao i u prethodnim funkcijama, i funkcijom `delete`, na `IgracDAO` instancu, ga uklanjamo. Validiramo funkciju kreiramo da u funkcijama `add` i `edit` ne bi pisali isti kod, već jednostavno pozvali funkciju. Uloga ove funkcije je da prepozna da li objekat postoji. Na kodu ispod se vidi funkcija:

```

private void validate(Igrac igrac) throws StatisticsException {
    List<String> errors = new ArrayList<>();
    if (igrac.getIdIgrac() == null) {
        errors.add("id igrača je obavezno");
    }
    if (igrac.getIme() == null || igrac.getIme().trim().isEmpty()) {
        errors.add("Ime je obavezno.");
    } else if (igrac.getIme().length() > 45) {
        errors.add("Ime ne sme biti duže od 45 karaktera");
    }
    if (igrac.getPrezime() == null ||
    igrac.getPrezime().trim().isEmpty()) {
        errors.add("Prezime je obavezno.");
    } else if (igrac.getPrezime().length() > 45) {
        errors.add("Prezime ne sme biti duže od 45 karaktera");
    }
    if (igrac.getTim_idTim() == null) {
        errors.add("id tima je obavezno");
    }
    if (!errors.isEmpty()) {
        throw new StatisticsException(errors);
    }
}

```

Funkcija prihvata objekat Igrac argument i baca izuzeetak StatisticsException. Potom kreiramo listu u koju ćemo smeštati validacijske greske. Ako bude bilo greški u ovoj listi, na kraju metode će biti bačen izuzetak koji enkapsulira (sadrži) listu grešaka. Ako ne bude bilo validacijskih grešaka lista će ostati prazna i izuzetak neće biti bačen. Ispod liste grešaka je nekoliko if iskaza kojima proveravamo da li su popunjena polja prilikom unosa podataka i bacamo grešku ako nisu. Pored pomenutih, tu je i funkcija findAll, kojom prolazimo sve objekte kojima transakcijom pristupamo.

```

public List<Igrac> findAll() {
    EntityManager em = EntityManagerWrapper.createEntityManager();
    try {
        return IgracDAO.getInstance().findAll(em);
    } finally {
        em.close();
    }
}

```

U try bloku vraćamo instancu IgracDAO funkcijom findAll iz IgracDAO klase. U StatistikaService klasi imamo pored findAll funkcije još 16 sličnih funkcija findAll1, findAll2, FindAll3..., FindAll16, kojima pravimo liste objekata statistika, sa određenimkara kteristikama, takvim da je idTima, od igrača čije statistike gledamo, isti, tj. da svi igraju za istu ekipu. Broj na kraju imena funkcije je isti kao idTim-a. Sve funkcije su iste samo što pozivaju različitu findAll funkciju iz StatistikaDAO klase. O tim funkcijama smo već pisali, rekli smo da ima 16

različitih funkcija i ovde se poziva odgovarajuća. Što se tiče UserService klase kod se nalazi na slici 35.

```
1 package dusan.project.statistics.model.service;
2
3 import dusan.project.statistics.exceptions.StatisticsException;
4 import dusan.project.statistics.model.dao.EntityManagerWrapper;
5 import dusan.project.statistics.model.dao.UserDAO;
6 import dusan.project.statistics.model.data.User;
7 import javax.persistence.EntityManager;
8
9 public class UserService {
10
11     private static final UserService instance = new UserService();
12
13     private UserService() {
14     }
15
16     public static UserService getInstance() {
17         return instance;
18     }
19
20     public boolean checkLogin(User user) throws StatisticsException {
21
22         EntityManager em = EntityManagerWrapper.createEntityManager();
23         try {
24             return UserDAO.getInstance().checkLogin(em, user);
25         } finally {
26             em.close();
27         }
28     }
29 }
```

Slika 35. UserService.java klasa

Posle kreiranja instance, praznog konstruktora i funkcije getInstance koja vraća vrednost instance, postoji samo jedna funkcija boolean tipa (vraća true ili false – tačno ili netačno) checkLogin, čija je namena da proverava preko transakcije sa EntityManager-om prijavu na sistem. U try bloku smo vratili true or false preko funkcije checkLogin (klase UserDAO) koja obrađuje instancu UserDAO klase, prihvatajući argumente em EntityManager-a i user-a. Try blok ima finally izlaz, što znači da kakva god da se operacija odvijala u try bloku, mora se završiti sa finally blokom, a tu se nalazi komanda kojom zatvaramo transakciju.

Exception

Exception ili izuzetak. U svaki projekat prilikom pisanja treba dodati izuzetak. To je klasa kojim se obuhvataju nepredviđene situacije u kojima se naša aplikacija ponaša mimo onoga što očekujemo. Dešavaju se različite situacije prilikom rada sistema, a pošto je on zavistan od rada određenih servera ili drugih aplikacija treba razmišljati o nepredviđenim okolnostima. Pored spoljnih uticaja, postoje i situacije kada se nešto u kodu ne slaže dovoljno dobro, pa sistem može da zakaže. Da bi se to izbeglo, piše se klasa Excetpion i u kodu se ona uključuje gde god je to moguće. Što se više izuzetaka obradi, sistem je bolji i manja je mogućnost da padne ili se desi nešto nepredviđeno. Naslici 36. nalazi se klasa StatisticExcetption.java.

```
1 package dusan.project.statistics.exceptions;
2
3 import java.util.Arrays;
4 import java.util.List;
5
6 public class StatisticsException extends Exception {
7
8     private final List<String> errorMessages;
9
10    public StatisticsException(List<String> errorMessages) {
11        super();
12        this.errorMessages = errorMessages;
13    }
14
15    public StatisticsException(String errorMessage) {
16        super(errorMessage);
17        this.errorMessages = Arrays.asList(errorMessage);
18    }
19
20    public StatisticsException(String errorMessage, Throwable cause) {
21        super(errorMessage, cause);
22        this.errorMessages = Arrays.asList(errorMessage);
23    }
24
25    public List<String> getErrorMessages() {
26        return errorMessages;
27    }
28 }
```

Slika 36. StatisticsException. java klasa

Klasa je produžetak biblioteke Exception, na početku pravimo listu stringova errorMessages. Potom funkciju koja je izuzetak i koji u sebi enkapsulira listu grešaka, koje treba prikazati korisniku. Ispod toga su dve funkcije koje imaju istu namenu samo prihvataju drugačije argumente. Situacija kada samo treba ispisati grešku ili kada pored greške treba pomenuti i neki argument zbog kog se greška dogodila.

Filter

Filter deo koda je u kome se nalazi samo jedna klasa `AccessFilter.java`. Ta klasa služi za obradu prijavljivanja na sistem. U njoj se nalazi nekoliko funkcija koje ćemo opisati. Klasa `AccessFilter` implementira biblioteku `Filter` sa svim njenim funkcijama. Na početku kreiramo listu stringova `NO_AUTH_URLs` sa vrednostima `"/index.jsp"`, `"/logout"`, `"/login"`. Lista URL-ova za koje nije potrebna autentifikacija. To su putanje za pristup početnoj stranici, login stranici i stranici za odjavu sa aplikacije, `logout`. Kreiramo prazan konstruktor. Potom tri override-ovane funkcije kao na kodu ispod:

```
@Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        String url = httpRequest.getServletPath();
        if (!NO_AUTH_URLs.contains(url)) {
            if (httpRequest.getSession().getAttribute("user") == null) {
                httpRequest.setAttribute("errors", new String[]{"Morate se
prijaviti na sistem."});
                httpRequest.getRequestDispatcher("index.jsp").forward(request, response);
            }
            chain.doFilter(request, response);
        }
    }

    @Override
    public void destroy() {
    }

    @Override
    public void init(FilterConfig filterConfig) {
    }
```

Sve tri funkcije imaju `@Override` anotaciju, koja omogućava prenamenu funkcija iznad kojih su napisane. U biblioteci `Filter`, koju naša klasa implementira već postoje funkcije `doFilter`, `destroy` i `init`, pa je potrebno override-ovati ih da bi mogli da ih modifikujemo. U funkciji `doFilter` koja kao argumente prima `ServletRequest`, `ServletResponse` i `FilterChain` prvo stvaramo jedan `HttpServletRequest`, a potom string kome korisnik pokušava da pristupi. Potom imamo `if` iskaz, ukoliko dobijeni URL nije u listi onih za koje nije potrebna autentifikacija potrebno je proveriti da li je korisnik ulogovan. Unutar tog `if` iskaza postoji još jedan. Provera, da li je korisnik ulogovan, se vrši tako što se proverí da li u sesiji ima atribut pod nazivom `"user"`. Ovaj atribut dodajemo u sesiju u `LoginServlet`-u kada korisnik prodje autentifikacionu proceduru. Atribut se briše iz sesije `LogoutServlet`-a, odnosno kada korisnik klikne na link za odjavu sa aplikacije. Ako u korisnikovoj sesiji nije pronađen atribut `"user"` tj. korisnik nije ulogovan. U zahtev, `Http Request`, u atribut `"errors"` smestamo niz sa jednom greskom koja informiše korisnika da se prijavi. Korisnika redirektujemo na stranicu `index.jsp`. Pored `doFilter` postoje `destroy` i `init` funkcije. `Destroy` i `init` se pozivaju iz veb kontejnera i ukazuju na to da se pozivaju iz service-a. Ove metode se pozivaju samo jednom u `doFilter` funkciji. `Destroy` se izvršava kada istekne odedeni vremenski period, a `init` kada se instancira filter u `doFilter`.

Controller

Poslednji deo backend-a je controller. U controller-u se nalaze servlet java klase kojima se šalju instrukcije na frontend, tačnije na jsp strane. Kontroler je završna faza biznis logike i to su klase kojima upravljamo našom aplikacijom. Svaka servlet klasa u kontroleru mora imati odgovarajuće mapiranje u web.xml fajlu. Mapirana servlet klasa u web.xml-u ima mapiranje na lokaciju i mapiranje na url (jedinstveni resursni lokator – veb adresu). U ovom projektu postoji više različitih tipova servleta u controller-u i klasu TypeOfAction.java, kojom smo definisali akcije koje koristimo u servlet klasama. Na slici 37. nalazi se TypeOfAction klasa.

```
1 package dusan.project.statistics.controller;
2
3 import static dusan.project.statistics.controller.TypeOfAction.values;
4
5 public enum TypeOfAction {
6
7     LIST("list"), ADD("add"), EDIT("edit"), DELETE("delete");
8
9     private final String action;
10
11     private TypeOfAction(String action) {
12         this.action = action;
13     }
14
15     public static TypeOfAction getForAction(String action) {
16         for (TypeOfAction typeOfAction : values()) {
17             if (typeOfAction.action.equals(action)) {
18                 return typeOfAction;
19             }
20         }
21         return LIST;
22     }
23
24 }
25
```

Slika 37. TypeOfAction klasa

Kao što primećujemo definisana su četiri tipa promenljivih LIST, ADD, EDIT i DELETE. Posle toga promenljiva “action” tipa string i napravljen konstruktor sa argumentom action. Kreirali smo jednu funkciju getForAction koja pozivanjem klase preko argumenta action vraća ili list (return LIST;) ili petljom for, kojom prolazi kroz sve opcije typeOfAction i pitalicom if kojom ispituje da li je zadati acton, neki od ponuđenih sem LIST, vraća ADD, EDIT ili DELETE.

Od servleta u controller-u postoje servleti koji odgovaraju svakom timu Evrolige. Postoji 16 timova i za svaki smo napravili po JSP stranu. Oni su pisani istim načinom. i objasnimo jedan od tih servleta. Te JSP strane su deo koji je posvećen korisniku, a pre dolaska na stranu tima koji nas interesuje, a posle uspešnog logovanja postoji strana timovi, za koju je napisana TimServlet.java klasa. Što se tiče administratorskog dela, tu postoje tri različite JSP strane i svakoj odgovara servlet IgracServlet.java, UtakmicaServlet.java i StatistikaServlet.java. Kada se pokrene projekat, prvo se pokreće index.jsp strana i da bi sa te strane otišli na administratorski panel ili na korisničke strane potrebno je logovanje. Za logovanje i za odjavu smo kreirali LoginServlet.java i LogoutServlet.java. Na kodu ispod prikazana je LoginServlet klasa.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String nalog = request.getParameter("nalog");
    String sifra = request.getParameter("sifra");
    User user = new User(nalog, sifra);
    boolean authenticationSuccessful = false;
    try {
        authenticationSuccessful{
UserService.getInstance().checkLogin(user);
    } catch (StatisticsException ex) {
        request.setAttribute("errors", ex.getErrorMessages());
    }
    if (!authenticationSuccessful) {
        request.setAttribute("errors", new String[]{"Neuspešna prijava
na sistem."});
    } else {
        request.getSession().setAttribute("user", user);
        System.out.println(user);
    }
    if ("bane".equals(user.getNalog())) {
        String nextPage = (authenticationSuccessful) ? "utakmica" :
"index.jsp";
        request.getRequestDispatcher(nextPage).forward(request,
response);
    } else if ("djole".equals(nalog)) {
        String nextPage = (authenticationSuccessful) ? "tim" :
"index.jsp";
        request.getRequestDispatcher(nextPage).forward(request,
response);
    }
}
```

Klasa LoginServlet je produžetak HttpServlet biblioteke, pa tako preuzima sve njene funkcije, za menjanje istih potrebno ih je override-ovati. Kao što primećujemo override-ovana je doPost funkcija. Prvo smo u doPost funkciji, koja je jedina u ovoj klasi, kreirali promenljive tipa string nalog i sifra i u njih smestili parametre sa unosa korisnika. Onda smo kreirali objekat tipa User argumentima nalog i sifra. Potom smo kreirali promenljivu authenticationSuccessful tipa boolean i setoval je na false. U bloku try proverili da li je login uspešan sa istancom UserService-a i funkcijom checkLogin i argumentom user. Hvatamo izuzetak sa catch-om i pitalicom if proveravamo da li je authenticationSuccessful false, ako jeste šaljem odgovor tipa String da prijava nije uspešna. Ako je uspešna otvaramo sesiju sa request.getSession().setAttribute. Na kraju klase posotji if pitalica kojom rešavamo mesto gde

nam se logovao korisnik. Ukoliko je „bane“ sledeća strana je sa ista kao i strana sa url mapiranjem utakmica, a ukoliko je „djole“ ista kao stana tim. Tačnije napravili smo nalog za administratora sa nalogom „bane“ i šifrom „bane“, a za korisnika „djole“, „djole“. Na slici 38. se nalazi LogoutServlet kojim se korisnik ili administrator odjavljuju sa servisa.

```
1 package dusan.project.statistics.controller;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 public class LogoutServlet extends HttpServlet {
10
11     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
12         throws ServletException, IOException {
13
14         request.getSession().invalidate();
15
16         request.getRequestDispatcher("index.jsp").forward(request, response);
17     }
18
19     @Override
20     protected void doGet(HttpServletRequest request, HttpServletResponse response)
21         throws ServletException, IOException {
22         processRequest(request, response);
23     }
24
25     @Override
26     protected void doPost(HttpServletRequest request, HttpServletResponse response)
27         throws ServletException, IOException {
28         processRequest(request, response);
29     }
30
31 }
```

Slika 38. LogoutServlet

Komandom `request.getSession().invalidate()`; se zatvara sesija, a komandom ispod servisa šalje na `index.jsp` (početnu) stranu. Time smo sa bilo koje pozicije iz aplikacije došli na početnu, odnosno vratili smo se na poziciju sa koje nam je ponovo potrebno logovanje. U svakom servletu nalaze se override-ovane funkcije `doGet` i `doPost`, kojima šaljemo parametre ka JSP. Funkcijom `doGet` parametri se pojavljuju u URL-u i šalje se mala količina podataka, podaci nisu enkriptovani, brža je metoda i može se bezbedno ponavljati u kratkim vremenskim intervalima. Dok funkcijom `doPost` šaljemo parametre u odvojenim linijama, ne postoji ograničenje u količini informacija, parametri su enkriptovani, koriste se za update-ovanje podataka na serveru, sporija je metoda.

Administratorski nalog se sastoji od tri JSP strane, `utakmice.jsp`, `igraci.jsp` i `statistike.jsp`. Svako od tih JSP strana odgovara servlet `UtakmicaServlet`, `IgracServlet`, `StatistikaServlet` i sva tri servleta imaju slične funkcije, pa ćemo objasniti jedan. `UtakmicaServlet.java` klasa je produžetak klase `HttpServlet` i pored override-ovanih funkcija `doGet` i `doPost`, koje su napisane na dnu klase, ima funkciju `processRequest`. Svaka servlet klasa na ovom projektu ima te tri

funkcije napisane na sličan način. Pošto su korisničke strane koncipirane tako da posle izbora tima korisnik ima mogućnost samo da pregleda podatke, u servletima koji su vezani za timove i TimServlet klasi postoji samo list opcija kojom izlistavamo podatke iz baze. Kod UtakmicaServlet, IgracServlet i StatistikaServlet potrebn je da kreiramo i opcije za dodavanje (add), menjanje (edit) i brisanje (delete). Na kodu ispod je prikazana funkcija processRequest klase UtakmicaServlet.

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)throws ServletException, IOException {
    String action = request.getParameter("action");
    TypeOfAction typeOfAction = TypeOfAction.getForAction(action);
    try {
        switch (typeOfAction) {
            case LIST: {
                break;
            }
            case ADD: {
                Integer idUtakmica =
Integer.parseInt(request.getParameter("idUtakmica"));
                Integer tim_idTim1 =
Integer.parseInt(request.getParameter("tim_idTim1"));
                Tim tim1 = new Tim(tim_idTim1);
                Integer brojPoenaTim1 =
Integer.parseInt(request.getParameter("brojPoenaTim1"));
                Integer tim_idTim2 =
Integer.parseInt(request.getParameter("tim_idTim2"));
                Tim tim2 = new Tim(tim_idTim2);
                Integer brojPoenaTim2 =
Integer.parseInt(request.getParameter("brojPoenaTim2"));
                Utakmica utakmica = new Utakmica(idUtakmica, tim1,
brojPoenaTim1, tim2, brojPoenaTim2);
                UtakmicaService.getInstance().add(utakmica);
                request.setAttribute("message", "Utakmica " +
utakmica.getIdUtakmica() + " uspešno dodata.");
                break;
            }
            case EDIT: {
                Integer idUtakmica =
Integer.parseInt(request.getParameter("idUtakmica"));
                Integer tim_idTim1 =
Integer.parseInt(request.getParameter("tim_idTim1"));
                Tim tim1 = new Tim(tim_idTim1);
                Integer brojPoenaTim1 =
Integer.parseInt(request.getParameter("brojPoenaTim1"));
                Integer tim_idTim2 =
Integer.parseInt(request.getParameter("tim_idTim2"));
                Tim tim2 = new Tim(tim_idTim2);
                Integer brojPoenaTim2 =
Integer.parseInt(request.getParameter("brojPoenaTim2"));
                Utakmica utakmica = new Utakmica(idUtakmica, tim1,
brojPoenaTim1, tim2, brojPoenaTim2);
```

```

        UtakmicaService.getInstance().edit(utakmica);
        request.setAttribute("message", "Utakmica " +
utakmica.getIdUtakmica() + " uspešno izmenjena.");
        break;
    }
    case DELETE: {
        Integer idUtakmica =
Integer.parseInt(request.getParameter("idUtakmica"));
        Utakmica utakmica = new Utakmica(idUtakmica);
        UtakmicaService.getInstance().delete(utakmica);
        request.setAttribute("message", "Utakmica je uspešno
izbrisana.");
        break;
    }
}
if (typeOfAction != TypeOfAction.LIST) {
    request.setAttribute("message", "Uspešno izvršena
akcija.");
}
} catch (StatisticsException ex) {
    request.setAttribute("errors", ex.getErrorMessages());
}
List<Utakmica> utakmice =
UtakmicaDAO.getInstance().findAll(EntityManagerWrapper.getEntityManager());
request.setAttribute("utakmice", utakmice);
request.getRequestDispatcher("/WEB-
INF/pages/utakmice.jsp").forward(request, response);
}

```

Na početku funkcije definišemo string action kome preko request.getParameter dodeljujemo vrednost sa JSP strane. Potom definišemo TypeOfAction, pomoću klase istog imena, koju smo ranije opisali i argumenta action. U try bloku postoji switch opcija, koja nam u zavisnosti od vrednosti typeOfAction nudi sledeće funkcionalnosti. Za LIST, switch opcija pravi break i izlazi iz try bloka i prelazi na funkcionalnost izlistavanja saržaja i redirekciju JSP strane, o tome kasnije. Za opciju ADD, prvo se Integer promenljiva idUtakmica tako što se preko Integer.parseInt string preinači u Integer, a preko request.getParameter pokupi string sa unosa. Potom se istim komandama kreira tim_idTim1 promnljiva, a preko nje obejkat tim1. Posle kreirnja tima1 promenjlive isto kao i idUtakmica, kreira se brojPoenaTim1. Sada imamo id utakmice objekat tipa tim i broja poena za prvi tim. Potpuno istim postupkom se kreiraju tim2 i brojPoenaTim2 i time imamo sve podatke koji su nam potrebni da bi napravili objekat tipa Utakmica. Kada se kreira taj obejkat, preko UtakmicaService se poziva instanca, a komandom add i argumentom (utakmica) dodajemo podatke nove utakmice. Poslednjom komandom ispisujemo tekst da je utakmica uspešno dodata i sa komandom break izlazimo iz try bloka. Potpuno isti princip je i za edit slučaj u opciji switch, s'tim što umesto add, u liniji kada pozivamo UtakmicaService.getInstance koristimo komandu edit i argument utakmica, kojom menjamo postojeće podatke objekta utakmica. Poslednja opcija u try bloku je DELETE i kod nje se drugačije konstruiše objekat Utakmica. Tu koristimo konstruktor sa samo jednim argumentom idUtakmica. Promenljivu idUtakmica pretvaramo u integer oblik sa Integer.parseInt komandom, a sa request.getParameter uzimamo sa unosa string. Pozivamo UtakmicaService instancu i koristimo komandu delete da bi obrisali podatke utakmice sa id utakmicom koju smo uzeli sa JSP strane. Na kraju try bolka postoji if pitalica, kojom pitamo da li je typeOfAction različit od LIST, ako jeste a izvršena je neka akcija to se ispisom

potvrđuje. Posle toga se hvata izuzetak `StatisticsException`. Nakraju funkcije kreiramo listu utakmica preko komande `findAll` sa instancom `UtakmicaDAO` i preko `EntityManagerWrapper`-a. Ukoliko nam je u switch-u `LIST`, odmah se prelazimo na ovaj deo funkcije. Rezultat koji dobijemo smeštamo u objekat utakmice, koji ćemo posle pozivati na JSP strani. Poslednjom komandom `request.getRequestDispatcher - forward` šaljemo podatke na adresu u zagradi - `/WEB-INF/pages/utakmice.jsp`.

Sve ostale servlet klase imaju neki oblik ovog koda prilagodjen datoj klasi. Tako postoje servleti, koji su zaduženi za administratorski panel `IgracServlet`, `StatistikaServlet`, pa oni isto imaju try blok i u njemu switch opcije sa mogućnostima `ADD`, `EDIT` i `DELETE`. Razlika je u tome što nam za kreiranje objekata `igrac` trebaju drugačiji argumenti, nego za utakmicu. Međutim svi argumenti su ili integer oblika, pa ih prebacujemo iz stringa u integer, u već opisanj formi, ili su nekog drugog objekta čiji je argument integer, pa postoji korak više kao u opisanom slučaju kada kreiramo objekte `tim1` i `tim2` u objektu `utkamica`. Isti su principi za `ADD` i `EDIT`, dok kod `DELETE` opcije brišemo objekat tako što ga kreiramo prethodno pomoću konstruktora koji ima samo jedan argumet, id tog objekta. Na kraju svih servleta nalazi se opcija gde kreiramo liste objekata i nazivamo ih `igraci` ili statistike i posle ih u JSP stranama pozivamo u tom obliku. To su bile strane koje će koristiti administrator, pa ih nazivamo administratorski panel.

Na korisničkom delu aplikacije nalazi se prvo `timovi.jsp` kome odgovara `TimServlet`. U `TimServlet`-u nemamo opcije za dodavanje, menjanje i brisanje, pa nam je funkcija znatno kraća. Kada kreiramo action i `typeOfAction` imamo samo `LIST` opciju, i kreiramo listu, ali ne timova, nego kreiramo listu utakmica koja će nam se prikazati na `timovi.jsp` strani. Po istom principu kao i kod `UtakmiceServlet`, razlika je u poslednjoj liniji, jer usmeravamo servlet ka `/WEB-INF/pages/timovi.jsp` strani. Posle strane `timovi.jsp` na kojoj korisnik ima mogućnost pregleda rezultata utakmica i izbora tima koji ga interesuje. Pritiskom na određeno dugme sistem ga upućuje na stranu tima koji ga interesuje. Tada nam se aktivira servlet zadužen za tu stranu, pa tako postoje servleti `ArmaniServlet`, `BarseServlet`, `BaskoniaServlet`, `BroseServlet`, `CskaServlet`, `DarussafakaServlet`, `EfesServlet`, `FenerbahceServlet`, `GalataServlet`, `MacabiServlet`, `OlympiackosServlet`, `PanathinakiosServlet`, `RealServlet`, `UnicsServlet`, `ZalgirisServlet` i `ZvezdaServlet`, koji odgovaraju jsp stranama sa imenima istim samo bez dodatka `Servlet`. Ovi servleti su pisani na isti način kao `TimServlet`, samo se umesto liste utakmica ovde kreira lista statistika i umesto komande `findAll`, koristimo slične komande, npr. komandu `findAll4` za `Csk-a`, o kojoj smo već pisali. Kreirali smo funkcije `findAll` sa dodatkom id Tima koji nam treba pa je id `Csk-a` 4 i `findAll4` je funkcija koja izlistava sve statistike igrača koji nastupaju za `Csk-a`, zbog query upita u funkciji. Posle liste statistika poslednjom komandom `forwar-ujemo` na stranu jsp koja nam je potrebna u slučaju `CskaServleta`, šaljemo podatke na stranu `cska.jsp`. Na slici 39. se nalazi funkcija `processRequest` `ArmaniServlet` klase.

```

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String action = request.getParameter("action");
    TypeOfAction typeOfAction = TypeOfAction.getForAction(action);
    switch (typeOfAction) {
        case LIST: {
            break;
        }
    }
    List<Statistika> statistike = StatistikaService.getInstance().findAll();
    request.setAttribute("statistike", statistike);
    request.getRequestDispatcher("/WEB-INF/pages/armani.jsp").forward(request, response);
}

```

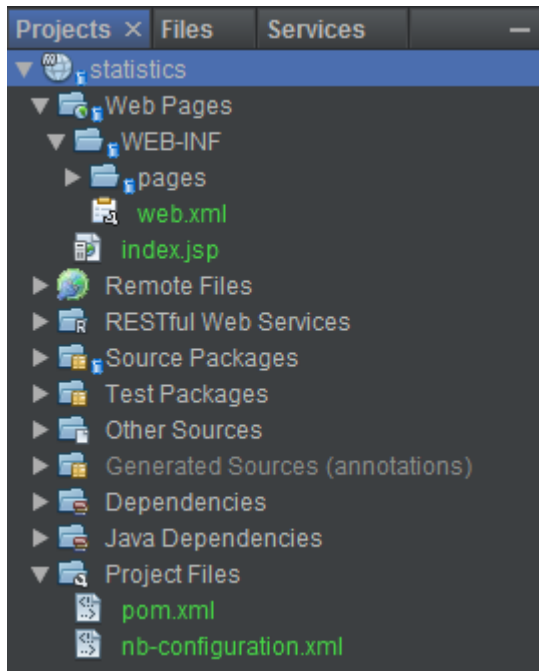
Slika 39. Funkcija processRequest iz ArmaniServlet.java

Pored processRequest svi ostali servleti koji upravljaju stranama timova imaju dve overrideovane klase o kojima smo pisali doGet i doPost isto napisane kao u klasi TimServlet.

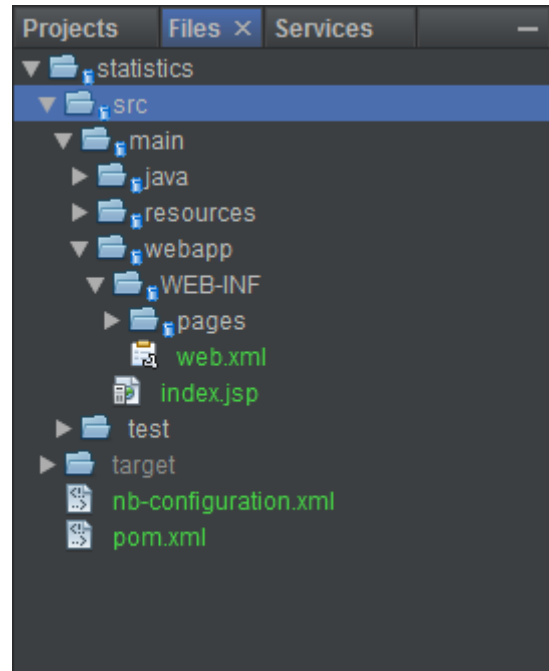
Opisom servleta smo opisali sve java klase. To je opis biznis logike na aplikaciji, tj. deo aplikacije koji upravlja procesima tako da dobijemo željeni efekat i da korisnici imaju mogućnosti na sistemu kakve smo zamislili na početku. Kada bi se aplikacija pustila u produkciju, tj. kada bi je postavili na internet da bude aktivna, suština održavanja bi se svodila da praćenje rada biznis logike, tj. održavanje java klasa. Hijerarhijom java klasa, odnosno načinom raspoređivanja klasa smo omogućili programerima koji bi se bavili održavanjem aplikacije da lakše pristupe klasama i da uvek mogu da dodaju neke parametre, čak i unarede biznis logiku.

7.2. Frontend aplikacije

Struktura fajlova kojima formiramo frontend smešteni su u datoteku main/webapp. NetBeans okruženje nam pruža dva pristupa fajlovima, o kojima smo već pisali, pa tako postoje Projects i Files pristup. Na slici 40. je prikazan Project tip, a na slici 41. Files.



Slika 40. Project meni



Slika 41. Files meni

Na slikama iznad je prikazana struktura frontend fajlova. Lako možemo primetiti, da je izdvojen fajl index.jsp, koji se nalazi u Web Pages ako gledamo Project meni, tj. u webapp ako gledamo Files meni. U tim folderima, koji god meni da gledamo nalazi se i folder WEB-INF. U pomenutoj datoteci se nalazi poddatoteka pages (strane) i web.xml konfiguracioni fajl o kome je već bilo reči. Sve ostale jsp strane nalaze se u datoteci pages. Podeli ćemo frontend u tri kategorije. Početna strana, koju čini index.jsp. Administratorski panel, koju čine stranice utakmice.jsp, igraci.jsp i statistike.jsp. Korisnički panel koju čine timovi.jsp kao strana iz koje idemo na pojedinačne timove, a strane timova su: armani.jsp, barsa.jsp, baskonia.jsp, brose.jsp, cska.jsp, darussafaka.jsp, efes.jsp, fenerbahce.jsp, galata.jsp, macabi.jsp, olympiacos.jsp, panathinaikos.jsp, real.jsp, unics.jsp, zalgiris.jsp i zveda.jsp.

Na aplikaciji su korišćene slike sa interneta, pa tako na sledećim veb adresama (linkovi ispod) se nalaze slike koje su korišćene na aplikaciji, a bile su dostupne 12.03.2017. na sledećim lokacijama:

- početna strana i sve korisničke strane koriste pozadinu plave boje, dostupno na <http://wallpapercave.com/wp/6ERktAi.png>
- početna strana sadrži logo Evrolige, dostupno na https://lh3.googleusercontent.com/iJwN6W909oEdb6a6UceSX93fdll9z-Z-YC8tajy8TRQT1DCTceJ6vcFgCBYX9LPD_S6E=w300
- početna strana sadrži karikaturu Željka Obradovića (trenera Fenerbahčea), dostupno na https://pbs.twimg.com/media/CigKC63WwAAH6_7.jpg
- na korisničkom panelu prva strana timovi sadrži sliku trofeja Evrolige, dostupno na http://www.eurohoops.net/wp-content/uploads/2015/04/14245459416_04c59ce611_b.jpg

index.jsp

Kada se pokrene program, prvo se pokrene index.jsp strana. Ideja je da to bude strana dobrodošlice sa nekim osnovnim informacijama o servisu (aplikaciji) i da imamo dugme kojim bi mogli da se logujemo na sistem. Svaka JSP strana nam počinje sa istim kodom, kojim definišemo vrstu pisma i uvodimo neke biblioteke čijim komandama se služimo na strani. Na slici 42. nalazi se vrh JSP strana.

```
<!DOCTYPE html>
<%@ page pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script type="text/javascript"
      src="http://cdnjs.cloudflare.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
    <script type="text/javascript"
      src="http://netdna.bootstrapcdn.com/bootstrap/3.3.2/js/bootstrap.min.js"></script>
    <link href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.3.0/css/font-awesome.min.css"
      rel="stylesheet" type="text/css">
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css"
      rel="stylesheet" type="text/css">
  </head>
```

Slika 42. početak index.jsp strane – kod

Kao što primećujemo postavljamo pageEncoding na pismo UTF-8. U head tagovima uvodimo biblioteke, koje omogućuju određene funkcionalnosti komandama kojima se kasnije služimo u kodu. Tako uvodimo jquery, bootstrap (javasript), font-awesome i bootstrap (CSS). Te biblioteke omogućavaju kodu da napravi funkcionalnost koje želimo. Naslov strane uvodimo kodom: `<title>Individualna statistika</title>`, a kodom iznad naslova kreiramo pozadinu:

```

<style>
  body {
    background-image:
      url("http://wallpapercave.com/wp/6ERktAi.png");
  }
  div { text-align: justify;
        text-justify: inter-word;
      }
</style>

```

Početna strana ima navigacioni panel na vrhu. To je, celom širinom strane traka na kojoj se nalaze ime aplikacije „Statistika“ u levom uglu i dugme „Prijava“ u desnom uglu. To smo postigli sledećim kodom:

```

<div class="navbar navbar-default navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target="#navbar-ex-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand">Statistika</a>
    </div>
    <div class="collapse navbar-collapse" id="navbar-ex-collapse">
      <ul class="nav navbar-nav navbar-right"></ul>
      <a class="btn btn-danger navbar-btn pull-right" data-
toggle="modal"
        data-target="#modal-signin">Prijava</a>
    </div>
  </div>
</div>

```

U prvom div tagu definišemo class-u navbar, u drugom otvaramo kontejner u kome definišemo navbar. Definišemo dugmiće koje ovde ne koristimo (ostavljamo ovako funkcionalnosti, zbog kasnijeg doradivanja aplikacije). Definišemo u a class-tagu ime navbara i na kraju dugme crvene boje (danger) na desnoj poziciji (right) i tipa modal. Target je (#modal-signin) bitan zato što ćemo kasnije kreirati prozor koji iskače na pritisak ovog dugmeta. Preko target-a pravimo vezu dugmeta i pop-out prozora.

Prilikom logovanja, postoji mogućnost da korisnik ili administrator unesu pogrešne podatke, ukoliko se to dogodi potrebno je da se ispiše greška, istu situaciju ćemo imati na svim stranama administratorskog panela. Problem sa greškama prilikom unosa rešavamo kodom kao sa slike 43. Na njoj se vidi da imamo tag sa c:if pitalicom, koja postavlja pitanje da li je errors prazan, ako nije ulazi se u dalji kod, koji sa for:Each petljom za sve promenljive error javlja grešku. Da bi greška bila ispisana potrebna je još jedan ovakav blok samo što uesto promenljive error stoji message, pa se tako prijavljene greške zapisuju u message (poruku) i ispisuju na veb strani.

```

<c:if test = "${not empty errors}">
  <div class="row">
    <div class="col-md-12">
      <div class="alert alert-danger" role="alert">
        <c:forEach items = "${errors}" var = "error">
          <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>
          <span class="sr-only">Error:</span>
          ${error} <br />
        </c:forEach>
      </div>
    </div>
  </div>
</c:if>

```

Slika 43. kod za grešku na index.jsp strani

Jednostavnim kodom ispisujemo pun naslov ispod nabara:

```

<div class="row">
  <div class="col-md-12 text-center">
    <h1 class="text-center">Individualni statistički parametri košarkaša
    Evrolige</h1>
  </div>
</div>

```

Ispod punog naslova otvaramo kontejner u koji stavljamo jumbotron polje, a u njega smeštamo tekst dobrodošlice i početno uputstvo. Na slici 44. se nalazi opisani kod.

```

<div class="container">
  <div class="row">
    <div class="col-md-12">
      <h4 class="jumbotron">
        Dobrodošli na sajt posvećen individualnoj statistici košarkaša Evrolige. Ovde ćemo
        se baviti individualnim učinkom igrača i detaljnom analizom pojedinih statističkih
        parametara na određenim utakmicama. Posle prijave na korisnički nalog
        imate mogućnost izbora timova koji vas interesuju i pristupa statistici
        igrača, odabranih timova. Ukoliko ste administrator, prijavite se sa administratorskim
        nalogom i imate mogućnost kreiranja, menjanja ili brisanja parametara.
        U svakom slučaju uživajte... &nbsp;  </h4>
      </div>
    </div>
  </div>
</div>

```

Slika 44. jumbotron na početnoj strani

Posle teksta smo postavili dve slike, logo Evrolige i karikaturu Željka Obradovića, trenera Fenerbahčea i najtrofejnijeg trenera u takmičenju. To smo postigli otvarajući kontejner i u njega smeštajući manje kontejnere sa kodom:

```




```

Širinu kontejnera smo rešili linijom: `<div class="col-md-3">`
gde nam prva slika ima širinu kontejnera 3, a druga 9.

Poslednja funkcionalnost na početnoj strani je prozor koji iskoči prilikom pritiska na prijava dugme. Kod ispod predstavlja primer za tu funkcionalnost:

```

<div class="modal fade" id="modal-signin" tabindex="-1" role="dialog">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-
          hidden="true">x</button>
        <h4 class="modal-title">Statistika</h4>
      </div>
      <form class="form-horizontal" role="form" action="login" method="POST">
        <div class="modal-body">
          <div class="form-group">
            <div class="col-sm-2">
              <label for="nalog" class="control-label">Nalog</label>
            </div>
            <div class="col-sm-10">
              <input type="text" class="form-control" id="nalog" name="nalog"
                placeholder="Korisničko ime">
            </div>
          </div>
          <div class="form-group">
            <div class="col-sm-2">
              <label for="sifra" class="control-label">Šifra</label>
            </div>
            <div class="col-sm-10">
              <input type="password" class="form-control" id="sifra"
                name="sifra" placeholder="Šifra">
            </div>
          </div>
          <div class="modal-footer">
            <a class="btn btn-default" data-dismiss="modal">Zatvori</a>
            <button type="submit" class="btn btn-
primary">Logovanje</button>
          </div>
        </form>
      </div>
    </div>
  </div>

```

Sa komandom `modal fade` u prvoj liniji definišemo prozor da je „nestajući“ tj. da kada se odradi komanda ispravno ili pogrešno prozor nestaje. Posle toga nam je veza sa targetom iz navbara preko `id`, za koji kaŕemo da je `modal-signin`. Posle smo definisali naslov (header) Statistika. U tagu `form` pravimo vezu sa backend-om preko `action` komande kojoj smo dodelili `login` vrednost i metod koju postavljamo na `POST`. Kreiramo nakon toga dva labela, trake koji će primati tekst od korisnika. Prvi label je `nalog`, koji ima naslov `Nalog` i prima tekst preko `input` komande gde definišemo tip `input`-a kao tekst, `id` definišemo kao `nalog`, ime kao `nalog` i prostor u kome se upisuje `nalog` svetlim slovima dok korisnik ne počne da kuca, stoji `Korisnički nalog`. Po istom principu formiran je i drugi label za `Šifru`. Potom smo kreirali klasu `modal-footer` koja nam elemente klase postavlja na dno porzora. Tu smo postavili dva dugmeta, „Zatvori“, za prekid i „Logovanje“ za komitovanje akcije, tj. za pokušaj logovanja kada se unese traŕeni tekst.

Administrativni panel

Administrativni panel čine tri JSP strane: utakmice.jsp, igrači.jsp i statistike.jsp. Administratoru su, kada se uloguje, u svakom trenutku preko navigacionog panela, dostupne sve tri strane. Na panelu levo se nalaze tri dugmeta, koja pritiskom prevode sa strane na stranu. Na navigacionom panelu sa desne strane se nalazi dugme „Izadi“ koje nas odjavljuje sa sistema i vraća na početnu stranu. Sve tri administratorske strane su pisane na sličan način. Prilikom otvaranja strane pojavljuju se tabele koje izlistavaju utakmice, igrače i statistike u zavisnosti na kojoj strani se nalazimo. Svaka vrsta tabele pored podataka sarži dugme „Izmeni“ i „Obriši“ kojim dati element tabele menjamo ili brišemo. Pored toga iznad tabele se nalazi dugme kojim dodajemo objekat strane na kojoj se nalazimo, znači utakmicu, igrača ili statistiku. Pošto su vrlo sličnog koda, objasnićemo igraci.jsp stranu. Razlika je u elementima objekta, igrača čine njegov id, Ime, Prezime, dok na primer kod Statistike imamo dosta više parametara (id, svaka statistička kategorija, id igrača, id tima). Posle toga imamo navbar, sličan kao kod index.jsp, s’tim što imamo dugmiće koje tamo nismo koristili. Na sledećem kodu prikazan je navbar sa igraci.jsp.

```
<div class="navbar navbar-default navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse"
        data-target="#navbar-ex-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand"><span>Statistika</span></a>
    </div>
    <div class="collapse navbar-collapse" id="navbar-ex-collapse">
      <ul class="nav navbar-nav">
        <li class="active">
          <a href="utakmica">Utakmice</a>
        </li>
      </ul>
      <ul class="nav navbar-nav">
      </ul>
      <ul class="nav navbar-nav">
        <li class="active">
          <a href="igrac">Igrači</a>
        </li>
      </ul>
      <ul class="nav navbar-nav">
        <li class="active">
          <a href="statistika">Statistike</a>
        </li>
      </ul>
      <ul class="nav navbar-nav navbar-right"></ul>
      <a class="btn btn-danger navbar-btn pull-right"
href="logout">Izadi</a>
    </div> </div>
  </div>
```

Na početku definišemo klasu navbar, a potom definišemo dugmiće koje ćemo ovde koristiti. Preko komande navbar-brand postavljamo naslov navbara. Iza naslova sledi definicija dugmića. Imamo prvo definisano dugme Utakmica. To je polje na navbar-u, koje kada pritisnemo, sistem nas prebaci na stranu sa url završetkom vrednosti promenljive href, a u ovom slučaju to je utakmica. Po istom principu smo definisali i strane igrac i statistika. Definiciju url završetka smo napravili u web.xml fajlu. Svi dugmići se nalaze na levom delu navbar-a, dok u desnom imamo definisano dugme za odjavu sa sistema. To dugme ima vrednost href-a "logout". Nakon navbar-a imamo kontejner u kome smo definisali naslov strane, u ovom slučaju Igrači. U sledeći kontejner smo smestili kod kojim upravljamo u slučaju grešaka, to je isti kod kao na index.jsp strani i već smo ga opisali. Postavljamo dugme za dodavanje igrača sa sledećim kodom:

```
<a class="btn btn-success pull-right" data-toggle="modal"
                                data-target="#modal-igrac-add">Dodaj
igracza</a>
```

Pritiskom na ovo dugme nam iskače prozor u kome nam se nude parametri koje treba da popunimo da bi dodali novog igrača u bazu. Kasnije ćemo dodati kod da bi to dugme dobilo funkcionalnosti, a vezu ćemo ostvariti preko modal-target-a koje smo definisali sa modal-igrac-add. Kreiramo tabelu. Ideja je da nam tabela ima u prvoj vrsti ime parametra, a da u kolonama ispod stoje vrednosti. Tako smo sledećim kodom kreirali prvu vrstu tabele sa sledećim paramterima:

```
<tr>
    <th>#</th>
    <th>Ime</th>
    <th>Prezime</th>
    <th>Tim za koji nastupa</th>
    <th>Izmeni</th>
    <th>Obriši</th>
</tr>
```

Na slici 45. prikazano je telo tabele, koje smo smestili u tbody tagove. Potom smo definisali c:forEach petlju koja nam prolazi kroz sve parametre koje servlet šalje iz baze podataka, a smestio je u promenljivu igraci. Tako imamo vrednosti promenljive items „\${igraci}”, sve oko reči igrači je sintaksa kojom se služi JSP strana.

```
<tbody>
    <c:forEach items="${igraci}" var="igrac">
        <tr>
            <td id="td_${igrac.idIgrac}">${igrac.idIgrac}</td>
            <td id="td_ime_${igrac.idIgrac}">${igrac.ime}</td>
            <td id="td_prezime_${igrac.idIgrac}">${igrac.prezime}</td>
            <td id="td_tim_idTim_${igrac.idIgrac}">${igrac.tim_idTim.imeTima}</td>
            <td>
                <a class="btn btn-info pull-right" data-toggle="modal"
                    data-target="#modal-igrac-edit" onclick="popuniEditFormu('${igrac.idIgrac}',
                        '${igrac.ime}', '${igrac.prezime}',
                        '${igrac.tim_idTim}', 'edit');" >Izmeni</a>
            </td>
            <td>
                <a class="btn btn-danger pull-right" data-toggle="modal"
                    data-target="#modal-igrac-delete" onclick="popuniDeleteFormu('${igrac.idIgrac}',
                        'delete');" >Obriši</a>
            </td>
        </tr>
    </c:forEach>
</tbody>
```

Slika 45. telo tabele sa igraci.jsp strane

U prvu vrstu smeštamo vrednosti id igrača, preko komande `${igrac.idIgrac}`, a čitamo je iz tabele `igrac` uzmi vrednost promenljive `idIgrac`. Po istom principu slažemo ostale parametre u vrsti dalje. Koristimo komandu preko koje uzimamo objekat, to uvek radimo preko `id`, pa tako uzimamo parametar objekta `igrac` preko `${igrac.idIgrac}`, a smeštamo vrednosti koje nas interesuju, pa stavljamo `${igrac.ime}` da bi nam sistem pokazao ime igrača sa `id`-jem koji smo tražili. Stavimo `${igrac.prezime}` za prezime, a `${igrac.tim_idTim.imeTima}` za tim, s tim što nam se ime tima pokazuje iz druge tabele preko stranog ključa, u ovom slučaju `tim_idTim`, pa tako čitamo iz tabele `igrac` uzmi strani ključ `tim_idTim` i preko njega nađi u tabeli gde je on primani ključ (`tim`) ime tima čiji je on primarni ključ. Na poslednja mesta stavljamo dva dugmeta. `Izmeni` i `Obriši`. Oba dugmeta su tipa (`data-toggle`) modal, sa targetom na modal-`igrac-edit` za `Izmeni` i modal-`igrac-delete`. Preko ovih targeta ćemo kasnije povezati kod za prozore koji će iskakati prilikom pritiska na dugme. Za oba dugmeta ćemo napisati javascript formu, pa ovde definišemo vezu sa skriptom preko `onclick` komandom i dodamo `popuniEditFormu` sa parametrima svih elemenata objekta `igrac` za `Izmeni` i `popuniDeleteFormu` samo sa argumentom `idIgraca`. Nakon toga samo sam zatvorili tagove.

Da bi nam dugmići koje smo formirali imali svoje funkcionalnosti, potrebno je da napišemo kod kojim će se prilikom pritiska na dugme pojavljivati prozori sa akcijom koja nam je potrebna. Na sledećem primeru je prikazan kod kojim formiramo funkcionalnost dugmeta za dodavanje igrača.

```
<div class="modal fade" id="modal-igrac-add">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-
          hidden="true">x</button>
        <h4 class="modal-title">Unesite podatke o novom igraču</h4>
      </div>
      <div class="modal-body">
        <form class="form-horizontal" role="form" action="igrac" method="POST">
          <input type="hidden" name="action" value="add" />
          <div class="form-group">
            <div class="col-sm-5">
              <label for="idIgrac" class="control-label">Id igrača</label>
            </div>
            <div class="col-sm-7">
              <input type="text" class="form-control" id="idIgrac"
                placeholder="Id igrača" name="idIgrac">
            </div>
          </div>
          <div class="form-group">
            <div class="col-sm-5">
              <label for="ime" class="control-label">Ime</label>
            </div>
            <div class="col-sm-7">
              <input type="text" class="form-control" id="ime" placeholder="Ime"
                name="ime">
            </div>
          </div>
          <div class="form-group">
            <div class="col-sm-5">
```

```

        <label for="prezime" class="control-label">Prezime</label>
    </div>
    <div class="col-sm-7">
        <input type="text" class="form-control" id="prezime"
        placeholder="Prezime" name="prezime">
    </div>
</div>
<div class="form-group">
    <div class="col-sm-5">
        <label class="col-xs-3 control-label">Tim</label>
    </div>
    <div class="col-sm-7">
        <input type="text" class="form-control" id="tim_idTim"
        placeholder="Tim" name="tim_idTim">
    </div>
</div>
<div class="modal-footer">
    <a class="btn btn-default" data-dismiss="modal">Zatvori</a>
    <button type="submit" class="btn btn-primary">Dodaj</button>
</div>
</form>
</div>
</div>
</div>
</div>

```

Prvim div tagom definišemo fade klasu, koja nam omogućava prozor iza koga je zamagljen ostali deo veb pregledača. U istom tagu definišemo id klase, i postavljamo ga na modal-igrac-add, čime pravimo vezu sa dugmetom, pošto nam je data-target definisan isto sa modal-igrac-add. Posle toga definišemo da je klasa modal-dialog i modal-content, tj. da imamo komunikaciju sa korisnikom. Definišemo naslov prozora (header) „Unesite podatke o novom igraču“. Potom definišemo telo prozora. Form tagom definišemo tip action i dodeljujemo mu igrac, a method POST, čime se povezujemo sa backend-om i šaljemo informacije servletu. Dok u liniji ispod definišemo value za input i dodeljujemo add čime u servletu pozivamo add tip u switch-u IgracServlet klase. Nakon toga imamo definisane div klase „form-group“ za pojedinačan parametar unosa. Definicija div klase „col-sm-5“ nam omogućava veličinu prostora za naziv parametra, a „col-sm-7“ veličinu prostora za unos vrednosti. Imamo tagve label i input. Label nam definiše preko for komande i vrednosti koju unosimo „idIgrac“ da id igrača dodeljujemo sledećim input-om. Input nam definiše nekoliko parametara, redom, tip unosa je tekst (type), promenljiva se definiše sa id, a mi joj dodeljujemo u ovom slučaju idIgrac, kasnije ime, prezime i td. Komandom placeholder dodeljujemo vrednost koja će se ispisivati na prostoru za unos vrednosti dok korisnik ne počene da unosi podatak. Name je naziv promenljive. Do kraja grupe se zatvore tagovi. Na isti način smo kreirali unos za ime, prezime i tim. Na kraju koda smo dodali dva dugmeta Zatvori i Dodaj. Dugme Zatvori sa data-dismiss="modal" definisali kao dugme koje nam zatvara otvoreni prozor. Dugme Dodaj sa type="submit" nam šalje podatke na servlet da se dalje obrađuju.

Prozor za menjanje podataka o igraču, ima vrlo sličan HTML kod, sa dodatkom javascripta, o kome će kasnije biti reči. Što se tiče HTML koda. postoje manje izmene. Na početku prvog taga div imamo id klase definisan sa modal-igrac-edit. Kod prvog input taga u modal-body div klase imamo edit na poziciji value (imali add kod dodavanja).

```
<input type="hidden" name="action" value="edit" />
```

Poromenu smo pravili kod input taga u modal-form div klasi, tj. u unosu za pojedinačni parametar.

```
<input type="text" class="form-control" id="idIgrac-edit" placeholder="Id igrača" name="idIgrac">
```

Tu nam id parametra ima vrednost idIgrac-edit. To je deo prozra kod koga unosimo parametar idIgrac. Tako ćemo dalje kroz kod imati ime-edit, prezime-edit i tim-edit, za svaki parametar koji unosimo. Poslednja razlika je u imenu dugmeta sa submit vrednošću na kraju prozora, Izmeni je naziv dugmeta.

Na slici 46. se nalazi HTML kod za prozor kojim brišemo objekat tipa igrač. Id klase u prvom tagu je modal-igrac-delete. U ovom prozoru nam je bitna konekcija sa objektom tipa igrač, njega ostvarujem preko input forme koja nam je hidden (sakrivena) i id promenljive je idIgrac-delete, pošto smo na servletu definisali da ćemo samo preko tog parametra pristupati objektu koji brišemo. Razlika u odnosu na prethodna dva prozora je i u naslovu prozora „Izbrišite igrača“ i na imenu poslednjeg dugmeta „Obriši“.

```
<div class="modal fade" id="modal-igrac-delete">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-hidden="true">x</button>
        <h4 class="modal-title">Izbrišite igrača</h4>
        <form class="" role="form" action="igrac" method="POST">
          <input type="hidden" name="action" value="delete" />
          <input type="hidden" id="idIgrac-delete" name="idIgrac" value="" />
          <div class="modal-footer">
            <a class="btn btn-default" data-dismiss="modal">Zatvori</a>
            <button type="submit" class="btn btn-primary">Obriši</button>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

Slika 46. HTML kod za delete prozor u igraci.jsp

Od koda na igraci.jsp strani sem zatvaranja tagova, ostalo je još i javascript, koje pišemo na kraju fajla u tagovima <script> i </script>. Na slici 50. je prikazan skript koji koristimo za edit i delete dugme. U kodu koji smo već opisali, kod tabele imali smo na kraju svake vrste dugmiće Izmeni i Obriši, kojima smo dodelili onclick vrednosti popuniEditFromu i popuniDeleteFromu, upravo te dve funkcije smo kreirali u scriptu kao na slici 47.

```

<script>

function popuniEditFormu(idIgrac, ime, prezime, tim_idTim) {
    document.getElementById("idIgrac-edit").value = idIgrac;
    document.getElementById("ime-edit").value = ime;
    document.getElementById("prezime-edit").value = prezime;
    document.getElementById("tim_idTim-edit").value = tim_idTim;
}

function popuniDeleteFormu(idIgrac) {
    document.getElementById("idIgrac-delete").value = idIgrac;
}

</script>

```

Slika 47. script sa igraci.jsp strane

Tako definisane funkcije uzimaju parametre iz zagrade da bi im pristupile. Komandom `document.getElementById` pristupa se unosu npr. `idIgrac-edit` iz edit forme, kod `popuniEditFormu` funkcije, i postavlja se na mesto podatka u servletu, koji dalje šalje u bazu. Tim postupkom pristupamo svim podacima i postavljamo ih na promenljive koje nam odgovaraju. Potpuno isti princip je za funkcije `popuniDeleteFormu`, samo što tom funkcijom pristupamo objektu samo preko id i kasnije ga brišemo.

Ostale dve strane administratorskog panela `utakmice.jsp` i `statistike.jsp` su pisane na potpuno isti način, razlika je jedino u parametrima tih objekata. Tako npr. za utakmicu imamo `idUtakmica`, `tim_idTim1`, `brojPoenaTim1`, `tim_idTim2`, `brojPoenaTim2`, pa smo kod kreiranja i tabele i prozora za dodavanje i menjanje podataka koristili te parametre i njih obrađivali. Kod `statistike.jsp` postoji `idStatistika`, `igrac_idIgrac`, `utakmica_idUtakmica` i svi statistički parametri kao što su poeni, minuti, indeks, asistencije i td. Koristili smo sve te parametre za kreiranje i tabele i prozora kojima menjamo ili dodajemo parametre. Za brisanje smo se u oba slučaja služili kao i kod `igraci.jsp` strane, pristupali smo samo preko id objekta i kasnije ga uklanjali. Time smo kreirali administrativni panel i bez ostalog koda za korisnički panel dobili smo deo aplikacije kojim upravljamo podacima. Poluarno nazvano CRUD (Create, Read, Update, Delete) aplikacija predstavlja sistem u kom možemo da kreiramo, čitamo, menjamo i brišemo podatke. Upravo sve te funkcionalnosti na našem administratorskom panelu imamo. Ono što je svestranost ovako kreiranog sistema je da možemo da menjamo podatke na više nivoa, pa tako imamo tim, kao nepromenljivu osnovu, u koji možemo da dodajemo igrače. Timove da spajamo preko utakmica, a kroz utakmice da kreiramo pojedinačne statistike za igrače koji nastupaju na njima.

Korisnički panel

Korisnički panel se sastoji od strane timovi.jsp, koja predstavlja svojevrsnu raskrslu, jer na njoj korisnik bira tim koji ga interesuje i ostalih strana za svaki tim pojedinačno: armani.jsp, barsa.jsp, baskonia.jsp, brose.jsp, cska.jsp, darussafaka.jsp, efes.jsp, fenerbahce.jsp, galata.jsp, macabi.jsp, olympiacos.jsp, panathinaikos.jsp, real.jsp, unics.jsp, zalgiris.jsp i zveda.jsp. Strane pojedinačnih timova su pisane na isti način, pa ćemo objasniti jednu od njih.

Strana timovi.jsp ima isto zaglavlje, pozadinu u navbar kao i sve dosad opisane strane, sa izmenom u odnosu na stale strane, jer u zaglavlju postoji samo jedno dugme sa href vrednošću tim. Takvo dugme nam na ovoj strani ne znači previše, jer pritiskom na njega prelazimo na stranu timovi.jsp (na njoj se već nalazimo) tj. ostajemo tu, ali nam znači takav navbar na stranama timova, jer se tim dugmetom vraćamo upravo na ovu stranu. Ispod navbara imamo u jednom kontejneru dva jumbotrone a jedan do drugog iste širine, to smo rešili komandom col-md-6. U prvi smo smestili tekst sa objašnjenjem o izboru timova, o tabeli na dnu strane i sezoni za koju je aplikacija pravljen. Ispod smo stavili sliku trofeja u Evroligi. U drugi jumbotron smo smestili tabelu sa imenima timova i dugmićima u nastavku imena na kojima piše „Dalje“. Korisnik bira tim tako što pritiska dugme u nastavku imena i servis ga prebacuje na stranu tima. Na slici 48. prikazan je deo koda za tabelu kojom biramo tim.

```
<th>Anadolu Efes Istanbul</th><td><a href="efes" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Baskonia Vitoria Gasteiz</th><td><a href="baskonia" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Brose Bamberg</th><td><a href="brose" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Crvena Zvezda</th><td><a href="zvezda" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>CSKA Moscow</th><td><a href="cska" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Darussafaka Dogus Istanbul</th><td><a href="darussafaka" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>EA7 Emporio Armani Milan</th><td><a href="armani" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>FC Barcelona Lassa</th><td><a href="barsa" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Fenerbahce Istanbul</th><td><a href="fenerbahce" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Galatasaray Odeabank Istanbul</th><td><a href="galata" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Macabi FOX Tel Aviv</th><td><a href="macabi" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Olympiacos Piraeus</th><td><a href="olympiacos" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Panathinaikos Athens</th><td><a href="panathinaikos" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Real Madrid</th><td><a href="real" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Unics Kazan</th><td><a href="unics" class="btn btn-success" role="button">Dalje</a></td></tr><tr>
<th>Zalgiris Kaunas</th><td><a href="zalgiris" class="btn btn-success" role="button">Dalje</a></td></tr></tr>
```

Slika 48. tabela za izbor tima sa timovi.jsp

Svaki od imena timova koji smo prvo pisali u tagovima <th> dobija vrednost href-a u tagu <td> npr. CSKA Moscow dobija vrednost cska, koji smo definisali u web.xml kao završetak url-a za servlet CskaServlet.java, a koji na strani cska.jsp u navbaru dodeljujemo href-u. Tako pravimo vezu dugmeta i strane. Ispod toga se nalazi tabela sa rezultatima, koju smo kreirali kao na strani utakmice, samo što nismo pravili dugmiće za menjanje i brisanje. Na slici 49. se nalazi kod za tu tabelu. Tabela je potpuno isto definisana kao već opisana tabela na strani igraci.jsp, samo ima drugačije parametre objekta.

```

<table class="table table-bordered table-hover"
  style="width:100%;text-align:left;background-color:whitesmoke;"
  >
  <thead>
    <tr>
      <th>#</th>
      <th>Domaćin</th>
      <th>Broj poena</th>
      <th>Gost</th>
      <th>Broj poena</th>
    </tr>
  </thead>
  <tbody>
    <c:forEach items="${utakmice}" var="utakmica">
      <tr>
        <td id="td_${utakmica.idUtakmica}">${utakmica.idUtakmica}</td>
        <td id="td_tim_idTim1_${utakmica.idUtakmica}">${utakmica.tim_idTim1.imeTima}</td>
        <td id="td_brojPoenaTim1_${utakmica.idUtakmica}">${utakmica.brojPoenaTim1}</td>
        <td id="td_tim_idTim2_${utakmica.idUtakmica}">${utakmica.tim_idTim2.imeTima}</td>
        <td id="td_brojPoenaTim2_${utakmica.idUtakmica}">${utakmica.brojPoenaTim2}</td>
      </tr>
    </c:forEach>
  </tbody>
</table>

```

Slika 49. kod za tabelu sa rezultatima na strani timovi.jsp

Poslednja strana za korisnika. U trenutku kada korisnik izabere stranu i pritisne na dugme dalje sistem ga prebacuje na stranu željenog tima. Pošto su sve strane pisane isto sa izuzetkom vrednosti dugmeta u navbaru, opišaćemo jednu od strana npr. cska.jsp. Zaglavlje strane cska.jsp za razliku od prethodno opisanih sadrži dodatak biblioteka kojima uvodimo highcharts grafikone. Na kodu ispod vidimo i kako.

```

<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/data.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
<script src="https://code.highcharts.com/modules/drilldown.js"></script>

```

Nakon zaglavlja dodajemo pozadinu kao što je već opisano, a potom i navbar. Navbar je pisan kao i na svim stranim, sa dodatkom dva dugmeta u levom uglu kao na kodu ispod.

```

<ul class="nav navbar-nav">
  <li class="active">
    <a href="tim">Timovi</a>
  </li>
</ul>
<ul class="nav navbar-nav">
  <li class="active">
    <a href="cska">CSKA Moscow</a>
  </li>
</ul>

```

Primećujemo da imamo dva dugmeta Timovi i Cska Moscow, a href komanda uzima vrednost u prvom slučaju tim, a u drugom cska. Pritiskom na prvo dugme sistem nas vraća na stranu za izbor timova, dok pritiskom na drugo ostajemo na istoj strani, ali se ona osvežuje (ponovo učitava). Ispod toga imamo šest grafikona. Zbog preglednosti prva tri smo stavili u jedan jumbotron, sa širinom kontejnera col-md-4 pojedinačno, što je trećina širine strane za ekran desktop računara ili lap-topa. Ispod tog jumbotrona stavili smo još jedan u koji smo smestili dva grafikona, prvi sa širinom col-md-8 i drugi sa širinom col-md-4. Ispod se nalazi još jedan jumbotron sa velikim grafikonom, pa tabela sa podacima svih statistika koje su unešene. Svi grafikoni su istog tipa, pa ćemo objasniti grafikon koji se odnosi na minute i poene.

Odvojimo grafikon u dva dela. HTML deo i script deo. Na slici 50. se nalazi HTML kod, a na slici 51. script kod.

```
<div id="containerMinutiPoeni" style="min-width: 310px; height: 400px; margin: 0 auto"></div>
<table id="datatable2">
  <thead style="display:none">
    <tr>
      <th></th>
      <th>Minuti</th>
      <th>Poeni</th>
    </tr>
  </thead>
  <tbody style="display:none">
    <c:forEach items="${statistike}" var="statistika">
      <tr>
        <th></th>
        <td>${statistika.minuti}</td>
        <td>${statistika.poeni}</td>
      </tr>
    </c:forEach>
  </tbody>
</table>
```

Slika 50. HTML kod grafikona minuti i poeni sa strane cska.jsp

HTML forma je forma tabele, u kom definišemo id div-a i pose toga stil. U liniji ispod definišemo id tabele. U tagu thead smo dodali za komandu style vrednost display:none, zato što ovi grafikon u osnovnoj formi ispisuju podatke ispod grafikona u nesređenom odliku, a te podatke ćemo imati i tabeli ispod svih grafikona, pa nije bilo potrebno stavljati ih na dva mesta. Ista situacija je i sa tbody tagom. Prvo smo definisali promenljive kolona (Minuti i Poeni). U telu smo koristili petlju c:forEach da nam za svaku statistiku iz baze unese podatak u tabelu, koju će grafikon da iscrtati. Iz baze smo komandama \${statistika.minuti} i \${statistika.poeni} uzeli podatke za minute i poene. Nakon toga smo zatvorili tagove. U scriptu smo definisali funkciju, a potom sa komandom \$('#containerMinutiPoeni').highcharts smo napravili vezu sa div tagom u kom smo id promenljivoj dodelili „containerMinutiPoeni” vrednost. Potom smo definisali tabelu iz koje grafikon uzima podatke, tabela : datatable2. Tom komandom smo ostvarili vezu sa tabelom, jer smo u kodu tabele definisali id sa datatable2. Tip nam je kolona, a naziv grafikona „Minuti i Poeni kroz sezonu“. Sa komandom allowDecimals, omogućujemo korišćenje decimala na grafikonu. Poslednjom komandom tooltip definišemo način pisanja na grafikonu.

```

<script>
$(function () {
    $('#containerMinutiPoeni').highcharts({
        data: {
            table: 'datatable2'
        },
        chart: {
            type: 'column'
        },
        title: {
            text: 'Minuti i Poeni kroz sezonu'
        },
        yAxis: {
            allowDecimals: true,
            title: {
                text: ''
            }
        },
        tooltip: {
            formatter: function () {
                return '<b>' + this.series.name + '</b><br/>' +
                    this.point.y + ' ' + this.point.name.toLowerCase();
            }
        }
    });
});
</script>

```

Slika 51. javascript kod za grafikon minuti i poeni na strani cska.jsp

Svi grafikoni na aplikaciji su napisani na opisan način, menjani su samo parametri objekta statistika. Na kraju strane posle grafikona smo smestili tabelu sa svim statistikama koje smo uneli. Tabela je kreirana kao i prethodne opisane u ovom radu. Na slici 52. je prikazano telo tabele, sa svim parametrima koje objekat statistika sadrži. Time smo prikazali kod strane cska.jsp, a i sve ostale, zato što su pisane po istom modelu.

```

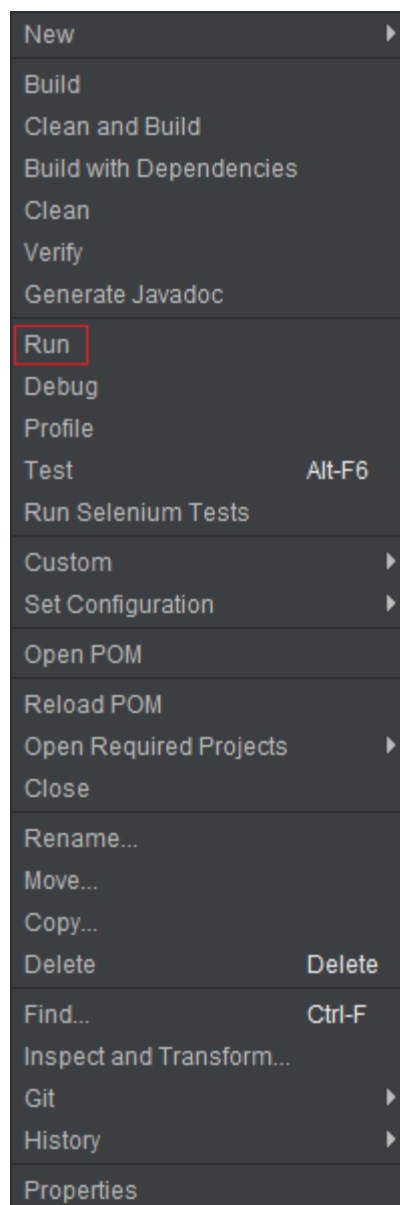
<tbody>
<c:forEach items="${statistike}" var="statistika">
<td id="td_igrac_idIgrac${statistika.igrac_idIgrac}">${statistika.igrac_idIgrac.ime}
    ${statistika.igrac_idIgrac.prezime}</td>
<td id="td_minuti${statistika.minuti}">${statistika.minuti}</td>
<td id="td_poeni${statistika.poeni}">${statistika.poeni}</td>
<td id="td_sutZa2Pogodjeni${statistika.sutZa2Pogodjeni}">${statistika.sutZa2Pogodjeni}</td>
<td id="td_sutZa2Promaseni${statistika.sutZa2Promaseni}">${statistika.sutZa2Promaseni}</td>
<td id="td_sutZa3Pogodjeni${statistika.sutZa3Pogodjeni}">${statistika.sutZa3Pogodjeni}</td>
<td id="td_sutZa3Promaseni${statistika.sutZa3Promaseni}">${statistika.sutZa3Promaseni}</td>
<td id="td_penaliPogodjeni${statistika.penaliPogodjeni}">${statistika.penaliPogodjeni}</td>
<td id="td_penaliPromaseni${statistika.penaliPromaseni}">${statistika.penaliPromaseni}</td>
<td id="td_ofanzivniSkokovi${statistika.ofanzivniSkokovi}">${statistika.ofanzivniSkokovi}</td>
<td id="td_defanzivniSkokovi${statistika.defanzivniSkokovi}">${statistika.defanzivniSkokovi}</td>
<td id="td_asistencije${statistika.asistencije}">${statistika.asistencije}</td>
<td id="td_ukradeneLopte${statistika.ukradeneLopte}">${statistika.ukradeneLopte}</td>
<td id="td_izgubljeneLopte${statistika.izgubljeneLopte}">${statistika.izgubljeneLopte}</td>
<td id="td_blokadeNapraavljene${statistika.blokadeNapraavljene}">${statistika.blokadeNapraavljene}</td>
<td id="td_blokadePrimljene${statistika.blokadePrimljene}">${statistika.blokadePrimljene}</td>
<td id="td_fauloviNapraavljeni${statistika.fauloviNapraavljeni}">${statistika.fauloviNapraavljeni}</td>
<td id="td_fauloviPrimljeni${statistika.fauloviPrimljeni}">${statistika.fauloviPrimljeni}</td>
<td id="td_indeks${statistika.indeks}">${statistika.indeks}</td>
</tr>
</c:forEach>
</tbody>

```

Slika 52. telo tabele sa dna strane cska.jsp

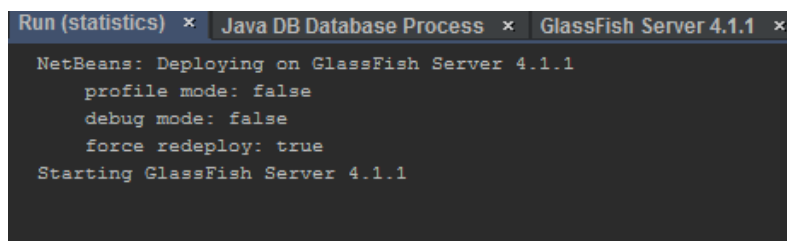
8. Funkcionalnost aplikacije

Kada smo završili sa pisanjem koda, potrebno je proveriti sve funkcionalnosti i izgled aplikacije koju smo kreirali. Tokom pisanja koda smo jednu po jednu funkcionalnost proveravali, prelazak na kucanja daljeg koda je zavisio od uspešnosti provera funkcionalnosti. U svakom trenutku prilikom pisanja koda imamo mogućnost da iz NetBeans okruženja pokrenemo projekat. Desnim klikom na ime projekta u meniju Projects, otvara nam se padajući meni u kom imamo više ponuđenih opcija. Komanda za pokretanje aplikacije je “Run”. Na



Slika 53. padajući meni

slici 53. prikazan je izgled padajućeg menija. Alternativa za pokretanje aplikacije je prečica tj. dugme F6. Kada se pokrene projekat u prozoru Output pojavljuju se tri taba. Prvi je Run (pokrenuta aplikacija), Drugi je Java DB Database Process (veza sa bazom) i GlassFish Server (veza sa serverom). Na slici 54. prikazan je izgled Output prozora za vreme pokretanja aplikacije.



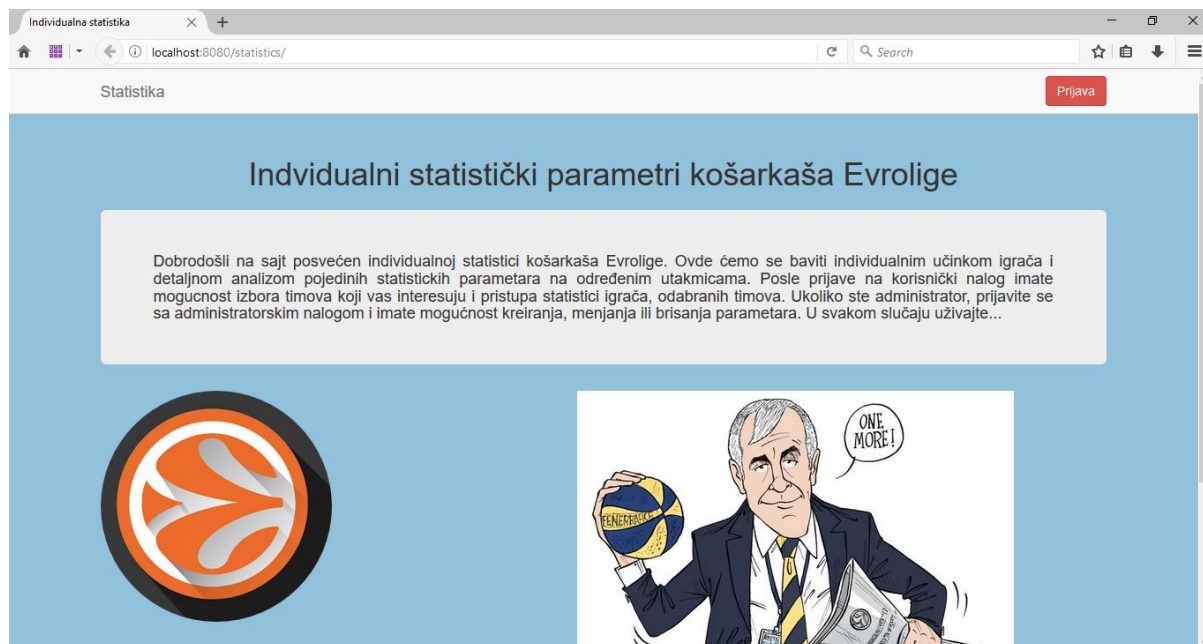
Slika 54. Output prozor

U Properties-u svakog projekta se mogu podešavati različite karakteristike. Između ostalog za veb aplikacije, ponuđeni su veb pretraživači koji su na raspolaganju i u kojima će se pokrenuta aplikacija prezentovati. U slučaju naše aplikacije konfigurisana je Mozilla kao pregledač u kom će se aplikacija izvršiti. Pored Run opcije, treba obratiti pažnju i na “Clean and Build” komandu, kojom se “čisti i bilduje” projekat. Važnost ove komande je u tome što Maven projekti pomoću ove komande dovlače u lokalni repozitorijum sve biblioteke koje projekat koristi, pa je ubuduće pokretanje projekta znatno brže, jer se svi potrebni resursi nalaze na lokalnoj mašini. Imamo i odvojene opcije Clean i Build. Sa opcijom Clean, NetBeans čisti nepotrebne parametre projekta. Tako da možemo i odvojeno koristiti te funkcionalnosti.

8.1. Izgled aplikacije

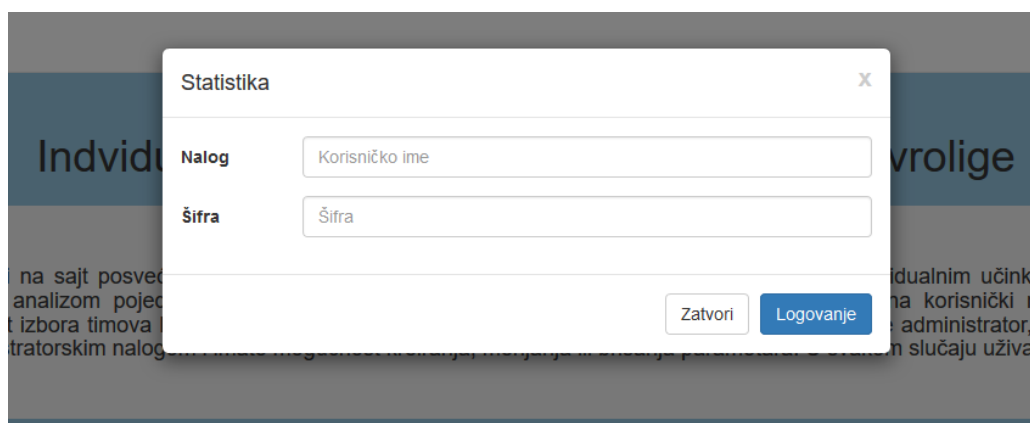
Početna strana

Na početnoj strani se nalaze tekst dobrodošlice sa uputstvom za korišćenje aplikacije. dugme za prijavu na sistem i karikatura najtrofejnijeg trenera Evrolige (Željka Obradovića). Na slici 55. nalazi se početna strana. Skrolanjem strane ka dnu izlistava se cela slika.



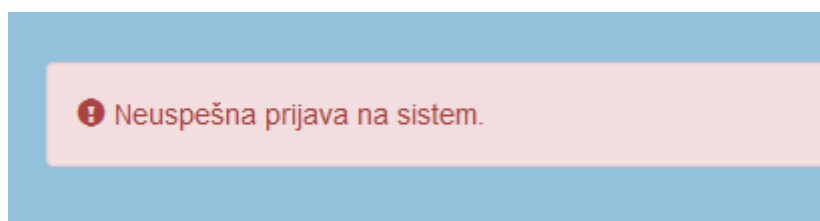
Slika 55. Početna strana aplikacije

Pritiskom na dugme za prijavu iskače prozor za logovanje, ostatak strane se zamagljuje i postaje nedostupan dok se ne unesu parametri za logovanje i klikne na dugme „Logovanje“ ili dok se ne zatvori prozor. Na slici 56. nalazi se izled početne strane kada se pritisne dugme za prijavu.



Slika 56. Prozor za prijavu na početnoj strani

Postoje dva odvojena panela o kojima smo pisali, administrativni i korisnički panel. Administrativni panel je za osobu koja bi bila zadužena za rad na aplikaciji sa ulogom kreiranja, menjanja i brisanja podataka sa sistema. Korisnički panel je predviđen za posetioce sajta koji su zainteresovani za podatke. Ukoliko posetilac sajta nema odgovarajuće parametre za prijavu ili greškom unese pogrešne, sistem će ga vratiti na početnu stranu sa dodatkom teksta “Neuspešna prijava na sistem” uokvirenog crvenom bojom na vrhu strane, kao na slici 57.



Slika. 57. Prijava greške prilikom prijave na sistem

Administrativni panel - Utakmice

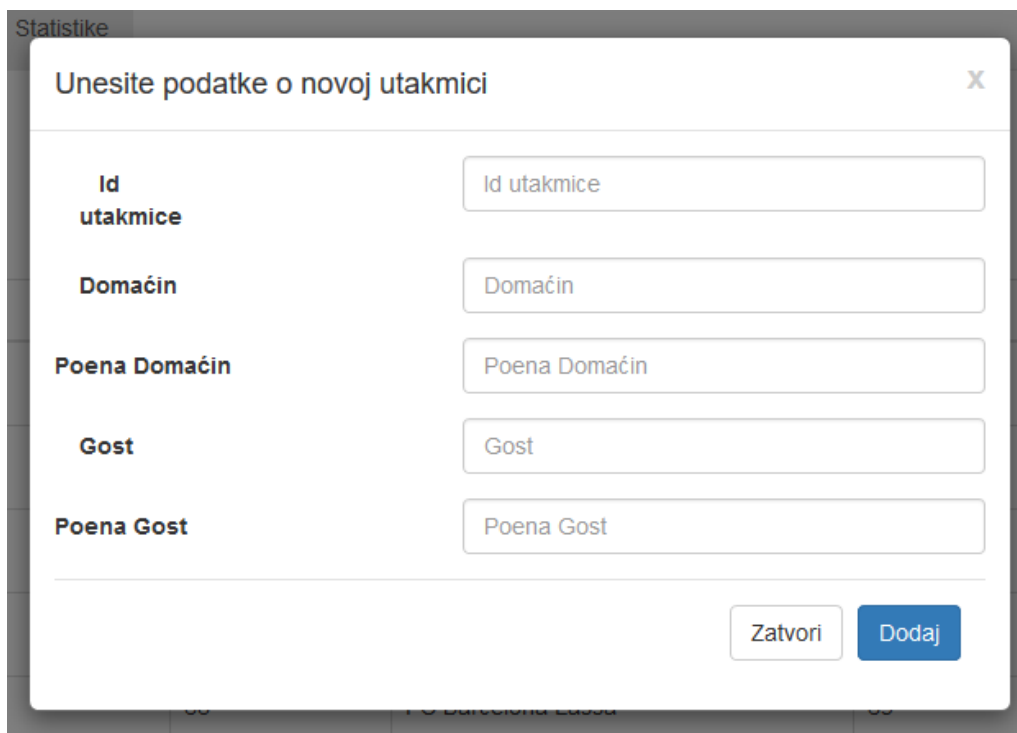
Na administrativni panel se prijavljujemo sa nalogom “bane” i šifrom “bane”. Kada se uojujemo otvara nam se stranica na kojoj su izlistane utakmice i na toj strani se možemo baviti podacima vezanim za utakmice. Na slici 58. je strana Utakmica.

#	Domaćin	Broj poena	Gost	Broj poena	Izmeni	Obriši
1	Real Madrid	83	Olympiacos Piraeus	65	Izmeni	Obriši
2	Galatasaray Odeabank Istanbul	84	CSKA Moscow	109	Izmeni	Obriši
3	EA7 Emporio Armani Milan	99	Macabi FOX Tel Aviv	97	Izmeni	Obriši
4	Crvena Zvezda	70	Darussafaka Dogus Istanbul	73	Izmeni	Obriši
5	Unics Kazan	63	FC Barcelona Lassa	69	Izmeni	Obriši
6	Fenerbahce Istanbul	67	Brose Bamberg	66	Izmeni	Obriši
7	Panathinaikos Athens	84	Zalgiris Kaunas	76	Izmeni	Obriši
8	Baskonia Vitoria Gasteiz	85	Anadolu Efes	84	Izmeni	Obriši
9	Olympiacos Piraeus	90	Anadolu Efes	66	Izmeni	Obriši

Slika 58. Strana Utakmica

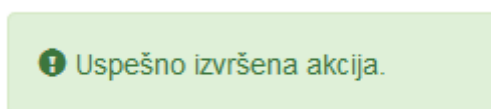
Skrolanjem na dole izlistava se tabela sa rezultatima do kraja strane. Svaka utakmica ima dva dugmeta na kraju, plavo dugme “Izmeni” i crveno “Obriši”. Iznad tabele nalazi se zeleno dugme “Dodaj utakmicu”. Pritiskom na zeleno dugme iskače prozor na kome možemo da unesemo podatke za dodavanje parametara utakmice. Ostatak strane se zamagljuje i

nedostupan je dok se ne doda utakmica ili dok se ne izađe iz prozora. Na slici 59. nalazi se prozor koji iskače prilikom pritiska na dugme “Dodaj utakmicu”.



Slika 59. Prozor za dodavanje nove utakmice

Unosimo podatke za utakmicu. Id (16), Domaćin (8, id Barselone), Poena Domaćin (72), Gost (9, id Fenerbahčea), Poena Gost (74). Pritiskamo na Dodaj dugme. Ispod naslova strane “Utakmice” ispisuje se, zelenom uokviren tekst “Uspešno izvršena akcija”. Na dnu tabele se dodaje utakmica pod rednim brojem 16. sa timovima i rezultatom koji smo uneli. Na slici 60. prikazana je poruka prilikom dodavanja, a na slici 61. dodata utakmica u tabeli.

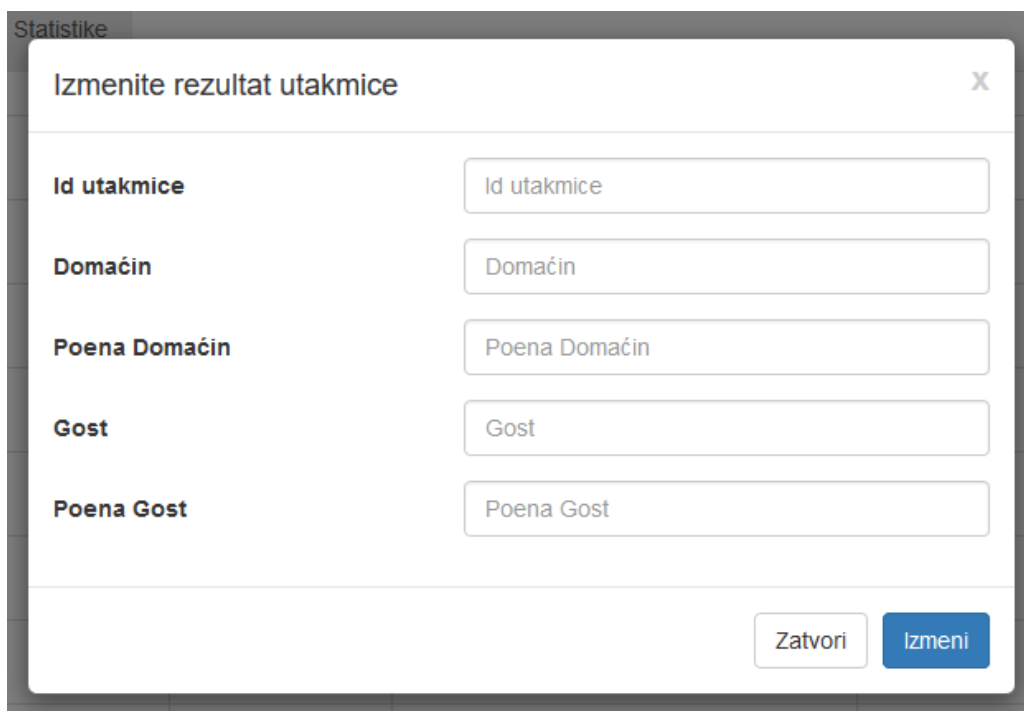


Slika 60. Tekst posle dodavanja utakmice

16	FC Barcelona Lassa	72	Fenerbahce Istanbul	74	Izmeni	Obriši
----	--------------------	----	---------------------	----	------------------------	------------------------

Slika 61. Tabela Utakmice, 16. pozicija posle unosa parametara

Posle svakog rezultata postoje dva dugmeta u produžetku “Izmeni” i “Obriši”. Ukoliko nismo zadovoljni sa unosom neke utakmice, imamo dve opcije da izmenimo unos ili da je potpuno izbrišemo. Ako želimo da je izmenimo, pritiskamo na dugme Izmeni. U tom trenutku nam izlazi prozor za izmenu parametara utakmica. Na slici 62. vidi se prozor za izmenu utakmice.



Slika 62. Prozor za menjanje utakmice

Unećemo iste parametre kao kod dodavanja utakmice, s'tim što ćemo za poene gosta umesto 74, staviti 73 (koliko je i bilo na toj utakmici) i pritisnuti Izmeni. Pojaviće nam se ista poruka kao i kod dodavanja utakmice, a parametri te utakmice će biti promenjeni (tačnije broj poena gosta). Na slici 63. imamo izgled tabele posle promene.

16	FC Barcelona Lassa	72	Fenerbahce Istanbul	73	<button>Izmeni</button>	<button>Obriši</button>
----	--------------------	----	---------------------	----	-------------------------	-------------------------

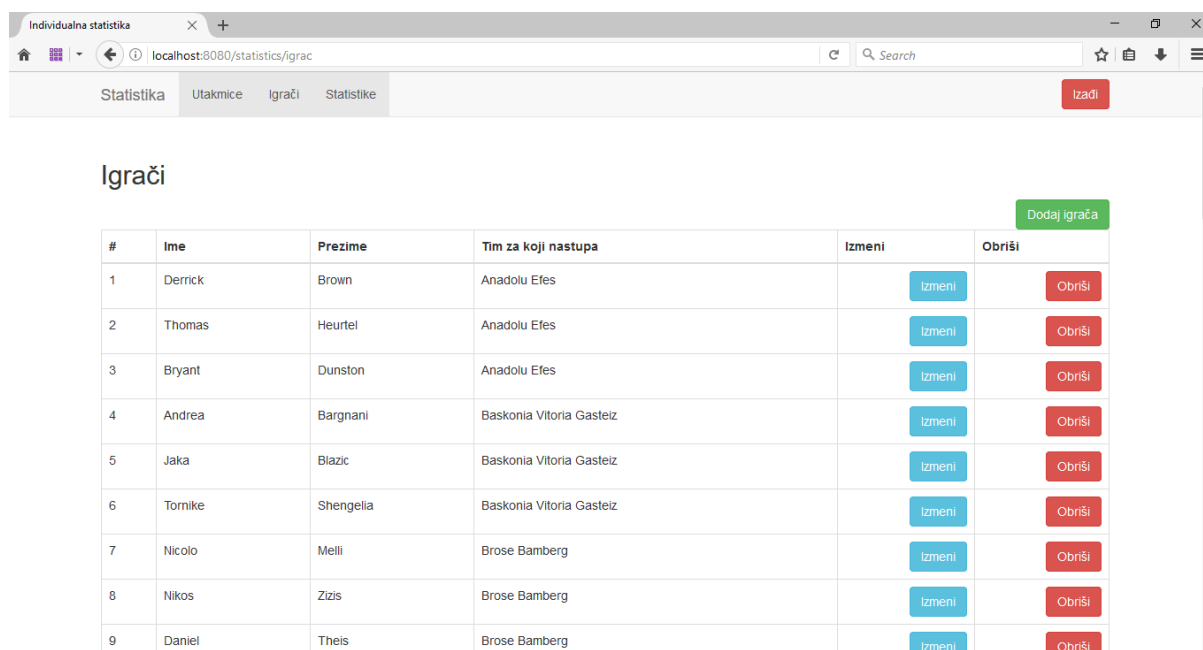
Slika 63. Izgled tabele Utakmice, pozicija 16, posle promene

Ukoliko nismo zadovoljni nekom utkmicom možemo je brisati, pritiskom na dugme Obriši. U tom slučaju iskaće nam prozor bez ikakvih parametara, samo sa dva dugmeta “Obriši” i “Zatvori”. Ukoliko pritisnemo Obriši iz sistema se briše data utakmica i ona nestaje iz tabele, a pojavljuje se poruka kao kod dodavanja i menjanja utakmica.

Pored funkcionalnosti vezane za CRUD objekta Utakmica na strani Utakmice, postoje i četiri dugmeta na navbar-u na vrhu. Skroz desno nalazi se crveno dugme, koje postoji na svim stranama, “Izadi”. Pritiskom na to dugme sistem nas bez ikakvih obaveštenja vraća na početnu stranu. Pored Izadi dugmeta, na administratorskom panelu imamo još tri dugmeta sive boje u levom uglu. To su “Utakmice”, “Igrači” i “Statistike”. Pritiskom na te dugmiće sistem će nas bez obaveštenja premeštati na te tri strane administratorskog panela. Na svakoj od administratorskih panela se na ist način dodaju, menjaju i brišu objekti. Kao što na strani Utakmice imamo utakmice nad kojima vršimo CRUD operacije, tako na strani Igrači imamo objekte tipa igrač koje možemo da dodajemo, menjamo i brišemo. Isto važi i za Statistike. Zato ćemo predstaviti te strane bez objašnjenja funkcionalnosti, jer su korišćeni isti obrasci kao i na strani koju smo opisali (Utakmice).

Igrači

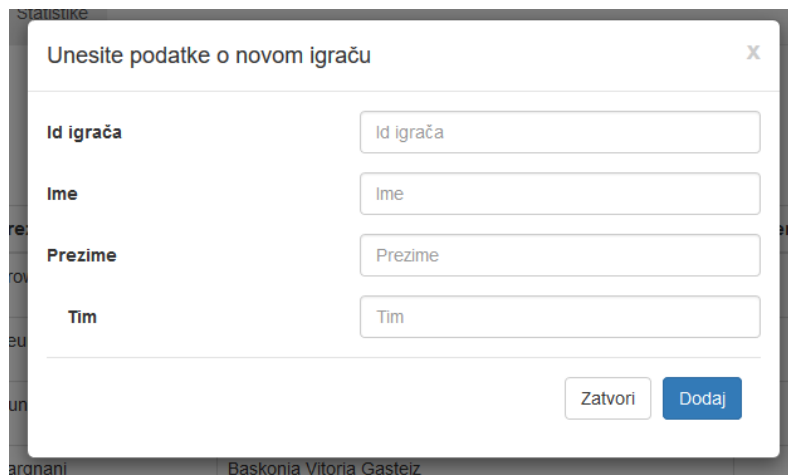
Pritiskom na dugme Igrači na navbar-u strane Utakmice, sistem nas prebacuje na stranu Igrači. To je strana posvećena igračima Evrolige. Na njoj se, pored već opisanog navbar-a, nalazi zeleno dugme za dodavanje “Dodaj Igrača”, tabela sa podacima igračima i dva dugmeta za svakog igrača “Izmeni” i “Obriši”. Na slici 64. se nalazi strana Igrači.



#	Ime	Prezime	Tim za koji nastupa	Izmeni	Obriši
1	Derrick	Brown	Anadolu Efes	Izmeni	Obriši
2	Thomas	Heurtel	Anadolu Efes	Izmeni	Obriši
3	Bryant	Dunston	Anadolu Efes	Izmeni	Obriši
4	Andrea	Bargnani	Baskonia Vitoria Gasteiz	Izmeni	Obriši
5	Jaka	Biazic	Baskonia Vitoria Gasteiz	Izmeni	Obriši
6	Tornike	Shengelia	Baskonia Vitoria Gasteiz	Izmeni	Obriši
7	Nicolo	Melli	Brose Bamberg	Izmeni	Obriši
8	Nikos	Zizis	Brose Bamberg	Izmeni	Obriši
9	Daniel	Theis	Brose Bamberg	Izmeni	Obriši

Slika 64.Strana Igrači

Svako od dugmića na ovoj strani radi po istom obrascu kao i dugmići na strani Utakmice, pa tako za Dodaj igrača i Izmeni iskaču prozori na kojima su parametri objekta Igrač. Unosimo paramtere i tako dodajemo ili menjamo objekat. Ukoliko ga dodamo igrač se prikazuje na dnu tabele sa najvećim id-jem. Ukoliko menjamo postojećeg igrača, on ostaje na svojoj poziciji samo mu se parametri menjaju. Na slici 65. nalazi se prozor za dodavanje igrača, a na slici 66. prozor za izmenu igrača. Pritiskom na dugme obriši pojavljuje nam se prozor bez parametara sa dva dugmeta kao kod Utakmice. Kod svake uspešne akcije pojavljuje se tekst uokviren zelenom, kao što smo opisali.



Unesite podatke o novom igraču

Id igrača

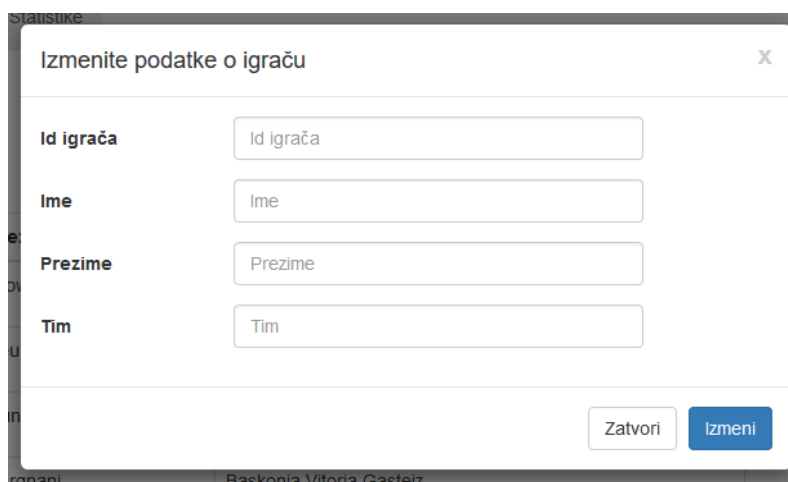
Ime

Prezime

Tim

Zatvori Dodaj

Slika 65. Prozor za dodavanje igrača



Izmenite podatke o igraču

Id igrača

Ime

Prezime

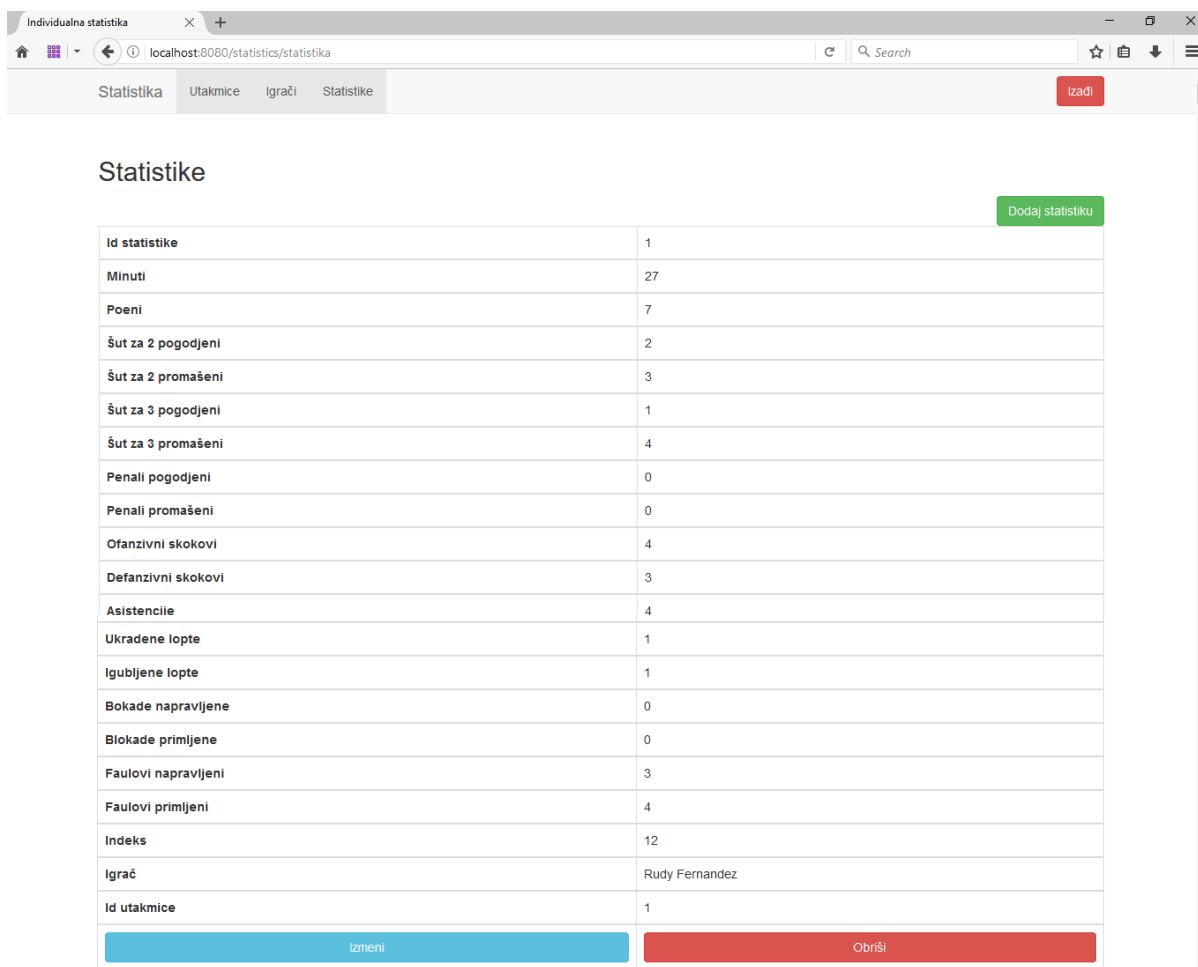
Tim

Zatvori Izmeni

Slika 66. Prozor za menjanje podataka o igraču

Statistike

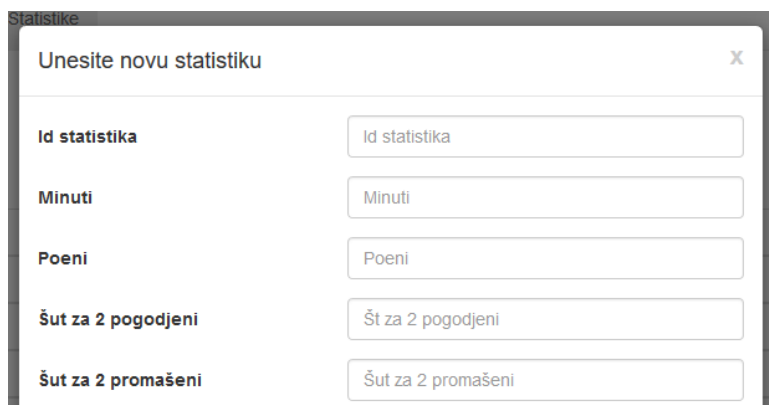
Pritiskom na dugme Statistike sistem nas prebacuje na stranu Statistike. Strana je pisana po istom obrascu kao i prethodne dve administrativne strane (Utakmice i Igrači). Imamo opisani navbar, zeleno dugme “Dodaj statistiku” i tabelu sa statistikama, gde svaka pojedinačna ima svoje dugme Izmeni i Obriši. Prozori za dodavanje, izmenu i brisanje su potpuno isti kao u prethodna dva slučaja, samo imaju više parametara, jer je statistika objekat od dvadeset i jednog parametra. Stranu Statistike smo razvukli zbog preglednosti parametara, pa ćemo je predstaviti iz dve slike. Na slici 67. nalazi se strana Statistika. Prva uneta statistika je za igrača Rudy Fernandez, koji nastupa za Real Madrid. Statistika ima idUtakmice kao parametar preko koga se vezuje za određenu utakmicu (u ovom slučaju prvu unetu – Real Madrid 83 : Olympiacos Pireus 65).



Id statistike	1
Minuti	27
Poeni	7
Šut za 2 pogodjeni	2
Šut za 2 promašeni	3
Šut za 3 pogodjeni	1
Šut za 3 promašeni	4
Penali pogodjeni	0
Penali promašeni	0
Ofanzivni skokovi	4
Defanzivni skokovi	3
Asistencije	4
Ukradene lopte	1
Igubljene lopte	1
Bokade napravljene	0
Blokade primljene	0
Faulovi napravljeni	3
Faulovi primljeni	4
Indeks	12
Igrač	Rudy Fernandez
Id utakmice	1

Slika 67. Strana Statistika

Na slici 68. Prikazan je početni deo prozora za dodavanje statistika, isti je obrazac kao i za dodavanje ostalih objekata. Time smo predstavili administrativni panel. Administrativac kada završi, pritiskom na izađi napušta sitem.



Unesite novu statistiku

Id statistika

Minuti

Poeni

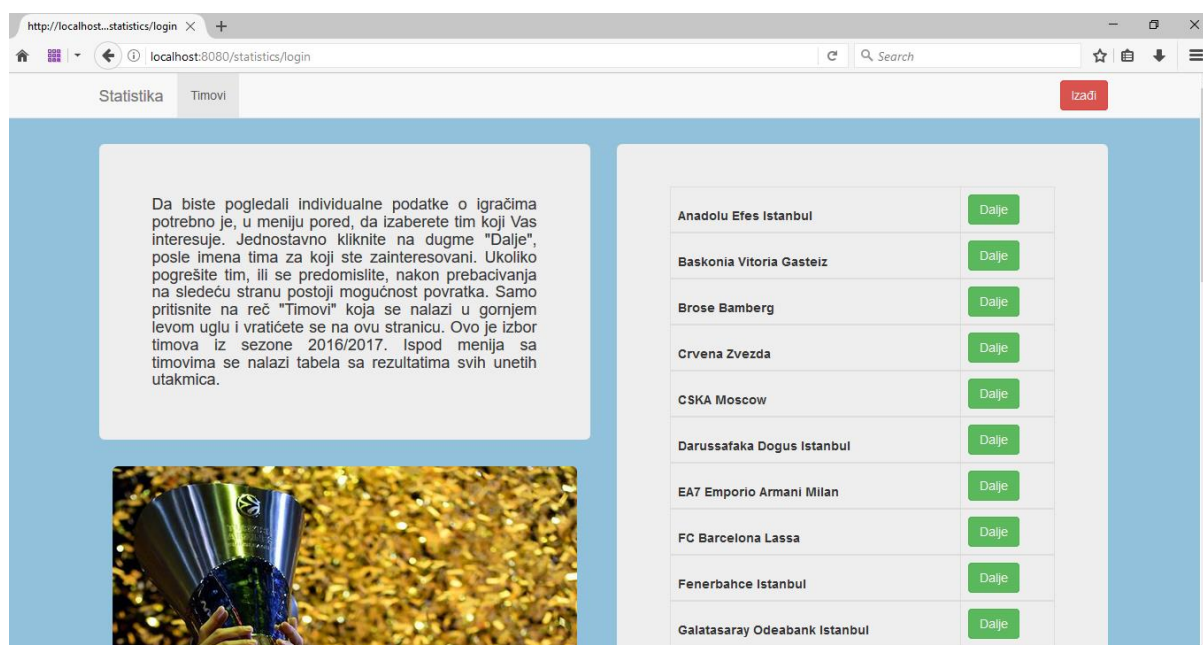
Šut za 2 pogodjeni

Šut za 2 promašeni

Slika 68. Početni deo prozora Dodaj Statistiku

Korisnički panel - Timovi

Na korisnički panel se prijavljujemo sa nalogom “djole” i šifrom “djole”. Kada se uojujemo otvara nam se stranica Timovi. Ta stranica posvećena je korisniku. Na vrhu stranice imamo navbar, kao i na stranama za administratora. Pored dugmeta izadi postoji i dugme Timovi kojim se osveževa stranica na kojoj se nalazimo. Ispod navbara sa leve strane se nalazi tekst sa uputstvom za korišćenje aplikacija (za korisnika), podatak o sezoni za koju je aplikacija pravljena i podatak o tabeli sa rezultatima na dnu strane na kojoj se nalazimo. Desno od teksta se nalazi tabela sa šesnaest timova učesnika takmičenja i iza svakog se nalazi zeleno dugme “Dalje”. Pitiskom na dugme prebacujemo se na stranu tima koji smo izabrali (ime tima je stajalo pre dugmeta). Ispod teksta stoji slika trofeja Evrolige. Ispod svega nalazi se tabela sa rezultatima koju korisnik može samo da pregleda. Na slici 69. nalazi se strana Timovi.



Slika 69. Strana Timovi

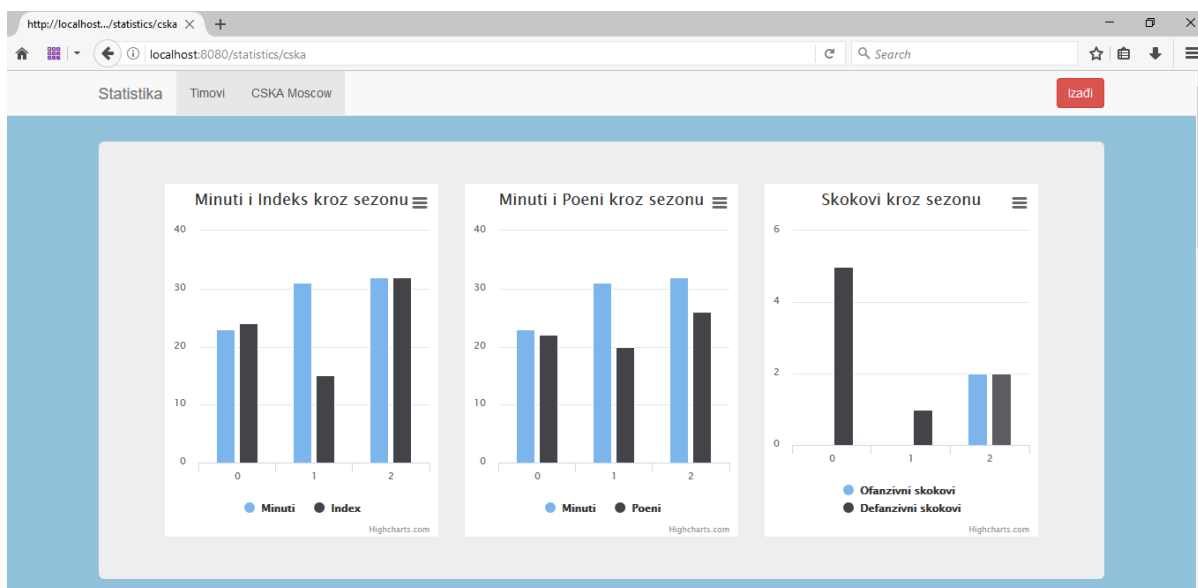
Skrolanjem na dole, silazi se na donji nivo strane na kome se nalazi tabela sa unetim rezultatima. Primećujemo razliku kod tabele Rezultati sa strane Timovi i tabele Utakimce sa strane Utakmice. I jedna i druga tabela imaju iste podatke, samo drugačiji naslov. Međutim, na administrativnom panelu tabela Utakmice ima funkcionalnosti CRUD nad podacima i administrator koji pristupa tabeli može da utiče na nju. Tabela Rezultati na korisničkoj strani, može samo da se pročita, tako da korisnik može samo da pogleda podatke. Na slici 70. nalazi se izgled strane Timovi kada se spusti strana do tebele Rezultati.

#	Domaćin	Broj poena	Gost	Broj poena
1	Real Madrid	83	Olympiacos Piraeus	65
2	Galatasaray Odeabank Istanbul	84	CSKA Moscow	109
3	EA7 Emporio Armani Milan	99	Macabi FOX Tel Aviv	97
4	Crvena Zvezda	70	Darussafaka Dogus Istanbul	73
5	Unics Kazan	63	FC Barcelona Lassa	69
6	Fenerbahce Istanbul	67	Brose Bamberg	66
7	Panathinaikos Athens	84	Zalgiris Kaunas	76
8	Baskonia Vitoria Gasteiz	85	Anadolu Efes	84
9	Olympiacos Piraeus	90	Anadolu Efes	66
10	Brose Bamberg	89	Unics Kazan	86
11	Macabi FOX Tel Aviv	82	Real Madrid	89
12	Darussafaka Dogus Istanbul	80	EA7 Emporio Armani Milan	81
13	CSKA Moscow	81	Panathinaikos Athens	77
14	Galatasaray Odeabank Istanbul	83	Crvena Zvezda	85

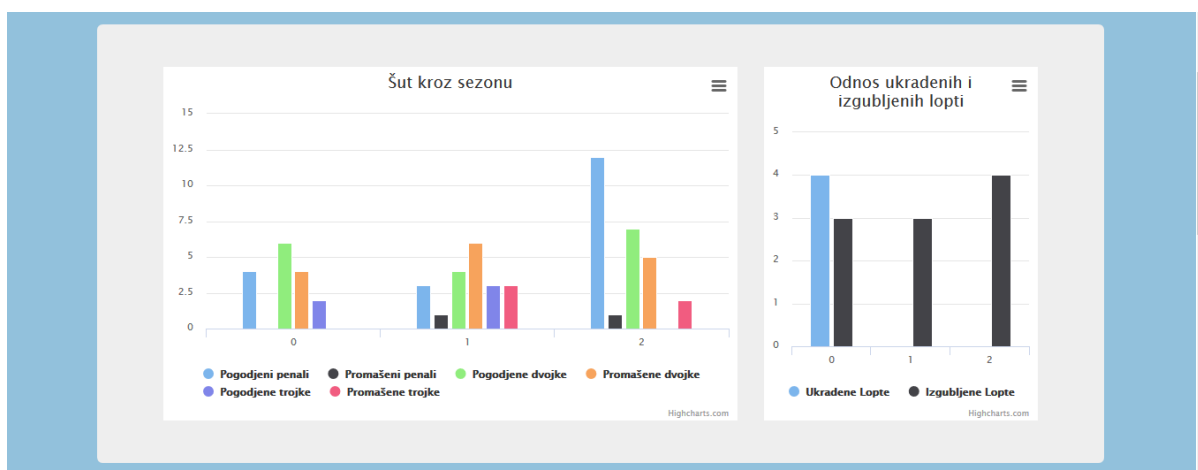
Slika 70. Izgled strane Timovi kada se spustimo do tabele Rezultati

Izabrani tim

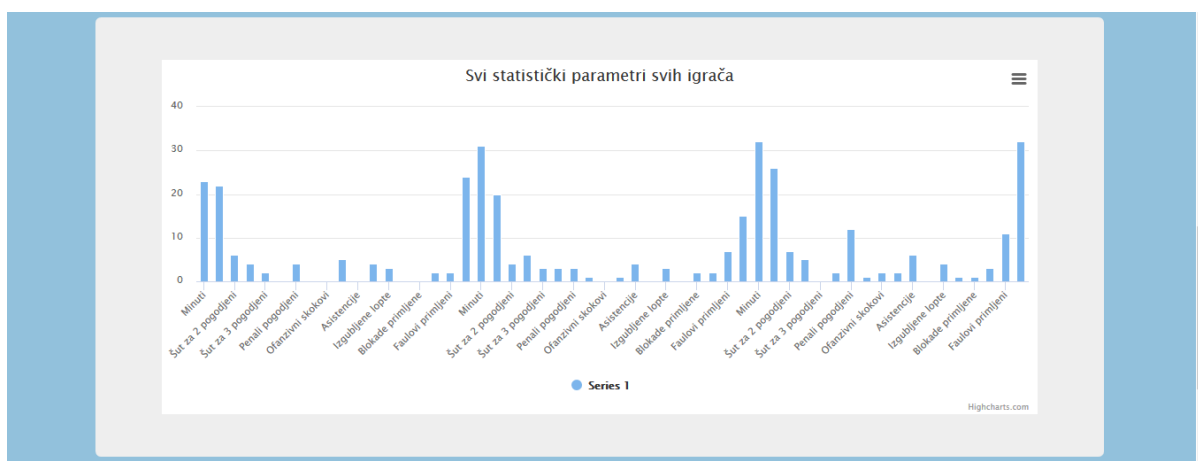
Izborom tima, sa strane Timovi, sistem nas šalje na stranu tima koji nas interesuje. Svi timovi su pisani po istom obrascu. Tako da se razlikuju samo zbog unetih podataka. Inače svi imaju istu strukturu, svaka strana ima navbar, šest grafikona i tabelu. Navbar se nalazi na vrhu strane i ima tri dugmeta. Na desnoj strani se nalazi crveno dugme Izadi, za izlazak sa sistema. Na levoj strani imamo dva siva dugmeta. Dugme Timovi, sa kojim se vraćamo na stranu Timovi i dugme sa imenom Tima sa kojim nam se osvežava strana. Na strani se nalazi šest grafikona gde se porede odeđene statističke kategorije i na dnu se nalazi tabela sa svim unetim podacima. Prikazaćemo izgled dva različita tima. Na svakoj od stranica smo različito pristupali podacima. Tako smo na stranici posvećenoj CSKA Moscow uporedili statistiku igrača Nando De Colo na tri utakmice. Na sledećoj strani posvećenoj Olympicos Piraeus smo uporedili tri igrača na jednoj utakmici. Parsiranje podataka zavisi od administratora, pošto on unosi ili briše podatke. Tako se može ispratiti samo jedan igrač ili više njih, na jednoj ili više utakmica. Na slici 71. nalazi se strana CSKA Moscow (prva tri grafikona). Na slici 72. nalazi se strana CSKA Moscow (četvrti i peti grafikon). Na slici 73. nalazi se strana CSKA Moscow (šesti grafikon). Na slici 74. nalazi se strana CSKA Moscow (tabela). Na slici 75. nalazi se strana Olympiacos Piraeus (prva tri grafikona). Na slici 76. nalazi se strana Olympiacos Piraeus (četvrti i peti grafikon). Na slici 77. nalazi se strana Olympiacos Piraeus (šesti grafikon). Na slici 78. nalazi se strana Olympiacos Piraeus (tabela).



Slika 71. Strana CSKA Moscow - prva tri grafikona



Slika 72. Strana CSKA Moscow - četvrti i peti grafikon

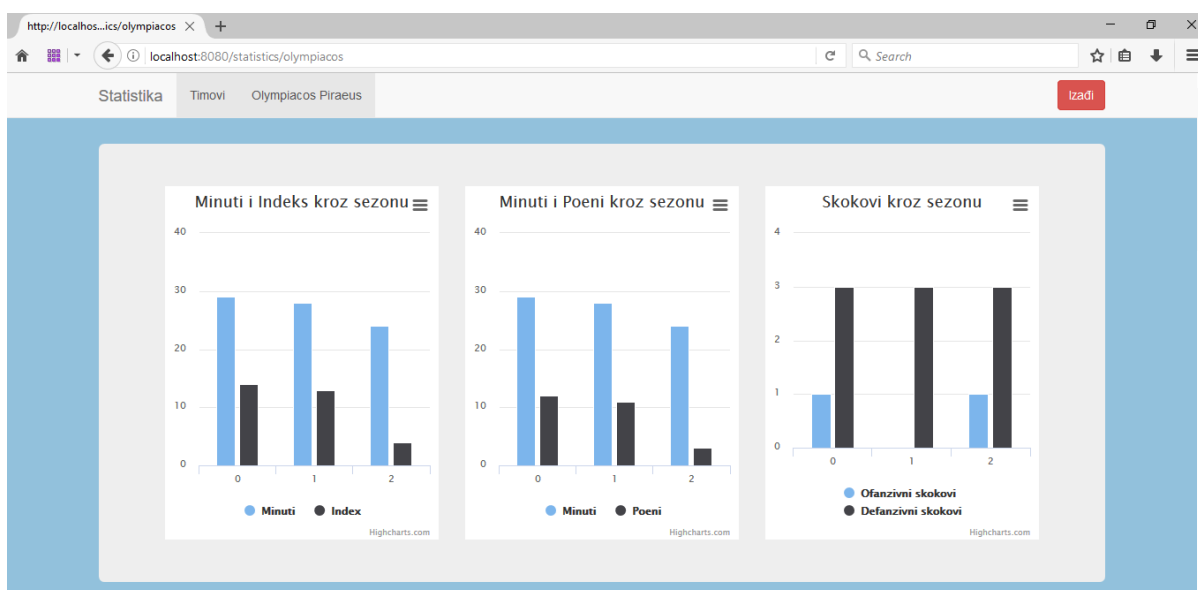


Slika 73. Strana CSKA Moscow - šesti grafikon

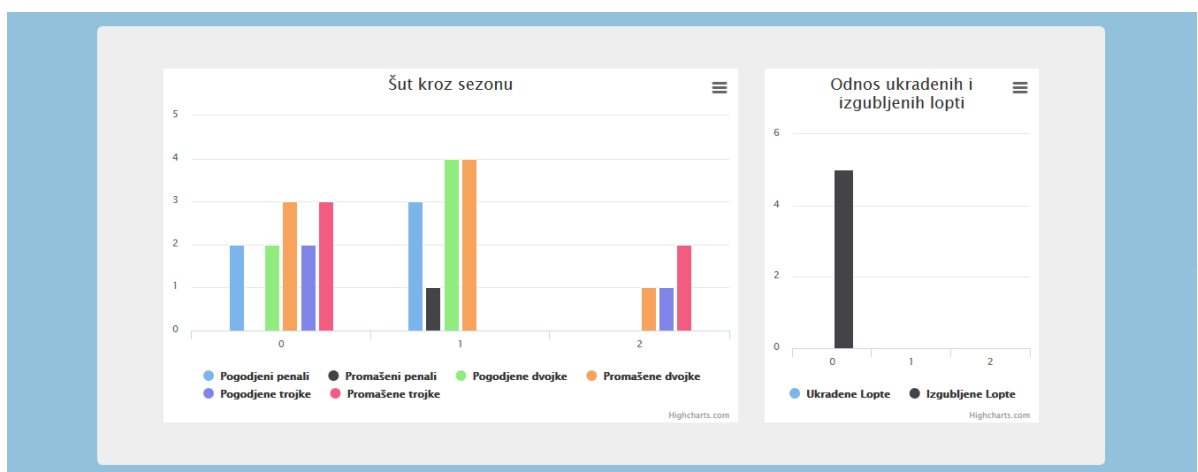
Tabela se statističkim parametrima

Igrač		min	pts	2/po	2/pr	3/po	3/pr	1/po	1/pr	Of sk	Def sk	As	Uk lo	Izg lo	Bl napr	Bl prim	Faul nap	Faul prim	Ind
Nando Colo	De	23	22	6	4	2	0	4	0	0	5	0	4	3	0	0	2	2	24
Nando Colo	De	31	20	4	6	3	3	3	1	0	1	4	0	3	0	2	2	7	15
Nando Colo	De	32	26	7	5	0	2	12	1	2	2	6	0	4	1	1	3	11	32

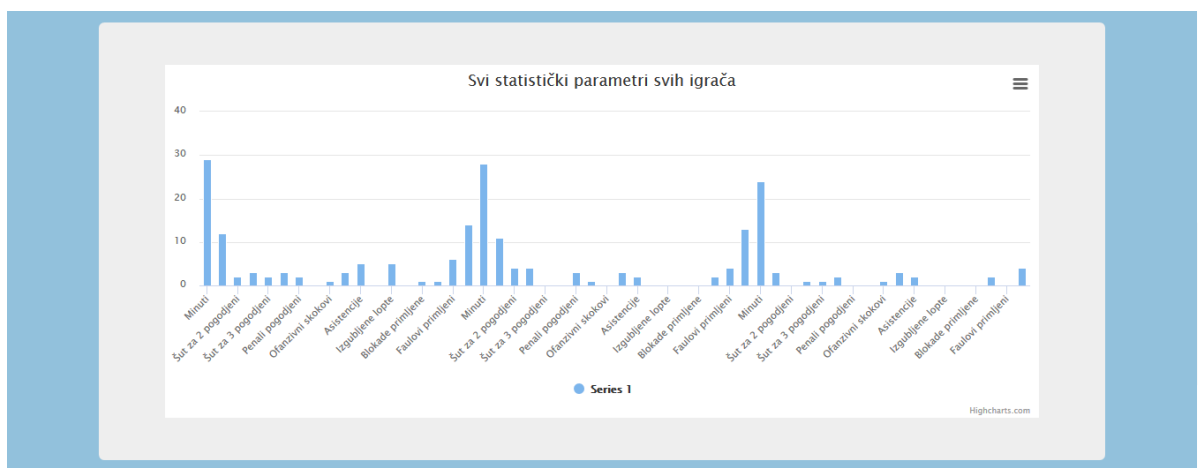
Slika 74. Strana CSKA Moscow – tabela



Slika 75. Strana Olympiacos Piraeus - prva tri grafikona



Slika 76. Strana Olympiacos Piraeus – četvrti i peti grafikon



Slika 77. Strana Olympiacos Piraeus – šesti grafikon

Tabela se statističkim parametrima

Igrač	min	pts	2/po	2/pr	3/po	3/pr	1/po	1/pr	Of sk	Def sk	As	Uk lo	Izg lo	BI napr	BI prim	Faul nap	Faul prim	Ind
Vassilis Spanoulis	29	12	2	3	2	3	2	0	1	3	5	0	5	0	1	1	6	14
Georgios Printezis	28	11	4	4	0	0	3	1	0	3	2	0	0	0	0	2	4	13
Vangelis Mantzaris	24	3	0	1	1	2	0	0	1	3	2	0	0	0	0	2	0	4

Slika 78. Strana Olympiacos Piraeus – tabela

Sve ostale strane pojedinačnih timova su zavisne od podataka iz baze i mogu se menjati i oblikovati kako administrator bude želeo. Suština je da u svakom trenutku mogu da se predstavljaju podaci u onom obliku u kom je to potrebno. Svi grafikoni su ovog oblika, a na dnu svake strane se nalaze tabele ovakvih oblika. Sa ovim smo opisali potpunu funkcionalnost aplikacije.

9. Zaključak

Projekti koji se kreiraju uz pomoć naprednih Java tehnologija, uglavnom, okupljaju timove programera različitih sposobnosti. Potrebno je konstruisati bazu, koristiti Java-u za kreiranje biznis logike (*backend-a*), formirati veb strane preko HTML5, CSS3, Javascript-a i Bootstrap-a. Toliko različitih tehnologija kad se isprepliće može lako da dovede do raznih vrsta grešaka i komplikacija tokom rada. Čak i najbolji *full stack developer-i* (programeri koji poznaju tehnologije za rad i na *backend-u* i na *frontend-u*), retko se usuđuju da sami kreiraju i održavaju aplikacije srednjeg ili većeg obima. Upravo sve pomenute tehnologije su korišćene za izradu opisane aplikacije. Veličinom i funkcionalnošću, implementirana veb aplikacija pripada redu srednjih. Puštanje u produkciju (rad) i održavanje ovakve aplikacije, obuhvatalo bi rad nekoliko zaposlenih. Programeri bi bili zaduženi za rad sa bazom i biznis logikom. Administratori servera bi bili zaduženi za održavanje konekcija i praćenje log datoteka. Administrator podataka bi se bavio unosom i održavanjem podataka.

Razvojna okruženja u kojima je kreirana aplikacija se koriste i u obrazovne svrhe, ali i u profesionalnom svetu. Administratorskim panelom smo obezbedili CRUD aplikaciju, pošto su funkcionalnosti administratorskog panela upravo funkcionalnosti CRUD-a. Administrator može da pristupi svim objektima koji su kreirani, da ih menja ili briše, kao i da dodaje nove. Korisnički panel pruža mogućnosti izbora tima, a potom i pristup isparsiranim podacima koje može da pregleda. Sa sportskog aspekta, mnogo je interesantniji korisnički nalog, pošto pruža uvid u sportske učinke košarkaša. Tako skaut (korisnik) može dobiti uvid u individualne učinke igrača i pomoću njih napraviti pripremu utakmice, jer podaci predstavljeni na način kao u projektu “govore” dosta o igraču. Dakle, pored spajanja naprednih tehnologija u jednu projektnu celinu, rad se bavi i sportskim aspektom.

Aplikacija je potpuno funkcionalna, i sve opisane funkcionalnosti su implementirane i lako proširive. Refaktorisanjem postojećeg koda, servis koji pruža aplikacija bi se unapredio i poboljšao. Dalji razvoj aplikacije u potencijalnim različitim pravcima, pre svega zavisi od korisnika. Dodavanjem novih grafikona i dijagrama, poboljšali bi analitiku individualnih statistika. Odvajanjem grafikona pojedinačnih statistika, rešili bi bolji uvid od utakmice do utakmice. Dodavanjem opcije za registrovanje, omogućili bi više naloga od dva pomenuta, s’tim što bi omogućili i komunikaciju neregistrovanog korisnika preko polja za poruke ili polja za mail adrese. Što se tiče unapređenja biznis logike, redefinisanjem jsp strana, dobili bi generičku stranu za izabrani tim, umesto svake pojedinačne, što bi ubrzalo sistem i smanjilo količinu koda i fajlova. Izmenom jsp strana, potrebno bi bilo promeniti i servlete da odgovaraju jedni drugima.

Ovaj rad, pored toga što je autoru omogućio savladavanje naprednih veb tehnologija kroz praktičan rad, mogao bi biti odlična alatka za ljude posvećene trenerskom i skauting poslu u košarci. Uz određenu analitiku video zapisa sa utakmica i druge gore opisane dorade koje se planiraju u budućnosti, ova veb aplikacija će postati veoma moćno pomoćno sredstvo za njih.

Spisak referenci

1. SQL (Structured Query Language), <http://searchsqlserver.techtarget.com/definition/SQL> (dostupno 21.03.2017.)
2. Hibernate Community Documentation, <https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/tutorial.html#tutorial-firstapp-setup> (dostupno 21.03.2017.)
3. Coderanch, <https://coderanch.com/t/218866/databases/Persistence-Hibernate> (dostupno 21.03.2017.)
4. Oracle, <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html> (dostupno 21.03.2017.)
5. Vlad Mihalcea's Blog, <https://vladmihalcea.com/2014/07/30/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/> (dostupno 21.03.2017.)
6. It-konket, <http://it-konekt.com/sr/comment/reply/203> (dostupno 21.03.2017.)
7. Bootstrap, <http://getbootstrap.com> (dostupno 21.03.2017.)
8. Bootstrap CSS, <http://getbootstrap.com/css/> (dostupno 21.03.2017.)
9. Bootstrap Components, <http://getbootstrap.com/components/> (dostupno 21.03.2017.)
10. Servlets tutorial, <https://www.tutorialspoint.com/servlets/> (dostupno 21.03.2017.)
11. W3school, HTTP Methods: GET vs. POST, https://www.w3schools.com/tags/ref_httpmethods.asp (dostupno 21.03.2017.)
12. Dzone/Java zone, <https://dzone.com/articles/what-servlet-container> (dostupno 21.03.2017.)
13. Gary Cornell, Cay S. Horstmann - Core JavaTM (Volume I, Fundamentals, Ninth Edition), Michigan, 2012. (dostupno 21.03.2017.)
14. JavaTpoint, ServletConfig Interface, <http://www.javatpoint.com/servletconfig> (dostupno 21.03.2017.)
15. JavaTpoint, ServletContext Interface, <http://www.javatpoint.com/servletcontext> (dostupno 21.03.2017.)
16. Oracle, Interface RequestDispatcher, <http://docs.oracle.com/javaee/6/api/javax/servlet/RequestDispatcher.html> (dostupno 21.03.2017.)
17. Oracle, JSP, <http://www.oracle.com/technetwork/articles/java/servlets-jsp-140445.html> (dostupno 21.03.2017.)
18. JavaTpoint, jsp:include action tag, <http://www.javatpoint.com/jsp-include-action> (dostupno 21.03.2017.)
19. The Java EE Tutorial, <http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html> (dostupno 21.03.2017.)
20. Tutolias Point, JSP – Standard Tag Library (JSTL) Tutorial, https://www.tutorialspoint.com/jsp/jsp_standard_tag_library.htm (dostupno 21.03.2017.)