# WEB APLIKACIJA ZA INTERAKCIJU TRENERA I IGRAČA TENISA

**Lončar, Bruno**

**VISOKO UČILIŠTE ALGEBRA**

ZAVRŠNI RAD

# WEB APLIKACIJA ZA INTERAKCIJU TRENERA I IGRAČA TENISA

Bruno Lončar

Zagreb, studeni 2019.

**Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi**

*„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.*

*U Zagrebu, 6.11.2019.*

*Bruno Loncar*

# Predgovor

Zahvaljujem svom mentoru, profesoru Aleksanderu Radovanu, na svim uputama i smjernicama koje mi je davao prilikom izrade ovog rada. Također želim zahvaliti Visokom učilištu Algebra na pruženom znanju u ove tri godine. Na posljetku, želim iskazati zahvalu svojoj obitelji i prijateljima koji su mi bili najveća podrška tijekom trajanja ovog studija.

**Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi**

# Sažetak

Cilj ovog završnog rada je da se kroz teorijski i praktični dio pokaže izrada informacijskog sustava koji koristi jedan servis za komunikaciju s više klijenata. Sustav je namijenjen trenerima i igračima tenisa, a svrha mu je praćenje podataka o treninzima koje treneri unose za svoje igrače. Rad započinje s planiranjem i postavljanjem zahtjeva sustava, te se nastavlja s modeliranjem sustava i izradom baze podataka s potrebnim tablicama i pohranjenim procedurama. U nastavku se opisuje izrada arhitekture REST servisa kroz slojeve s naglaskom na tehnologiju izrade, skalabilnost, asinkronu komunikaciju s bazom podataka, te sigurnost i autorizaciju. U idućim poglavljima rada se opisuje izrada klijenata koji koriste servis. Prvi i glavni klijent je jednostranična web aplikacija napravljena u Angular-u. Prilikom izrade opisana je arhitektura Angular aplikacije, njezine komponente i servisne klase, te korištenje Angular Material biblioteke za izradu dizajna i korisničkog sučelja aplikacije. Drugi klijent je nativna Android aplikacija koja je namijenjena isključivo trenerima, a čija je svrha osnovni pregled podataka o igračima i treninzima. U radu su prikazane specifične mogućnosti Android platforme koje doprinose ukupnim funkcionalnostima sustava poput aktivnosti, fragmenata, lokalne pohrane i višejezičnosti. Na kraju rada je opisano stavljanje u pogon sustava i njegovih komponenti.

**Ključne riječi:** informacijski sustav, teniski treninzi, REST servis, Angular, Android.

# Sažetak (na engleskom jeziku)

The goal of this paper is to show, in theory, and practice, the development of an information system that uses a service to communicate with multiple clients. The system is intended for tennis coaches and players with the purpose of tracking data of trainings that coaches add for their players. This paper begins with planning and setting the system requirements and continues with system modeling and database creation with all the necessary tables and stored procedures. Later, the paper describes the development of REST service architecture through multiple layers with emphasis on production technology, scalability, asynchronous communication with the database, security, and authorization. Further on, the paper describes the production of client applications that use the service. The first and the main client is a single-page web application built with Angular technology. During the production, the paper describes the Angular architecture, its components, service classes and the usage of Angular Material library for designing the user interface of the application. The second client is a native Android application that coaches can use to view the basic data of their players and trainings. The paper shows how specific possibilities of the Android platform can contribute to the entire system such as activities, fragments, local storage, and internationalization. Finally, the paper describes the deployment of the system and its components.

**Key words:** information system, tennis trainings, REST service, Angular, Android.

# Sadržaj

# 1. Uvod

Tenis je popularan sport koji broji mnoštvo članova diljem cijeloga svijeta. Mnogi profesionalci i amateri svakodnevno treniraju sa svojim trenerima i postižu različite rezultate. Takvi se rezultati zbog njihove količine i loše organizacije teško prate i lako gube. Na tržištu se nalaze generička rješenja koja često ne odgovaraju specifičnim potrebama nekog sporta stoga treneri često odustaju od istih.

Deuce je aplikacija namijenjena upravo trenerima i igračima tenisa. Omogućuje im da se prijavljuju u sustav kojim se prate podatci o treninzima i rezultatima koji se postižu kroz treninge. Trener dodaje svoje igrače te za njih unosi treninge. Svaki trening sadrži podatke o mjestu i datumu održavanja, trajanju te različitim ocjenama koje trener unosi za svaki trening. Trener ima uvid u podatke o unesenim treninzima te kalendar na koji može dodavati treninge i tako pratiti buduće treninge. Igrači se također mogu registrirati i prijavljivati u sustav vidjeti podatke o svojim treninzima kao i informacije o budućim treninzima.

Sustav je dostupan „u oblaku“ te mu se pristupa preko jednostranične web aplikacije (engl. *Single Page Application*, skraćeno SPA) i nativne Android aplikacije. Naglasak rada stavljen je na web aplikaciju koja podržava sve funkcionalnosti sustava, dok Android aplikacija podržava osnovne funkcionalnosti.

Klijentske aplikacije komuniciraju s REST servisom izrađenom u .NET Core Web API tehnologiji. Kroz izradu servisa prikazana je izrada slojevite arhitekture, modeliranje, autorizacija, sigurnost i asinkroni rad. Također, u radu se pokazuje planiranje i izrada baze podataka te potrebnih objekta u njoj na primjeru MySQL baze podataka.

Prilikom izrade rada korištene su najbolje prakse izrade informacijskih sustava te novije tehnologije u za izradu web i mobilnih rješenja poput Angular-a, .NET Core-a i JavaScripta.

# 2. Planiranje informacijskog sustava

Iako mnogi programeri vole odmah krenuti s fazom razvoja sustava tj. programiranjem, to se nije pokazalo kao dobra praksa u razvoju složenijih sustava zbog nedostatka cilja, misije i vizije samog sustava. Bez adekvatnog planiranja brzo dolazi do arhitekturnih problema i drugih zapreka koje je skupo ispravljati u fazi razvoja, dok je ispravljanje istih problema u fazi planiranja mnogo efikasniji i jeftiniji način. Također, iskustvo je pokazalo da je rezultat izrade sustava bez planiranja najčešće ispod očekivanja. Time se dolazi do zaključka da je faza planiranja presudna faza koja određuje ishod cijelog sustava te da preskakanje te faze dovodi do neuspjeha samog sustava [1].

Iz tog razloga, životni ciklus aplikacije kojom se bavi ovaj rad započinje upravo planiranjem.

## 2.1. Ciljevi i vizija

Sustav koji se razvija namijenjen je praćenju teniskih treninga i rezultata koji se postižu na tim treninzima. Sudionici sustava su treneri koji unose različite podatke o svojim igračima te pregledavaju unesene podatke.

Potreba za izradom ovakvog sustava se pojavila iz činjenice da mnogi treneri žele na smislen i strukturiran način pratiti kako njihovi igrači napreduju kroz treninge, no zbog specifične prirode sporta često to ne mogu raditi na cjelovit način koristeći neke od trenutnih rješenja na tržištu. Kroz ovaj sustav želi im se omogućiti da neovisno o uređaju koji koriste za interakciju sa sustavom i lokaciji na kojoj se nalaze, uvijek mogu pristupiti svojim podatcima ili unositi nove.

Da bi se to postiglo, potrebno je napraviti registraciju i prijavu u sustav za njih. Preporuka je da se kod implementacije korisničkih računa implementira i neka vrsta zaštite tih podataka kroz zaštitu lozinke. U ovom slučaju lozinke trebaju biti *hashirane*.

Nakon registracije i prijave, korisnici trebaju biti preusmjereni na početnu kontrolnu ploču svog računa. U slučaju trenera, trebaju mu se pokazati današnji odrađeni i neodrađeni treninzi te njegovi igrači. Trener mora moći ulaziti u pojedine treninge s kontrolne ploče kako bi vidio dodatne informacije o tom treningu. Također, trener treba unositi nove treninge u sustav. Prilikom unosa podataka isti se trebaju validirati kako bi podatci koji se unose bili

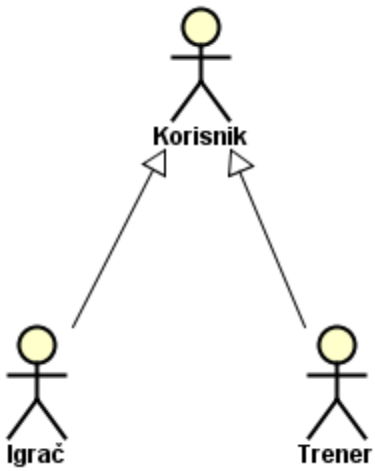
u konzistentnom stanju. Trener također mora moći ući u profile svojih igrača i na njima vidjeti sve prethodne i buduće treninge za njih. Prilikom pregleda profila treba vidjeti i analitiku tih treninga prikazanu tablično i grafički. Trener treba imati mogućnost dodavanja novih igrača i to tako da upiše njihovo ime i prezime ili email adresu te da sustav pronađe sve igrače koji odgovaraju unesenim podatcima. Trener treba imati i pristup svom profilu gdje može uređivati podatke o sebi i svoju sliku.

U slučaju igrača, nakon registracije i prijave se također treba preusmjeriti na početnu kontrolnu ploču na kojoj imaju uvid u svoje podatke o treninzima. Igrači ne mogu unositi podatke o treninzima, već samo pregledavati podatke koje treneri unose o njima. Igrač, kao i trener, treba vidjeti analitiku svojih treninga te uređivati podatke o svom profilu i svoju sliku.

## 2.2. Analiza zahtjeva funkcionalnosti

Iz prethodno definirane vizije i ciljeva sustava kreće se s analizom zahtjeva i zapisivanjem istih u obliku strukturiranog, tabličnog zapisa. U tablici (Tablica 2.1) se u lijevom stupcu nalazi redni broj zahtjeva funkcionalnosti i naziva klijenta koji implementira određenu funkcionalnost. Funkcionalnost koje će se implementirati u oba klijenta (u Angular i Android aplikaciji) naznačene su s oba naziva u lijevom stupcu.

Sudionici sustava su treneri i igrači (Slika 2.1. Sudionici sustava) koji se generaliziraju kao korisnici. Korisnik opisuje aktivnosti koje i trener i igrač mogu raditi.




Slika 2.1. Sudionici sustava

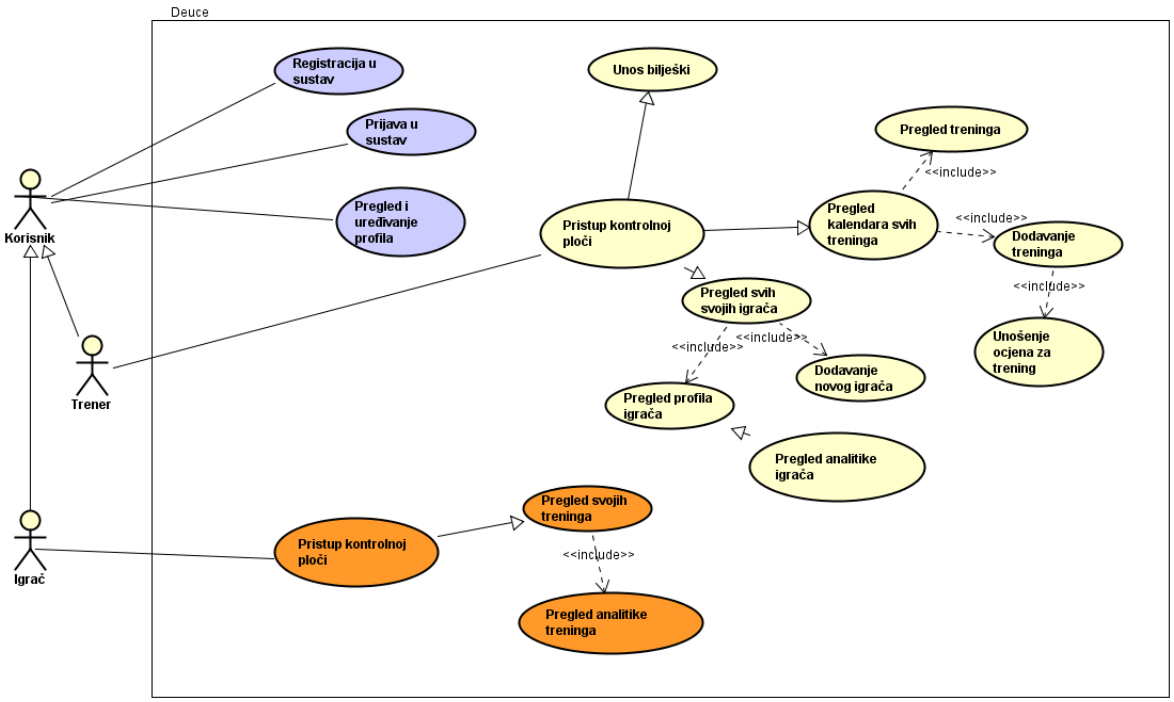
Tablica 2.1. Funkcionalni zahtjevi sustava

| | |
|---|---|
| **Korisnik** | |
| Angular F1, Android F1 | Korisnik će se moći registrirati u sustav i prilikom registracije odabrati želi li napraviti račun kao trener ili kao igrač. |
| Angular F2, Android F2 | Korisnik će se moći prijavljivati u sustav ako je prethodno registriran kako bi pristupio svom računu. |
| Angular F3, Android F3 | Sustav će prilikom registracije unesenu lozinku *hashirati* kako bi se zaštitili podatci i pristup računu u slučaju sigurnosnog incidenta. |
| Angular F4 | Korisnik će moći pristupiti svom profilu te uređivati osobne podatke i sliku. |
| **Trener** | |
| Angular F5, Android F4 | Trener će moći pregledavati sve buduće i prošle treninge u obliku kalendara kako bi imao strukturiran uvid u svoje aktivnosti. |
| Angular F6 | Trener će moći unositi bilješke i podsjetnike na kontrolnoj ploči kako bi na jednom mjestu imao uvid u svoje obveze i zabilješke. |
| Angular F7 | Trener će moći unositi nove treninge u sustav u odrađenom ili neodrađenom obliku te lako raspoznavati koji su treninzi završeni, a koje tek treba unijeti. |
| Angular F8 | Trener će moći unositi različite predefinirane kategorije ocjena za svaki trening kako bi kasnije pratio napredak igrača u svakoj kategoriji. |
| Angular F9 | Trener će moći dodavati nove igrače putem modula za pretragu igrača koristeći njihov email ili ime i prezime. |
| Angular F10, Android F5 | Trener će moći pregledavati profile igrača i vidjeti njihove osobne podatke kao i podatke o prethodnim i budućim treninzima. |
| Angular F11. | Trener će moći pregledavati analitiku unesenih treninga prikazanu grafički i tablično. |

| Igrač | |
|---|---|
| Angular F12, Android F6. | Igrač će moći pregledavati sve svoje buduće i prošle treninge kako bi imao strukturiran uvid u odrađene i buduće treninge. |
| Angular F13. | Igrač će moći pregledavati analitiku svojih unesenih treninga prikazanu grafički i tablično. |

## 2.3. Slučajevi korištenja

Slučajevi korištenja (engl. *Use cases, skraćeno UC*) su način dokumentiranja zahtjeva funkcionalnosti na način koji ih povezuje sa sudionicima koji ih mogu koristiti. Slučajevi korištenja mogu biti prikazani tablično ili grafički. Na slici (Slika 2.2. Slučajevi korištenja sustava) se grafički (dijagramom) prikazuju slučajevi korištenja sustava.




Slika 2.2. Slučajevi korištenja sustava

## 2.4. Tehničke potrebe sustava

S tehničke strane, sustav treba biti razvijen koristeći moderne tehnologije izrade informacijskih sustava. Sustav u ovom slučaju predstavlja REST servis koji se nalazi „u

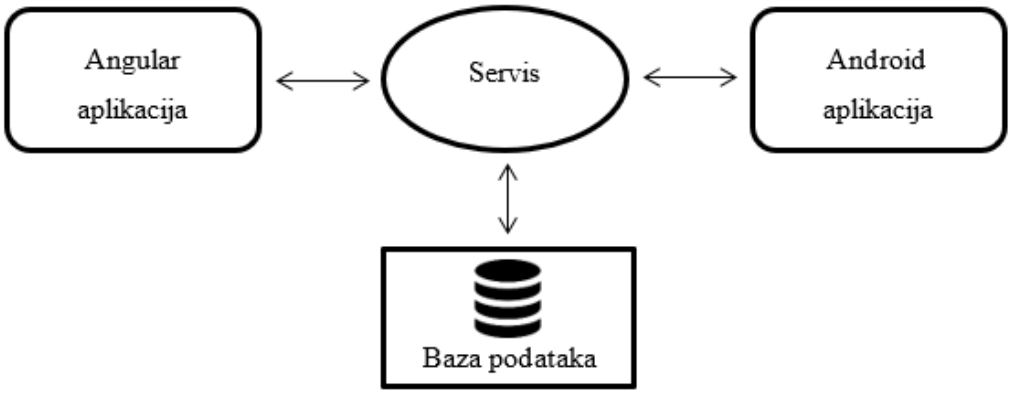
oblaku“ i kojemu se pristupa koristeći dva različita klijenta. U vrijeme izrade ovog rada, na raspolaganju je mnogo tehnologija s kojima se može izraditi takav servis. Tehnologija u kojem će se izrađivati servis će biti .NET Core, programski okvir za izradu modernih servisa koji nije vezan samo za jednu platformu, npr. Microsoft Windows, već se može nalaziti na različitim virtualnim okruženjima.

Servis koji se razvija treba mjesto na koje će se spremati i s kojeg će se dohvaćati podatci. U ovom slučaju to je relacijska baza podataka napravljena u MySQL-u. MySQL je baza podataka koja je, kao i .NET Core, neovisna o platformi na kojoj se nalazi te se može izvršavati na bilo kojem virtualnom okruženju. Baza podataka treba biti napravljena tako da podržava trenutno definirane funkcionalnosti kao i mogućnost da se proširuje ovisno o novim funkcionalnostima.

Prvi klijent koji koristi servis je web aplikacija. Želi se napraviti moderna web aplikacija koja će biti u obliku jednostranične aplikacije. To je vrsta aplikacije u kojoj se jedna stranica dinamički ponovno ispisuje kod novih zahtjeva koristeći JavaScript umjesto da se od servera dohvaćaju nove stranice za nove zahtjeve. Tehnologija u kojoj će se izrađivati ovaj klijent je Angular. To je programski okvir otvorenog kôda (engl. *Open-source framework*) koji omogućuje izradu jednostraničnih aplikacija.

Drugi klijent je nativna Android aplikacija. To je aplikacija napravljena isključivo za pokretanje na Android platformi i karakteriziraju je visoke performanse i visok stupanj pouzdanosti. Također, takve aplikacije imaju pristup različitim hardverskim postavkama pametnog telefona što omogućuje stvaranje složenijih aplikacija koje imaju potrebu korištenja istih postavki.

Arhitektura komponenti sustava je prikazana na slici (**Error! Reference source not found.**).




Slika 2.3. Arhitektura komponenti sustava

# 3. Modeliranje sustava

Svaki informacijski sustav sastoji se od podataka, procesa i okidača koji pokreću procese nad podatcima. Spomenuti podatci se čuvaju u različitim bazama podataka i to u oblicima pogodnim za računalnu obradu i manipulaciju podataka.

Da bi sustav koji se izrađuje odgovarao standardima struke, nužno je da se model čuvanja podataka pravilno postavi već na početku. Na temelju modela podataka se dalje izrađuje servis i njegove komponente, kao i klijentske aplikacije, stoga su izmjene modela podataka skupe za resurse jer pokreću lanac izmjena koji se treba napraviti da bi sustav ponovno bio u željenom stanju.

Prilikom modeliranja sustava koriste se prethodno definirani zahtjevi funkcionalnosti.

## 3.1.1. Definiranje modela podataka

### Korisnik

Modeliranje se započinje korisnikom sustava. Model bilježi podatke o emailu i lozinki kako bi se korisnik uspješno mogao prijaviti u sustav. Model također sadrži i osobne podatke poput imena i prezimena korisnika. Uz te podatke, model sadrži i datum registracije korisnika. S obzirom na to da trener i igrač imaju jednaka polja, za obojicu se može koristiti isti model: korisnik. Prilikom donošenja takve odluke važno je raspoznavati tip korisnika kako bi se znalo je li korisnik trener ili igrač, stoga se povezuje s drugim modelom: tipom korisnika.

### Tip korisnika

Tip korisnika je model koji povezuje korisnika s tipom koji može biti trener ili igrač. Prebacivanje tipa korisnika u zasebni model omogućuje i dodavanje novih tipova korisnika ako sustav pokaže potrebu za tim.

### Trener-igrač

Trener-igrač povezuje trenera sa svojim igračima. Svaki trener može imati više igrača a igrač može imati samo jednog trenera po trenutnoj specifikaciji. Ipak, imajući u vidu da se specifikacija može promijeniti s rastom sustava, i da će možda igrač imati i više treninga, trenutni model omogućuje da se takva promjena jednostavno izvede bez promjene modela.

### Bilješka

Trener može pisati različite bilješke koje se prikazuju na njegovoj kontrolnoj ploči. U trenutnom obliku bilješka je obični tekst, a model povezuje taj tekst s trenerom kojem pripada.

### Trening

Trening je model koji sadrži informacije o datumu i vremenu treninga, mjestu održavanja, trajanju, treneru koji održava trening i igraču koji dolazi na trening. Trening također sadrži podatke o obrađenosti treninga i generalnim pozitivnim i negativnim komentarima vezanim za taj trening. Sva polja osim komentara su obvezna kako bi model bio u konzistentnom stanju.

### Ocjena treninga

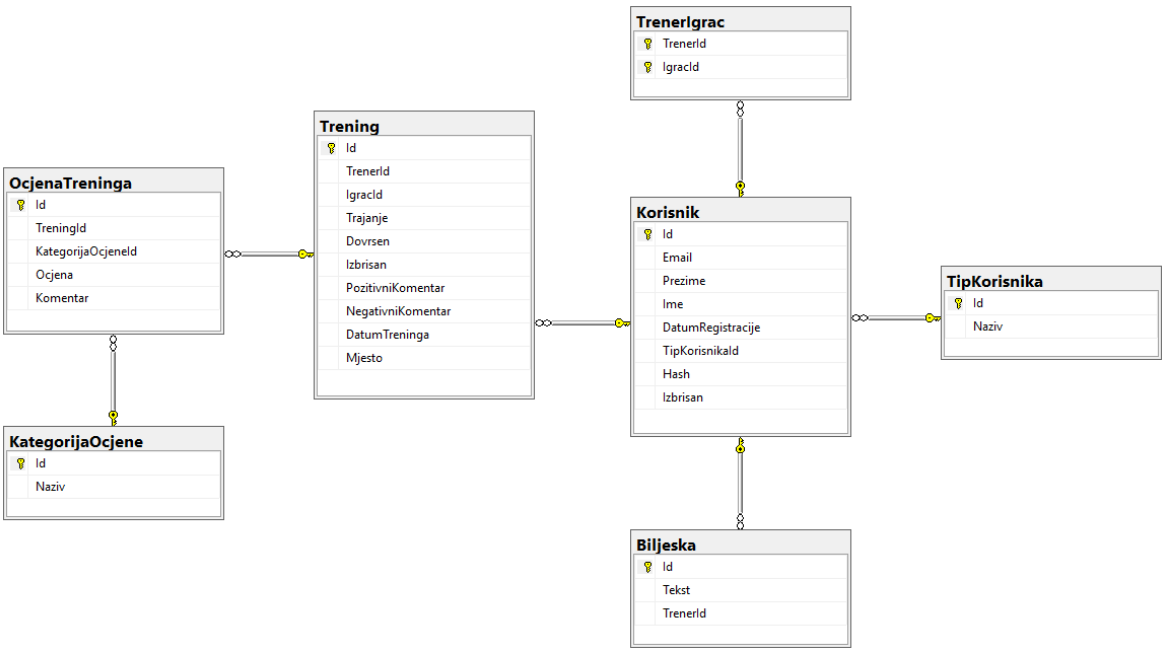
Svaki trening osim generalnog pozitivnog i negativnog komentara sadrži i ocjene koje se za taj trening unose. Model ocjena treninga sadrži informacije o treningu koji se ocjenjuje, predefiniranoj kategoriji ocjene koja se unosi, vrijednosti ocjene koja se unosi od 1 do 10 te komentaru na unesenu ocjenu.

### Kategorija ocjene

Kategorija ocjene je model koji sadrži nazive različitih predefiniranih kategorija koje su dostupne za ocjenjivanje. Primjer nekih od kategorija koje su dostupne su: forhend, bekhend, kondicija, servis, izdržljivost, slice itd.

### Ocjena treninga

Svaki trening osim generalnog pozitivnog i negativnog komentara sadrži i ocjene koje se za taj trening unose. Model ocjena treninga sadrži informacije o treningu koji se ocjenjuje, predefiniranoj kategoriji ocjene koja se unosi, vrijednosti ocjene koja se unosi od 1 do 10 te komentaru na unesenu ocjenu.




Slika 3.1. Dijagram modela sustava

### 3.1.2. Veze između modela sustava

Na dijagramu modela sustava (**Error! Reference source not found.**) navedene su tablice i veze kojima su povezane. Veze između tablica definiraju ovisnosti među njima. To znači da podatci jedne tablice ovise o podatcima druge tablici.

Na primjeru sustava koji se obrađuje, tablica „Korisnik“ ovisi o tablici „TipKorisnika“ i to kroz stupac „TipKorisnikaId“. Tablica „Bilješka“ ovisi o tablici „Korisnik“ kroz stupac „TrenerId“ koji definira kojem treneru pripada bilješka. Tablica „TrenerIgrac“ ovisi o tablici Korisnik i to s oba stupca: „TrenerId“ i „IgracId“. Tablica „Trening“ također ovisi o tablici „Korisnik“ sa stupcima „TrenerId“ i „IgracId“. Tablica „OcjenaTreninga“ ovisi o tablici trening sa stupcem „TreningId“ koji definira za koji trening je vezana unesena ocjena te ovisi o tablici „KategorijaOcjene“ koja kroz stupac „KategorijaOcjeneId“ definira o kojoj se kategoriji unesene ocjene radi.

# 4. Izrada baze podataka

## 4.1. Općenito o bazama podataka

Baza podataka (engl. *Database*, skraćeno DB) je alat za prikupljanje i organizaciju podataka [2]. U računarstvu je de facto standard za pohranu podataka te se dijeli na različite vrste poput: relacijske, objektno-orijentirane, hijerarhijske, dokumentne, grafičke, NoSQL baze [3]...

Podatci se u bazu podataka mogu dodavati, ažurirati, brisati i čitati (engl. *Create, Read, Update, Delete,* skraćeno CRUD) koristeći standardizirane metode za pristup i manipulaciju istima.

Baza podataka koju koristi sustav kojim se rad bavi je relacijska baza podataka. To je tip baze podatka koja se sastoji od tablica s formalno definiranim stupcima koje mogu biti međusobno povezane različitim relacijama pomoću primarnih i stranih ključeva. Baze podataka osim tablice sadrže i druge elemente poput: procedura, funkcija, pogleda, okidača, indeksa, dijagrama i korisnika koji pristupaju bazi.

## 4.2. Izrada tablica baze po modelu podatka

S obzirom na to da je u prethodnom poglavlju definiran model podataka, on služi kao uputa pri izradi tablica koje se nalaze u bazi podataka. Samim time, fokus ovog poglavlja stavlja se na sintaksu izrade baze podataka, ograničenja na tablicama i njihovim stupcima te procedurama kojima se izvršava CRUD nad podatcima.

Prethodno definirani model preslikava se u tablice koje baza podataka sadrži. Tako baza sadrži sljedeće tablice:

- Korisnik
- TipKorisnika
- TrenerIgrac
- Biljeska
- Trening
- OcjenaTreninga
- KategorijaOcjene

Da bi se tablice napravile, potrebno je komunicirati s bazom podataka. Baza podataka koristi "*Data definition language* - DDL" jezik za stvaranje objekta u bazi te rad s podatcima. Različiti prodavači (engl. *vendors*) baza podataka imaju različitu sintaksu DDL-a kojeg koristi njihova baza. Razlike u sintaksi između različitih baza su najčešće u nijansama, no da bi se baza koristila potrebno je poznavati upravo nijanse sintakse te baze.

Sustav koji prati ovaj rad koristi MySQL bazu podataka i sintaksa izrade tablica prikazana je u kodu (Kôd 4.1) na primjeru tablice Korisnik:

```
CREATE TABLE Korisnik
(
    Id INT NOT NULL AUTO_INCREMENT,
    Email NVARCHAR(130) NOT NULL,
    Prezime VARCHAR(130) NOT NULL,
    Ime VARCHAR(130) NOT NULL,
    DatumRegistracije DATETIME NOT NULL,
    TipKorisnikaId INT NOT NULL,
    Hash VARCHAR(255) NOT NULL,
    Izbrisan BOOLEAN NOT NULL,
    PRIMARY KEY (Id),
    FOREIGN KEY (TipKorisnikaId) REFERENCES TipKorisnika(Id),
    UNIQUE UQ_EMAIL_TIP (Email, TipKorisnikaId)
);
```

Kôd 4.1. DDL za izradu tablice Korisnik

Prilikom izrade tablica definiraju se njezini stupci i tipovi podataka te ograničenja na tablici i stupcima koja osiguravaju da baza podataka bude u konzistentnom stanju.

Na primjeru tablice „Korisnik" definirani su sljedeći stupci:

- ID – jedinstveni identifikator korisnika koji je obavezno polje te se sa svojstvom `AUTO_INCREMENT` osigurava da se ID inkrementalno povećava sa svakim novim retkom u tablici
- Email – tekst koji je obavezno polje i označava email s kojim se korisnik prijavljuje u sustav
- Prezime – tekst koji je obavezno polje i označava osobni podatak korisnika
- Ime – tekst koji je obavezno polje i označava osobni podatak korisnika

- DatumRegistracije – obavezno polje koje označava datum dodavanja računa u sustav
- TipKorisnikaId – obavezno polje koje razdvaja račun trenera od računa igrača
- Hash – obavezno polje koje sadrži *hashiranu* lozinku korisnika
- Izbrisan – obavezno polje koje definira jeli korisnik aktivan ili izbrisan

Na tablici „Korisnik“ je također definirano ograničenje `UNIQUE(Email, TipKorisnikaId)` koje omogućuje da se korisnik s istim Emailom može pojaviti samo jednom kao trener i samo jednom kao igrač.

Sukladno izradi tablice „Korisnik“, u bazu podataka su unesene i ostale tablice koristeći istu metodu.

## 4.3. Unos inicijalnih podataka u bazu

Nakon stvaranja baze podataka te definiranja tablica, potrebno je unijeti inicijalne podatke u bazu bez kojih sustav ne može funkcionirati.

S obzirom na arhitekturu baze podataka, jedini inicijalni podatci koji su potrebni za rad sustava su podatci o tipu korisnika. Ti podatci se mogu jednostavno unijeti koristeći *INSERT* naredbu. Podatci o tipu trenera se unose na sljedeći način:

```
INSERT INTO TipKorisnika (Naziv) VALUES
      ('Trener'),('Igrac');
```

Izvršavanjem *INSERT* naredbe, tablica je popunjena s dva tipa korisnika što se može provjeriti izvršavanjem *SELECT* naredbe:

```
SELECT * FROM tipkorisnika;
```

Koja nakon izvršavanja vraća dva retka podataka koji su prikazani u tablici (Tablica 4.1).

Tablica 4.1. Rezultat SELECT naredbe nakon dodavanja tipova korisnika

| **ID** | **Naziv** |
|---|---|
| 1 | Trener |
| 2 | Igrač |

## 4.4. Izrada pohranjenih procedura na bazi podataka

Unosom inicijalnih podataka, baza podataka je spremna za obavljanje CRUD operacija. Podatci se dohvaćaju koristeći *SELECT* naredbe, umeću koristeći *INSERT* naredbe, mijenjaju koristeći *UPDATE* naredbe i brišu koristeći *DELETE* naredbe.

Takve naredbe mogu se izvršavati direktno iz programskog koda, no to nije najbolja opcija rukovanja s bazom podataka zato što su upiti na bazu podataka u programskom kodu skloniji pogreškama, sigurnosnim incidentima (npr. *SQL Injection*), često se ne mogu ispravljati greške (engl. *debug*) te se prilikom izmjene upita treba ponovno sagraditi cijela aplikacija (engl. *rebuild*).

Rješenje za gore navedene probleme krije se u pohranjenim procedurama (engl. *stored procedures*). To su funkcije na bazi podataka koje vraćaju već otprije pripremljene tablice ili izmjenjuju podatke, tako da se iz programskog koda samo poziva naziv procedure i ulazni parametri, a na bazi se događa logika dohvata ili izmjene podataka.

Za izradu procedura na bazi također se koristi *DDL* jezik sa sintaksom prikazanom u kodu (Tablica 4.2Kôd 4.2).

```
CREATE PROCEDURE GetTreninziZaIgraca(id INT)
BEGIN
    SELECT * FROM Trening t
    INNER JOIN Korisnik k ON k.id = t.igracid
    WHERE t.IgracId = id AND t.Izbrisan = FALSE
    ORDER BY t.datumTreninga DESC;
END;
```

Kôd 4.2. DDL za izradu procedure za dohvat svih treninga za igrača

Stvaranje procedure započinje ključnim riječima `CREATE PROCEDURE`, te unosom imena procedure i ulaznim parametrima. Na primjeru procedure „GetTreninziZaIgraca“, ulazni parametar je ID igrača čiji se treninzi žele dohvatiti.

Tijelo procedure se započinje s ključnom riječi `BEGIN` i završava s `END`, a unutar tijela se piše što procedura treba raditi. Na ovom primjeru procedura dohvaća sve neizbrisane

treninge čiji `IgracId` odgovara ulaznom parametru `id` te ih poreda po datumu treninga, od najnovijih do najstarijih.

## 4.5. Popis svih pohranjenih procedura

U tablici (Tablica 4.2) su prikazane sve procedure koje se nalaze u bazi podataka (poredane abecedno).

Tablica 4.2. Popis pohranjenih procedura

| | |
|---|---|
| DeleteKorisnik | GetTreneriZaIgraca |
| DeleteTrenerIgrac | GetTrening |
| DeleteTrening | GetTreninziZaIgraca |
| GetBiljeska | GetTreninziZaTrenera |
| GetGeneralniKomentari | LoginKorisnik |
| GetIgraciZaTrenera | PostDovrsenTrening |
| GetKategorickiPrikaz | PostLog |
| GetKategorijaOcjene | PostOcjenaTreninga |
| GetKategorijeOcjena | PostTrenerIgrac |
| GetKategorijeOcjenaZaTrening | PostTrening |
| GetKomentariOcjena | PutBiljeska |
| GetKorisnik | PutKorisnik |
| GetOcjeneZaIgraca | PutTrening |
| GetOcjeneZaTrening | SearchIgraci |
| GetSumarniPrikazZaIgraca | UploadProfilePicture |
| GetSumarniPrikazZaTrenera | |

# 5. Web servis

## 5.1. Uvod u web servise

„Usluge weba ili web servisi (engl. *Web services*) su programski sustavi oblikovani tako da podržavaju interoperabilne interakcije putem mreže. Takvi sustavi nisu namijenjeni krajnjim korisnicima (ljudima) već drugim aplikacijama i programima. Namjena im je da omoguće korištenje podataka i operacija drugim programima.“ [4]

Sudionici usluga weba su:

- Pružatelj usluge (engl. *service provider*) – na primjeru aplikacije Deuce to je .NET Core aplikacija
- Zahtjevatelj usluge (engl. *service requester*) – na primjeru aplikacije Deuce to su Angular web aplikacija i Android aplikacija

Današnje organizacije se sve više okreću web servisima za izradu svojih web rješenja s obzirom na to da su se aplikacije izrađene u arhitekturama poput MVC-a pokazale kao ne skalabilne te s vremenom postaju teške za održavanje. Korištenje *mikro-servisne arhitekture* pokazalo se kao puno bolje rješenje s obzirom na to da je kod takve arhitekture održavanje jednostavnije, aplikacije su skalabilnije, te su labavo spojene (engl. *loosely coupled*) što omogućuje da svaki dio ili komponenta sustava funkcionira bez obzira na druge komponente.

Također, dolaskom novih tehnologija za izradu web i mobilnih rješenja, poput Angular-a, React-a i Vue-a, koje za dohvat i manipulaciju podataka koriste isključivo web servise, servisi su postali sve učestaliji u IT sustavima.

## 5.2. Zahtjevi i odgovori

Web servisi funkcioniraju na principu slanja zahtjeva (engl. *request*) na server i dobivanje odgovora (engl. *response*) od servera na taj zahtjev. To su paketi informacija koje jedno računalo šalje drugom sa svrhom komunikacije. Klijent najčešće šalje zahtjev prema

serveru, dok server šalje odgovor klijentu. Zahtjev je definiran metodom, a najčešće metode su:

- GET – dohvat podataka sa servera
- POST – slanje podataka na server
- PUT – izmjena podataka na serveru
- DELETE – brisanje podataka sa servera

Osim metode, zahtjev ima i druga svojstva, poput tijela zahtjeva (engl. *request body*) u kojem se nalazi sadržaj koji se šalje na server POST ili PUT metodom, parametara koji se najčešće šalju GET metodom i prisutni su u URL-u zahtjeva, adrese servera koji prima zahtjev te adrese klijenta na koji se treba poslati odgovor.

Odgovor je paket informacija koje server šalje klijentu nakon zaprimljenog zahtjeva. Odgovor se sastoji od tijela i statusnog koda koji klijentu govori je li zahtjev uspješno odrađen. Neki od statusnih kodova koji se pojavljuju u Deuce aplikaciji su:

- 200 OK – zahtjev je uredu, odgovor je ispravan
- 400 Bad request – nešto nije uredu sa zahtjevom
- 401 Unauthorized – zahtjev je dobar ali korisnik nije autoriziran za pristup
- 404 Not found – zahtjev koji klijent traži ne postoji
- 500 Internal server error – dogodila se pogreška na serveru prilikom obrade zahtjeva

Tijelo zahtjeva i odgovora je u JSON (*JavaScript object notation*) formatu. To je lagani (engl. *lightweight*) format namijenjen komunikaciji između računala je te ga ljudi lako mogu pisati i čitati. U kodu (Kôd 5.1) je prikazan primjer modela „Korisnik“ u JSON obliku.

```
{
   "id":20,
   "email":"ivan.ivic@gmail.com",
   "prezime":"Ivan",
   "ime":"Ivić",
   "datumRegistracije":"2019-10-27T21:33:09",
   "tipKorisnikaId":2,
   "izbrisan":false
}
```

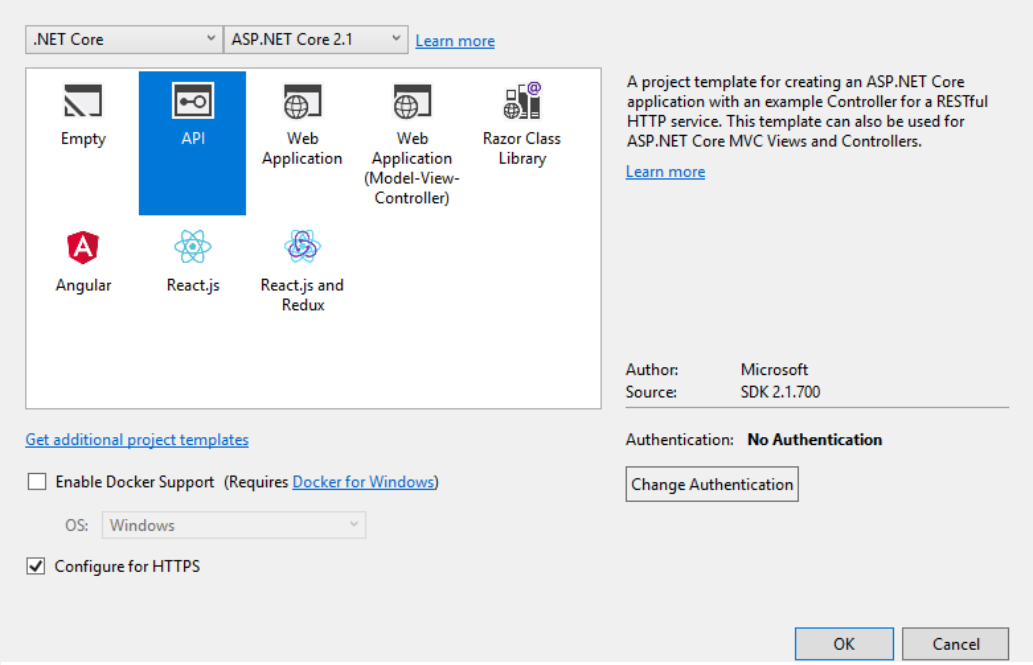
Kôd 5.1. Model Korisnik u JSON obliku

## 5.3. Tehnologija izrade web servisa

Prvi korak u izradi web servisa je odabir tehnologije u kojoj će se web servis implementirati. Na tržištu postoje mnoge tehnologije za implementaciju web servisa [5], a neki od najpoznatijih su: Node.js, Ruby, Python, .NET Core Web API itd.

Tehnologija koja se koristi za izradu ovog rada je **.NET Core Web API**. To je pozadinska ili bekend (engl. *backend*) tehnologija razvijena od strane Microsoft-a koja se može izvršavati na bilo kojem operacijskom sustavu (engl. *cross-platform*), a ne samo Microsoft Windows-u. Primarna zadaća .Net Core Web API-a je da obavlja *backend* logiku, da komunicira s bazom podataka, te da pruža uslugu (podatke) klijentima koji zahtijevaju te usluge. Web API koristi C# programski jezik za svoju implementaciju.

Preduvjet za izradu projekta je instalacija .NET Core programskog paketa, koji za Windows korisnike dolazi s instaliranom *Visual Studio* aplikacijom te se projekt izrađuje kroz grafičko sučelje kako je prikazano na slici ().




Slika 5.1. Izrada .NET Core Web API-ja kroz grafičko sučelje

Kod Linux korisnika se .NET Core preuzima kao samostalni programski paket te se projekt izrađuje kroz komandnu liniju i to tako da komanda započinje s ključnim riječima `dotnet new`, nakon kojih slijede parametri koji definiraju tip projekta, u ovom slučaju

`webapi`, te naziv projekta koji se stvara definiran s parametrom `-n`. Primjer naredbe za stvaranje Webapi projekta s nazivom Deuce:

```
$ dotnet new webapi -n Deuce
```

Prilikom izvršavanja navedene naredbe u komandnoj liniji, stvorit će se novi projekt sa svim potrebnim dokumentima i referencama.

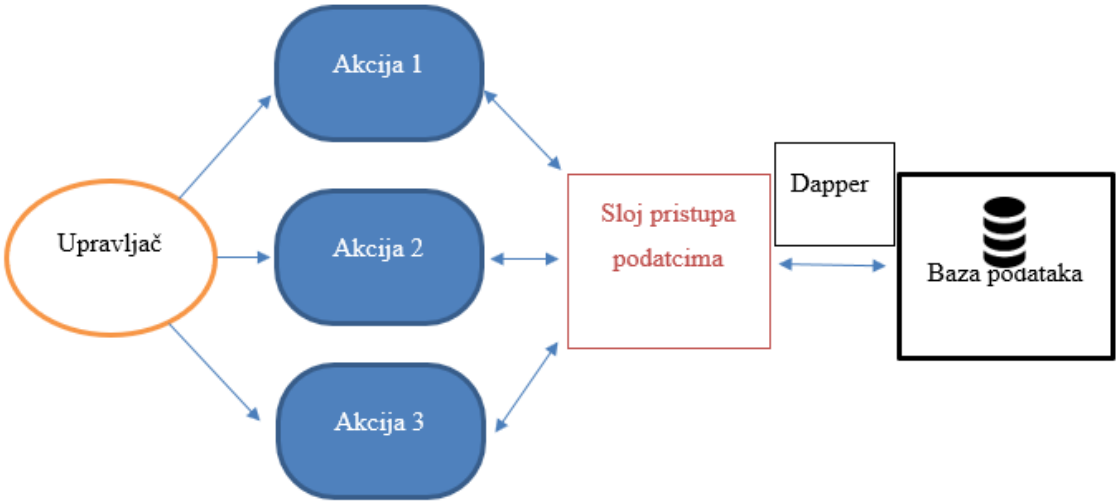
## 5.4. Arhitektura web servisa

Web servis pruža uslugu kroz upravljače (engl. *controllers*) koji su izloženi kroz različite putanje. Upisivanjem ispravne putanje, poziva se određena akcija (metoda) na upravljaču koja odrađuje poslovnu logiku zahtjeva i vraća rezultate.

Poslovna logika se obavlja kroz određena poslovna pravila definirana za svaki poslovni slučaj. Poslovna logika se najčešće stavlja u odvojeni sloj, sloj poslovne logike (engl. *Business logic layer,* skraćeno BLL) koji se poziva iz upravljača, no s obzirom na to da sustav koji se izrađuje nema veliki broj kompleksnih poslovnih pravila, sloj poslovne logike se obavlja upravo u upravljaču.

Nakon obrade poslovne logike, zahtjev se prosljeđuje sloju pristupa podatcima (engl. *Database access layer,* skraćeno DAL). Sloj pristupa podatcima komunicira s bazom podataka te nad njom vrši CRUD operacije koristeći upite ili pohranjene procedure. Za komunikaciju s bazom podataka postoje različite tehnologije poput: ADO.NET, DataSet, Entity Framework, te tehnologija koju ovaj sustav koristi - *Dapper*. To je programski okvir koji mapira tablice iz baze podataka u objekte u C# programskom jeziku.

Na slici (**Error! Reference source not found.**) je prikazan dijagram koji ilustrira kako se zahtjev kreće od upisivanja ispravnog URL-a i aktivacije upravljača, preko upravljačkih akcijskih metoda, prema sloju pristupa podatcima koji podatke dohvaća ili zapisuje u bazu, a zatim se zahtjev vraća prema upravljaču koji klijentu vraća odgovor.




Slika 5.2. Dijagram toka zahtjeva

## 5.5. Izrada modela servisa

Modeli servisa su C# klase koje se preslikavaju na tablice koje se nalaze u bazi podataka. Svaki model podijeljen je na tri regije: konstruktori (engl. *constructors*) koji postavljaju objekt u ispravno stanje, svojstva (engl. *properties*) koja definiraju polja koja se nalaze na modelu i odgovaraju stupcima u tablicama te *overrides* koji izmjenjuju zadana svojstva klase poput ispisa objekta `toString()` i jednakosti `equals()`. U kodu (Kôd 5.2) je prikazan model „Bilješka“.

```
public class Biljeska
{
    #region Constructors
    public Biljeska()
    {
    }

    public Biljeska(int id, int trenerId, String tekst)
    {
        Id = id;
        TrenerId = trenerId;
        Tekst = tekst;
    }
    #endregion
```

```
        #region Properties
        public int Id { get; set; }
        public int TrenerId { get; set; }
        public String Tekst { get; set; }
        #endregion

        #region Overrides
        public override bool Equals(object obj)
        {
            if (obj == null || GetType() != obj.GetType())
            {
                return false;
            }
            return this.Id == (obj as Biljeska).Id;
        }
        #endregion
    }
```

Kôd 5.2. Model Bilješka na servisu u C#-u

.NET Core Web API omogućuje da se bez napora model servisa iz programskog koda pretvara u JSON objekt koji se šalje u odgovoru i obrnuto: da se bez napora JSON objekt koji se šalje u tijelu zahtjeva dohvaća u programskom kodu.

### 5.5.1. Izrada ViewModel-a servisa

Ponekad je u odgovoru potrebno poslati model koji ne odgovara tablicama koje se nalaze u bazi podataka, s obzirom na to da se upitom na bazu mogu vratiti potpuno nove tablice s proizvoljnim stupcima koji se izvode iz postojećih tablica. U takvom slučaju se govori o ViewModel-ima. ViewModeli su klase koje ne odgovaraju tablicama u bazi, već nekoj izvedenoj tablici iz baze.

## 5.6. Izrada sloja pristupa podatcima

Sloj pristupa podatcima je sloj koji komunicira s bazom podataka. Sastoji se od različitih metoda koje pozivaju određene pohranjene procedure kojima prosljeđuju parametre potrebne za njihovo izvršavanje.

Za pristup podatcima ovaj sustav koristi objektno-relacijsko mapiranje, skraćeno ORM. To znači da se tablice, koje se dobivaju upitima, automatski pretvaraju u C# objekte. ORM klijent koji se koristi u ovom sustavu je „Dapper“.

Klasa koja se koristi za pristup podatcima je klasa `Repo` koja implementira sučelje `IRepo`. U sučelju su definirane sve metode koje se trebaju moći pozvati iz *backend*-a dok je klasa `Repo` trenutno jedina implementacija tog sučelja. Ovakva izrada sloja pristupa podatcima omogućuje laku promjenu načina na koji se dohvaćaju podatci, ako to u budućnosti bude potrebno.

Klasa `Repo` na sebi ima svojstvo `IDbConnection` koje predstavlja otvorenu konekciju na *data source*. To je sučelje čija je implementacija klasa `MySqlConnection` i koja predstavlja otvorenu konekciju prema MySQL bazi podataka.

Prilikom stvaranja klase `MySqlConnection` potrebno joj je proslijediti konekcijski string (engl. *connection string)* do baze podataka. Konekcijski string nije dobro stavljati izravno u kod već se isti stavlja u konfiguracijski dokument *appsettings.json* u kojem se čuvaju svi osjetljivi podatci aplikacije. U kodu (Kôd 5.3) se nalazi primjer konekcijskog string-a u konfiguracijskoj datoteci.

```
"ConnectionStrings": {
   "MySQLConnectionString":
   "Server=localhost;Database=deuceDB;Uid=root;Pwd=pwd9423;"},
```

Kôd 5.3. Konekcijski string u JSON obliku

### 5.6.1. Asinkroni dohvat podataka

Metode koje se nalaze u DAL-u su asinkrone. To znači da se prilikom pozivanja neke od metoda stvara nova dretva u kojoj se metoda izvršava. Tako program ne treba blokirano čekati dok baza podataka ne vrati rezultate, već može odrađivati druge operacije dok ga metoda ne obavijesti da su rezultati baze stigli. Primjer metode dohvaćanja treninga:

```
public async Task<Trening> GetTrening(int id)
{
    using (IDbConnection conn = Connection)
    {
        var result = await conn.QueryAsync<Trening>("GetTrening",
            new { id }, commandType: CommandType.StoredProcedure);
        return result.FirstOrDefault();
    }
}
```

Kôd 5.4. Metoda dohvaćanja treninga na servisu

Iz navedenog koda (Kôd 5.4) vidljivo je kako je cijela metoda označena ključnom riječi `async` dok je dio u kojoj se dohvaćaju podatci iz baze (dio u kojem se čeka na podatke iz baze) označena s ključnom riječi `await`. Ključna riječ `await` suspendira izvršavanje `async` metode dok se rezultati ne dobiju natrag [6]. U kombinaciji s asinkronim metodama koristi se povratni tip `Task<T>` koji predstavlja povratni objekt u asinkronom načinu rada. Iz Task-a se može dobiti informacija o tome je li posao obavljen i ako je, koji su rezultati [7].

## 5.7. Izrada upravljača i usmjeravanje

Nakon izrađenih modela i sloja pristupa podatcima, potrebno je izraditi upravljače koji će rukovati s dolaznim HTTP zahtjevima te slati odgovarajući odgovor. Po konvenciji, naziv svake upravljačke klase mora završavati s ključnom riječi *Controller*.

Svaki upravljač mora imati pristup sloju pristupa podatcima, sučelju `IRepo`, te konfiguraciji projekta koju predstavlja sučelje `IConfiguration`. Stvaranjem bazne klase `DeuceControllerBase` koja inicijalizira ta dva sučelja i koja implementira sučelje `ControllerBase` rješava se taj problem. Svi sljedeći upravljači trebaju naslijediti klasu `DeuceControllerBase` kako bi imali pristup `IRepo`-u i `IConfiguration`-u.

Atribut `[ApiController]` se stavlja ispred klase upravljača i osigurava da je upravljač dostupan na određenoj URL putanji (čini `[Route]` atribut obveznim), osigurava automatske HTTP 400 odgovore ako zahtjev nije valjan, te omogućuje korištenje `[FromBody]`, `[FromRoute]` i ostalih atributa koji se koriste za dohvaćanje parametara iz zahtjeva [8].

Primjer koda upravljača Korisnik:

```
[Route("api/[controller]")]
[ApiController]
public class KorisnikController : DeuceControllerBase
{
    public KorisnikController(IRepository repo,
        IConfiguration config): base(repo, config) {}

    [HttpGet, Authorize]
    public async Task<ActionResult<Korisnik>> Get()
    {
        try
        {
            return await _repo.GetKorisnik(GetUserId());
        }
        catch (Exception e)
        {
            _repo.PostLog(new Log(e.Message));
            return StatusCode(500, "Internal server error");
        }
    }
  }
```

Kôd 5.5. Kod upravljača Korisnik

U navedenom kodu (Kôd 5.5) prikazan je upravljač s akcijskom metodom `Get()`. Atribut `[Route]` definira putanju preko kojeg se može pristupiti upravljaču. U ovom slučaju to je putanja *„/api/korisnik“*.

Metoda `Get()` ima atribut `[HttpGet]` koji označava s metodu na zahtjevu s kojom se dolazi do akcijske metode na upravljaču. Drugi atribut je `[Authorize]` koji označava da samo autorizirani (prijavljeni) korisnici mogu pristupiti toj akcijskoj metodi. Akcijske

metode su također asinkrone, poput metoda na sloju pristupa podatcima te im je povratni tip `Task<T>`.

Hvatanje pogrešaka se vrši na akcijskoj metodi čije je tijelo u *Try-catch* bloku i ako se dogodi pogreška, sustav je bilježi, a korisniku vraća statusni kod 500 – *Internal server error.*

### 5.7.1. Popis upravljača

Servis može imati više upravljača te je poželjno da se upravljači grupiraju po funkciji koju obavljaju. S obzirom na potrebe sustava, u tablici (Tablica 5.1) definirani su sljedeći upravljači:

Tablica 5.1. Popis upravljača, njihovih ruta i funkcija

| **Ime upravljača** | **Ruta** | **Funkcija** |
|---|---|---|
| KorisnikController | /api/korisnik | Dijeljene akcije za igrače i trenere |
| OcjeneController | /api/ocjene | Rad s ocjenama i analitikom igrača |
| TrenerController | /api/trener | Akcije za trenere |
| TreninziController | /api/treninzi | Dohvat treninga i podataka o treninzima |
| AuthController | /api/treninzi | Autentikacija korisnika |

## 5.8. Autorizacija i sigurnost

### 5.8.1. JWT autorizacija

U prethodnim poglavljima opisano je kako se pristupa resursima na servisu putem URL-ova koji se okidaju na različite akcijske metode. Na primjer: ako trener (čiji je ID 7) želi dohvatiti svoje igrače, dovoljno je da u pretraživač unese adresu „/api/trener/igraci/7“ i servis će mu vratiti sve njegove igrače. No, ako trener čiji je ID 8, ili netko tko uopće nije trener, pokuša unijeti istu adresu „/api/trener/igraci/7“, dohvatit će se igrači koji pripadaju treneru koji ima ID 7 i dogodit će se sigurnosni incident.

Da bi se tome stalo na put, potrebno je implementirati neku vrstu autorizacije koja će prilikom svakog zahtjeva identificirati zahtjevatelja te ovisno o njegovim ovlastima. Jedan od načina za to napraviti je kroz implementaciju JWT autorizacije.

JWT stoji za *JSON Web Token* i predstavlja otvoreni standard koji definira kompaktni i sigurni način za prijenos informacija između računala u obliku JSON objekta. [9]

Prilikom prijave u sustav, provjeravaju se uneseni podatci te se prilikom uspješne prijave u odgovoru šalje JWT token. JWT token sadržava podatke o ID-u prijavljenog korisnika te njegovu rolu. Klijent tada sprema JWT u svoju lokalnu pohranu i prilikom svakog sljedećeg zahtjeva ga šalje u zaglavlju. Kada server zaprimi zahtjev, prvo iščita njegov token iz zaglavlja te ako je ispravan i klijent ima prava na određeni resurs, taj mu se resurs vraća.

Token se prilikom prvog slanja klijentu potpisuje privatnim ključem pošiljatelja i tako se osigurava da se token ne može krivotvoriti.

Iz tokena se sa strane servera vrlo lako mogu dobiti podatci o prijavljenom korisniku stoga prilikom zahtjeva nije potrebno navoditi ID kao što je gore opisano („/api/trener/igraci/7“) već je dovoljno da korisnik unese „/api/trener/igraci“, a server će iz JWT-a izvući o kojem se korisniku radi.

### 5.8.2. *Hashiranje* zaporki

Prilikom registracije korisnika, njegova zaporka se *hashira*. To znači da se u bazu podataka ne upisuje u formatu čitljivog teksta (engl. *plain text*) već kao „probavljen“ (engl. *disgested*) tekst. *Hashiranje* je nepovratan način probavljanja teksta tj. nakon što je tekst *hashiran,* ne može se vratiti u prvotni oblik.

Ovakav način spremanja zaporki omogućava da se u slučaju curenja podatka lozinke ne mogu tako lako dešifrirati i korisnički računi će ostati zaštićeni.

*Hashiranje* koje koristi ovaj sustav je SHA-256. To je *hashiranje* u kojem se ulazni parametar pretvara u probavljeni tekst fiksne duljine od 64 znakova.

# 6. Angular aplikacija

## 6.1. Uvod u Angular

Angular je programski okvir (engl. *framework*) otvorenog koda (engl. *open-source*) razvijen od strane Google-a za izradu jednostraničnih aplikacija. [10]

Preduvjet za izradu Angular projekta je instalacija Angular CLI programskog paketa. Angular CLI je alat komandne linije za stvaranje i građenje Angular aplikacija. Ostali preduvjeti su instalacija Node.js-a (JavaScript *runtime* okruženja) i npm package manager-a. Kako bi se projekt napravio i pokrenuo, potrebno je u komandnoj liniji izvršiti sljedeće naredbe:

```
$ ng new Deuce
$ ng serve
```

Izvršavanjem navedene naredbe stvorit će se novi direktorij sa svim potrebnim dokumentima i referencama za valjani Angular projekt. Nakon toga će se projekt pokrenuti lokalno i moći će mu se pristupiti preko komandne linije.

## 6.2. Komponente

Komponente su osnovni građevni blok Angular aplikacije. Angular Aplikacija se sastoji od stabla komponenata (engl. *component tree*) [11]. Svaka komponenta sadrži:

- HTML datoteku koja predstavlja pogled (engl. *view*)
- Skriptnu datoteku koja predstavlja upravljač
- Datoteku stila komponente

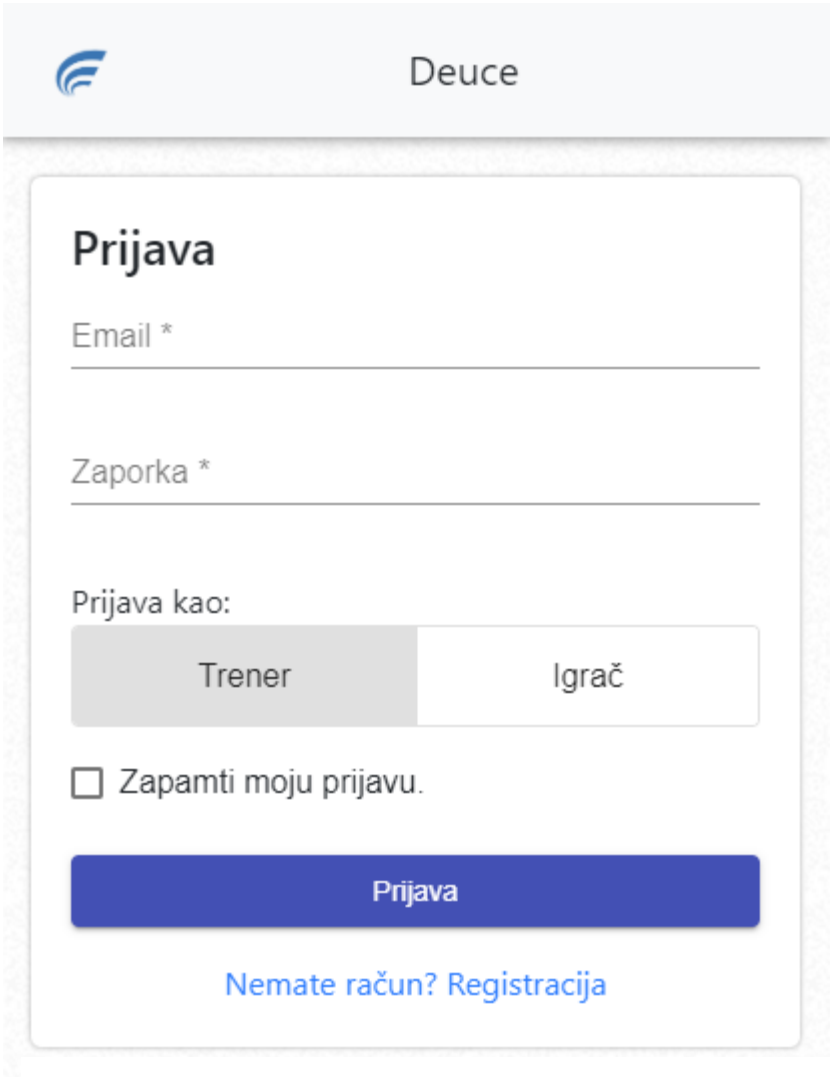
Skriptna datoteka se piše u TypeScript programskom jeziku, programskom jeziku otvorenog koda napravljenom od strane Microsoft-a, koji se prevodi u JavaScript koji je razumljiv pretraživaču.

Datoteka stila se može pisati u običnom CSS-u ili se prilikom izrade može odabrati *preprocessor* stila poput Sass/SCSS, Less ili Stylus-a. Za izradu aplikacije Deuce odabran je Sass, *preprocessor* skriptni jezik koji se prevodi u CSS stilove [12].

### 6.2.1. Popis komponenata Deuce aplikacije

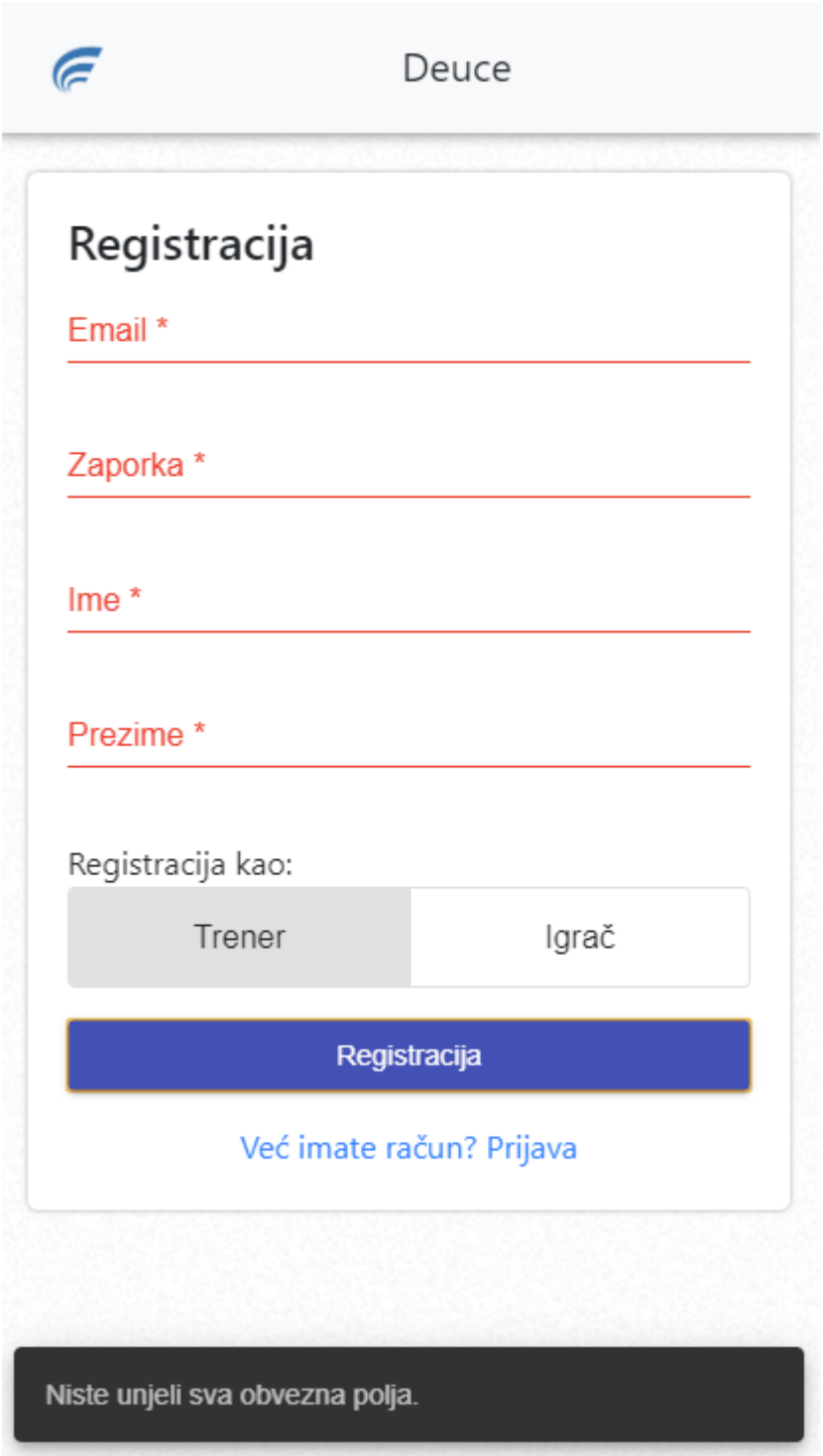
Komponenta prijave u sustav

Korisnik se može prijaviti u sustav kao trener ili kao igrač unosom svojih vjerodajnica (Slika 6.1. Komponenta prijave u sustav).




Slika 6.1. Komponenta prijave u sustav

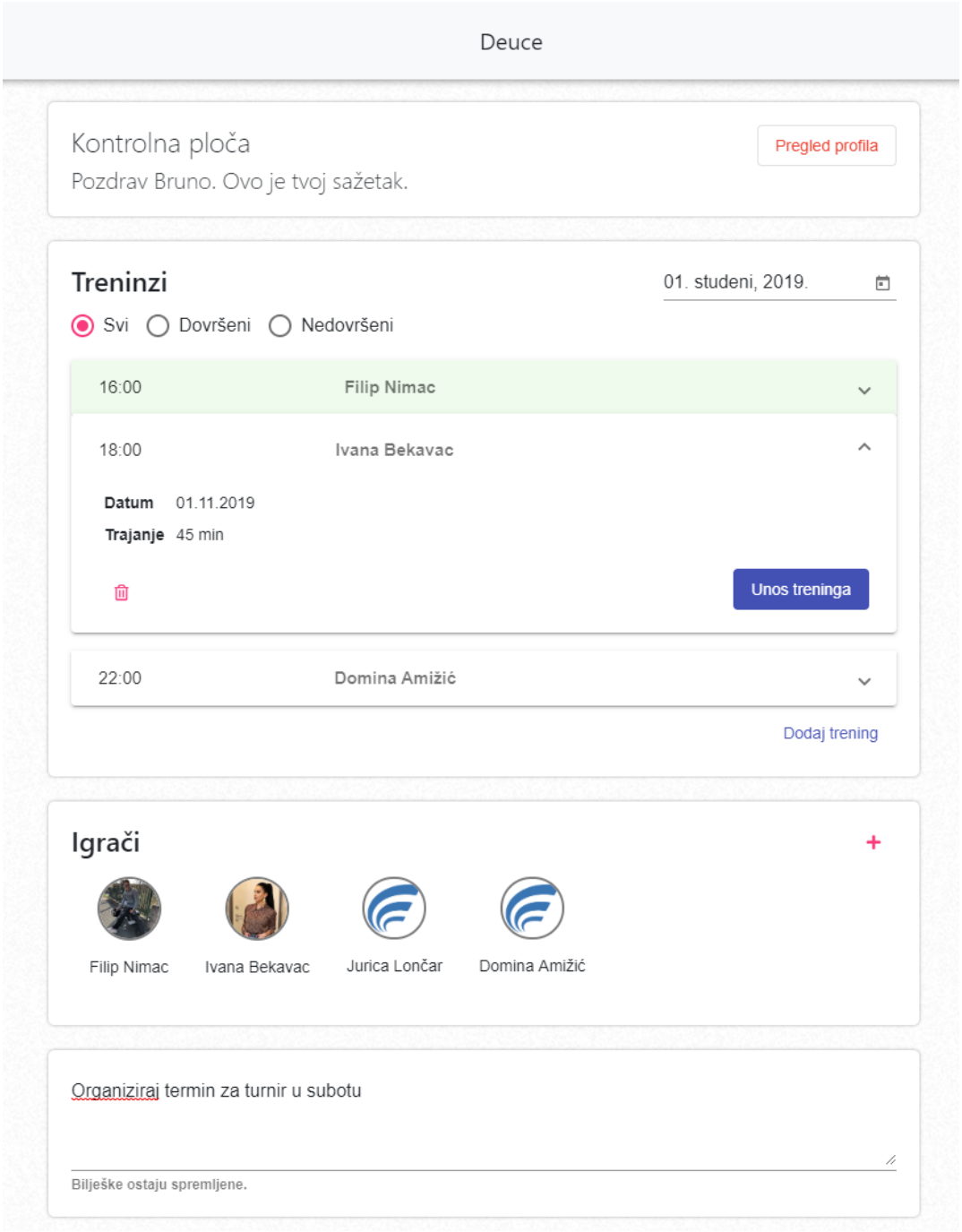
Korisnik može izraditi svoj račun unosom potrebnih podataka. Na slici (**Error! Reference source not found.**) je prikazana validacija forme koja provjerava jesu li sva obvezna polja ispunjena te ako nisu, obavještava korisnika o pogreškama.




Slika 6.2. Komponenta izrade novog računa

### Komponenta kontrolne ploče trenera

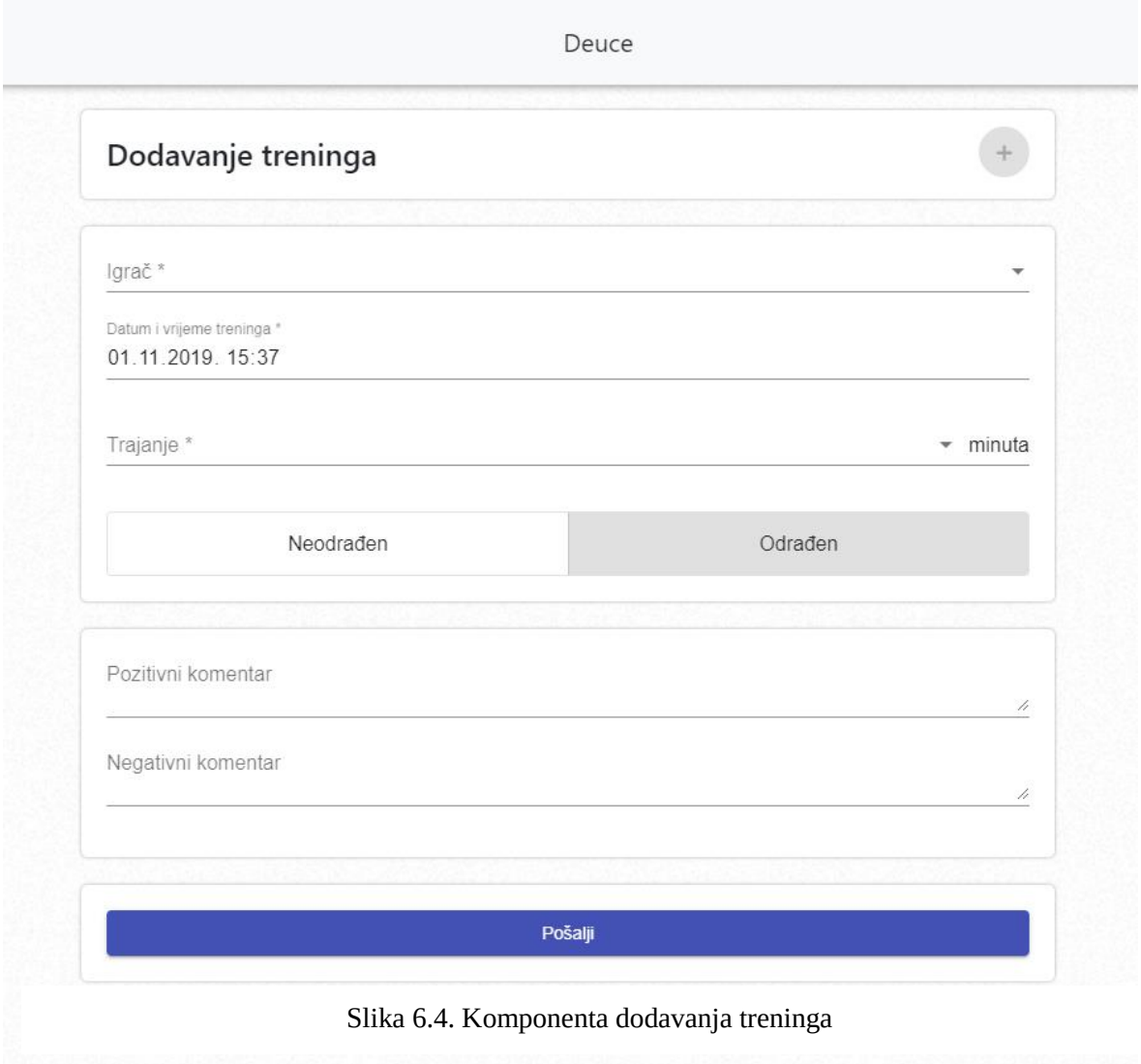
Trenera se prilikom prijave preusmjerava na kontrolnu ploču gdje ima uvid u kalendar treninga koje može pregledavati po datumu te filtrirati po statusu dovršenosti treninga. Treninzi označeni zelenom bojom su odrađeni, a bijeli neodrađeni. Trener na kontrolnoj ploči vidi i sve svoje igrače te može klikom na njih pristupiti njihovom profilu. Na dnu stranice trener ima svoje bilješke koje može izmjenjivati i koje ostaju spremljene (Slika 6.3. Komponenta trenerove kontrolne ploče).




Slika 6.3. Komponenta trenerove kontrolne ploče

Komponenta dodavanja treninga

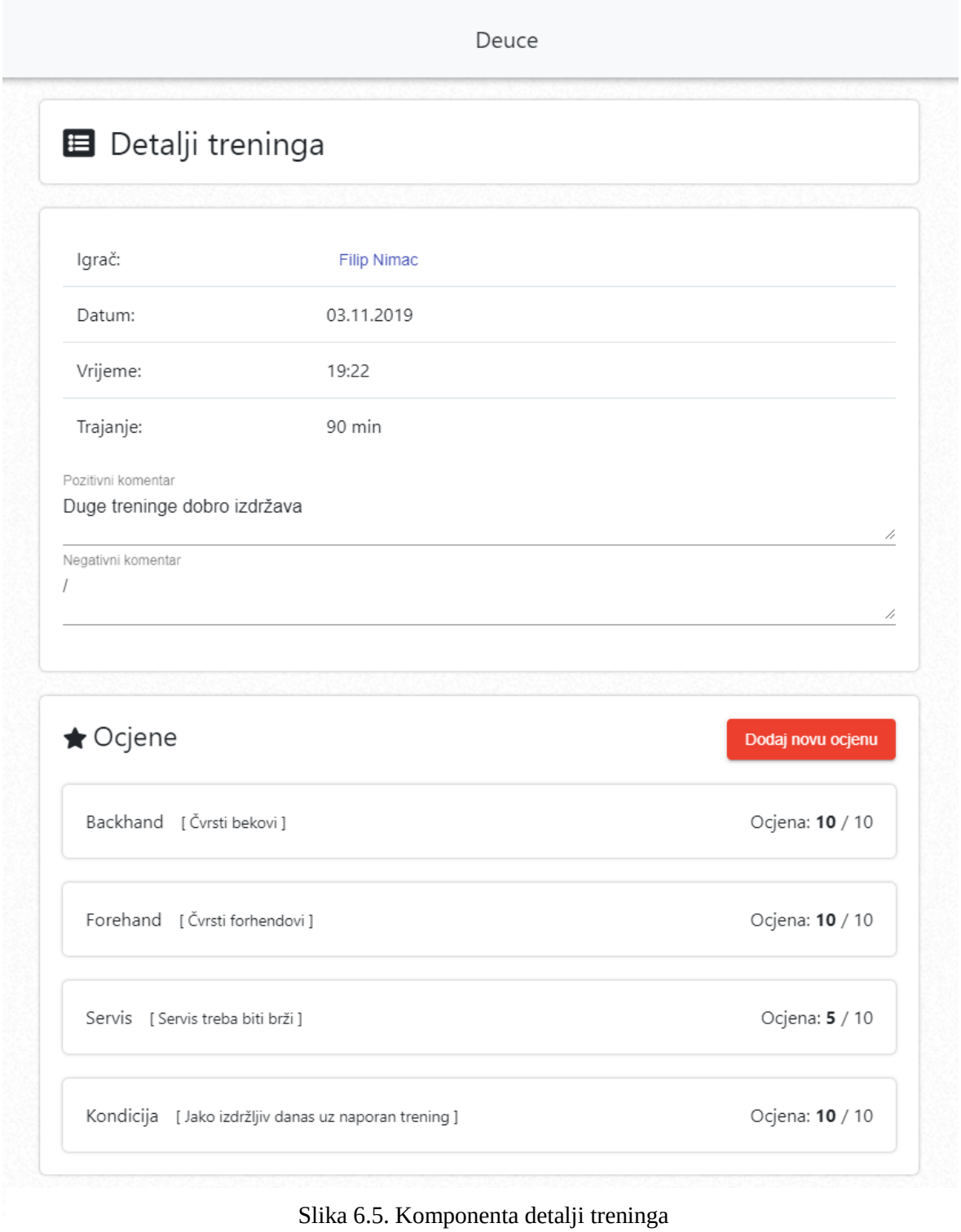
Trener klikom na gumb „Dodaj trening“ može dodati trening za nekog od svojih igrača. Prilikom dodavanja treninga, trener odabire igrača i trajanje treninga iz padajućih izbornika te datum i vrijeme treninga. Trener može dodati već odrađen trening te za njega unijeti pozitivni ili negativni komentar ili neodređen trening za koji može unijeti podatke kasnije (Slika 6.4. Komponenta dodavanja treninga).




Slika 6.4. Komponenta dodavanja treninga

Komponenta detalji treninga

Trener može pregledavati unesene podatke o treningu te unositi ocjene po kategorijama za svaki trening (Slika 6.5. Komponenta detalji treninga).



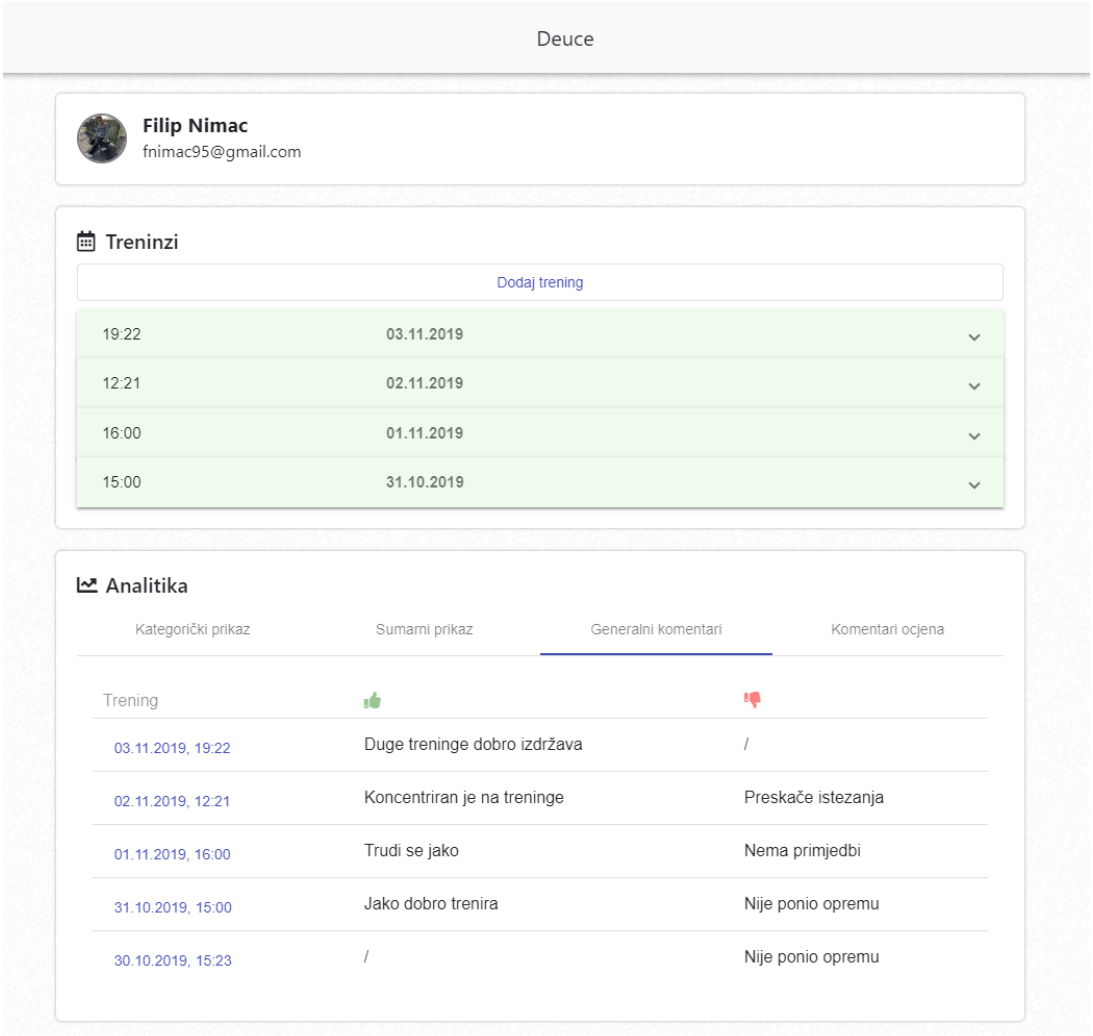

Slika 6.5. Komponenta detalji treninga

### Komponenta pregleda profila igrača

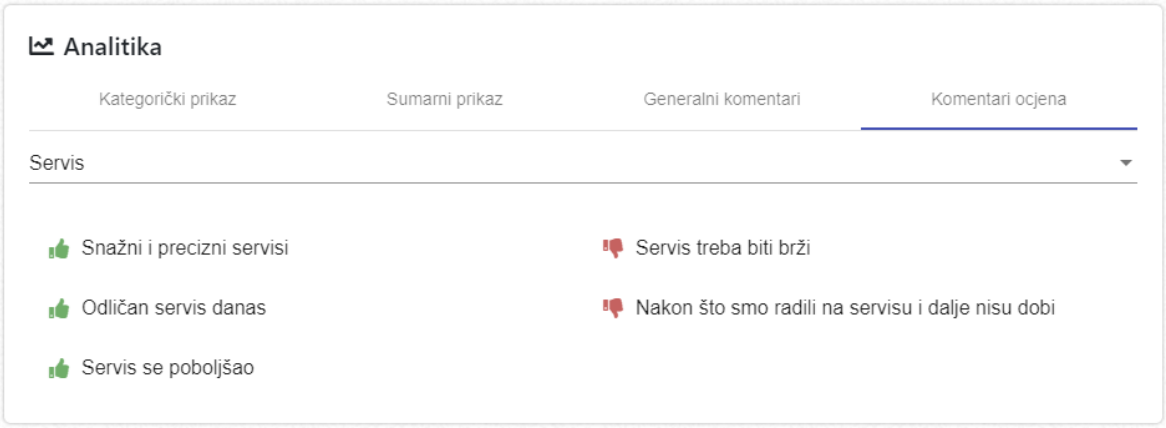
Trener može vidjeti profile svojih igrača, dok igrači vide samo svoj profil te su na njega preusmjereni nakon uspješne prijave. Na ovoj komponenti se nalaze svi treninzi određenog igrača te analitika igrača podijeljena u četiri kartice:

- **Kategorički prikaz**: korisnik odabire određenu kategoriju ocjenjivanja te vidi grafikon koji prikazuje kretanje ocjena za tu kategoriju kroz određeno vrijeme
- **Sumarni prikaz**: radar grafikon koji prikazuje prosjek svih ocjena za svaku od kategorija za koju su unesene ocjene (Slika 6.8. Analitička kartica „Sumarni prikaz“)
- **Generalni komentari**: prikaz komentara u obliku liste za svaki trening za koji su uneseni pozitivni ili negativni komentari (Slika 6.6. Komponenta pregleda profila igrača)

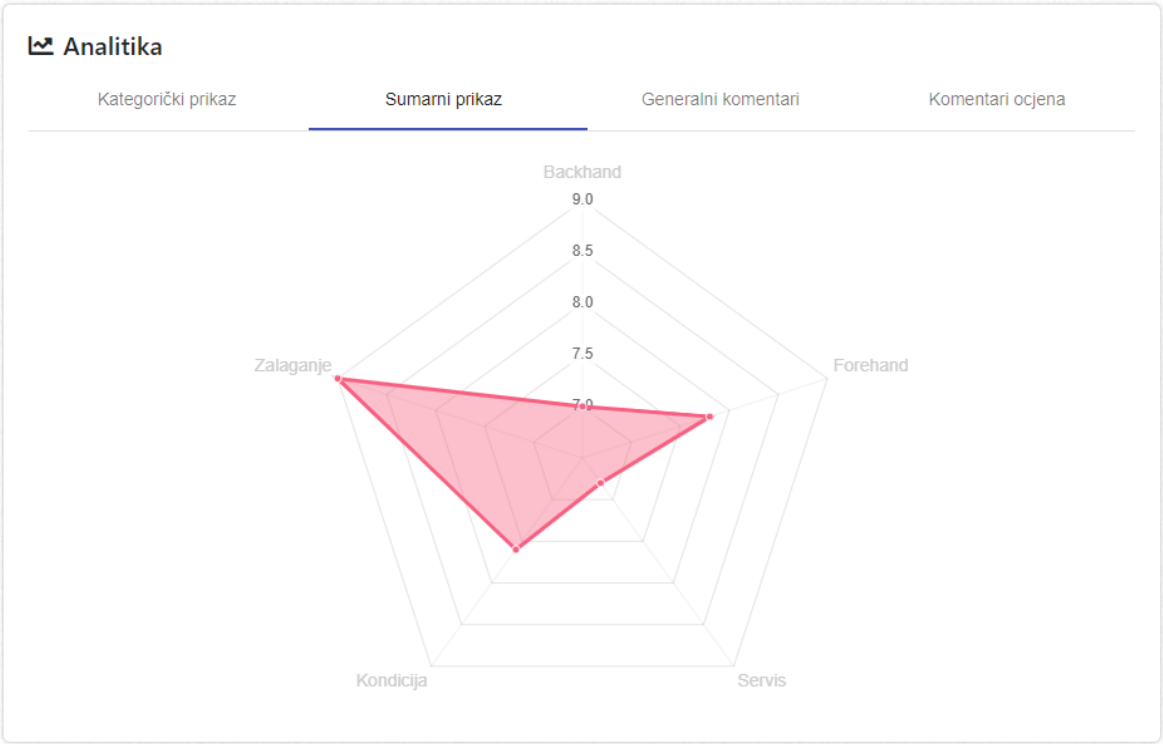
- **Komentari ocjena**: korisnik odabire određenu kategoriju ocjenjivanja te mu se ispisuju uneseni komentari za tu kategoriju (Slika 6.7. Analitička kartica „Komentari ocjena“)




Slika 6.6. Komponenta pregleda profila igrača



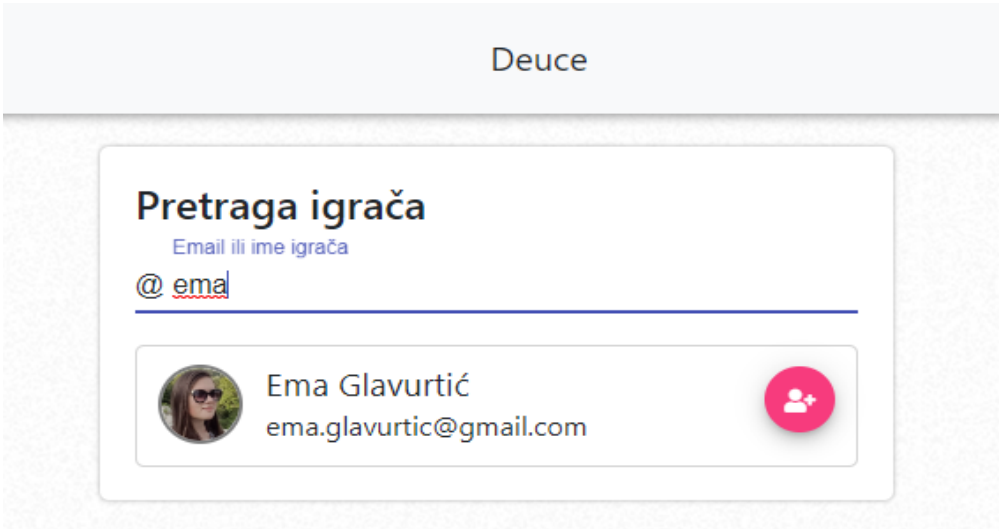

Slika 6.7. Analitička kartica „Komentari ocjena“




Slika 6.8. Analitička kartica „Sumarni prikaz“

### Komponenta pretrage i dodavanja igrača

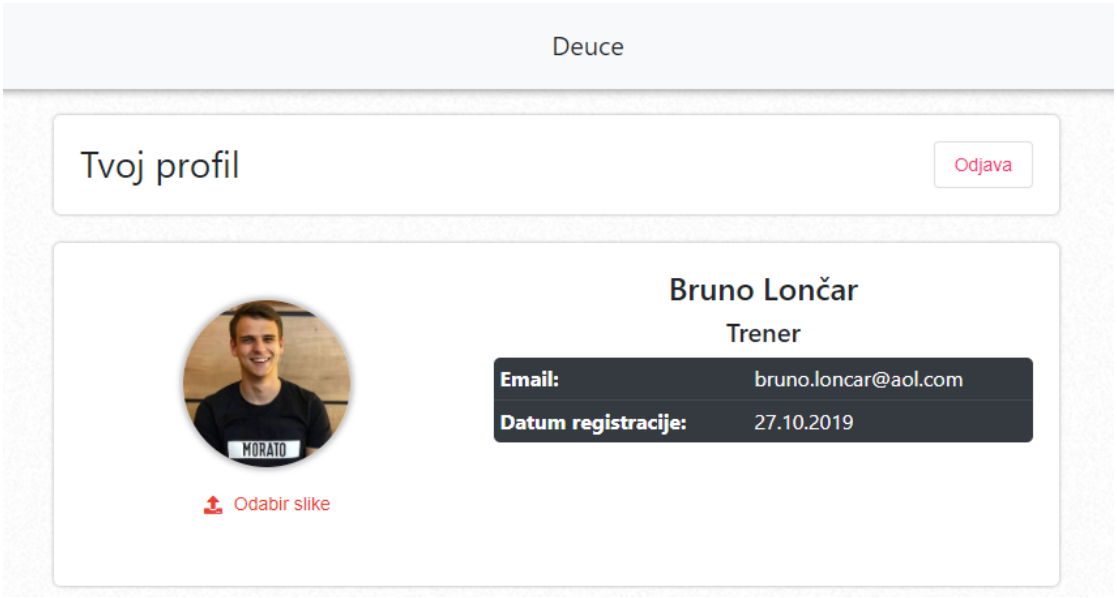
Trener može pretraživati igrače po njihovom imenu, prezimenu i emailu te ih dodavati kao svoje igrače (Slika 6.9).




Slika 6.9. Komponenta pretrage i dodavanja igrača

### Komponenta uređivanja profila

Korisnik može promijeniti sliku profila koja se prikazuje u sustavu te vidjeti svoje osnovne podatke (Slika 6.10).




Slika 6.10. Komponenta uređivanja profila

### 6.2.2. Izgled i dizajn komponenata

U cijeloj Angular aplikaciji korišten je Angular Material. To je biblioteka komponenata namijenjena korištenju u Angular aplikacijama. Angular Material aplikacijama daje moderan izgled koji lijepo izgleda na svim uređajima. Aplikacija je *responsive* tj. može se može pregledavati na monitorima, tabletima te mobilnim uređajima te ovisno o ekranu ima prilagođen raspored i skaliranje.

### 6.2.3. Servisne klase

Komponente su građevni blokovi koji određuju kako će izgledati izgled i raspored aplikacije. Ako se želi komunicirati sa servisom, onda nije dobro takve stvari stavljati u kod od komponenti već u zasebne klase koje se brinu o toj komunikaciji.

Deuce ima četiri servisne klase:

- **DeuceService**: zadužena za prijavu i registraciju korisnika
- **TrenerService**: zadužena za slanje zahtjeva koji se odnose na trenera
- **TreningService**: zadužena za dohvat i dodavanje podataka o treninzima
- **OcjenaService**: zadužena za dodavanje ocjena i pregled analitike igrača

Servisne klase se sastoje od metoda koje pozivaju rute na servisu. Na slučaju aplikacije Deuce, svaka metoda na servisnoj klasi se preslikava na metodu koja se poziva na servisu. Primjer metode `getTrening()` na servisnoj klasi:

```
getTrening(id): Observable<any> {
    let token = localStorage.getItem('jwt');

    return this.http.get<any>(this.url + 'treninzi/' + id, {
      headers: new HttpHeaders({
        "Authorization": "Bearer " + token,
        "Content-Type": "application/json"
      })
```

```
    });
}
```

Kôd 6.1. Metoda dohvaćanja treninga na servisnoj klasi

Iz navedenog koda (Kôd 6.1) je vidljivo da metoda prilikom slanja zahtjeva u zaglavlje uključuje JWT koji je spremljen u lokalnoj pohrani aplikacije, a koji se dobije prilikom prijave u sustav. U metodi je definiran URL, metoda s kojom se šalje na server, te tijelo zahtjeva.

### 6.2.4. Putanje i kontrola pristupa

Putanje aplikacije su definirane u posebnoj datoteci („app-routing.module.ts“) koja se uvozi (engl. *import*) u glavni modul aplikacije. Svaka putanja je povezana s komponentom koju prikazuje.

Na svaku putanju osim putanja „/login“ i „/register“ postavljena je zaštita da samo prijavljeni korisnici mogu pristupiti tim putanjama. Takva zaštita je napravljena pomoću klase `AuthGuard` koja implementira sučelje `CanActivate` i istoimenu metodu čija je svrha dohvat JWT-a iz lokalne pohrane te provjeravanje njegove ispravnosti.

U tablici (Tablica 6.1) su popisane sve putanje aplikacije te komponenta do koje vode. Komponente koje primaju dodatne parametre označene su sa znakom :, npr. :id.

Tablica 6.1. Putanje Angular aplikacije s njihovim komponentama.

| **Putanja** | **Komponenta** |
|---|---|
| /login | Komponenta prijave u sustav |
| /register | Komponenta izrade novog korisničkog računa |
| /dash | Komponenta kontrolne ploče trenera |
| /treninzi/dodaj | Komponenta dodavanja treninga |
| /treninzi/dodaj/:id | Komponenta dodavanja treninga za uneseni nedovršeni trening. |

| | |
|---|---|
| /treninzi/:id | Komponenta detalji treninga |
| /igrac/:id | Komponenta pregleda profila igrača |
| /igrac | Komponenta pregleda profila igrača za prijavljenog igrača |
| /igraci/dodaj | Komponenta pretrage i dodavanja igrača |
| /profil | Komponenta uređivanja profila |

# 7. Android aplikacija

## 7.1. Svrha i ciljevi izrade Android aplikacije

Svrha izrade Android aplikacije je da se pristup sustavu proširi na još jednog klijenta, u ovom slučaju Android aplikaciju, koji se naslanja na postojeću arhitekturu sustava. Android aplikacija može, kao i Angular aplikacija, pozivati metode na servisu i tako komunicirati s ostatkom sustava.

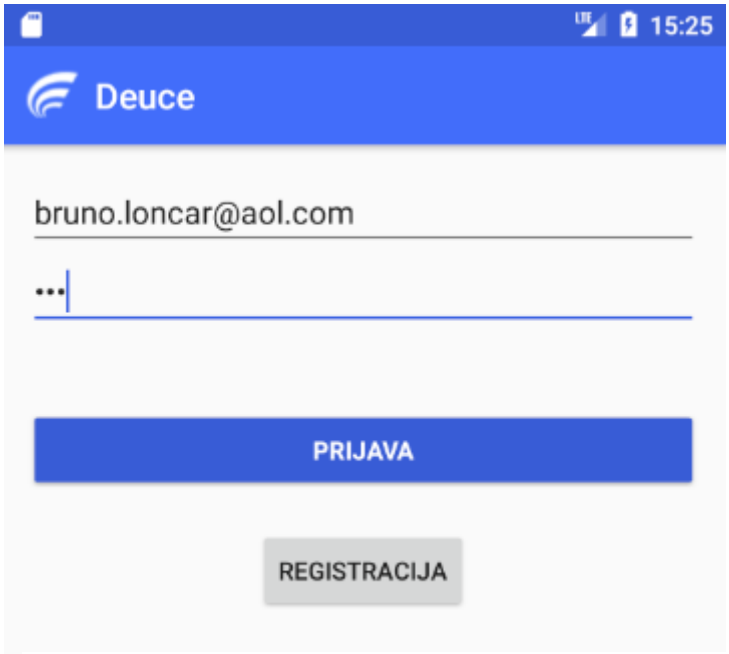
Android kao platforma nudi mnoge mogućnosti *out of the box*, stoga se ovim dijelom rada želi pokazati primjena takvih mogućnosti i kako one mogu doprinijeti ciljevima sustava. Primjer takvih mogućnosti su: izrada rasporeda aplikacije kroz aktivnosti i fragmente, jednostavna promjena jezika aplikacije, stranica s postavkama, nativne navigacije i izbornici, adapteri da prilagođeni prikaz podataka te ugrađeni kalendar.

Preko Android aplikacije i igrači i treneri mogu napraviti račun. U aplikaciju se prijaviti mogu samo treneri te oni imaju osnovni uvid u popis svojih igrača, kalendaru s dovršenim i nedovršenim treninzima te grafikon sumarnog prikaza ocjena za svakog igrača.

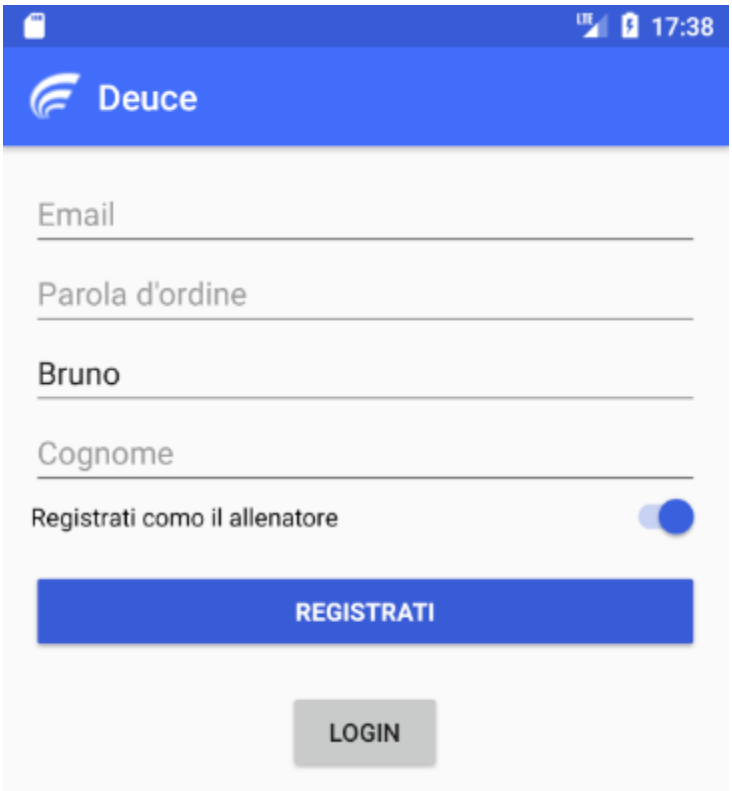
## 7.2. Izgled i ekrani aplikacije

Ekrani prijave i registracije u sustav

Korisnicima se prilikom prvog pokretanja aplikacije otvara ekran za prijavu u sustav (Slika 7.1). Ako korisnik nema račun, može ga napraviti klikom na gumb „Registracija“ (Slika 7.4. Kartica Kalendar u Android aplikaciji).

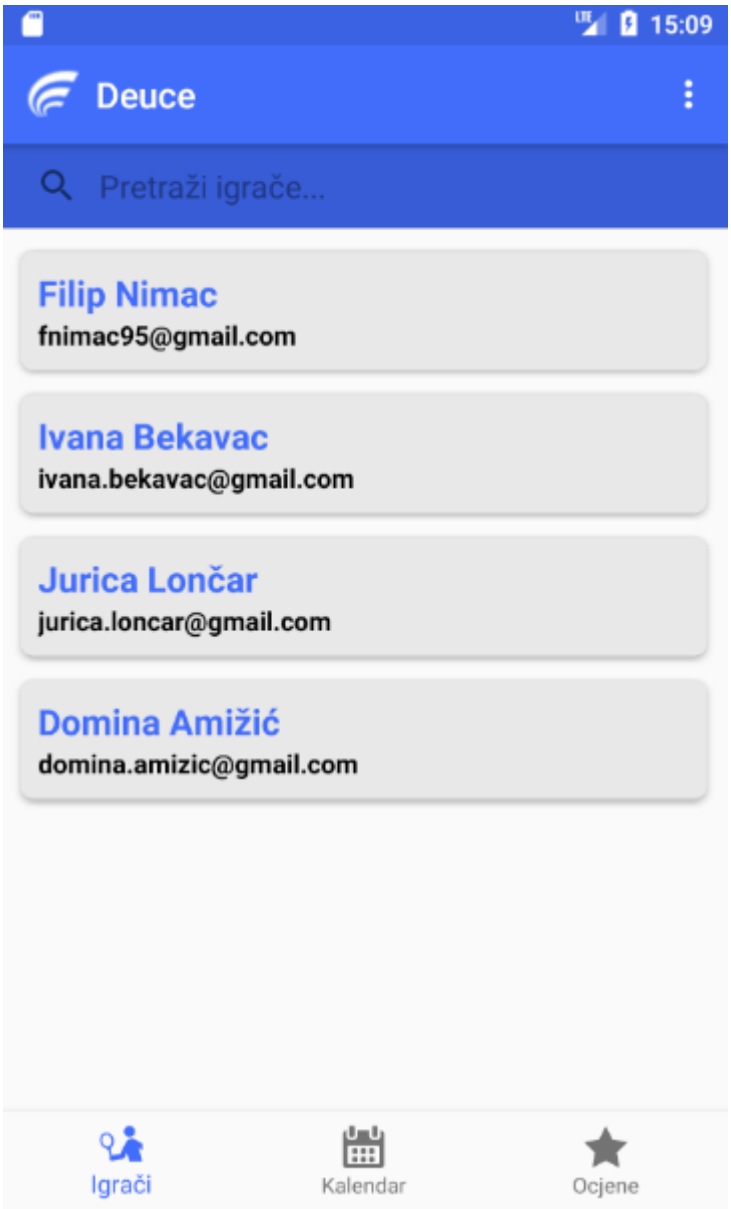



Slika 7.1. Ekran prijave u Android aplikaciju




Slika 7.2. Ekran registracije dok je aplikacija na Talijanskom jeziku

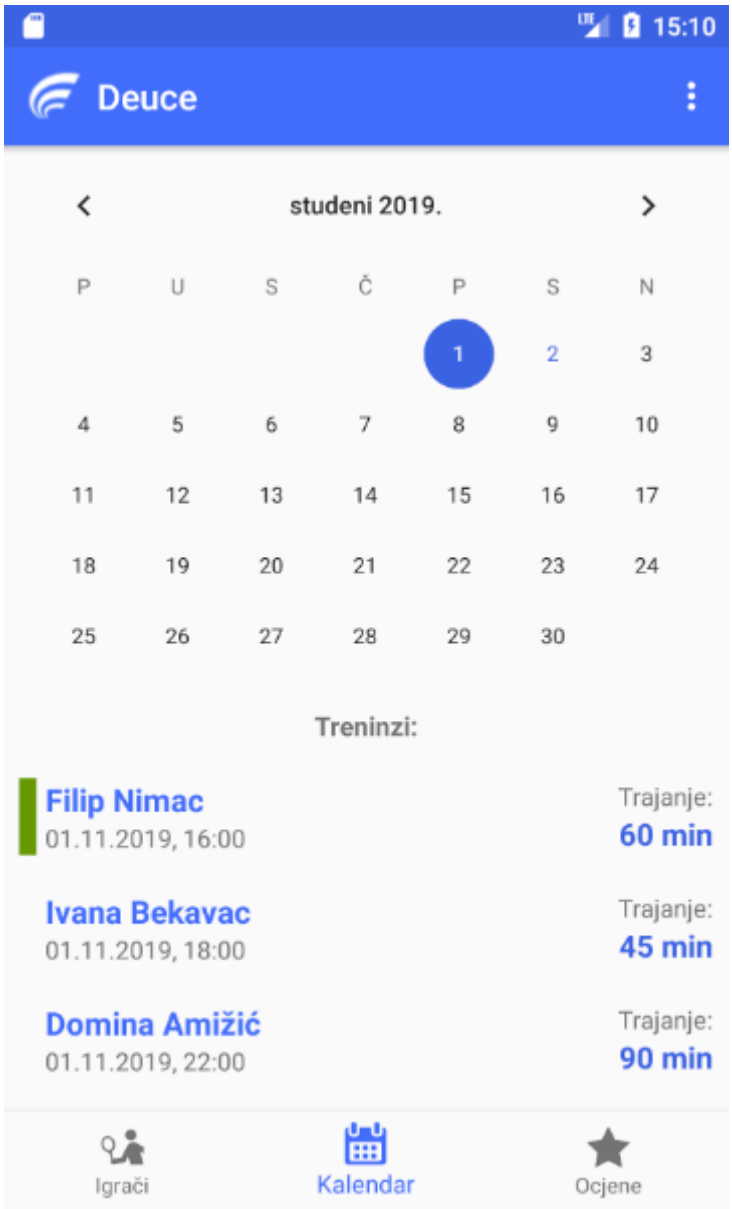
Trenera se prilikom uspješne prijave preusmjerava na glavnu aktivnost aplikacije koja se sastoji od tri kartice (Slika 7.3. Kartica igrači u Android aplikaciji). Kartica koja se prikaže s prijavom je kartica „Igrači“. U njoj trener vidi popis svojih igrača s njihovim osnovnim podatcima. Trener može pretraživati sve svoje igrače koristeći okvir za pretragu na vrhu kartice. Klikom na igrača, trenera se  preusmjerava na karticu ocjene s pregledom ocjena za odabranog korisnika.




Slika 7.3. Kartica igrači u Android aplikaciji

Kartica Kalendar

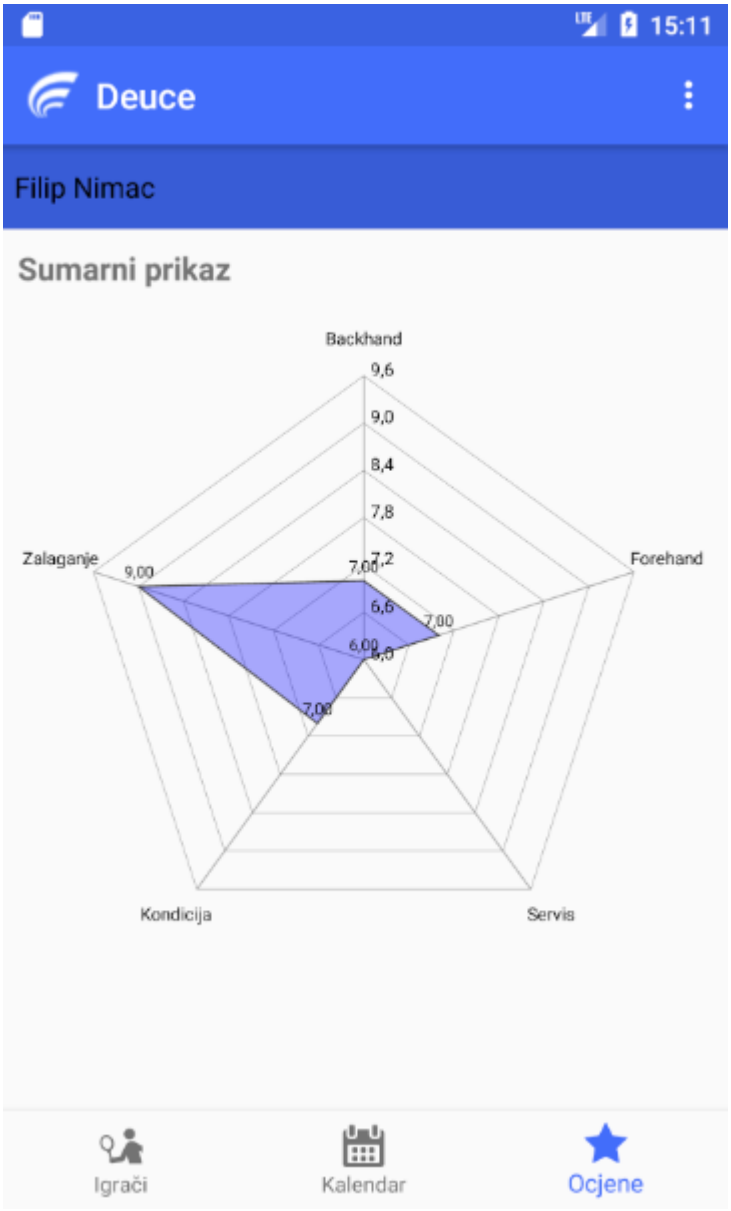
Odabirom kartice Kalendar, treneru se otvara kalendar s odabranim današnjim datumom (Slika 7.4. Kartica Kalendar u Android aplikaciji). Ispod kalendara se prikazuju svi treninzi za današnji datum te informacije o njima. Odrađeni treninzi imaju zelenu oznaku s lijeve strane dok neodrađeni nemaju. Trener može odabrati bilo koji datum te će mu se osvježiti lista treninga za odabrani datum.




Slika 7.4. Kartica Kalendar u Android aplikaciji

Kartica ocjene

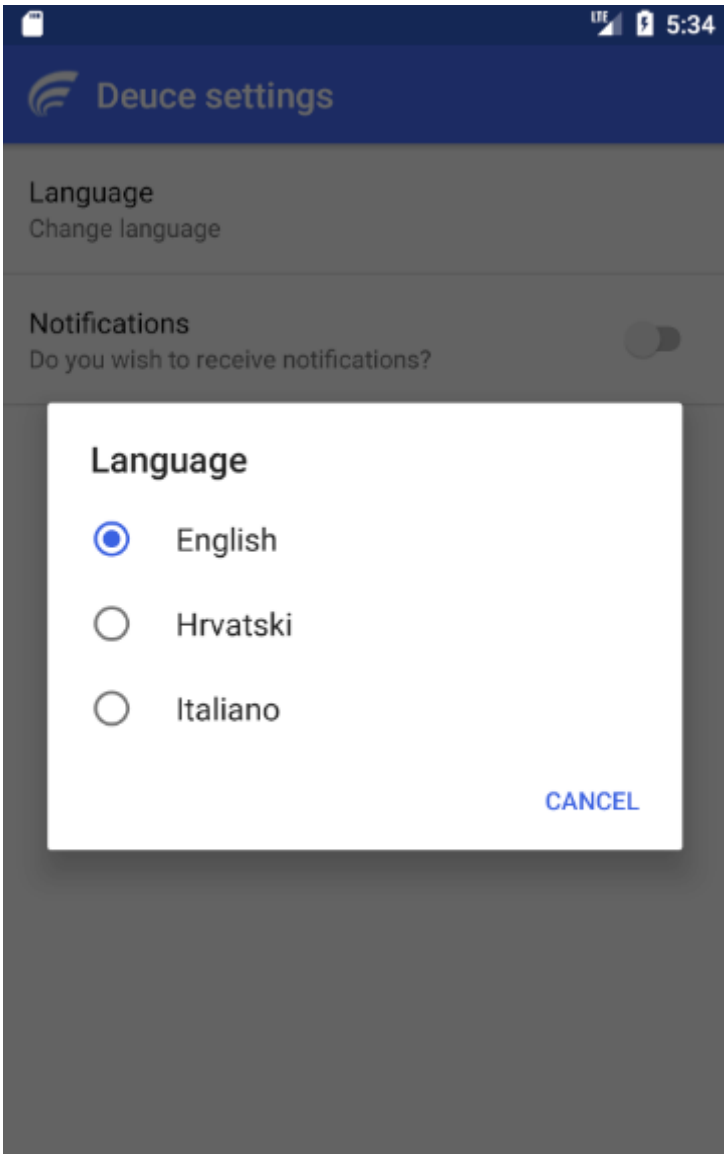
Odabirom kartice ocjene, treneru se prikazuje padajući izbornik s njegovim igračima. Odabirom igrača iz padajućeg izbornika, iscrtava se radar grafikon sa sumarnim prikazom svih ocjena (Slika 7.5. Kartica Ocjene u Android aplikaciji).




Slika 7.5. Kartica Ocjene u Android aplikaciji

### Aktivnost Postavke

Trener može iz izbornika koji se nalazi u gornjem desnom kutu odabrati opciju „Postavke“. Treneru se prilikom odabira te opcije otvara nova aktivnost koja ima nativni izgled postavki svih Android aplikacija (Slika 7.6). Takva aktivnost napravljena je koristeći `PreferenceFragment`, ugrađene klase koja napuhava (engl. *inflate*) aktivnost postavki. U postavkama korisnik može promijeniti jezik aplikacije.




Slika 7.6. Aktivnost postavke dok je aplikacija na Engleskom jeziku

## 7.3. Korištene mogućnosti i tehnologije prilikom izrade

### 7.3.1. Aktivnosti i fragmenti

Ekrani Android aplikacije su aktivnosti koje se sastoje od programskog koda u Javi koji obrađuje logiku aktivnosti i rasporeda (engl. *layout*) u XML formatu [13]. Svaka aktivnost nasljeđuje klasu `AppCompatActivity` koja definira klasu kao aktivnost te pruža mogućnost prepisa (engl. *override*) metoda životnog ciklusa aktivnosti.

Aktivnost na sebi može imati manje dijelove zvane Fragmenti koji se iscrtavaju unutar aktivnosti. Glavna aktivnost aplikacije zove se `TabbedActivity`, koja ovisno o odabranoj kartici na dnu aplikacije prikazuje određeni fragment.

### 7.3.2. Internacionalizacija

Internacionalizacija (engl. *Internationalization*, skraćeno I18N) je postupak prilagodbe aplikacije za podršku višejezičnosti. Android platforma ima ugrađeni mehanizam koji postavlja jezik aplikacije ovisno o postavljenom jeziku na uređaju. Kako bi se aplikacija mogla prevesti, potrebno je sve tekstne resurse izvući u resursnu datoteku.

Resursna datoteka je dokument u XML formatu u kojoj su nabrojani svi *stringovi* aplikacije sa svojim identifikatorom. Prilikom izrade višejezičnosti, taj se dokument duplicira i stringovi se prevode na željeni jezik. Kako bi Android znao o kojem se jeziku radi, potrebno je dodati ekstenziju na kraj imena datoteke ovisno o jeziku, npr. „.hr“ za prijevod na hrvatski.

Aplikacija Deuce je dostupna na hrvatskom, engleskom i talijanskom jeziku, a jezik se može promijeniti kroz postavke aplikacije ili promjenom jezika cijelog uređaja.

### 7.3.3. Retrofit

Retrofit je REST klijent za Javu i Android [14]. Omogućuje da se na jednostavan način šalju zahtjevi na servis te da se dohvaća odgovor od servisa u JSON obliku. Retrofit zahtjeva da se izradi sučelje s popisom svih metoda koje se mogu pozvati na servisu. Deklarirane metode se moraju naznačiti (engl. *annotate*) kako bi Retrofit znao o kojoj se metodi slanja radi te koja je putanja.

```
    @GET("trener/igraci")
    Call<List<Korisnik>> getIgraci(@Header("Authorization") String
token);
```

U navedenom kodu je vidljivo kako se za dohvaćanje treninga za trenera koristi GET metoda na putanju („trener/igraci“) s anotacijom `@GET("trener/igraci")`. U tijelo zahtjeva se stavlja JWT token dobiven prilikom prijave u sustav i naznačuje se s anotacijom `@Header("Authorization")`. Potrebno je izraditi i klasu `RetrofitCI` koja ima *singleton pattern* tj. objekt se stvara samo prvi put kad je zatražen te se u memoriji zabilježi njegova referenca, a svakim sljedećim traženjem se vraća taj objekt. U toj klasi se nalaze i postavke Retrofit klijenta poput putanje do servisa te sigurnost pristupa.

### 7.3.4. GSON

GSON je Java biblioteka koja se koristi za pretvaranje Java objekata u JSON oblik i obrnuto. Kako bi se objekt mogao pretvoriti u JSON objekt, potrebno je naznačiti polja koja će se pretvoriti kao na sljedećem primjeru za klasu `LoginPodatci` koja se šalje u tijelu zahtjeva prijave na sustav gdje su polja koja se serijaliziraju u JSON objekt označena s anotacijama `@Expose` i `@SelializedName()`:

```
public class LoginPodatci {
    @Expose @SerializedName("email")
    private String email;

    @Expose @SerializedName("lozinka")
    private String lozinka;

    @Expose @SerializedName("tip")
    private int tip;
}
```

Kôd 7.1. Java klasa s naznačenim poljima za serijalizaciju u JSON objekt

# 8. *Deployment* sustava

Stavljanje sustava u pogon (engl. *deployment*) je skup aktivnosti kojima se softverski sustav postavlja u stanje dostupno za uporabu. S obzirom na različitu prirodu svakog informacijskog sustava, ovaj proces je teško definirati, no najčešće se sastoji od instalacije, konfiguracije, testiranja i posljednjih promjena prije upogonjenja [15].

Sustav se prilikom *deployment*-a postavlja na poslužitelja, od kojih su najpoznatiji Amazon Web Services (AWS) i Azure.

## 8.1. *Deployment* baze podataka

Baza podataka se prilikom *deployment*-a migrira sa svim svojim objektima i podatcima na nekog od poslužitelja. Na poslužitelju se odabire baza podataka koju se želi migrirati te poslužitelj automatski stvara konfiguraciju koja je potrebna za ispravan rad baze podataka. Prilikom *deployment*-a je potrebno stvoriti korisnike koji mogu pristupati bazi podataka te im dodijeliti potrebne uloge za pristup resursima na bazi.

Također, potrebno je definirati i ulazna pravila (engl. *inbound rules*) čija je svrha definirati pravila pristupa bazi podataka. Jedno od ulaznih pravila je s koje adrese se može pristupati bazi podataka. Preporuka je da se takvo pravilo unese i postavi na adresu servisa.

Na primjeru AWS-a, komponenta na koju se stavlja baza podataka zove se „Amazon Relational Database Service“ (Amazon RDS).

## 8.2. *Deployment* servisa

Servis se postavlja na virtualno računalo (engl. *virtual machine*), a operacijski sustav virtualnog računala ovisi o platformi na kojoj se servis može izvršavati. U slučaju ASP.NET aplikacija, to je Windows. S obzirom na to da je servis ovog sustava napravljen u .NET Core tehnologiji, koja je *cross-platform*, to može biti Windows ili Linux.

Prilikom postavljanja servisa na poslužitelja, potrebno je promijeniti konekcijski *string* koji odgovara podatcima migrirane baze podataka kako bi servis mogao komunicirati s tom bazom.

Na primjeru AWS-a, servis se stavlja na „Elastic Beanstalk“ koji se brine o postavljanju konfiguracije servera za ispravan rad servisa.

## 8.3. *Deployment* Angular aplikacije

Angular aplikacija zahtjeva Node.js za posluživanje. AWS kroz „EC2“ komponentu omogućuje *out of the box* prijenos Angular aplikacije. Potrebno je stvoriti instancu „EC2“ komponente te prilikom izrade odabrati da se komponenta konfigurira za rad s Node.js serverom.

U Angular aplikaciji je prije *deploy*-a potrebno izmijeniti putanje do servisa s kojim aplikacija komunicira.

## 8.4. *Deployment* Android aplikacije

Kako bi se Android aplikacija stavila u pogon, potrebno je imati otvoren „Android Developer“ račun. Izrada takvog računa košta 25 dolara. Aplikacija se *deploy*-a putem „Android Development Console“ u koju se preko grafičkog sučelja unose podatci o aplikaciji.

U aplikaciji je potrebno izmijeniti putanju do servisa te je izgraditi (engl. *build*) u APK obliku. Izgrađeni APK se tada dodaje u *Production release* koji se nalazi u Development konzoli. Nakon što instalacijski čarobnjak (engl. *installation wizard*) završi, aplikacija postaje javno dostupna na „Trgovini Play“ [16].

# Zaključak

Životni ciklus svakog informacijskog sustava započinje planiranjem. Prilikom planiranja potrebno je definirati ciljeve i viziju, zahtjeve sustava te tehničku izvedbu. Nakon inicijalnog postavljanja plana, sustav je potrebno modelirati. Modeli sustava se pojavljuju u svim komponentama sustava stoga je važno imati viziju kako će sustav izgledati od početka kako bi se izbjegle promjene koje su kasnije skupe za ispravljanje.

Implementacija sustava započinje izradom baze podataka koja predstavlja okosnicu sustava. Prilikom izrade baze važno je obratiti pozornost na strukturu tablica te ograničenja na tablicama kako bi baza bila u konzistentnom stanju. Preporuka je da se upiti nikad ne vrše izravno iz programskog koda već da se pripreme procedure koje se pozivaju po potrebi.

Zbog problema sa skaliranjem i slabom modularnosti, monolitna se arhitektura pokazala kao loša za izradu kompleksnijih sustava. Upravo zbog toga, sve veći broj organizacija se okreće razvoju servisne arhitekture koju može koristiti više klijenata istovremeno. Prilikom izrade takve arhitekture, potrebno je obratiti pozornost na sigurnost, modularnost i asinkronost. Također, većina modernih klijenata u vrijeme izrade ovog rada koristi REST servise za dohvat i slanje podataka stoga servisna arhitektura postaje standard u izradi informacijskih sustava.

Prilikom izrade klijenata preporuka je da se koriste mogućnosti koje klijentska platforma nudi jer se tako ubrzava proces razvoja i smanjuje mogućnost pojave pogrešaka.

Zahvaljujući modernim *cloud* servisima, aplikacije se na jednostavan način mogu postavljati u pogon bez velikih znanja o servera na kojem se poslužuju.

Imajući u vidu navedene činjenice, možemo zaključiti da se s dobrim planiranjem, slijeđenjem dobrih praksi prilikom izrade informacijskih sustava, te korištenjem specifičnih mogućnosti pojedinih platformi, mogu izraditi kvalitetna i održiva programska rješenja koja donose vrijednost korisnicima.

# Popis kratica

| | | |
|---|---|---|
| API | Application Programming Interface | Sučelje za programiranje aplikacija |
| APK | Android Package | format izgrađene android aplikacije |
| AWS | Amazon Web Services | Amazon poslužitelj u oblaku |
| BLL | Business Logic Layer | sloj poslovne logike |
| CRUD | Create, Read, Update, Delete | Izrada, Čitanje, Izmjena, Brisanje |
| DAL | Data Access Layer | sloj pristupa podatcima |
| DB | Database | baza podataka |
| DDL | Data Definition Language | jezik za rad s bazama podataka |
| HTTP | Hypertext Transfer Protocol | protokol prijenosa na webu |
| I18N | Internationalization | internacionalizacija |
| JWT | JSON Web Token | web token u JSON obliku |
| ORM | Object-Relationship Mapping | objektno-relacijsko mapiranje |
| SHA | Secure Hash Algorithm | sigurni algoritam *hashiranja* |
| SPA | Single Page Application | Jednostranična aplikacija |
| UAT | User Acceptance Testing | testiranje prihvaćanja od korisnika |
| UC | Use Case | Slučaj korištenja |
| XML | EXtensible Markup Language | jezik za označavanje podataka |

# Popis slika

# Popis tablica

# Popis kôdova

# Literatura


[1] ECOMPUTERNOTES, INFORMATION SYSTEMS PLANNING, http://ecomputernotes.com/mis/information-and-system-concepts/informationsystemsplanning, listopad 2019.

[2] MICROSOFT, OSNOVE BAZA PODATAKA, https://support.office.com/hr-hr/article/osnove-baza-podataka-a849ac16-07c7-4a31-9948-3c8c94a7c204, listopad. 2019.

[3] CSHARPCORNER, TYPES OF DATABASE MANAGMENT SYSTEMS, https://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/, listopad. 2019.

[4] VISOKO UČILIŠTE ALGEBRA, INTEROPERABILNOST INFORMACIJSKIH SUSTAVA, prezentacija Web Services

[5] MEDIUM, BACKEND TECH STACK, https://medium.com/educalabs/choosing-a-backend-tech-stack-in-2018-b415e5f436d3, listopad. 2019.

[6] MICROSOFT, AWAIT OPERATOR, https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/await, listopad. 2019.

[7] CSHARPCORNER, TASK AND THREAD, https://www.c-sharpcorner.com/article/task-and-thread-in-c-sharp/, listopad. 2019.

[8] MICROSOFT, NETCORE ANNOTATIONS, https://docs.microsoft.com/en-us/aspnet/core/web-api/index?view=aspnetcore-2.1#annotate-class-with-apicontrollerattribute, listopad. 2019.

[9] JWT, JWT INTRODUCTION, https://jwt.io/introduction/, listopad. 2019.

[10] ANGULAR, DOCUMENTATION, https://angular.io/guide/setup-local, listopad. 2019.

[11] ANGULAR, COMPONENTS, https://angular.io/api/core/Component#description, listopad. 2019.

[12] WIKIPEDIA, SASS, https://en.wikipedia.org/wiki/Sass_(stylesheet_language), listopad. 2019.

[13] ANDROID, COMPONENTS, https://developer.android.com/guide/components/, listopad. 2019.

[14] VOGELLA, RETROFIT, https://www.vogella.com/tutorials/Retrofit/article.html, listopad. 2019.

[15] WONDERSHARE, SOFTWARE DEPLOYMENT, https://pdf.wondershare.com/business/what-is-software-deployment.html, listopad 2019.

[16] JLELSE, PUBLISHING ANDROID APP, https://android.jlelse.eu/publishing-an-android-app-da3502c652af, listopad 2019.