SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5996

IZRADA WEB APLIKACIJE ZA PRAĆENJE REZULTATA VESLAČKIH TRENINGA

Mihovil Ćaćić

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 11. ožujka 2019.

ZAVRŠNI ZADATAK br. 5996

Pristupnik: **Mihovil Čačić (0036485899)**
Studij: Računarstvo
Modul: Programsko inženjerstvo i informacijski sustavi

Zadatak: **Izrada web aplikacije za praćenje rezultata veslačkih treninga**

Opis zadatka:

Veslačka sekcija treba imati razvijenu programsku potporu koja će im olakšat praćenje rezultata treninga. Svaki trening sastoji se od više vježbi s pripadnim rezultatom. Korisnik treba moći pratiti informacije o vlastitim rezultatima putem svojeg profila (npr. prosječno vrijeme, najbolje vrijeme, prikaz rezultata u vremenu). Potrebno je omogućiti stvaranja grupa korisnika. Članovi grupe imaju mogućnost međusobne komunikacije porukama, prikaza poretka članova po vježbama, prikaza grupnih rezultata za odabrane članove i slično. Zadatak ovog završnog rada je ostvariti kvalitetnu implementaciju traženih funkcionalnosti u vidu web aplikacije. U radu je potrebno proučiti i prikazati mogućnost radnog okvira Spring za razvoj aplikacije na poslužitelju te radnog okvira React za razvoj korisničkog prikaza na klijentskoj strani, što uključuje prilagođenost za mobilne uređaje. Web aplikaciju potrebno je ispitat i sa strane funkcionalnosti i sa strane performansi izvođenja.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 14. lipnja 2019.

Mentor:
Doc. dr. sc. Alan Jović

Predsjednik odbora za završni rad modula:
Izv. prof. dr. sc. Ivica Botički

Djelovođa:
Doc. dr. sc. Mirjana Domazet-Lošo

SADRŽAJ

1. UVOD

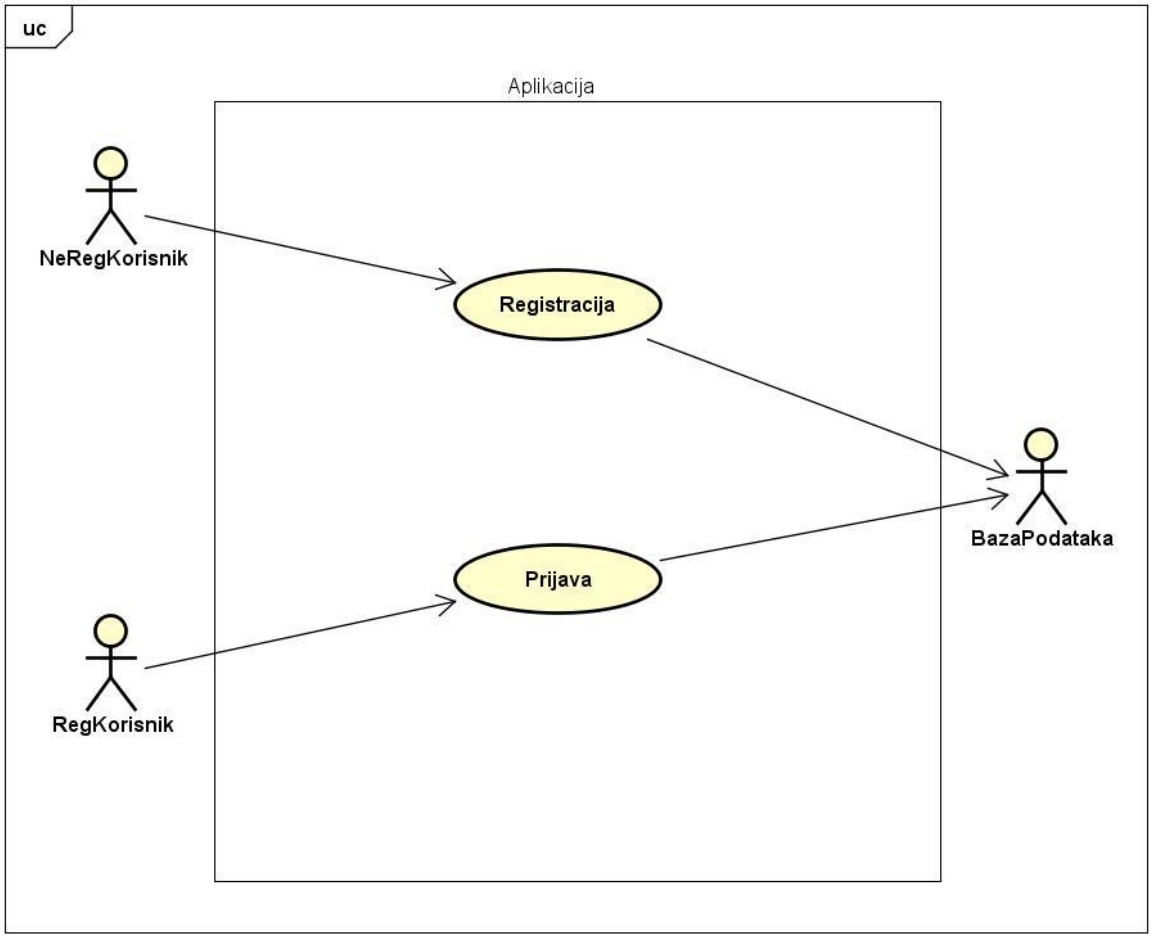
Prilikom sudjelovanja u radu Veslačke sekcije Fakulteta elektrotehnike i računarstva javila se potreba za praćenjem, kako osobnog, tako i napretka ostalih članova sekcije, te usporedbu rezultata članova. Rješenje smo pronašli u vidu web aplikacije koja bi omogućila spremanje rezultata veslačkih treninga, te pružila uvid u iste. Time bi korisnici mogli pratiti pomake u vlastitim rezultatima treninga.

U drugome poglavlju opisani su arhitektura i način oblikovanja sustava, funkcionalni zahtjevi, i način komunikacije između pojedinih dijelova sustava. U trećem poglavlju su prikazane korištene tehnologije i alati, te mogućnosti i prednosti njihova korištenja. U četvrtom poglavlju prikazana je implementacija na poslužiteljskoj i klijentskoj strani. U petom poglavlju je iznesen zaključak, te plan nastavka rada na sustavu. Konačno, u šestome se nalazi popis literature.

2. ARHITEKTURA I OBLIKOVANJE SUSTAVA

2.1. Funkcionalni zahtjevi sustava

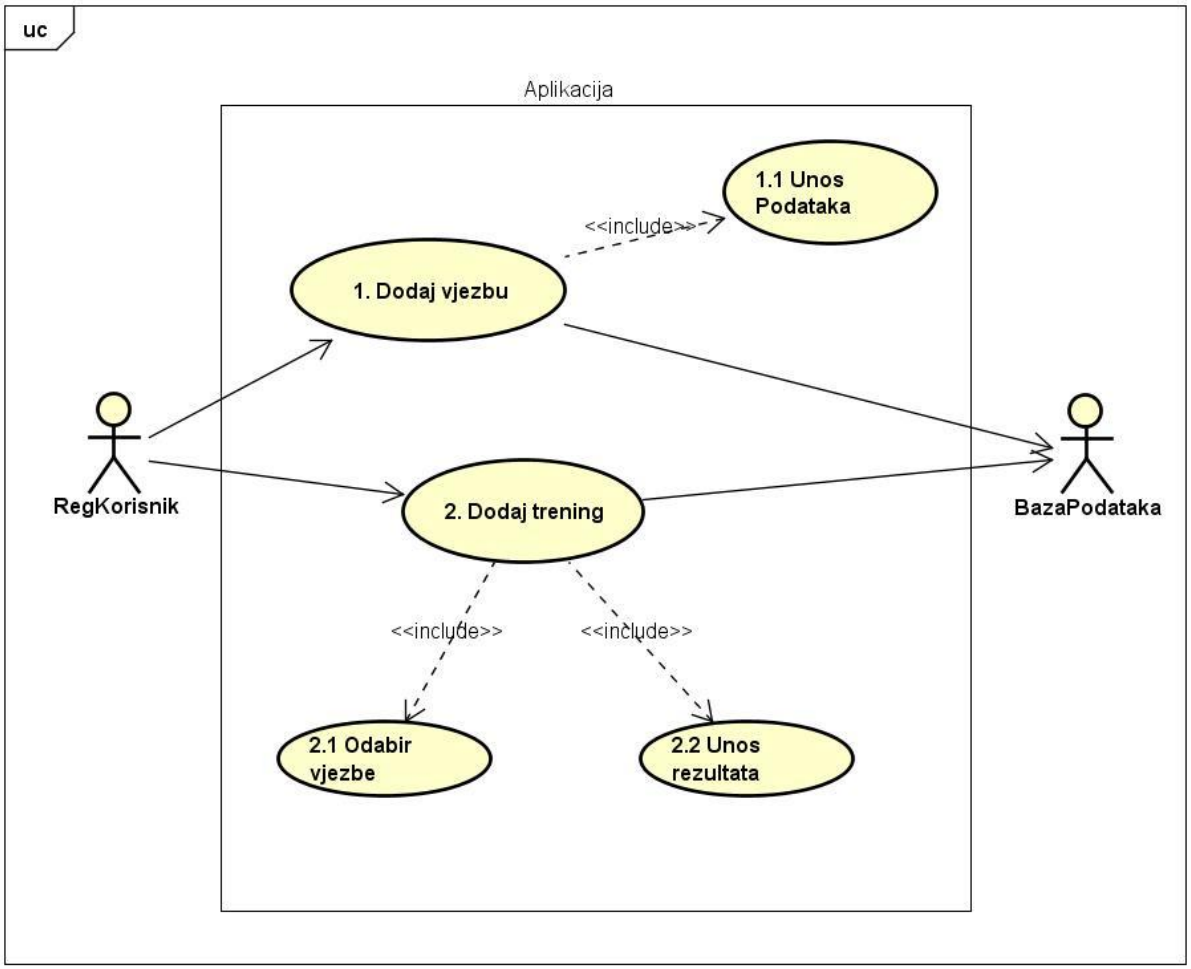
Aktori ovoga sustava su **NeRegKorisnik** i **RegKorisnik** (u nastavku teksta: Korisnik), čiji su obrasci uporabe sustava prikazani na slici 1. **NeRegKorisnik** ima mogućnost registracije, gdje unosi tražene podatke, ime, prezime, email, lozinka, te prosječno vrijeme na 500 m tijekom testne vježbe (1000 m ili 2000 m) koje predstavlja referentnu vrijednost na temelju koje će se uspoređivati ostali treninzi. Nakon uspješne registracije korisnik ima mogućnost prijave u sustav. Tek unošenjem ispravnih podataka prilikom prijave, korisnik dobiva mogućnost korištenja svih daljnje navedenih funkcionalnosti.



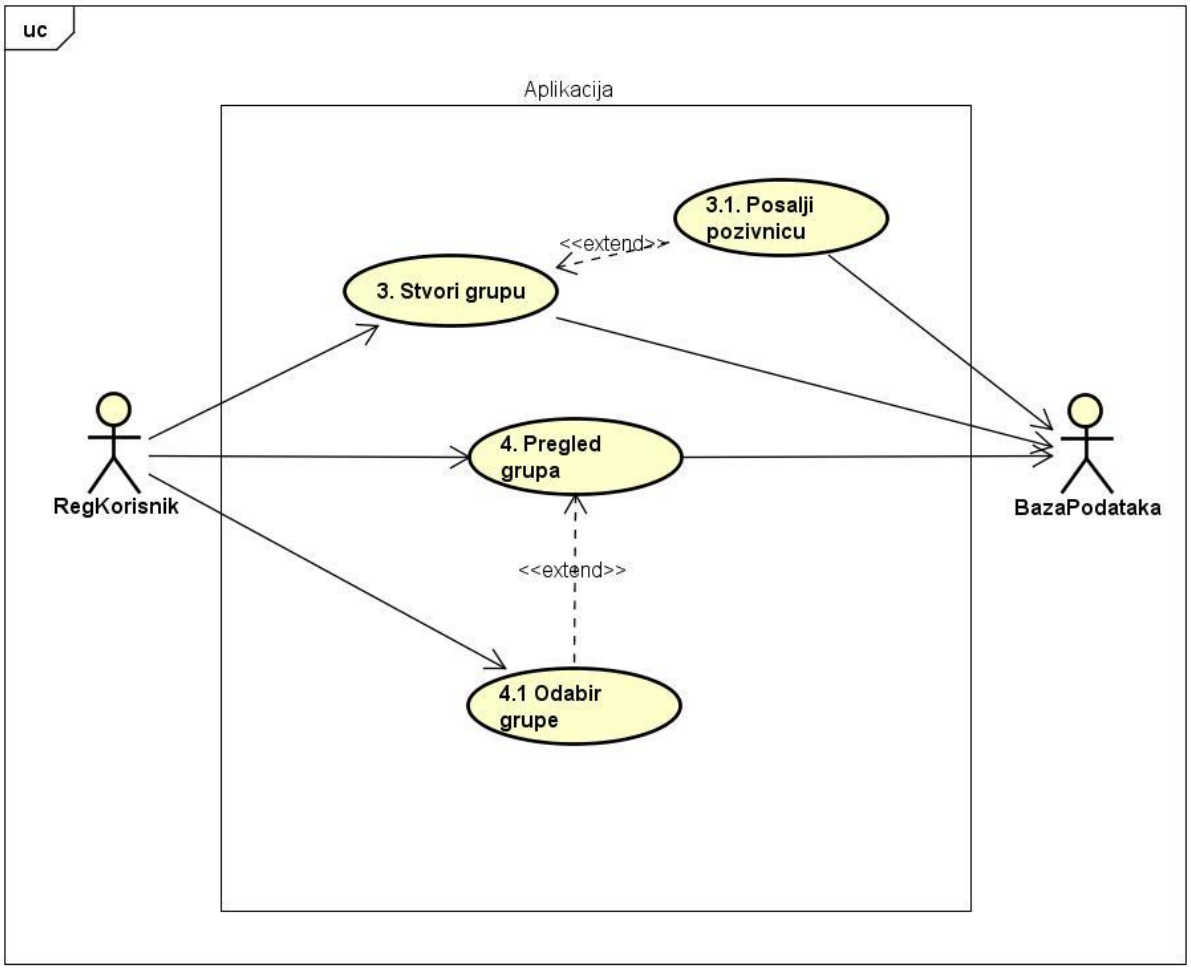
Slika 1. Obrazac uporabe - prijava i registracija

Ako među ponuđenim vježbama ne postoji tražena vježba, Korisnik ima mogućnost dodati vježbu, koja će nakon uspješnog dodavanja biti vidljiva svim ostalim korisnicima. Prilikom dodavanja pojedine vježbe, potrebno je unijeti obavezne vrijednosti, naziv, distance (predstavljaju dužinu ili vrijeme trajanja, jedne relacije), očekivano vrijeme za svaku distancu, te po potrebi napomenu.

Prilikom dodavanja novog treninga, kao što je vidljivo na slici 2, Korisnik mora odabrati jednu od ponuđenih vježbi, datum i rezultat za svaku distancu odabrane vježbe. Nakon ispravno unesenih rezultata zapis se sprema u bazu podataka.



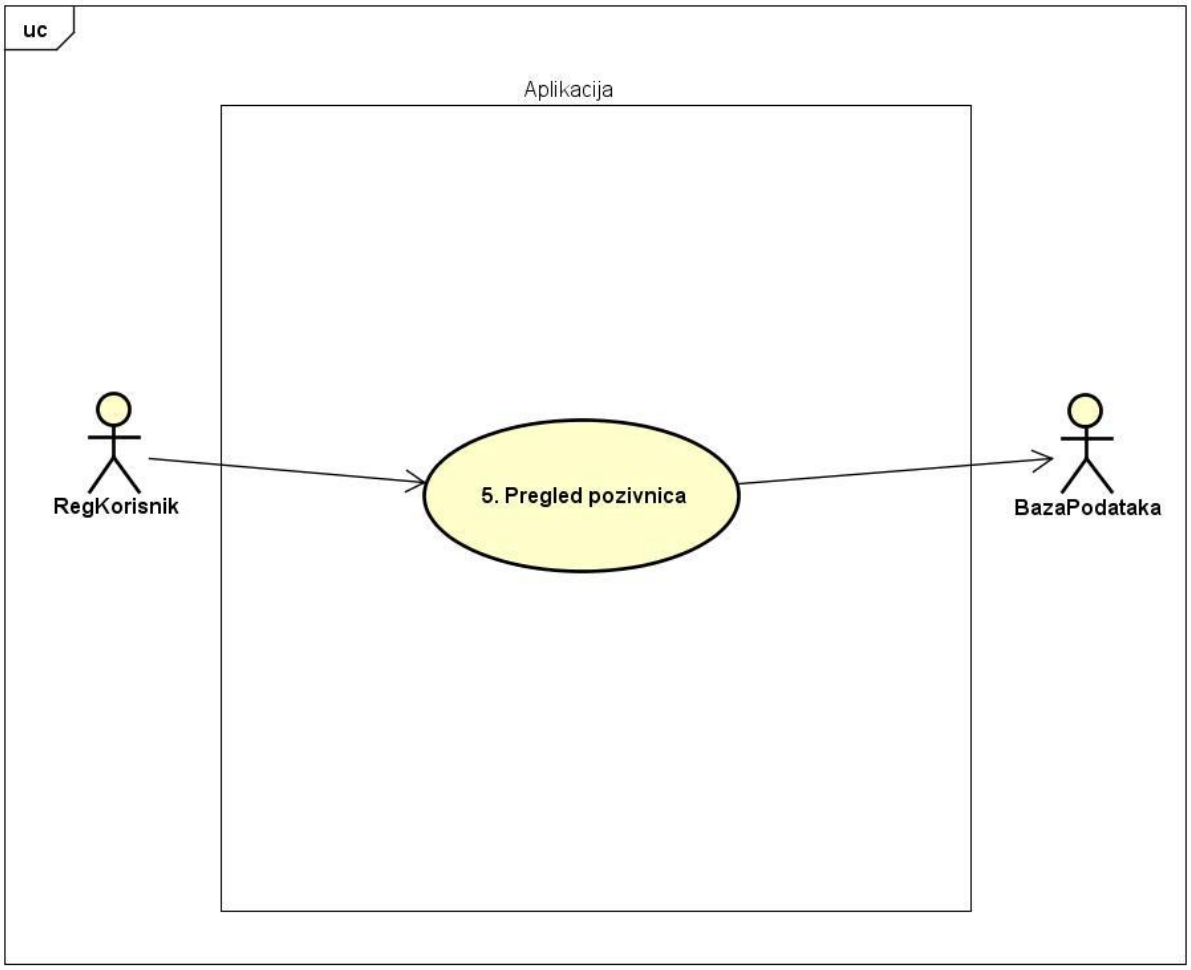
Slika 2. Obrazac uporabe – trening i vježba



Slika 3. Obraza uporabe - grupe

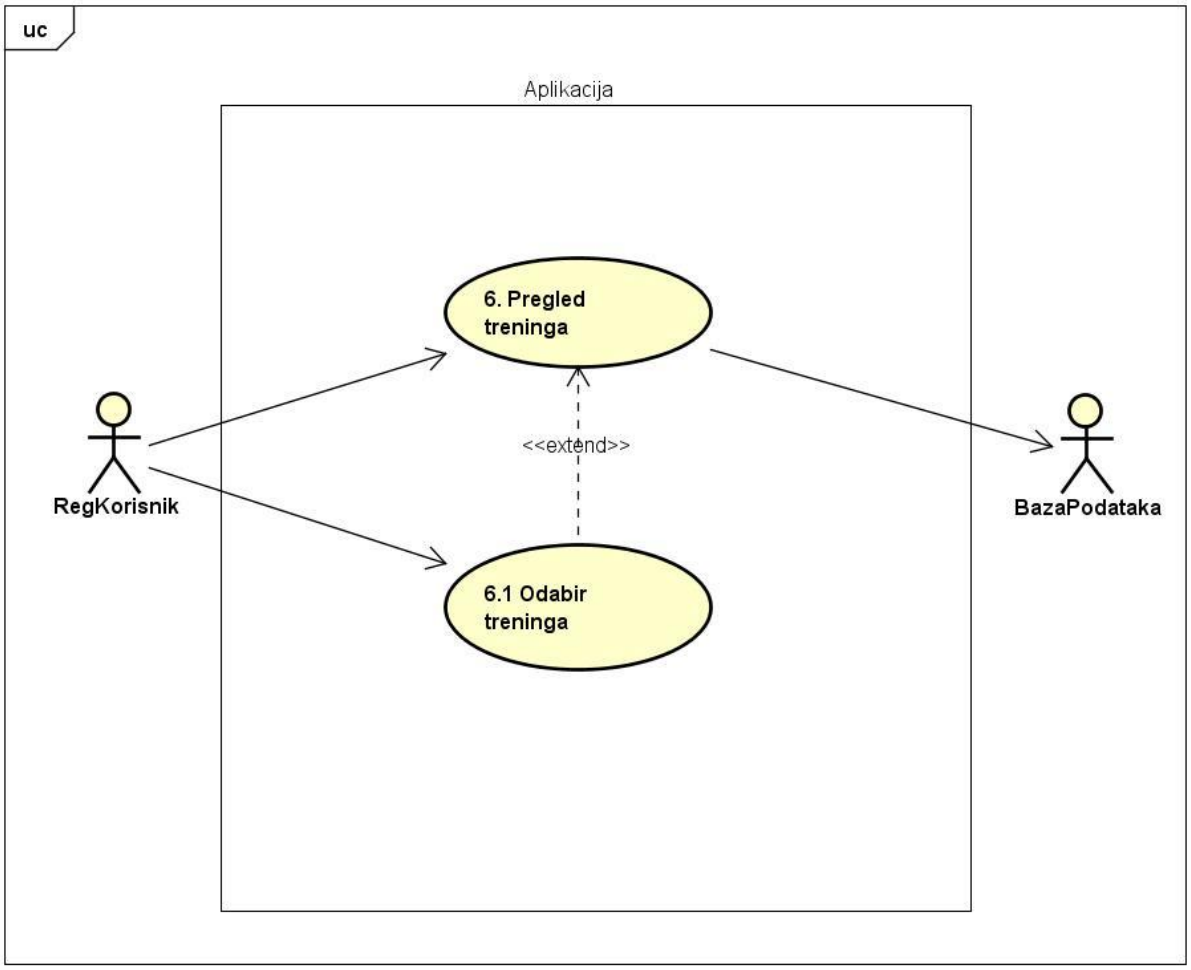
Na slici 3 prikazane su funkcionalnosti vezane za rad s grupama. Svaki Korisnik ima mogućnost stvoriti grupu, te pozvati druge Korisnike. Prilikom pregleda grupa, Korisnik dobije na uvid sve grupe u kojima se i sam nalazi. Nakon odabira jedne od ponuđenih grupa, prikaže se poredak članova, te popis svih članova te grupe.

Da bi Korisnik postao član neke grupe, vlasnik grupe mora pozvati Korisnika da se pridruži grupi. Korisniku se zatim pozivnica prikaže prilikom pregleda svih pozivnica, koja je prikazana na slici 4. Za svaku od prikazanih pozivnica Korisnik ima mogućnost odbiti ili prihvatiti članstvo za pojedinu grupu.



Slika 4. Obrazac uporabe - pozivnice

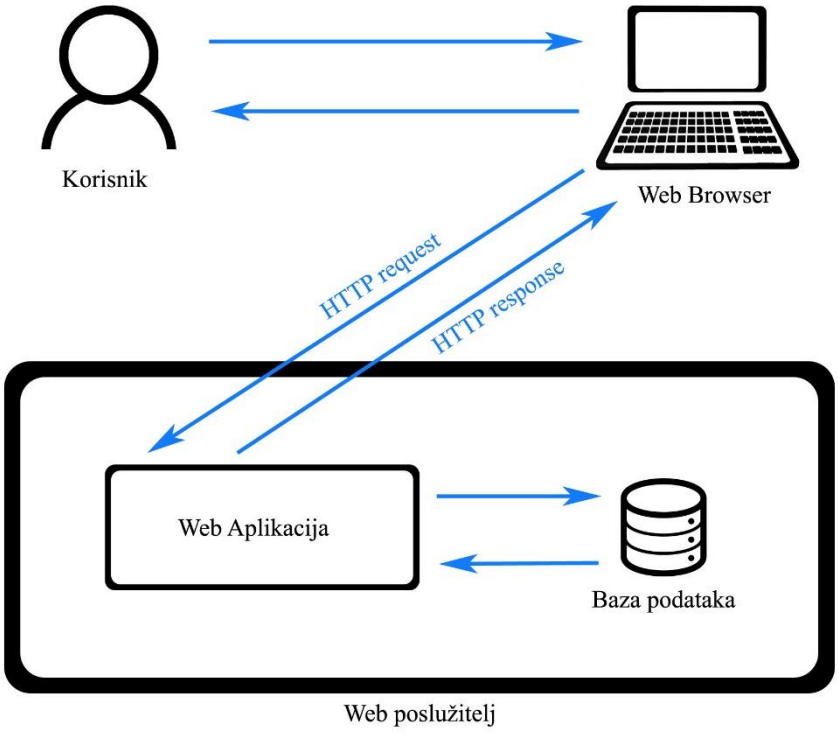
Glavna funkcionalnost koja nam omogućuje praćenje napretka je pregled treninga, prikazana na slici 5, gdje se korisniku prikazuju svi treninzi poredani po datumu kreiranja. Korisnik zatim ima mogućnost odabira jednog od ponuđenih treninga, čime dobiva uvid u svoje rezultate i očekivane rezultate za svaku distancu vježbe odrađene na odabranom treningu.



Slika 5. Obrazac uporabe - pregled treninga

2.2. Skica sustava

Web aplikacija je platformski neovisna, odnosno za pristup je potreban samo web preglednik (eng. *Web Browser*), preko kojega, pomoću HTTP zahtjeva (eng. *HTTP request*) pristupamo funkcionalnostima naše web aplikacije. Korisnik šalje HTTP zahtjev, kojeg web poslužitelj prima i obrađuje. Aplikacija na temelju poslanog HTTP zahtjeva pokreće određenu funkcionalnost, te po potrebi pristupa bazi podataka. Nakon izvršavanja pokrenutog procesa, web aplikacija, odnosno web poslužitelj vraća korisniku HTTP odgovor (eng. *HTTP response*) u kojem se nalaze odgovarajući podaci i informacija o uspješnosti izvršavanja poslanog HTTP zahtjeva. Opisani način komunikacije između dijelova sustava prikazan je na slici 6.



Slika 6. Skica sustava

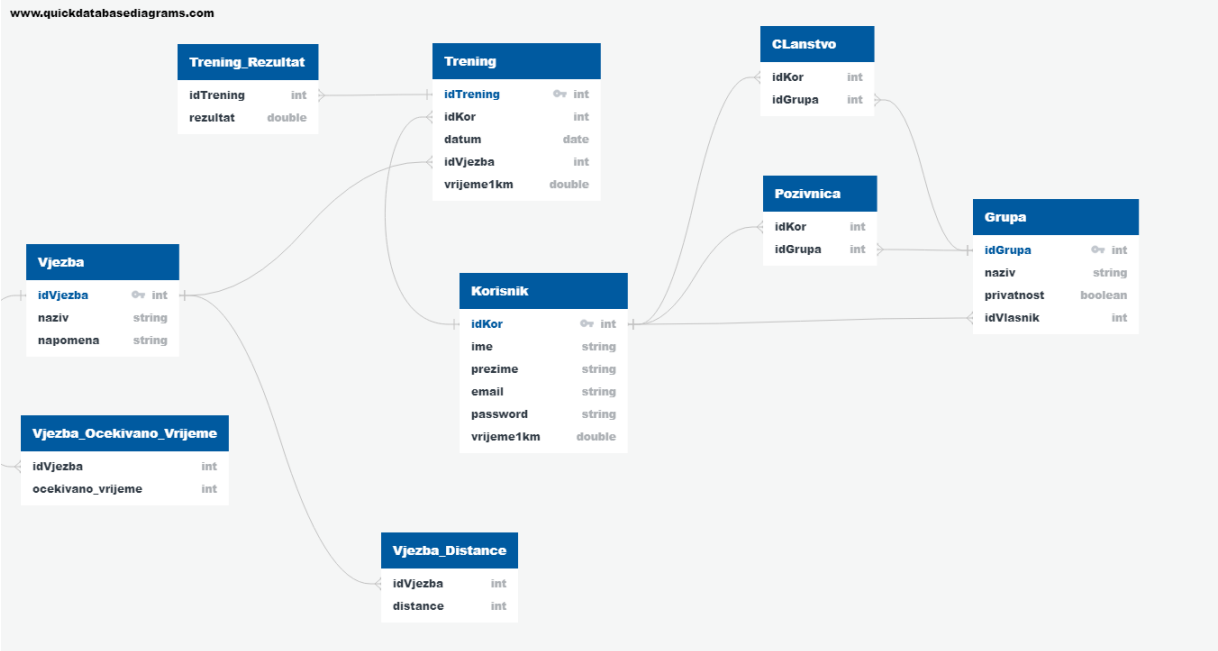
Web poslužitelj

Web poslužitelj se nalazi na udaljenom računalu koje ima svoju statičku IP adresu, pomoću koje mu pristupamo. Na njemu je pokrenuta web aplikacija koja čeka korisničke zahtjeve, te baza podataka u koju aplikacija sprema podatke [6].

Baza podataka

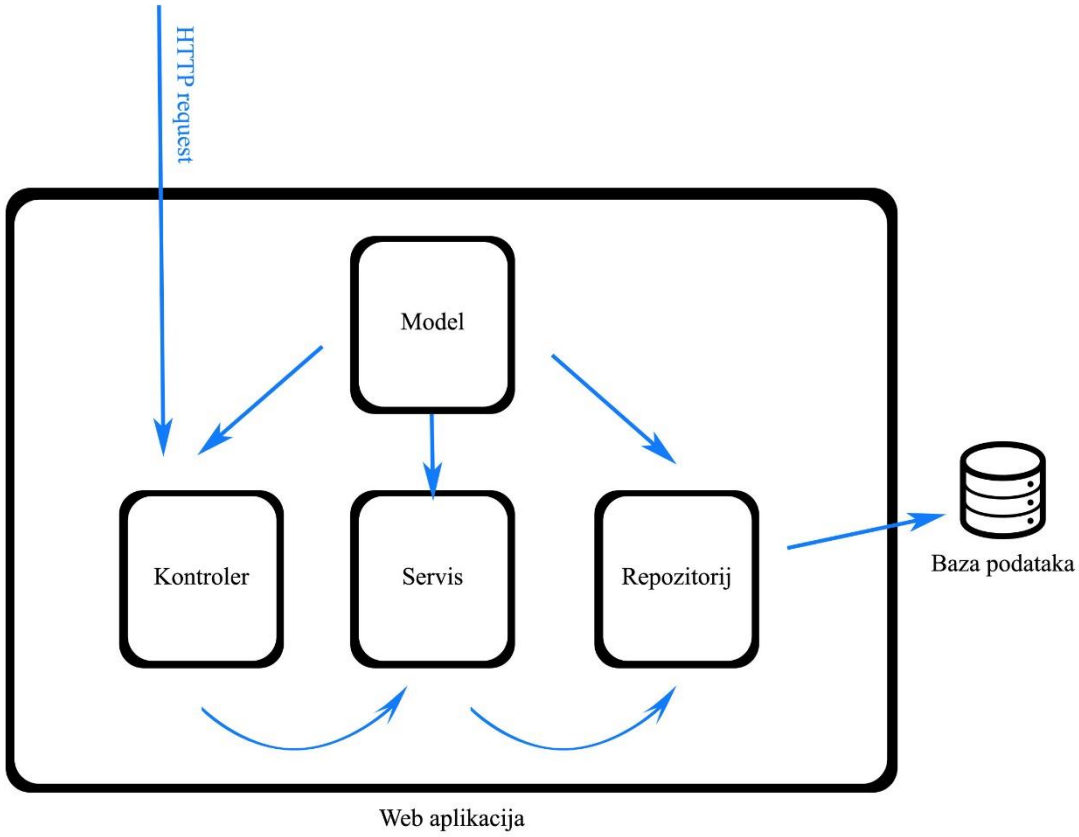
Baza podataka je skup podataka pohranjen u elektroničkom obliku na točno određeni i organizirani način. Baze se razlikuju po načinu organizacije spremanja podataka. Za potrebe ove aplikacije korištena je relacijska baza podataka, gdje su podaci spremljeni u tablice, odnosno relacije, koje imaju vlastite atribute. Takav način spremanja podataka, te korištenje sustava za upravljanje bazom podataka omogućuje efikasno rukovanje s podacima.

Sustav za upravljanje bazom podataka (eng. *Database Management System-DBMS*) je program koji pruža korisnicima grafičko sučelje za manipulaciju s podacima, ali i brine o načinu dohvaćanja, spremanja i ažuriranja podataka.



Slika 7. Baza podataka - relacijski model

Na slici 7 prikazan je fizički, odnosno relacijski model baze podataka. Entitet **Korisnik**, jednoznačno je određen svojim, automatski generiranim, primarnim ključem **idKor**, te ostalim atributima: imenom, prezimenom, emailom, lozinkom (*password*), i referentom vrijednošću koja služi za uspoređivanje rezultata treninga (*vrijeme1km*). Entitet **Trening** sadrži sljedeće atribute: **idTrening**, koji je ujedno i primarni ključ, **idKor**, koji predstavlja korisnika koji je odradio određeni trening, **datum**, **idVjezba**, koji predstavlja odrađenu vježbu na pojedinom treningu, i **vrijeme1km**, koji predstavlja korisnikovu referentnu vrijednost u vrijeme odrađivanja treninga (trenutno korisnikovo *vrijeme1km* može biti drugačije nego u vrijeme odrađivanja treninga). Relacija **Trening_Rezultat** se sastoji od dva atributa, **idTrening** i **rezultat**, koji zapravo predstavljaju listu rezultata za svaku distancu za pojedini trening. Entitet **Vjezba** ima svoj primarni ključ **idVjezba**, **naziv**, te opcionalno napomenu, koja je predviđena za informacije o pauzama između distanci, tempu izvođenja vježbi ili pak nešto treće. Vježba ima dvije povezane relacije, **Vjezba_Distance**, koja predstavlja popis distanci koje se izvode za pojedinu vježbu, te **Vjezba_Ocekivano_Vrijeme** koja predstavlja očekivana vremena izvođenja za svaku distancu vježbe. Atribut **ocekivano_vrijeme** u relaciji **Vjezba_Ocekivano_Vrijeme** označava broj sekundi koje je potrebno dodati na korisnikovo referentno vrijeme, **vrijeme1km**, kako bi dobili ciljanu vremensku vrijednost za pojedinu distancu. Relacija **Grupa** ima atribute **idGrupa**, **naziv**, **idVlasnika**, koji predstavlja korisnika koji je stvorio grupu, te **privatnost**, koja ako je postavljena na vrijednost *true* označava da samo vlasnik ima mogućnost pregleda treninga članova grupe, inače, svi ostali članovi imaju mogućnost pregleda treninga ostalih članova grupe. Relacije **Clanstvo** i **Pozivnice** predstavljaju popis članova grupe, te popis pozvanih korisnika po grupama koji još nisu odbili niti prihvatili pozivnicu.



Slika 8. Skica aplikacije

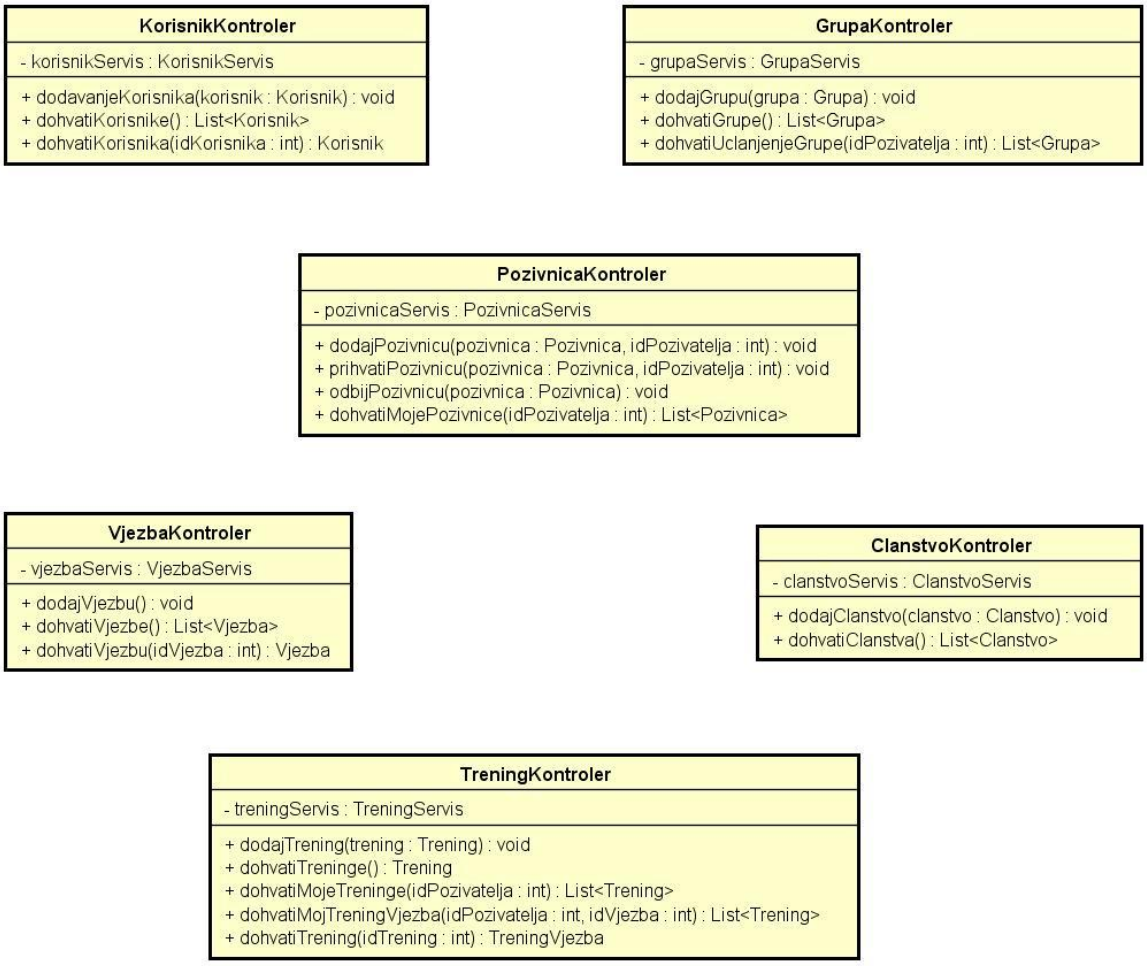
Slika 8 prikazuje model aplikacije, odnosno način komuniciranja između komponenti unutar same web aplikacije. Kontroleri su razredi koji zaprimaju HTTP zahtjeve, te na temelju njih određuju koja će se funkcionalnost izvršiti. Nakon zaprimanja zahtjeva, aplikacija prolazi kroz sve razrede koji su označeni kao kontroleri, te u njima traži metodu koja odgovara primljenom zahtjevu. Ako postoji, tražena metoda delegira posao određenoj usluzi (eng. *Web service*), te mu predaje potrebne parametre koje je dobila prilikom zaprimanja HTTP zahtjeva. Usluge (servisi) predstavljaju srž web aplikacije, u njima se odvija poslovna logika i sve funkcionalnosti koje aplikacija pruža. Oni pomoću repozitorija, koji imaju ugrađene funkcije za komunikaciju s bazom podataka, dohvaćaju i spremaju podatke u bazu podataka. Sve navedene komponente rukuju s modelima, koji predstavljaju naše entitete u bazi podataka.

2.3. Dijagram razreda

2.3.1 Kontroleri

Kontroleri su dio sustava, prikazani na slici 9, koji komuniciraju s okolinom. Oni rukuju s HTTP zahtjevima te na temelju njih pozivaju određene metode. Svaki od prikazanih kontrolera kao člansku varijablu ima odgovarajući razred koji pruža određene usluge (eng. *Web service*), čije funkcije koristi kako bi se izvršila određena funkcionalnost.

`KorisnikKontroler` sadrži metode: **`dodavanjeKorisnika(Korisnik korisnik)`** koja služi za dodavanje novog korisnika prilikom registracije, **`dohvatiKorisnike()`** koja vraća listu svih registriranih korisnika, te **`dohvatiKorisnika(int idKorisnika)`** koja dohvaća korisnika čiji *id* odgovara predanome parametru.



Slika 9 Dijagram razreda – Kontroleri

GrupaKontroler ima jednu člansku varijablu tipa **GrupaServis**, te metode **dodajGrupu(Grupa grupa)** koja se poziva prilikom stvaranja nove grupe, **dohvatiGrupe()** koja vraća sve postojeće grupe, i **dohvatiUclanjeneGrupe(int idPozivatelja)** koja vraća listu svih grupa kojih je korisnik član.

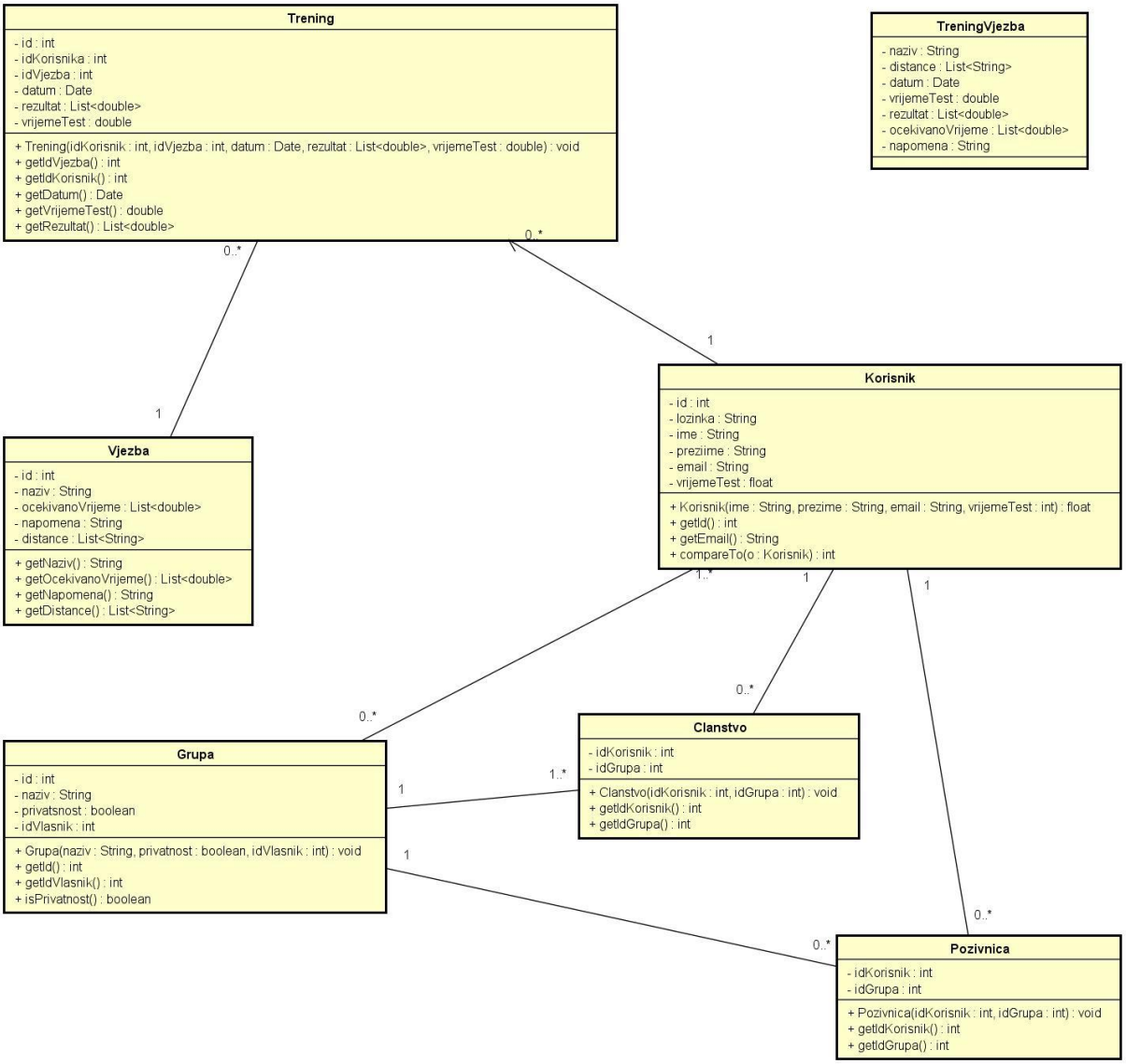
PozivnicaKontroler sadrži servis **PozivnicaServis**, metodu **dodajPozivnicu(Pozivnica pozivnica, int idPozivatelja)** koja se poziva prilikom slanja pozivnice određenom članu za određenu grupu, metode **prihvatiPozivnicu()** i **odbijPozivnicu()** koje se pozivaju kada korisnik prihvati ili odbije poziv za članstvo u nekoj grupi, te **dohvatiMojePozivnice()** koja vraća sve pozivnice od određenog korisnika.

VjezbaKontroler ima člansku varijablu tipa **VjezbaServis** i metode **dodajVjezbu(Vjezba vjezba)**, **dohvatiVjezbe()** i **dohvatiVjezbu(int idVjezbe)**, koje služe za dodavanje nove vježbe , dohvaćanje svih postojećih vježbi i dohvaćanje točno određene vježbe.

ClanstvoKontroler se sastoji od jednog razreda tipa **ClanstvoServis** i metoda **dodajClanstvo(Clanstvo clanstvo)** i **dohvatiClanstva()** koje služe za dodavanje novog članstva i dohvaćanje svih postojećih.

TreningKontroler uz uslugu **TreningServis** sadrži i metode za dodavanje novog treninga **dodajTrening(Trening trening)**, dohvaćanje svih treninga pojedinog korisnika **dohvatiMojeTreninge(int idPozivatelj)**, dohvaćanje svih treninga na kojima se izvodila odabrana vježba **dohvatiMojTreningVjezba(int idPozivatelj, int idVjezba)**, te za dohvaćanje pojedinog treninga **dohvatiTrening(int idTrening)**.

2.3.2 Modeli



Slika 10. Dijagram razreda – Modeli

Modeli prikazani na slici 10 su razredi koji opisuju entitete u bazi podataka. Usluge (eng *web services*) sustava koje pružaju funkcionalnosti koriste modele prilikom rukovanja s podacima dobivenim iz baze podataka. Modeli nisu izravno spojeni s bazom, već se pomoću sučelja **`JpaRepository`** na temelju njih stvaraju relacije u bazi podataka.

Razred **`Korisnik`** predstavlja korisnika sustava, a sastoji se od članskih varijabli: **`id`** tipa *int*, **`lozinka`**, **`ime`**, **`prezime`**, **`email`**, koji su tipa *String*, te **`vrijemeTest`** tipa *float*, koje predstavlja referentno vrijeme korisnika. Sadrži konstruktor, koji prima ime, prezime, lozinku i referentno vrijeme, metode za dohvaćanje varijabli **`id`** i **`email`**, i metodu **`compareTo()`** koju nadjačava. Jedan korisnik može odraditi nula ili više treninga, biti član više grupa, dobiti nula ili više pozivnica za učlanjenje u grupu.

Razred **`Vjezba`** sadrži člansku varijablu **`id`**, **`naziv`**, **`ocekivanoVrijeme`** koja je lista *double* vrijednosti koje predstavljaju očekivana vremena svake distance, **`napomena`**, te **`distance`** tipa *List<String>* koje predstavljaju duljinu ili vrijeme trajanja svake relacije u jednoj vježbi. Jedna vježba se može izvoditi na više različitih treninga.

`Trening` također ima vlastiti **`id`**, **`idKorisnika`** koji predstavlja korisnika koji je odradio određeni trening, **`datum`** tipa *Date*, **`idVjezba`** koja označava vježbu koja se izvodila na pojedinom treningu, **`vrijemeTest`** koje predstavlja korisnikovu referentnu vrijednost u vrijeme izvođenja treninga, te člansku varijablu tipa *List<double>* koja predstavlja korisnikova vremena za svaku distancu odrađene vježbe na tome treningu. Na jednom treninga se može izvoditi najviše jedna vježba, te isti trening može pripadati samo jednome korisniku.

Razred **`Grupa`** se sastoji od varijable **`id`**, **`naziv`**, **`idVlasnika`** koji predstavlja *id* korisnika koji je stvorio grupu, i **`privatnost`**. U jednoj grupi može biti najmanje jedan, vlasnik grupe, ili više članova, te je moguće poslati pozivnice za više korisnika.

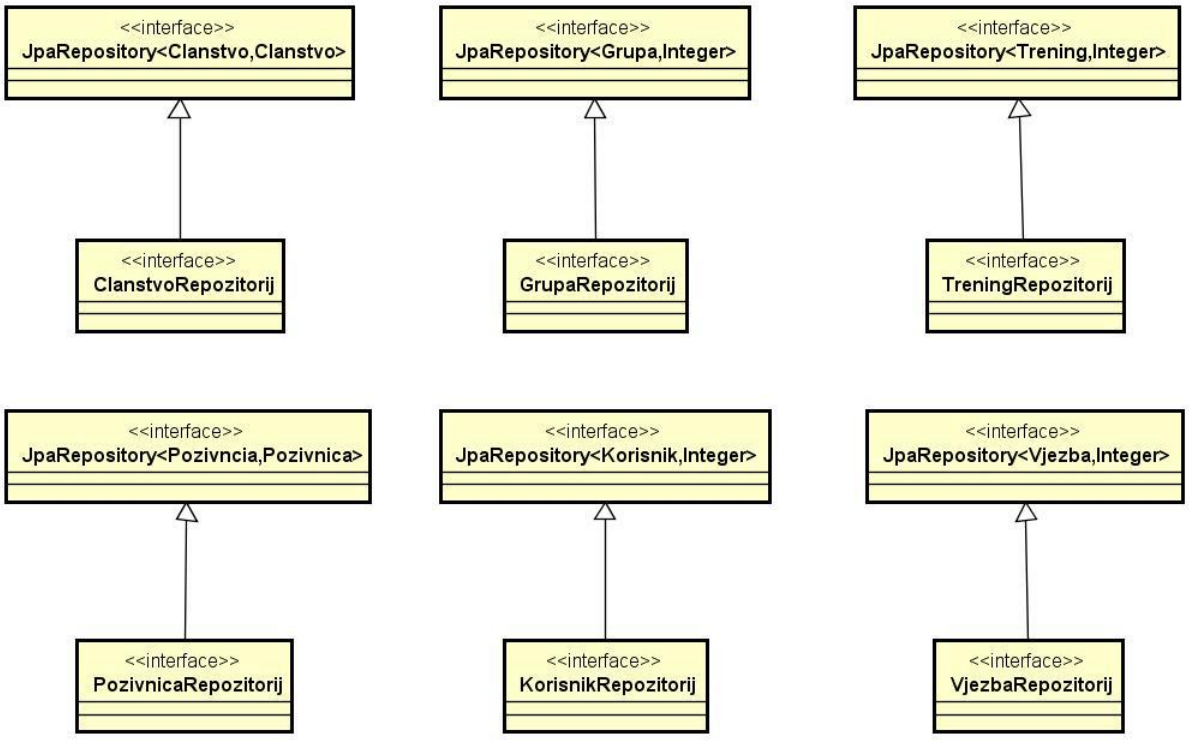
Razred **`Clanstvo`** i **`Pozivnica`** imaju varijable **`idKorisnik`** i **`idGrupa`**. U razredu **`Clanstvo`** te dvije vrijednosti predstavljaju informaciju o članstvima korisnika u grupi, dok u razredu **`Pozivnica`** predstavljaju pozive u grupe za određene korisnike.

Razred **`TreningVjezba`** sa svojim varijablama, **`naziv`**, **`distance`**, **`datum`**, **`vrijeme test`**, **`rezultat`**, **`ocekivanoVrijeme`** i **`napomena`**, objedinjuje informacije objekta **`Trening`** i

pripadajućeg objekta **`Vjezba`** čiji *id* odgovara varijabli **`idVjezba`** objekta **`Trening`**, te služi za lakše rukovanje s podacima na klijentskoj strani.

2.3.3 Repozitoriji

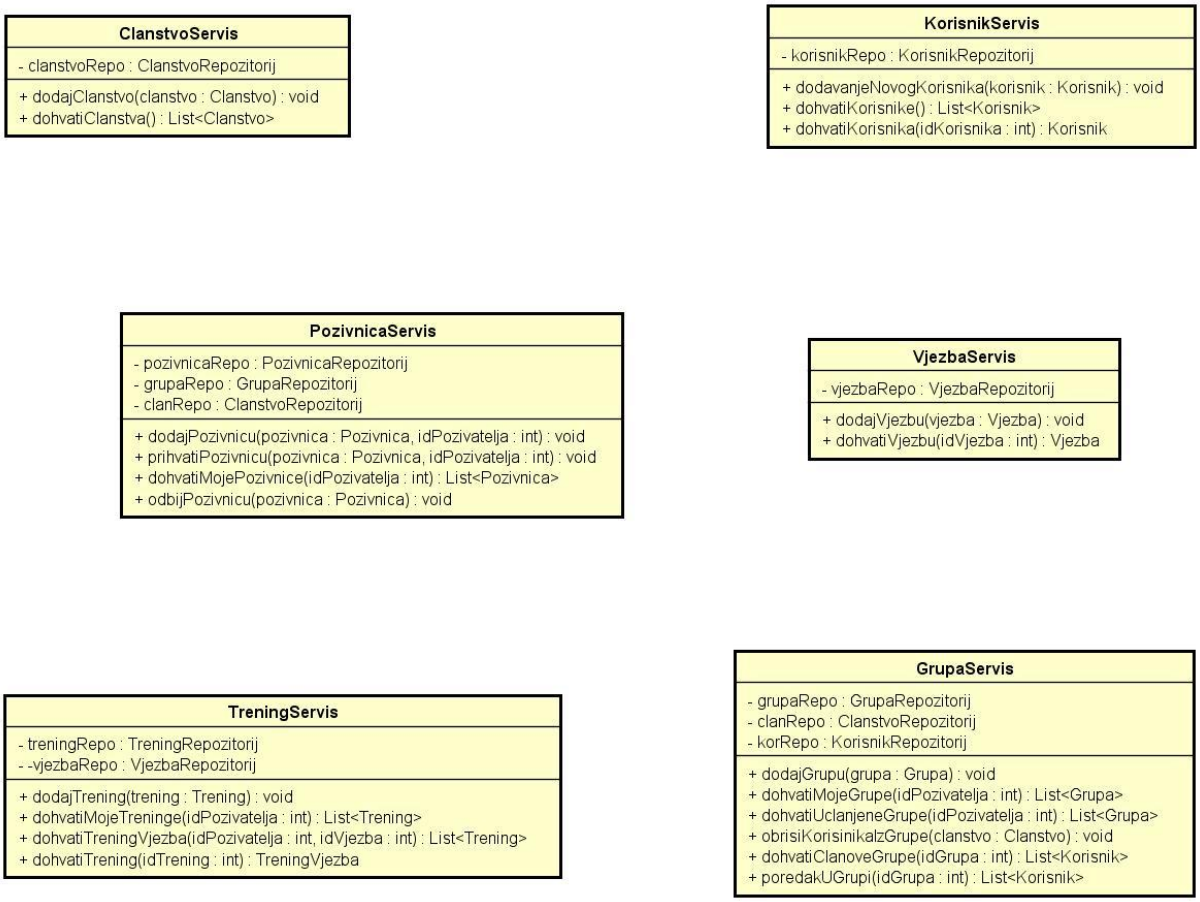
Repozitoriji prikazani na slici 11 su sučelja koja nasljeđuju parametrizirano sučelje **`JpaRepository<T,ID>`** koje ima ugrađene metode za dohvaćanje i spremanje podataka u bazu podataka. Prilikom nasljeđivanja sučelja **`JpaRepository<T,ID>`**, kao prvi parametar „T“ potrebno je predati razred koji odgovara tablici u bazi podataka nad kojom će naš repozitorij obaviti operacije, te kao drugi parametar razred koji predstavlja primarni ključ naše tablice.



Slika 11. Dijagram razreda - repozitoriji

Sustav sadrži **`ClanstvoRepozitorij`**, **`GrupaRepozitorij`**, **`TreningRepozitorij`**, **`PozivnicaRepozitorij`**, **`KorisnikRepozitorij`**, i **`VjezbaRepozitorij`**. Svaki od navedenih se brine za dohvaćanje i spremanje jednog od prije spomenutih modela.

2.3.4 Usluge



Slika 12. Dijagram razreda - Usluge

Usluge (servisi) su razredi, prikazani na slici 12, koji sadrže metode u kojima se odvija poslovna logika sustava. Njihove funkcije se pozivaju od strane kontrolera, a prilikom pružanja određene funkcionalnosti, pomoću repozitorija dohvaćaju podatke iz baze koje dobiju u obliku pripadajućeg modela.

`ClanstvoServis` ima člansku varijablu tipa **`ClanstvoRepozitorij`**, i dvije metode koje služe za unos novog članstva i dohvaćanje postojećih.

`KorisnikServis` sadrži primjerak sučelja **`KorisnikRepozitorij`**, metode za dodavanje novog korisnika u sustav, dohvaćanje postojećih korisnika, te dohvaćanje korisnika s traženom *id* vrijednošću.

PozivnicaServis ima tri različita repozitorija, **PozivnicaRepozitorij**, **GrupaRepozitorij** i **ClanstvnoRepozitorij**. I metode za dodavanje nove pozivnice, prihvaćanje pozivnice, dohvaćanje svih pozivnica koje se odnose na pojedinog korisnika, i metodu za odbijanje pozivnice.

VjezbaServis koristi **VjezbaRepozitorij**, te metode za dodavanje nove vježbe i dohvaćanje vježbe koja sadrži odgovarajući id.

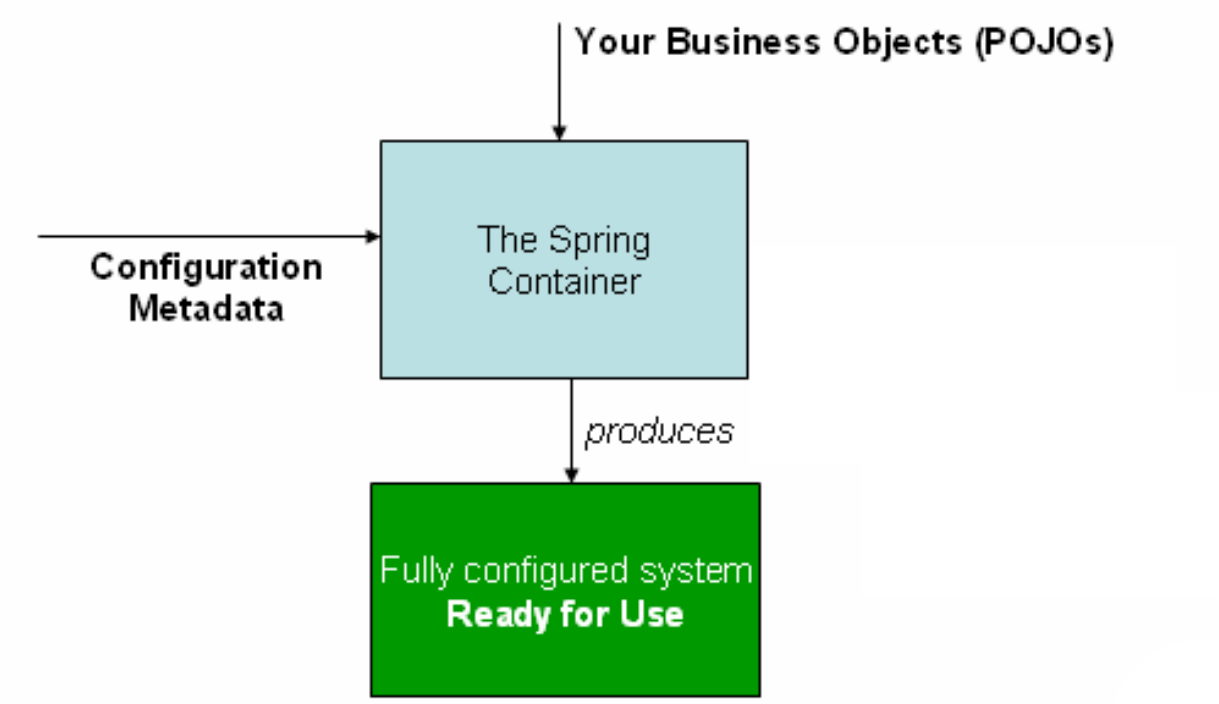
TreningServis ima dva primjerka repozitorija, jedan je tipa **TreningRepozitorij**, a drugi tipa **VjezbaRepozitorij**. Te metode služe za dodavanje novog treninga, dohvaćanja svih treninga pojedinog korisnika, dohvaćanje svih treninga pojedinog korisnika na kojima se izvodila određena vježba, Te dohvaćanje treninga koji sadrži odgovarajući *id*.

GrupaServis sadrži **GrupaRepozitorij**, **ClanstvoRepozitorij** i **KorisnikRepozitorij**. Ujedno sadrži i metode za stvaranje nove grupe, dohvaćanje grupa koje je pojedini korisnik kreirao, dohvaćanje grupa čiji je određeni korisnik član, brisanje pojedinog korisnika iz grupe, dohvaćanje svih članova neke grupe, i metodu za određivanje poretka u određenoj grupi.

3. KORIŠTENE TEHNOLOGIJE I ALATI

3.1. Mogućnosti korištenja radnog okvira Spring

Spring Framework je radni okvir programskoj jezika Java, koji je prvi put predstavljen 2002. godine, te je od tada stekao veliku popularnost zbog svoje jednostavnosti i učinkovitosti. Jedna od glavnih prednosti radnog okvira Spring je što omogućava programeru da se više fokusira na rješavanje poslovnih problema a manje na konfiguraciju sustava, ali ostavljajući mnogo prostora za mogućnost vlastite konfiguracije.



Slika 13. IoC Container (Inversion of Controle)

Spring Container predstavlja srž radnog okvira Spring. On uzima naše razrede, koji su predstavljeni kao objekti POJO (eng. *Plain Old Java Object*), i na temelju konfiguracijskih datoteka, koje mogu biti u obliku xml datoteka, anotacija ili Java koda, stvara objekte, međusobno ih povezuje te upravlja njima tijekom njihovog životnog ciklusa [4].

Radni okvir Spring je modularan i pruža mogućnost zasebnog korištenja svakog od modula. U izradi naše aplikacije koristili smo module za izradu web aplikacija, Spring MVC i Spring REST.

Moduli Spring MVC i REST se temelje na jednom centralnom razredu **`DispatcherServlet`** koji zaprima sve HTTP zahtjeve. On je zadužen, nakon primljenog HTTP zahtjeva, odrediti koji kontroler je potrebno pozvati.

3.2. Mogućnosti korištenja biblioteke React

React je biblioteka programskog jezika JavaScript, koja se koristi za izradu korisničkog sučelja, te predstavlja komponentu „prikaz“ (eng. *View*) u našoj MVC arhitekturi [8]. React dozvoljava i korištenje drugih radnih okvira i biblioteka. Aplikacija bazirana na biblioteci React je zamišljena kao jedno stranična (eng. *single-page*) aplikacija, odnosno, fizički postoji samo jedna HTML datoteka, čije se komponente i stanja po potrebi prikazuju i osvježavaju. Komponente, koje su temeljne jedinice React aplikacije, su hijerarhijski organizirane. Svaka komponenta mora implementirati **`render()`** metodu, koja je zadužena za prikaz pojedine komponente [2]. Komponente sadrže stanje (eng. *State*), koje je promjenjivo i svojstva (eng. *Properties*) koja se prenose prilikom definicije komponente, te ih nije moguće naknadno mijenjati (eng. *read-only*).

Model prikaza objekata (eng. *Document Object Model – DOM*) je reprezentacija HTML dokumenta organizirana kao stablo, te je takva spremljena u memoriji. Prilikom promjena na našoj stranici, potrebno je stvoriti novi model prikaza objekata (DOM), odnosno novu strukturu stabla, te prikazati sve njezine elemente.

React koristi virtualni model prikaza objekata (eng. *Virtual DOM*), koja predstavlja strukturu stabla JavaScript objekata, koja nije prikazana korisniku. Prilikom inicijalnog pokretanja, na temelju React elemenata stvara se virutalni model prikaza objekata (Virtual DOM), te se na temelju njega, u memoriji stvara model prikaza objekata (DOM), čiji se elementi moraju prikazati korištenjem **`render()`** metode. Kod promjena na stranici, imamo dva primjerka virtualne (Virtual DOM) strukture koje se uspoređuju, te na temelju njihove razlike se određuju elementi koje je potrebno ponovno dodati u model prikaza objekata (DOM) i prikazati.

3.3. Ostale korištene tehnologije i alati

Java je objektno orijentirani programski jezik. Programi napisani u Javi se izvršavaju na virtualnom stroju. Stoga se isti kod može izvoditi na svim operacijskim sustavima za koje postoji Javin virtualni stroj [7]. Za pokretanje Javinih programa na računalu, potrebno je instalirati JRE (*Java Runtime Environment*) u kojem se zapravo nalazi JVM (*Java Virtual Machine*) odnosno virtualni stroj. Za potrebe razvoja Javinih programa potrebno je instalirati JDK (Java Development Kit) koji uz virtualni stroj sadrži i javac (prevodilac za Javu).

PostgreSQL je sustav za upravljanje bazama podataka. Otvorenog je koda, te objektno-relacijskog tipa [3].

Intellij IDE je integrirana razvojna okolina koja pruža podršku za pisanje programa u Java programskom jeziku.

Git je sustav za upravljanje verzijama koda pogodan, kako za velike, tako i za manje projekte

HTML (*HyperText Markup Language*) je prezentacijski jezik za izradu web stranica. HTML datoteke su tekstualnog formata, te koristeći različite znakove (eng. *Tags*) opisuju kako će se nešto prikazati na našem web pregledniku.

CSS (*Cascading Style Sheet*) je stilski jezik koji služi za oblikovanje HTML dokumenta.

Astah je program za izradu UML dijagrama. Za izgradnju ovog sustava korištena je njegova podršku za izradu obrazaca uporabe (eng. *Use Case*) koji su služili prilikom definiranja funkcionalnih zahtjeva i izradu dijagrama razreda.

Heroku je platforma u oblaku koja programerima pruža mogućnost prevođenja, pokretanja i izvršavanja aplikacije u potpunosti na web serveru, te takva postaje dostupna svim korisnicima s pristupom internetu.

4. IMPLEMENTACIJA

4.1 Implementacija sustava na poslužiteljskoj strani

Razred **`KorisnikKontroler`** označen je anotacijom *@RestController* te je zadužen za pokretanje funkcionalnosti vezane uz korisnika. Spomenuti razred koristi model **`Korisnik`** kao parametar i kao povratne vrijednosti u svojim metodama. Sadrži člansku varijablu tipa **`KorisnikServis`** čije metode poziva prilikom pokretanja određenih funkcionalnosti

```
1.  package ZavrsniRad.RowingApp.Kontroleri;
2.
3.  import ZavrsniRad.RowingApp.Modeli.EmailRaz;
4.  import ZavrsniRad.RowingApp.Modeli.Korisnik;
5.  import ZavrsniRad.RowingApp.Servisi.KorisnikServis;
6.  import com.fasterxml.jackson.databind.node.TextNode;
7.  import org.springframework.beans.factory.annotation.Autowired;
8.  import org.springframework.http.MediaType;
9.  import org.springframework.web.bind.annotation.*;
10.
11. import java.util.List;
12.
13. @RestController
14. @RequestMapping("/korisnik")
15. public class KorisnikKontroler {
16.
17.     @Autowired
18.     KorisnikServis korisnikServis;
19.
20.     @PostMapping("/dodaj")
21.     public void dodvanjeKorisnika(@RequestBody Korisnik korisnik){
22.         korisnikServis.dodavanjeNovogKorisnika(korisnik);
23.     }
24.
25.     @GetMapping("/dohvati")
26.     public List<Korisnik> dohvatiKorisnike(){
27.         return korisnikServis.dohvatiKorisnike();
28.     }
29.
30.     @GetMapping("/dohvati/{id}")
31.     public Korisnik dohvatiKorisnika(@PathVariable("id") int idKorisnika){
32.        return korisnikServis.dohvatiKorisnika(idKorisnika);
33.     }
34.
35.     @GetMapping("/dohvatiIdKorisnika")
36.     public int dohvatiId(@RequestBody EmailRaz emRaz){
37.         return korisnikServis.dohvatiId(emRaz.getEmail());
38.     }
39. }
```

Slika 14. Implmentacija – KorisnikKontroler

```
1.  package ZavrsniRad.RowingApp.Servisi;
2.
3.  import ZavrsniRad.RowingApp.Modeli.Korisnik;
4.  import ZavrsniRad.RowingApp.Repozitoriji.KorisnikRepozitorij;
5.  import org.springframework.beans.factory.annotation.Autowired;
6.  import org.springframework.stereotype.Service;
7.
8.  import java.util.List;
9.  import java.util.NoSuchElementException;
10.
11.
12. @Service
13. public class KorisnikServis{
14.
15.     @Autowired
16.     KorisnikRepozitorij korisnikRepo;
17.
18.     public void dodavanjeNovogKorisnika(Korisnik korisnik){
19.         try{
20.             if(korisnikRepo.findById(korisnik.getId()).get().getEmail().equals(koris
    nik.getEmail())) { //Provjerava postoji li već korisnik sa unesenom adresom
21.                 System.out.println("Vec postoji korisnik sa odabranim emailom")
22.             }
23.
24.         }catch(NoSuchElementException ex){
25.             korisnikRepo.save(korisnik);
26.         }
27.     }
28.
29.     public List<Korisnik> dohvatiKorisnike(){
30.         return korisnikRepo.findAll();
31.     }
32.
33.     public Korisnik dohvatiKorisnika(int idKorisnika){
34.         return korisnikRepo.findById(idKorisnika).get();
35.     }
36.
37.     public int dohvatiId(String email){
38.         return korisnikRepo.findByEmail(email).getId();
39.     }
40.
41. }
```

Slika 15. Implmentacija – KorisnikServis

Implementacija razreda **`KorisnikServis`**, vidljiva na slici 15, sadrži metode koja obavljaju funkcionalnosti vezane uz korisnika, dodavanje novog korisnika, dohvaćanje svih korisnika i dohvaćanje *id* vrijednosti korisnika s određenom email adresom. Navedene metode spomenute usluge poziva **`KorisnikKontroler`**, što je vidljivo na slici 14. U metodama prikazane usluge pozivaju se funkcije sučelja **`KorisnikRepozitorij`** koje dohvaćaju i spremaju podatke iz baze.

Razred **`Korisnik`** prikazan na slici 16 modelira korisnika sustava. Dodavanjem anotacija *@Entity* i *@Table(name="Korisnik")* smo definirali da nam taj razred predstavlja relaciju, s nazivom Korisnik, u bazi podataka. Označavanjem članske varijable **`private int id`** s anotacijom *@Id* smo definirali primarni ključ relacije, a s anotacijom *@GeneratedValue* označili da će se vrijednost atributa **`id`** u relaciji automatski povećati prilikom unosa svake n-torke. Razred korisnik implementira metodu **`int compareTo()`** sučelja **`Comparable<T>`**, te vraća vrijednost 1 ukoliko korisnik ima bolje referentno vrijeme.

```java
package ZavrsniRad.RowingApp.Modeli;

import javax.persistence.*;

@Entity
@Table(name = "Korisnik")
public class Korisnik implements Comparable<Korisnik>{

    @Id
    @GeneratedValue
    private int id;

    @Column(nullable= false)
    private String lozinka;

    @Column(nullable = false)
    private String ime;

    @Column(nullable = false)
    private String prezime;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private double vrijemeTest;

    public Korisnik(){

    }

    public Korisnik(String ime, String prezime, String email, double vrijemeTest) {

        this.ime = ime;
        this.prezime = prezime;
        this.email = email;
        this.vrijemeTest = vrijemeTest;
    }

    public String getLozinka() {
        return lozinka;
    }

    public void setLozinka(String lozinka) {
        this.lozinka = lozinka;
    }

    public int getId() {
        return id;
    }
```

Slika 16 Implementacija – Korisnik

Na slici 17 prikazana je implementacija sučelja **KorisnikRepozitorij** koje nasljeđuje sučelje **JpaRepository<Korisnik,Integer>**. Metodu za dohvat svih zapisa relacije **Korisnik findAll()**, kao i metodu za dohvat n-torke s određenom vrijednosti primarnog ključa **findById(int id)**, i mnoge druge, prikazani repozitorij naslijedio je od sučelja **JpaRepository**. A sam je deklarirao metodu **findByEmail(String email)** koju nije potrebno implmentirati, a vraća korisnika s određenom email adresom.

```
1.  package ZavrsniRad.RowingApp.Repozitoriji;
2.
3.  import ZavrsniRad.RowingApp.Modeli.Korisnik;
4.  import org.springframework.data.jpa.repository.JpaRepository;
5.
6.  public interface KorisnikRepozitorij extends JpaRepository<Korisnik,Integer> {
7.      Korisnik findByEmail(String email);
8.  }
```

Slika 17. Implementacija – KorisnikRepozitorij

Sučelje **TreningRepozitorij**, prikazano na slici 18, ne deklarira niti jednu vlastitu metodu, već sve metode za rad s zapisima relacije **Trening** koristi od naslijeđenog sučelja **JpaRepository.**

```
1.  package ZavrsniRad.RowingApp.Repozitoriji;
2.
3.  import ZavrsniRad.RowingApp.Modeli.Trening;
4.  import org.springframework.data.jpa.repository.JpaRepository;
5.
6.  public interface TreningRepozitorij extends JpaRepository<Trening,Integer> {
7.  }
```

Slika 18. Implementacija TreningRepozitorij

```java
import java.util.Date;
import java.util.List;

@Entity
public class Trening implements Comparable<Trening> {
    @Id
    @GeneratedValue
    private int id;

    @Column(nullable = false)
    private int idKorisnika;

    @Column(nullable = false)
    private int idVjezba;

    @Column(nullable = false)
    private java.util.Date datum;

    @Column(nullable = false)
    @ElementCollection
    private List<Double> rezultat;

    @Column(nullable = false)
    private double vrijemeTest;

    public Trening(){

    }

    public Trening(int idKorisnika, int idVjezba, Date datum, List<Double> rezultat, double vrijemeTest) {
        this.idKorisnika = idKorisnika;
        this.idVjezba = idVjezba;
        this.datum = datum;
        this.rezultat = rezultat;
        this.vrijemeTest = vrijemeTest;
    }

    public int getIdVjezba() {
        return idVjezba;
    }

    public void setIdVjezba(int idVjezba) {
        this.idVjezba = idVjezba;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getIdKorisnika() {
        return idKorisnika;
    }

    public void setIdKorisnika(int idKorisnika) {
        this.idKorisnika = idKorisnika;
    }
```

Slika 19. Implementacija – Trening

Na slici 19 vidljiva je implementacija razreda **`Trening`**, na temelju koje je stvorena relacija **`Trening`** u bazi podataka. Slično kao u primjeru razreda **`Korisnik`**, i članske varijable razreda **`Trening`** su označene anotacijama: *@Id* za definiranje primarnog ključa, *@GeneratedValue* za automatsko inkrementiranje atributa id, i *@Column(nullable=false)* za definiranje ograničenja vrijednosti atributa koje ne dozvoljava stvaranje zapisa u relaciji s vrijednosti označenog atributa jednakoj *null*.

`TreningKontroler` prikazan na slici 20 zadužen je za pozivanje određenih funkcionalnosti vezane uz treninge. Sadrži primjerak usluge **`TreningServis`**, vidljiv na slici 21, te njegove metode poziva za obavljanje funkcionalnosti dodavanja treninga, dohvaćanja svih treninga iz relacije, dohvaćanje svih treninga određenog korisnika, dohvaćanje treninga na temelju predane id vrijednosti traženog treninga i dohvaćanje svih treninga na kojima se odrađivala odabrana vježba.

Razred **`TreningServis`** koristeći sučelja **`TreningRepozitorij`** i **`VjezbaRepozitorij`** dohvaća podatke iz relacije **`Trening`** i relacije **`Vjezba`**, rukuje s njima, obrađuje ih i sprema u bazu ili kao povratnu vrijednost vraća kontroleru, koji dalje podatke pomoću HTTP protokola šalje na klijentsku stranu (eng. *Frontend*).

```java
package ZavrsniRad.RowingApp.Kontroleri;

import ZavrsniRad.RowingApp.Modeli.Trening;
import ZavrsniRad.RowingApp.Modeli.TreningVjezba;
import ZavrsniRad.RowingApp.Modeli.Vjezba;
import ZavrsniRad.RowingApp.Servisi.TreningServis;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/trening")
public class TreningKontroler {

    @Autowired
    TreningServis treningServis;

    @PostMapping("/dodaj")
    public void dodajTrening(@RequestBody Trening trening){
        treningServis.dodajTrening(trening);
    }

    @GetMapping("/dohvati")
    public List<Trening> dohvatiTreninge(){
        return treningServis.dohvatiTreninge();
    }

    @GetMapping("/dohvati/mojiTreninzi/{pozivatelj}")
    public List<Trening> dohvatiMojeTreninge(@PathVariable("pozivatelj") int idPozivatelja){
        return treningServis.dohvatiMojeTreninge(idPozivatelja);
    }

    @GetMapping("/dohvati/mojiTreninzi/{pozivatelj}/vjezba/{vjezba}") //Vraca listu treninga za pojedinu vjezbu
    public List<Trening> dohvatiMojTreningVjezba(@PathVariable("pozivatelj") int idPozivatelja, @PathVariable("vjezba") int idVjezba){
        return treningServis.dohvatiTreningVjezba(idPozivatelja,idVjezba);
    }

    @GetMapping("/dohvati/{idTrening}")
    public TreningVjezba dohvatiTrening(@PathVariable("idTrening") int idTreninga){
        return treningServis.dohvatiTrening(idTreninga);
    }
```

Slika 20. Implementacija – TreningKontroler

```java
@Service
public class TreningServis {

    private final int brojZadnjihDistanci=20;

    @Autowired
    TreningRepozitorij treningRepo;
    @Autowired
    VjezbaRepozitorij vjezbaRepo;

    public void dodajTrening(Trening trening){
        treningRepo.save(trening);
    }

    public List<Trening> dohvatiTreninge(){
        return treningRepo.findAll();
    }

    public List<Trening> dohvatiMojeTreninge(int idPozivatelja){
        List<Trening> mojiTreninzi=new LinkedList<>();
        List<Trening> sviTreninzi=treningRepo.findAll();
        Collections.sort(sviTreninzi);
        for(Trening trening:sviTreninzi){
            if(trening.getIdKorisnika()==idPozivatelja){
                mojiTreninzi.add(trening);
            }
        }
        return mojiTreninzi;
    }

    public List<Trening> dohvatiTreningVjezba(int idPozivatelja, int idVjezbe){
        List<Trening> mojiTreninzi=dohvatiMojeTreninge(idPozivatelja);
        List<Trening> mojTreningVjezba=new LinkedList<>();

        for(Trening trening:mojiTreninzi){
            if(trening.getIdVjezba()==idVjezbe)
                mojTreningVjezba.add(trening);
        }
        return mojTreningVjezba; //lista treninga za odabranu vjezbu
    }

    public List<GrafPrikaz> dohvatiZadnjeDistance(int idPozivatelja){//prikaz treninga

       return prikaziTreninge(dohvatiMojeTreninge(idPozivatelja));
    }

    public List<GrafPrikaz> dohvatiZadnjeDistanceZaPojedinuVjezbu(int idPozivatelja,int
     idVjezbe){ //prikaz treninga za pojedinu vjezbu
        return prikaziTreninge(dohvatiTreningVjezba(idPozivatelja,idVjezbe));
    }

    public List<GrafPrikaz> prikaziTreninge(List<Trening> treninzi){ //Na temelju preda
    ne liste stvara podatke za prikaz
        List<GrafPrikaz> listaPrikaza=new LinkedList<>();
        int brojZapisa = 0;

        for(Trening trening:treninzi){
            List<Double> ocekVrijeme=vjezbaRepo.findById(trening.getIdVjezba()).get().g
    etOcekivanoVrijeme();

            for(int i=0;i<ocekVrijeme.size();i++){ //postavljamo ocekivano vrijeme
```

Slika 21 Implementacija – TreningServis

4.2 Implementacija prikaza na klijentskoj strani

```
1.  import React, { Component } from 'react'
2.  import { Button, FormGroup, FormControl, ControlLabel } from "react-bootstrap";
3.  import "./Registracija.css";
4.  export class Registracija extends Component {
5.
6.      constructor(props) {
7.          super(props);
8.
9.          this.state = {
10.           email: "",
11.           password: "",
12.           ime:"",
13.           prezime:"",
14.           vrijemeTest:""
15.         };
16.       }
17.
18.
19.     validateForm() {
20.         return this.state.email.length > 0 && this.state.password.length > 0;
21.     }
22.
23.     handleChange = event => {
24.         this.setState({
25.           [event.target.id]: event.target.value
26.         });
27.       }
28.
29.       handleSubmit = event => {
30.         event.preventDefault();
31.         Axios.post('https://rowingbackend.herokuapp.com/users/sign-up',{
32.
33.                 email: this.state.email,
34.                 password: this.state.password,
35.                 ime:this.state.ime,
36.                 prezime:this.state.prezime,
37.                 vrijemeTest:this.state.vrijemeTest
38.
39.         }).then(resp =>{ this.setState({token:resp.data.token})})
40.
41.
42.             console.log("Korisnik registriran");
43.             this.props.history.push({
44.                 pathname: '/admin/login'
45.             });
46.
47.       }
48.     render() {
49.         return (
50.             <div className="Login">
51.         <form onSubmit={this.handleSubmit}>
52.           <FormGroup controlId="email" bsSize="large">
53.             <ControlLabel>Email</ControlLabel>
54.             <FormControl
55.               autoFocus
56.               type="email"
57.               value={this.state.email}
58.               onChange={this.handleChange}
59.             />
60.           </FormGroup>
61.           <FormGroup controlId="password" bsSize="large">
```

Slika 22. Komponenta Registracija

Prilikom unosa putanje *„/admin/registracija*“ u korijenskom elementu (<dvi id=“root“>) se prikaže razred **`Registracija`** koji nasljeđuje razred **`React.Component`**. Nakon spomenutog nasljeđivanja, razred **`Registracija`** je i sam postao Reactova komponenta. Sve Reactove komponente, pa tako i razred **`Registracija`**, moraju ponuditi implementaciju metode **`render()`** koja određuje na koji način će se element prikazati

Kao što se vidi na slici 22, komponenta **`Registracija`** sadrži polja za unos podataka potrebnih za registraciju korisnika. Tijekom unošenja podataka u polja, istovremeno se mijenja stanje komponente (eng. *State*). Nakon unosa podataka i pritiskom na dugme „*Registriraj se*“ poziva se metoda **`handleSubmit()`** u kojoj se šalje HTTP zahtjev sa pripadnim podacima. Nakon registriranja, korisnika se preusmjerava na stranicu prijave u sustav.

Na slici 23 prikazana je komponenta **`Login`**. Slično kao i komponenta **`Registracija`**, ona sadrži dva polja za unos email adrese i lozinke. Nakon unosa podataka i pritiska šalje se HTTP zahtjev s putanjom „'https://rowingbackend.herokuapp.com/users/sign-up“ .

Korišten je gotov predložak [9] Reactove aplikacije i modificiran kako bi odgovarao potrebama aplikacije. U preuzetom predlošku se nalazi komponenta **`Dashboard`** prikazana na slici 24, a prikazuje dvije komponente **`Card`**. U jednoj komponenti **`Card`** se nalazi dijagram s rezultatima i očekivanim vremenima, a u drugome komponenta **`PopisTreninga`**.

Komponenta **`PopisTreninga`** pomoću HTTP zahtjeva dohvaća listu svih treninga pojedinog korisnika, te ih prikazuje u obliku liste.

```
1.  import React, { Component } from "react";
2.  import { Button, FormGroup, FormControl, ControlLabel } from "react-bootstrap";
3.  import "./Login.css";
4.  import Axios from 'axios';
5.
6.  export default class Login extends Component {
7.    constructor(props) {
8.      super(props);
9.
10.     this.state = {
11.       email: "",
12.       password: "",
13.       token:''
14.     };
15.   }
16.
17.   validateForm() {
18.     return this.state.email.length > 0 && this.state.password.length > 0;
19.   }
20.
21.   handleChange = event => {
22.     this.setState({
23.       [event.target.id]: event.target.value
24.     });
25.   }
26.
27.   handleSubmit = event => {
28.     event.preventDefault();
29.     Axios.post('https://rowingbackend.herokuapp.com/login',{
30.         "email": this.state.email,
31.         "lozinka": this.state.password
32.     }).then(resp =>{ this.setState({token:resp.data.token})})
33.
34.     if(this.state.token!==''){
35.         console.log("Korisnik logiran");
36.         this.props.history.push({
37.             pathname: '/admin/home',
38.             state: { loginInfo: {
39.                 email:this.state.email,
40.                 token: this.state.token
41.                 }
42.             }
43.         });
44.     }
45.   }
46.
47.   render() {
48.     return (
49.       <div className="Login">
50.         <form onSubmit={this.handleSubmit}>
51.           <FormGroup controlId="email" bsSize="large">
52.             <ControlLabel>Email</ControlLabel>
53.             <FormControl
54.               autoFocus
55.               type="email"
56.               value={this.state.email}
57.               onChange={this.handleChange}
58.             />
59.           </FormGroup>
60.           <FormGroup controlId="password" bsSize="large">
61.             <ControlLabel>Password</ControlLabel>
```

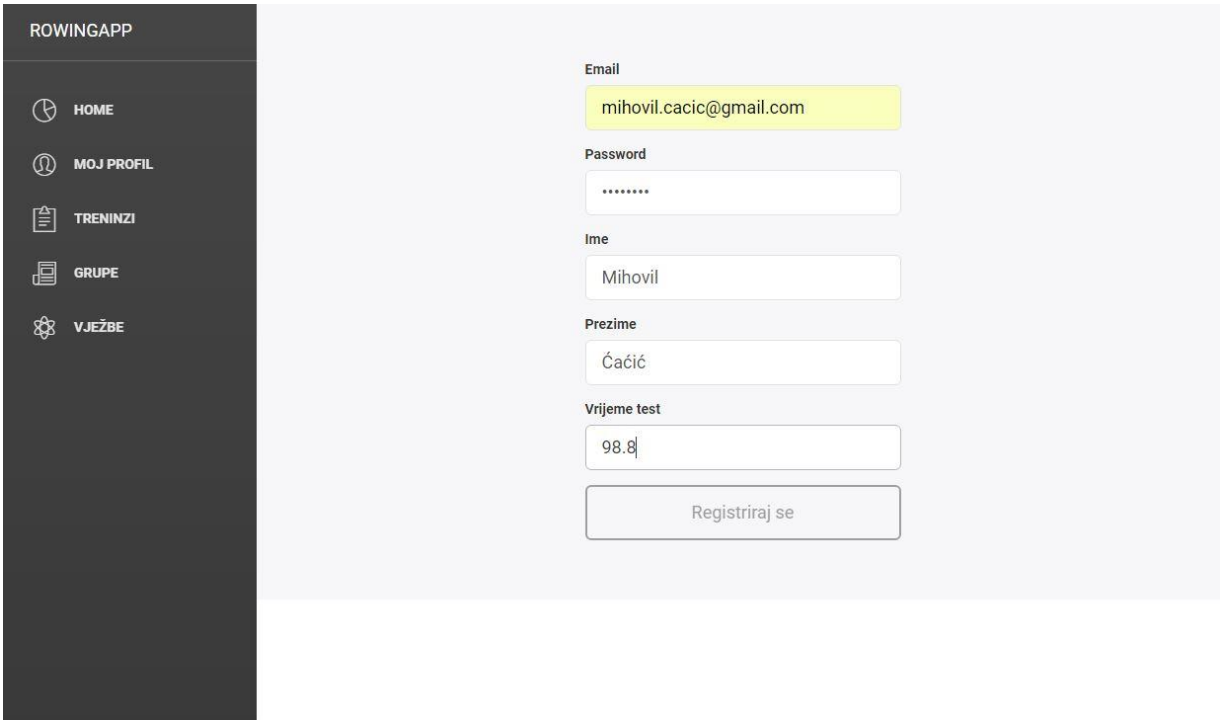
Slika 23. Komponenta Login

```
mport React, { Component } from "react";
import ChartistGraph from "react-chartist";
import { Grid, Row, Col } from "react-bootstrap";
import { Card } from "components/Card/Card.jsx";
import { StatsCard } from "components/StatsCard/StatsCard.jsx";
import { Tasks } from "components/Tasks/Tasks.jsx";
import {
  dataPie,
  legendPie,
  optionsSales,
  responsiveSales,
  legendSales,
  dataBar,
  optionsBar,
  responsiveBar,
  legendBar
} from "variables/Variables.jsx";
import axios from "axios";
import PopisTreninga from "components/PopisTreninga/PopisTreninga";

class Dashboard extends Component {

async componentWillMount(){

   /* console.log(this.props.location.state.loginInfo.email)*/
    await this.setState({
      email:this.props.location.state.loginInfo.email,
      token:this.props.location.state.loginInfo.token
    })

    axios.defaults.headers.common['Authorization']='Bearer ' +this.state.token;
render() {

    return (
      <div className="content">
        <Grid fluid>

          <Row>
            <Col md={8}>
              <Card
                statsIcon=""
                id="s"
                title=""
                category=""
                stats=""
                content={
                  <div className="ct-chart">
                    <ChartistGraph
                      data={this.state.dataSales}
                      type="Line"
                      options={optionsSales}
                      responsiveOptions={responsiveSales}
                    />
                  </div>
                }
                legend={
                  <div className="legend">{this.createLegend(legendSales)}</div>
                }
              />
            </Col>
            <Col md={4}>
```

Slika 24 Komponenta Dashboard

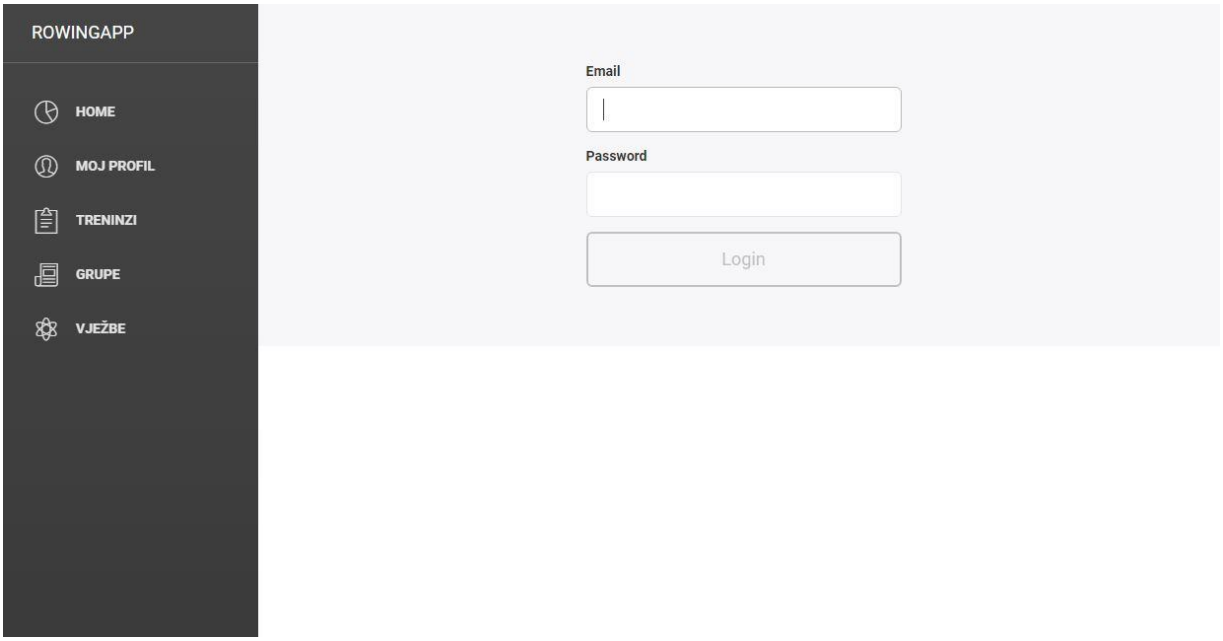
4.3 Korisničko sučelje

Da bi korisnik pristupio mogućnostima koje pruža web aplikacija, prvo mora kreirati svoj korisnički račun. Prilikom registracije potrebno je ispuniti sva polja s odgovarajućim podacima, kao što je prikazano na slici 25.



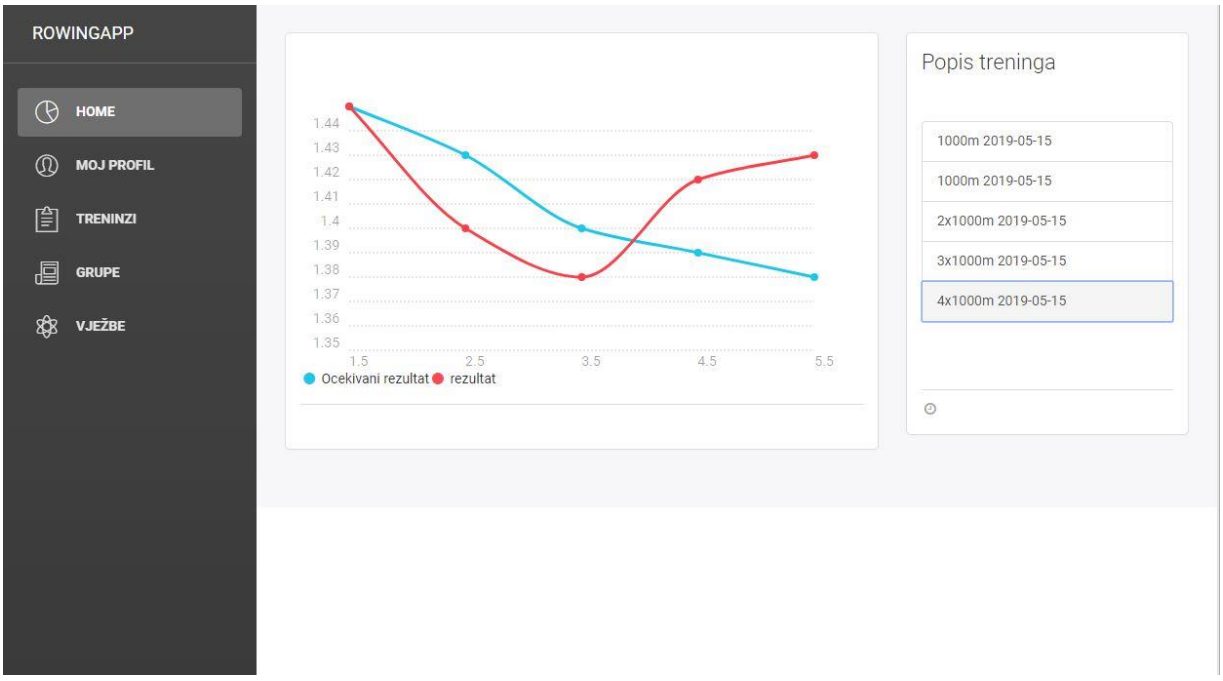
Slika 25. Korisničko sučelje – registracija

Nakon uspješne registracije korisnik se mora prijaviti u sustav s korisničkim podacima koje je unio prilikom registracije. Tek nakon ispravno unesene kombinacije email adrese i lozinke, korisnik ima mogućnost pristupiti ostalim funkcionalnostima sustava. Do tada, nije bio u mogućnosti slati HTTP zahtjeve na poslužiteljski dio sustava. Izgled korisničkog sučelja za prijavu prikazan je na slici 26.



Slika 26 Korisničko sučelje - prijava u sustav

Nakon uspješne prijave, korisnika se preusmjeri na početnu stranicu sustava, prikazanu na slici 27. Na lijevoj strani se nalazi izbornik, dok se na krajnjem desnoj strani nalazi popis svih treninga. Odabirom jednog od ponuđenih treninga prikaže se graf s vrijednostima rezultata treninga i očekivanim vrijednostima.



Slika 27. Korisničko sučelje - početna stranica

5. ZAKLJUČAK

Izrađena web aplikacija od značajne je koristi za praćenje rezultata veslačkih treninga, jer vrlo lako možemo pogledati rezultate za određeni vremenski period, te doći do zaključka o napretku u proteklome razdoblju. Od sljedeće akademske godine, 2019./2020., plan je testirati aplikaciju na članovima veslačke sekcije Fakulteta elektrotehnike i računarstva, te prikupiti povratnu informaciju o korištenju aplikacije. U planu je također i izrada mobilne aplikacije, koja bi korisnicima omogućila unos rezultata tijekom treninga. Prilikom izrade mobilne aplikacije koristila bi se već postojeća baza podataka, i uz manje preinake web aplikacije, moguće je ostvariti komunikaciju između web aplikacije i mobilne aplikacije koja bi koristila usluge koje pruža već izrađena web aplikacija [5]. Razmotrit će se i opcija spajanja na veslački ergometar pomoću tehnologije Bluetooth, te time prikupiti podatke o treningu iz ergometra i spremiti ih u bazu podataka.

6. LITERATURA

[1] Pivotal Softwere, Spring, 13.6.2019., https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[2] Facebook Inc., React, 13.6.2019. , https://reactjs.org/

[3] The PostgreSQL Global Development Group, PostgreSQL, 13.6.2019. , https://www.postgresql.org/docs/

[4] Tutorials Point, Spring IoC Container, 13.6.2019., https://www.tutorialspoint.com/spring/spring_ioc_containers.htm

[5] Trifork, Creating an Android app for your website with Spring Android and REST, 13.6.2019. , https://blog.trifork.com/2011/02/07/creating-an-android-app-for-your-website-with-spring-android-and-rest/

[6] FastWebHost.In, What is Web Server and Different Types of Web Servers, 13.6.2019., https://www.fastwebhost.in/blog/what-is-web-server-and-different-types-of-web-servers/

[7] JavaTPoint, JVM (Java Virtual Machine) Architecture, 13. 6. 2019., https://www.javatpoint.com/internal-details-of-jvm

[8] Wikipedia, Model-View-Controller, 13.6.2019., https://hr.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[9] Creative Tim, Material Dashbord React, 13.6.2019., https://www.creative-tim.com/product/material-dashboard-react

7.POPIS SLIKA

8.NASLOV, SAŽETAK I KLJUČNE RIJEČI NA HRVATSKOM JEZIKU

8.1. Naslov

Izrada web aplikacije za praćenje rezultata veslačkih treninga

8.2. Sažetak

U radu „Izrada web aplikacije za praćenje rezultata veslačkih treninga“ opisan je način izgradnje ovog web sustava. Prikazane su funkcionalnosti koje sustav pruža. Opisane su tehnologije i alati koji su se koristili prilikom izgradnje sustava, te njihove prednosti. Na kraju je prikazana implementacija sustava na poslužiteljskoj i klijentskoj strani.

8.3 Ključne riječi

Java, Spring Boot, React, postgresql, baza podataka, web aplikacija, MVC arhitektura, veslački treninzi

9. NASLOV, SAŽETAK I KLJUČNE RIJEČI NA ENGLESKOM JEZIKU

9.1 Naslov

Development of Web Aplication For Tracking Results of Rowing Training Programs

9.2 Sažetak

This thesis demonstrates the development of a web application for tracking results of rowing training programs using Spring and React technologies. It also provides insight into functional requirements of the system and describes the Spring and React technologies.

9.3. Ključne riječi

Java, Spring Boot, React, database, Postgresql, rowing training, web application, MVC architecture