WEB APLIKACIJA ZA PRAĆENJE NATJECATELJSKIH TURNIRA

Radoš, Anđela

Graduate thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:051888

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-10-13**

Repository / Repozitorij:

Repository of University Department of Professional Studies

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

ANĐELA RADOŠ

Z A V R Š N I R A D

WEB APLIKACIJA ZA PRAĆENJE NATJECATELJSKIH TURNIRA

Split, rujan 2023.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Napredno programiranje web aplikacija otvorenog kôda

Z A V R Š N I R A D

Kandidat: Anđela Radoš

Naslov rada: Web aplikacija za praćenje natjecateljskih turnira

Mentor: Marina Rodić, viši predavač

Split, rujan 2023.

SADRŽAJ

Sažetak 1

Summary 1

1. Uvod 2

2. Korištene tehnologije 3

2. 1. Python 3

2. 2. Django 4

2. 3. Djoser 5

2. 4. MySQL 5

2. 5. JavaScript 6

2. 6. React 6

2. 7. Redux 8

2. 8. HTML 8

2. 9. CSS 9

3. Opis aplikacije 10

3. 1. Funkcionalnost 10

3. 2. Integracija razvojnog okvira Django i biblioteke React 11

3. 3. Odvajanje poslužiteljskog dijela aplikacije od korisničkog 13

3. 4. Korisničke uloge 13

4. Implementacija poslužiteljskog dijela 15

4. 1. Konfiguracija instaliranih alata 15

4. 2. Dodjeljivanje mrežnih putanja 17

4. 3. Modeli 18

4. 4. Serijalizatori 22

4. 5. Baza podataka 23

4. 6. Pogledi i pomoćne funkcije 25

5. Implementacija klijentskog dijela 35
5. 1. Početna stranica 35
5. 2. Navigacijske trake 35
5. 3. Autentikacija 38
5. 4. Tijek prvenstva 44
5. 5. Mehanizmi odlučivanja 53
5. 6. Tijek utakmice 58
5. 7. Elo rating 59
5. 8. Paket @g-loot 61
5. 9. Profili 62
5. 10. Dizajn i teme 62
6. Zaključak 64
Literatura 65

Sažetak

U ovom radu je opisana aplikacija za praćenje i upravljanje nogometnim natjecateljskim turnirima. To je alat koji upraviteljima nogometnih turnira služi za kreiranje prvenstava i detaljno izvještavanje uživo o događajima s nogometnih utakmica, a korisnicima omogućuje pravovremeno praćenje tih događaja. Uz to, mogu se pratiti i analizirati neke sveukupne statistike o natjecanjima, klubovima i igračima. Osim praćenja turnira, aplikacija korisnicima nudi korisnički profil, novosti i forum. Različite korisničke uloge imaju različite dozvole korištenja. Aplikacija je izrađena korištenjem tehnologija Django i React, kao bazu podataka koristi MySQL, a dizajn je oblikovan u stilskom jeziku CSS. Cilj rada, osim pružanja centralizirane platforme za praćenje prvenstava je i prikazati spajanje i međudjelovanje više različitih tehnologija u izradi kompletne aplikacije.

Ključne riječi: CSS, Django, MySQL, nogomet, React

Summary

Web application for tracking football championships

This thesis describes an application for managing and tracking competitive football tournaments. It is a tool that helps tournament managers create championships, provide detailed live coverage of events from football matches and allows users to follow these events in real-time. Furthermore, some overall statistics about competitions, clubs and players can be tracked and analyzed. In addition to tracking tournaments, the application offers user profile, news and a forum. Different user roles have different usage permissions. The application was built using Django and React technologies, uses MySQL for the database, and the look is designed in CSS style language. The aim of this thesis, along with providing a centralized platform for tracking championships, is to show the synthesis and interaction of several different technologies in the creation of a full-stack application.

Keywords: CSS, Django, MySQL, football, React

1. Uvod

Nogomet je bez dvojbe najpoznatiji sport na svijetu i kako vrijeme prolazi, samo plijeni sve više pažnje. Iako je počeo kao jednostavna igra s minimalnim pravilima i opremom, s vremenom se komercijalizirao i u njega se počelo ulagati sve više novca. To je dovelo do osnivanja profesionalnih liga, izgradnje velikih stadiona po cijelom svijetu, razvoja marketinških i medijskih strategija te uvođenja sponzora i reklama. Osim dolaska na stadion i gledanja televizije, nogomet se može pratiti i preko aplikacija. Ovisno o tome zanimaju li korisnika samo rezultati ili želi imati uvid u sve aspekte nogometa, te aplikacije su prilagodljive.

Ova aplikacija je internetska, dizajnirana je za desktop i jednostavna za korištenje jer nogometne informacije dijeli u kategorije. Naglasak stavlja na praćenje rezultata uživo, a kombinira elemente društvenih mreža i stranice za prijenos događaja (engl. *live update*).

U drugom poglavlju definirane su tehnologije i alati korišteni za izradu aplikacije. Opisana je instalacija i konfiguracija tih tehnologija, njihova primjena u ovoj aplikaciji te neke njihove prednosti i mane.

Treće poglavlje daje uvid u opći opis aplikacije. Objašnjene su njezine mogućnosti i namjena, a opisano je i kako funkcionira. U trećem poglavlju se također opisuju prednosti i mane aplikacija kojima je poslužiteljski dio odvojen od klijentskog te su uspoređena dopuštenja pojedinih korisničkih uloga u ovoj aplikaciji.

Četvrto poglavlje detaljno opisuje implementaciju poslužiteljskog dijela aplikacije, a tu je opisana i implementacija baze podataka. Opisano je i kako su se koristili alati i tehnologije u izradi poslužiteljskog dijela.

Peto poglavlje se bavi implementacijom klijentskog dijela gdje se ubraja i oblikovanje dizajna. Nakon ovog poglavlja slijedi zaključak.

2. Korištene tehnologije

Ovo poglavlje opisuje tehnologije koje su se koristile u izradi ove aplikacije. Aplikacija se sastoji od poslužiteljskog i klijentskog dijela. Poslužiteljski dio je napisan u programskom jeziku Python korištenjem razvojnog okvira Django i ima ulogu pristupa i obrade podataka, komunikacije s bazom podataka, provođenja HTTP upita i autentikacije korisnika, koja je implementirana pomoću biblioteke Djoser. Korištena baza podataka je MySQL. Za izradu klijentskog dijela korišteni su programski jezik JavaScript, prezentacijski jezik HTML, biblioteke React i Redux te stilski jezik CSS. Ovaj dio zadužen je za oblikovani prikaz podataka korisniku te upravljanje stanjem aplikacije.

2. 1. Python

Python je objektno orijentirani programski jezik opće namjene koji se najčešće koristi za razvoj web stranica, automatizaciju zadataka, analizu i vizualizaciju podataka. Utemeljio ga je Guido van Rossum 1990. godine u Nizozemskoj. Ime je dobio po seriji Monty Python's Flying Circus, a zamišljen je kao nasljednik programskog jezika ABC.

Kao programski jezik, Python je intuitivan i jednostavan za učenje. Zbog toga mu je područje primjene široko, a često ga koriste i programeri početnici. Za razlikovanje programskih blokova koristi uvlačenje, a ne vitičaste zagrade kao većina programskih jezika. Ne zahtijeva korištenje posebnih interpunkcijskih znakova za kraj naredbe, a navođenje tipova podataka se može i izostaviti. Također ima i opsežnu standardnu biblioteku koja je dostupna svima. Bilo da se radi o manipulaciji datotekama i medijima, komunikaciji s bazama podataka, jediničnom testiranju i ostalim funkcionalnostima, programeri se mogu poslužiti paketima standardne biblioteke umjesto da pišu vlastiti kôd. Još neke od prednosti korištenja Pythona su čitljivost kôda, automatsko upravljanje memorijom i neovisnost o operativnom sustavu, dok su mu neke od mana sporost, veća potrošnja memorije i nedostatak podrške za razvoj mobilnih aplikacija.

Programski jezici najsličniji Pythonu su Java, Scala i Ruby. Python je interpreterski jezik, što znači da se programski kôd izvodi liniju po liniju, a ne sve odjednom. Stoga se programi u Pythonu ne trebaju prevoditi (engl. *compile*) da bi se pokrenuli. U ovom završnom radu se Python koristio za poslužiteljski dio, a korištena verzija je 3.10.

2. 2. Django

Django je razvojni okvir otvorenog kôda pisan u programskom jeziku Python. Nastao je 2003. godine, a ime je dobio po gitaristu Djangu Reinhardtu. Osmišljen je kako bi ubrzao i olakšao razvoj web aplikacija.

Neke od najvažnijih funkcionalnosti koje Django nudi su:

- modeli podataka
- rukovanje HTTP upitima pomoću pogleda (engl. *views*)
- komunikacija s bazom podataka
- jedinično testiranje
- administracijsko sučelje
- rad s formama
- autentikacija
- migracije

Modeli su Python klase ekvivalentne tablicama (entitetima) baze podataka. Modeli su potklase klase `django.db.models.Model` i imaju svojstva koja predstavljaju atribute tablica u bazi.

```
class Substitution(models.Model):
    id = models.AutoField(primary_key=True, null=False, unique=True)
    playerout = models.IntegerField(blank=False, null=False)
    playerin = models.IntegerField(blank=False, null=False)
    minute = models.TextField(blank=False, null=False)
    matchid = models.IntegerField(blank=False, null=False)
    team = models.IntegerField(blank=False, null=False)
```

Ispis 1: Model podataka za izmjenu igrača

Pogledi su funkcije koje obrađuju HTTP upite, dohvaćaju podatke iz baze, generiraju odgovore i provode validaciju (provjeru ispravnosti) podataka. Za jedinično testiranje Django posjeduje ugrađene pakete koji nude testne klase, testne klijente i tvrdnje (engl. *assertions*). Administratorsko sučelje je aplikacija koja omogućuje zaštitu podataka te brzo i jednostavno upravljanje podacima iz baze od strane administratora. Django forme ili obrasci su elementi korisničkog sučelja koji se koriste za prikaz HTML formi. Primaju unos korisnika, obrađuju ga i šalju (engl. *submit*). Migracije u Djangu omogućuju izmjene

strukture i organizacije baze podataka (engl. *database schema*). Te promjene uključuju dodavanja, izmjene i brisanje tablica, polja u tablicama, promjene tipova podataka i sl. Naredba `makemigrations` kreira migracije koje se temelje na izmjenama baze, a naredba `migrate` provodi te promjene. Korištena verzija Djanga u ovom radu je 4.0.6.

2. 3. Djoser

Djoser je biblioteka za provjeru autentičnosti za Django. Omogućuje skup pogleda (engl. *views*) za rukovanje registracijom, prijavom i odjavom korisnika, ponovnim postavljanjem lozinke i aktivacijom korisničkog računa. Također generira autentikacijske tokene na osnovu korisničkog imena, emaila i lozinke i provjerava ispravnost tokena. Djoser ima ugrađene putanje (engl. *routes*, *endpoints*) za upravljanje korisničkim računima, a u ovome radu su se koristile sljedeće:

- `/users` za dohvaćanje svih korisničkih računa
- `/users/me` za dohvaćanje podataka trenutno prijavljenog korisnika
- `/users/activation` za aktivaciju korisničkog računa nakon registracije
- `/users/reset_password` za zahtjev za ponovno postavljanje lozinke
- `/users/reset_password_confirm` za postavljanje nove lozinke
- `/jwt/create` za dodjeljivanje jwt tokena
- `/jwt/verify` za validaciju jwt tokena

2. 4. MySQL

MySQL je sustav za upravljanje bazama podataka. Napisan je u programskim jezicima C i C++ i dostupan je za većinu operativnih sustava. Potpuno je besplatan i otvorenog kôda i uključuje mnogo sigurnosnih značajki (enkripcija, autentikacija, korisničke dozvole i sl.). MySQL ima visoke performanse i brzinu rada pa stoga može uspješno obrađivati ogromne količine podataka, što opet ovisi o veličini baze podataka, hardveru, upitima na bazu (engl. *query*) i drugim čimbenicima. MySQL ima i mehanizam predmemorije (engl. *caching*) gdje se pohranjuju podaci iz baze kojima se često pristupa, što također doprinosi brzini. Područje primjene mu je široko, od mobilnih i web aplikacija i igrica, preko financijskih servisa pa do ugrađenih sustava. MySQL se može koristiti i u računalnom oblaku (engl. *cloud*). Ima i grafičko korisničko sučelje gdje se shemom baze podataka može upravljati pomoću grafičkih komponenti (engl. *widgets*) u okruženju MySQL Workbench.

2. 5. JavaScript

JavaScript (skraćeno JS) je programski jezik koji se prvenstveno koristi za mrežne (web) aplikacije. Izvršava se na klijentskoj strani aplikacije, odnosno pokreće se izravno u pregledniku (engl. *browser*). Prilikom izrade web aplikacija, koristi se u kombinaciji s jezicima HTML i CSS. JavaScript je multiparadigmatski programski jezik, što znači da podržava više programskih stilova, odnosno pristupa programiranju. Može se pisati u proceduralnom stilu (kao niz naredbi koje se izvršavaju po redu), objektno-orijentiranom stilu (definiranje objekata sa svojim svojstvima, atributima i funkcijama, temelji se na prototipu), funkcionalnom stilu (gdje se kôd izrađuje primjenom i slaganjem funkcija), u stilu programiranja vođenog događajima (engl. *event-driven*, gdje komponente programa međusobno komuniciraju slanjem signala i odgovorima na te signale) i u asinkronom stilu (tehnika koja omogućuje izvođenje dugotrajnih zadataka u programu, a da za to vrijeme može izvršavati i druge događaje, umjesto čekanja da se ti zadaci završe). U ovom radu korištena je kombinacija navedenih paradigmi.

2. 6. React

React je biblioteka za programski jezik JavaScript koja se koristi za izradu interaktivnih korisničkih sučelja. Najpogodnija je za jednostranične aplikacije (engl. *single-page application - SPA*, internetske aplikacije sa samo jednom stranicom), web trgovine, financijske aplikacije, društvene mreže, aplikacije za prijenos događaja, za učenje stranih jezika i sl.

Jedna od prednosti ove biblioteke su komponente. To su samostalni blokovi kôda koje za povratnu vrijednost imaju HTML elemente. Dijele se na klase i funkcijske komponente i uvijek se pišu velikim početnim slovom. Funkcijske komponente su mnogo jednostavnije od klasa i definiraju se kao i obične funkcije jezika JavaScript. Mogu primati i argumente, ali ne moraju. Ti argumenti su atributi dodatka JSX (sintaksa ugrađena u JavaScript koja omogućuje korištenje HTML elemenata izravno u kôdu) i prosljeđuju se kao objekti. Također, argumenti funkcijskih komponenti se mogu izvući (engl. *extract*) i proslijediti eksplicitno. Za upravljanje stanjem (engl. *state*) u funkcijskim komponentama se koriste i posebne funkcije, tzv. „React hooks“. Pozivaju se unutar funkcijskih komponenti. Funkcija `useState` vraća par stanje-funkcija za ažuriranje stanja. Kada se pozove funkcija

ažuriranja, React osvježava (engl. *re-render*) stanje novom vrijednošću. Osim funkcije `useState`, korisna *hook* funkcija je `useEffect`. Služi za upravljanje sporednim efektima (engl. *side effects*), kao npr. dohvaćanje podataka iz baze, postavljanje rukovanja događajima (engl. *event handling*), pokretanje animacija, upravljanje bibliotekama JavaScript-a i sl. Funkcija `useEffect` prima dva argumenta, funkciju koja predstavlja efekt koji će se izvršiti nakon svakog prikazivanja komponente i niz varijabli o kojima efekt ovisi (ako se neka od varijabli unutar niza promijeni, efekt se ponovno izvršava). Ovaj niz može biti i prazan. Nasuprot funkcijskim komponentama, klase imaju metode životnog ciklusa (slično konstruktorima i destruktorima), nasljeđuju klasu `React.Component` i stanjem upravljaju ugrađenim metodama, umjesto *hook* funkcijama.

```
const PlayerPhotos = ({ id, isAuth }) => {
  const [plyrs, setPlyrs] = useState([]);
  useEffect(() => {
    fetch(`/api/players/`)
      .then((response) => response.json())
      .then(data => setPlyrs(data.filter(x => x.currentteam == id)));
  }, []);

  return (
    <div>
      <span className="inline-span">
        <h1 className={contactSubtitle}>Players</h1>
        {isAuth && <button className="card-button-goto">Add</button>}
      </span>
      <div className="players-container">
        {plyrs.map((item) => {
          return (
            <div onClick={() => navigate(`/players/${item.id}`)}>
              <h2 className={`${contactSubtitle} ${playerTeamName}`}>
                {item.jerseynumber} {item.jerseyname}
              </h2>
              <p className={`${sub} ${position}`}>{item.position}</p>
              <img src={item.profilephoto} className="player"></img>
            </div>
          );
        })}
      </div>
    </div>
  );
};
export default PlayerPhotos;
```

Ispis 2: Komponenta za fotografije igrača u klubu uz korištenje *hook* funkcija

2. 7. Redux

Redux je biblioteka za upravljanje stanjem web aplikacija koja ima jednostavan API za skladištenje stanja. Centralizirana je što znači da je sve vezano uz stanje smješteno na jednom mjestu koje se naziva skladište (engl. *store*), koje je zajedničko svim komponentama aplikacije za čitanje i upisivanje stanja. Redux koristi tzv. reduktore (engl. *reducers*), čiste funkcije (engl. *pure function*), koji na temelju trenutnog stanja i neke radnje (engl. *action*, objekt koji predstavlja događaj ili promjenu stanja) mijenja trenutno stanje i vraća novo. Funkcija koja povezuje React i Redux je `connect`. Njome se stvara nova komponenta (spremnik) koja se povezuje na Redux skladište. Ova funkcija kao parametar može primati funkciju `mapStateToProps` i objekt koji sadrži funkcije za otpremanje (engl. *dispatch*). Funkcija `connect` rezultira novom komponentom, povezanom na Redux skladište koja ima pristup podacima stanja i akcija u skladištu.

```
const mapStateToProps = state => ({
    isAuthenticated: state.auth.isAuthenticated,
    user: state.auth.user
  });

export default connect(mapStateToProps, {checkAuthenticated,
loadUser})(ForumPost);
```

Ispis 3: Primjer povezivanja komponente objave foruma na Redux skladište

2. 8. HTML

HTML je standardni jezik oznaka (engl. *markup*, *tag*) za izradu strukture i sadržaja web stranica. Internetski preglednici s web poslužitelja primaju HTML dokumente i prikazuju ih kao multimedijske internet stranice. Elementi jezika HTML su oznake, odnosno *tagovi*. Svaki element predstavlja dio stranice, okruženi su izlomljenim zagradama (< i >), imaju početnu i završnu oznaku i mogu se graditi hijerarhijski. *Tagovi* mogu imati i neke atribute koji omogućuju dodatne informacije o njima, kao što su visina, širina, stil, poveznica, tekst itd. Neki od najčešćih tagova koji se koriste su `<h1>` (od 1 do 6, za naslove i veći tekst), `<p>` (za paragraf teksta), `<a>` (za poveznice na druge internet stranice ili putanje), `<img>` (za slike), `<ul>` i `<ol>` (za pobrojane i nepobrojane liste), `<head>` (za naslov web stranice, meta informacije i sl.), `<div>` (za grupu elemenata na koje se primjenjuje neki stil i sl.).

2. 9. CSS

CSS (Cascading Style Sheets) je stilski jezik koji opisuje izgled i format HTML dokumenata. Osmišljen je kako bi se dizajn odvojio od strukture, odnosno sadržaja dokumenta, što olakšava stiliziranje HTML komponenti i dodavanje vizualnih efekata kao što su boje, fontovi, animacije i razmještaj.

Dizajn oblikovan u ovom programskom jeziku može biti jako kompleksan i prilagođavati se različitim veličinama ekrana. Sintaksa se sastoji od određenih engleskih riječi u obliku parova selektor-deklaracijski blok. Selektor označava HTML element (ili više njih) na koje se određeni dizajn primjenjuje, a deklaracijski blok definira taj dizajn.

```
.forum-card {
    margin-left: 100px;
    margin-right: 100px;
}

.forum-icons {
    display: flex;
    align-items: flex-end;
    color: silver;
}

.forum-icons-light {
    display: flex;
    align-items: flex-end;
    color: black;
}

.discussion-wrapper {
    position: fixed;
}

.misc-inline-title {
    display: inline;
    color: goldenrod;
    font-family: Games;
    font-size: 30px;
}
```

Ispis 4: Dizajn stranice za forum

3. Opis aplikacije

Ovo poglavlje pokriva funkcionalnost aplikacije, uspoređuje korisničke uloge i opisuje integraciju razvojnog okvira Django i biblioteke React na način gdje je poslužiteljski dio aplikacije odvojen od klijentskog. Aplikacija je namijenjena organizatorima nogometnih prvenstava kako bi u stvarnom vremenu donosili izvještaje o utakmicama u tijeku te korisnicima koji bi te izvještaje pratili. Naziv aplikacije je „Tournzilla“, što je nastalo spajanjem engleskih riječi *tournament* (prvenstvo) i *godzilla* (moćno čudovište).

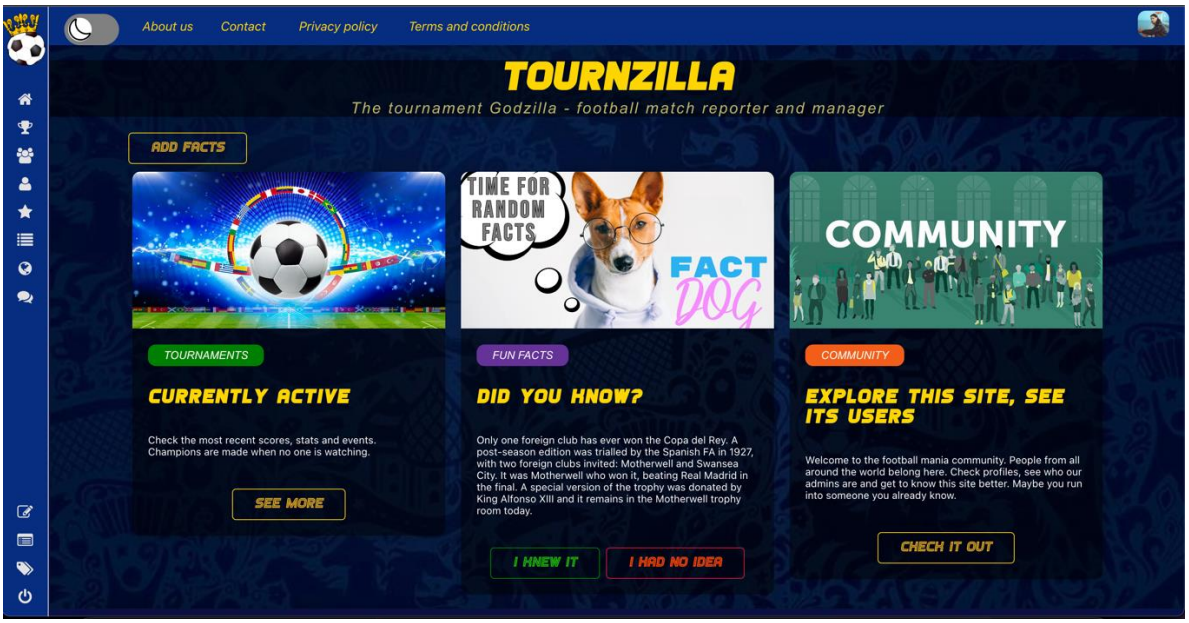
3. 1. Funkcionalnost

Kada korisnik otvori aplikaciju dočeka ga početna stranica (slika 1). Većina njenog sadržaja je statična i tu se nalaze:

- poveznica na sve utakmice u tijeku
- zanimljivosti (engl. *fun facts*)
- poveznica na listu svih registriranih korisnika

Sadržaj je u obliku kartica, a s lijeve strane je fiksirana navigacijska traka po kojoj se korisnik kreće i otvara sadržaje. Moguće je pregledati sva evidentirana prvenstva koje kreiraju i uređuju administratori i detalje o njima, kao što su rasporedi i ishodi utakmica, grupne kvalifikacije, eliminacije, podaci o organizaciji prvenstva (otvaranje, maskota, troškovi), medalje i najbolji igrači. Odabirom pojedine utakmice na tablici eliminacija otvaraju se detalji te utakmice. Ako je ona u tijeku, administratori ih mogu uređivati pomoću alata koji su dostupni samo njima dok ih korisnik može samo pregledati, ali pravovremeno, jer se sadržaj osvježava svakih 5 sekundi. Zatim su tu timski i nogometaški profili. I oni sadrže razne statistike koje ažuriraju administratori, uglavnom o postignućima timova i igrača (pobjede, porazi, golovi, vrijednost pojedinog igrača na tržištu, najbolji strijelci itd.). Registrirani korisnici mogu uređivati listu omiljenih timova kako bi na jednom mjestu imali uvid u njihove utakmice u tijeku. Aplikacija pruža i prikaz rangiranja timova i igrača, koje se ažurira po završetku svakog prvenstva pomoću metode za evaluaciju zvane ELO rating. Novosti na stranici dodaju administratori, a registrirani korisnici ih mogu komentirati, za razliku od foruma, gdje svi registrirani korisnici mogu dodavati i rasprave i komentare o nogometu. Prateći aktivnost korisnika dodjeljuju mu se tzv. značke. Većina informacija koje se prikazuju na stranici je pohranjena u bazi podataka, a samo nepromjenjivi podaci, kao što su države, vrste nogometnih postava i neke slike su

dio strukture aplikacije. U aplikaciju su ugrađene i dvije teme, svijetla i tamna.



Slika 1: Početna stranica aplikacije

3. 2. Integracija razvojnog okvira Django i biblioteke React

Razvojni okvir Django, jedan od najpotpunijih razvojnih okvira za razvoj web aplikacija i biblioteka React, koja je pogodna za izradu web stranica i jednostavno rukovanje korisničkim sučeljem, imaju neka zajednička svojstva koja ih čine dobrom kombinacijom u izradi web aplikacije. Neka od tih svojstava su skalabilnost, sveobuhvatna dokumentacija, popularnost kod razvojnih programera, podržavanje arhitekture MVC (engl. *model-view-controller*) i oba su otvorenog kôda. Jedan on načina integracije ove dvije tehnologije je taj gdje React služi za usmjeravanje (engl. *routing*) i prikaz stranica (engl. *render*), a Django omogućuje podatke iz baze i autentikaciju te oni međusobno komuniciraju putem HTTP upita. Ovakva kombinacija tehnologija može poboljšati performanse u slučaju većih i kompleksnijih aplikacija. U tom slučaju može se koristiti biblioteka Redux kao posrednik, kako bi više komponenti moglo dijeliti isto stanje.

Za postavljanje razvojnog okvira Django potrebno je na operativnom sustavu imati instaliran programski jezik Python i konfigurirane varijable okruženja (engl. *environment variables*) kako bi se mogao instalirati i alat pipenv (upravitelj paketima i virtualno okruženje). Zatim se pomoću tog alata aktivira virtualno okruženje i instalira Django. Kada je instaliran, pomoću njega se kreira osnovna struktura projekta u koju se dodaju

komponente, modeli, predlošci, funkcionalnosti i sl. Nakon toga se pokreće naredba za dodavanje i sinkronizaciju baze podataka s modelima entiteta u projektu. Po završetku svih ovih koraka može se lokalno pokrenuti poslužitelj kojemu se pristupa pomoću porta 8000.

```
$ cd Desktop
$ pip3 install pipenv
$ pipenv shell
$ pipenv install django
$ django-admin startproject tournzilla
$ python manage.py startapp api
$ python manage.py migrate
$ python manage.py runserver
```

Ispis 5: Popis naredbi za postavljanje razvojnog okvira Django

Klijentski dio se instalira korištenjem naredbe `npx create-react-app client`, koja će kreirati gotov projekt sa svim osnovnim ovisnostima, modulima i konfiguracijama i može se pokrenuti lokalno na portu 3000. Umjesto toga, bolji izbor je dodati putanju za klijentski dio u postavke poslužiteljskog dijela, tako da se aplikaciji pristupa pomoću porta za poslužitelj. Na taj način Django poslužuje i klijentski dio aplikacije. Nakon svakih promjena u kôdu klijentskog dijela, potrebno je pokrenuti naredbu `npm run build` kojom se ažurira direktorij `build` u klijentskom dijelu, jer Django samo locira taj direktorij, a ne sinkronizira promjene na klijentu. Ispis 6 prikazuje tu konfiguraciju.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'client/build'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

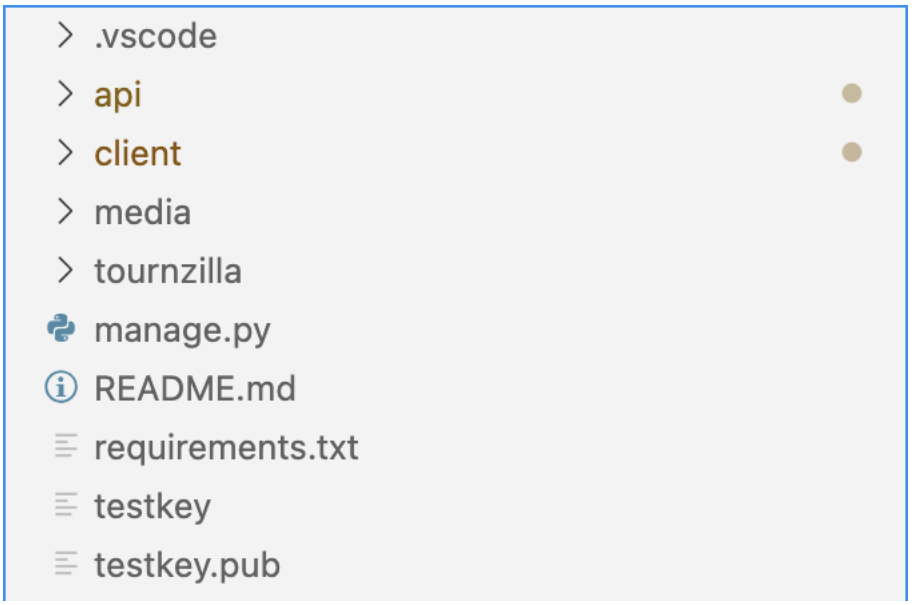
Ispis 6: Dodavanje klijentskog direktorija u datoteku settings.py

3. 3. Odvajanje poslužiteljskog dijela aplikacije od korisničkog

Logika poslužiteljskog dijela je odvojena od klijentskog. Time je svaki dio samostalan i lakše je njime upravljati. Na primjer, ako je potrebno, moguće je promijeniti poslužiteljski dio bez promjene klijentskog i obrnuto. Odvajanje omogućuje i ponovno korištenje bilo kojeg od dijelova u drugim projektima (engl. *reusability*), a stvara i privid uredne strukture aplikacije. Ova aplikacija je organizirana u 4 direktorija (prikazano na slici 2):

- `/tournzilla` sadrži konfiguraciju poslužiteljskog dijela, listu modula i URL-ove
- `/api` sadrži modele entiteta, predloške i implementaciju usmjernika (engl. *router*)
- `/media` pohranjuje sve fotografije koje se dodaju korištenjem aplikacije
- `/client` sadrži cijeli klijentski kôd

U istoj razini se nalazi i datoteka `manage.py` koja služi kao ulazna točka za interakciju s aplikacijom putem terminala (migracije entiteta, pokretanje poslužitelja, testiranja i sl.).



Slika 2: Struktura aplikacije

3. 4. Korisničke uloge

Korisnik s ulogom administratora je ujedno i upravitelj turnira. Ta vrsta korisnika ima sve ovlasti nad aplikacijom. Administrator može kreirati turnire, uređivati statistike i rezultate utakmica u tijeku, dodavati, uređivati i brisati prvenstva, timove i igrače. Također, može upravljati korisnicima, brisati ih ili im dodijeliti ulogu administratora. Biblioteka

Djoser za model korisnika nudi polja `is_staff` i `is_admin` koja se mogu koristiti za razlikovanje korisničkih uloga. Ovdje je dovoljno jedno polje jer su svi administratori upravitelji turnira (koristi se `is_staff`). Jedina radnja koju administrator ne može provesti, a obični prijavljeni korisnik može, je prijava neprimjerenog ponašanja korisnika. To je zato jer administrator ima mogućnost uklanjanja tuđih korisničkih računa pa nema potrebe za prijavama (ako primijeti neprimjereno ponašanje nekog korisnika, može mu izbrisati račun), dok ostali prijavljeni korisnici moraju podnijeti prijave administratoru koji ih pregleda i odlučuje što učiniti s tim korisnicima. Iako administrator može uređivati rezultat i statistike utakmica, to može činiti samo dok je utakmica u tijeku i to samo ako se igra (dakle, ne može u pauzi za poluvrijeme ili pauzama između produžetaka utakmice) jer su utakmice jako ovisne o vremenu i međusobno. Ako se dogodi da administrator propusti utakmicu, osiguran je mehanizam odlučivanja (opisan u petom poglavlju) koji omogućuje nesmetan tijek prvenstva i odluku o pobjedniku i sve potrebno za sljedeću utakmicu (postava igrača, sudac, sudionici) i dodaje statistike trenutnoj utakmici i sudionicima (pobjednik, status, rezultat). Za objavu novosti na stranici zaduženi su isključivo administratori jer su to službene novosti (iako ih komentirati mogu svi prijavljeni korisnici), a na forumu bilo koji prijavljeni korisnik može započeti raspravu i komentirati. Detaljan prikaz korisničkih uloga se nalazi u tablici 1.

Tablica 1: Korisničke uloge

	Administrator	**Prijavljeni korisnik**	**Anonimni /registrirani nepotvrđeni korisnik**
Pregled sadržaja	+	+	+
Kreiranje prvenstava, timova, igrača	+	-	-
Ažuriranje rezultata	+	-	-
Objava novosti	+	-	-
Komentiranje	+	+	-
Objava rasprava	+	+	-
Prijava neprimjerenog ponašanja	-	+	-
Dobivanje znački	+	+	-
Favoriti	+	+	-

4. Implementacija poslužiteljskog dijela

Ovo poglavlje opisuje način na koji je postavljen i implementiran poslužiteljski dio aplikacije, od instalacije i postavki alata, preko izrade logike API poziva pa do povezivanja baze podataka. Bit će objašnjen koncept preslikavanja relacijskih entiteta u objekte (ORM, engl. *Object-relational mapping*), definiranje i dodjeljivanje putanja pogledima i izvedba pomoćnih funkcija za upravljanje modelima. Uređivač cijelog kôda je Visual Studio Code.

4. 1. Konfiguracija instaliranih alata

Instalacijom razvojnog okvira Django i kreiranjem projekta dobiva se osnovna struktura aplikacije i konfiguracijske datoteke. Unutar datoteke `settings.py` dodaju se aplikacije i paketi koje programer želi integrirati u projekt kako bi ga prilagodio potrebama. U datoteci već postoje neke instalirane aplikacije:

- `django.contrib.admin` je alat za administracijsko sučelje, upravljanje modelima, korisnicima i ostalim sadržajem
- `django.contrib.auth` je aplikacija za korisničke uloge, dozvole i provjere ispravnosti ulaznih korisničkih podataka i sl.
- `django.contrib.contenttypes` je alat za rad s modelima
- `django.contrib.sessions` je aplikacija koja pohranjuje informacije o sesijama za svaki korisnički račun
- `django.contrib.messages` je aplikacija koja omogućuje poruke korisnicima nakon obrade korisničkog unosa podataka
- `django.contrib.staticfiles` locira postavljene statičke datoteke i poslužuje ih prilikom pokretanja aplikacije

Osim zadanih aplikacija, u datoteku je instalirano još nekoliko alata. Kao što je već spomenuto ranije, instaliran je paket Djoser, koji pomoću svojih putanja za autentikaciju olakšava rad s korisničkim računima i ulogama. Za izradu API-a instaliran je alat (Django) Rest Framework. Pomoću njega moguće je kreirati modele podataka s odgovarajućim serijalizatorima koji će ih pretvarati u objekte formata JSON za HTTP upite. Za pristup aplikaciji s više domena instaliran je alat Cors headers. Tu se postavljaju izvori s kojih je dozvoljen, a s kojih zabranjen pristup. Na primjer, može se dodati postavka `CORS_ALLOW_ALL_ORIGINS=True` za dozvolu pristupa sa svih izvora, ali to za sobom vuče

moguće posljedice po pitanju sigurnosti aplikacije, zato s ovom postavkom treba oprezno postupati, ovisno o vrsti podataka s kojima aplikacija radi. Kako bi ovaj alat bio funkcionalan, potrebno je dodati i tzv. posrednički softver (engl. *middleware*), koji se primjenjuje na nadolazeće HTTP upite i odlazne odgovore tako što na njih dodaje odgovarajuća CORS zaglavlja. Posrednički softver alata Cors headers mora biti smješten ispred ostalih posredničkih softvera kako bi imao svrhu. Za alat Rest framework izrađen je dodatak treće strane Simple JWT (JSON web token), čija je uloga omogućiti značajke za autentikaciju, točnije proizvesti tokene, provjeravati njihovu ispravnost i ograničiti pristupe pojedinim putanjama API-a. Moguće je i postaviti prilagođeno zaglavlje kojim će počinjati generirani token. U ovom slučaju to je `string` „JWT“, a postavka koja se koristi je `AUTH_HEADER_TYPES`. Dodatno, za onemogućavanje već korištenih i zastarjelih tokena koristi se dodatak ovog paketa, Simple JWT Token blacklist. Za sve tokene se kreiraju entiteti u bazi podataka ovisno o vrsti tokena pa tako postoje tokeni na crnoj listi (engl. *blacklisted*) i aktivni i valjani tokeni (engl. *outstanding*). Na kraju instaliranih alata se dodaje konfiguracijska klasa `api.apps.ApiConfig` koja sadržava aplikaciju sa svim njezinim komponentama. Prva riječ (`api`) odnosi se na naziv aplikacije.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'djoser',
    'rest_framework',
    "corsheaders",
    'rest_framework_simplejwt',
    'rest_framework_simplejwt.token_blacklist',
    'api.apps.ApiConfig',
]
SIMPLE_JWT = {'AUTH_HEADER_TYPES': ('JWT',),}
CORS_ALLOW_ALL_ORIGINS = True
REST_FRAMEWORK = {
 'DEFAULT_PERMISSION_CLASSES':
  ['rest_framework.permissions.IsAuthenticated'],
 'DEFAULT_AUTHENTICATION_CLASSES':
  ('rest_framework_simplejwt.authentication.JWTAuthentication',),
}
```

Ispis 7: Postavke instaliranih alata

4. 2. Dodjeljivanje mrežnih putanja

Kao otpremnik mrežnih putanja (engl. *URL dispatcher*) koristi se datoteka `urls.py`. Tu se navode putanje uparene s odgovarajućim funkcijama (pogledima) i nekim jedinstvenim skraćenim imenima, odnosno aliasima. To znači da će poslužitelj na posjećenoj lokaciji izvršiti određene funkcije. Putanje se mogu definirati na više načina, a jedan od njih je jednostavno spremanje u listu. Svaka putanja mora počinjati kosom crtom (/), a može sadržavati i varijable, čiji tip podataka je potrebno navesti.

```
urlpatterns = [
    path('/teams/', views.getTeams, name="teams"),
    path('/teams/<int:pk>/', views.getTeam, name="team"),
    path('/tournaments/', views.getChampionships, name="tournaments"),
    path('/facts/', views.getFacts, name="facts"),
    path('/facts/<int:pk>/', views.getFact, name="fact"),
    path('/facts/create/', views.createFact, name="factcreate"),
    path('/users/', views.getUsers, name='users'),
    path('/users/<int:pk>/', views.getUser, name='user'),
    path('/forum/', views.getForum, name="forum"),
    path('/forum/<int:pk>/', views.getForumPost, name="forumpost"),
    path('/players/', views.getPlayers, name="players"),
    path('/players/<int:pk>/', views.getPlayer, name="player"),
    path('/news/', views.getNews, name="news"),
    path('/news/<int:pk>/', views.getNewsDetails, name="newsdetails"),
    path('/reports/', views.getReports, name="reports"),
    path('/reports/<int:rid>/', views.getReport, name="report"),
    path('/referees/', views.getReferees, name="referees"),
    path('/referees/<str:rname>/', views.getReferee, name="referee"),
    path('/groups/<int:cid>/', views.getGroups, name="groups"),
    path('/formations/<int:mid>/', views.getFormations,
name="formations"),
    path('/goals/<int:mid>/', views.getGoals, name="goals"),
    path('/goals/<int:mid>/<int:gid>/', views.getGoal, name="goal"),
    path('/yellowcards/<int:mid>/', views.getYellowCards,
name="yellowcards"),
    path('/redcards/<int:mid>/', views.getRedCards, name="redcards"),
    path('/penalties/<int:mid>/', views.getPenalties,
name="penalties"),
    path('/tournaments/<int:cid>/matches/', views.getMatches,
name="matches"),
    path('/forumcomments/<int:pid>/<int:cid>/', views.getForumComment,
name="forumcomment")
] + static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

Ispis 8: Popis putanja aplikacije

4. 3. Modeli

Modeli su klase koje služe za prikaz podataka iz baze. Svaki model predstavlja jedan entitet, odnosno tablicu, a svojstva modela (engl. *properties*) atribute, odnosno polja u tablici. Logika svojstava je slična postavljanju atributa u tablicu baze podataka pa tako postoje primarni ključevi, ključne riječi za jedinstvenost, mogućnost vrijednosti `NULL`, vrste podataka slične onima u bazi (`BooleanField`, `DateField`, `TextField`, `IntegerField`, `AutoField` itd.), oznake za obavezna polja, zadane vrijednosti itd.

S obzirom da u aplikaciji postoji korisnički profil, kreiran je model `User`. Tvore ga osobni podaci, autentikacijski podaci (korisničko ime, lozinka i email), podaci vezani uz dozvole (vrsta korisnika, aktivacija računa), podaci vezani uz korištenje aplikacije (broj objava, značke i sl.) i fotografija. Email i korisničko ime su jedinstvena svojstva, a uz lozinku su jedina obavezna svojstva. Ova klasa proširuje ugrađenu klase `AbstractUser` i `PermissionsMixin`. Objekti ove klase dolaze od klase `UserAccountManager`, koja posjeduje metode za upravljanje korisničkim računima, odnosno na osnovu unesenih korisničkih podataka kreira osobni, administratorski ili račun za osoblje.

```
class UserAccountManager(BaseUserManager):
    def create_user(self, email, username, password=None):
        if not email or not username:
            raise ValueError('Email and username must be provided.')
        email = self.normalize_email(email)
        user = self.model(email=email, username=username)
        user.set_password(password)
        user.is_staff=False
        user.save()
        return user
    def create_staffuser(self, email, username, password):
        user = self.create_user(email,username,password=password,)
        user.staff = True
        user.save()
        return user
    def create_superuser(self, email, username, password):
        user = self.create_user(email,username,password=password,)
        user.staff = True
        user.admin = True
        user.save()
        return user
```

Ispis 9: Klasa za upravljanje korisničkim računom

```
class User(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(max_length=255, unique=True)
    username = models.CharField(max_length=255, unique=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    knownfacts = models.IntegerField(default=0)
    unknownfacts = models.IntegerField(default=0)
    achievements = models.IntegerField(default=0)
    fullname = models.TextField(default='???', blank=True)
    gender = models.TextField(default=None, null=True, blank=True)
    status = models.TextField(default="", blank=True)
    birthday = models.DateField(default=None, null=True, blank=True)
    country = models.TextField(default="")
    hobbies = models.TextField(default="???", blank=True)
    membersince = models.DateField(default=str(date.today()))
    instagram = models.TextField(default="", blank=True)
    facebook = models.TextField(default="", blank=True)
    youtube = models.TextField(default="", blank=True)
    favorites = models.IntegerField(default=0)
    profilephoto =
models.TextField(default="../../media/profile_default_img.jpg")

    objects = UserAccountManager()
    USERNAME_FIELD = 'username'
    REQUIRED_FIELDS = ['email']

    def __get_email__(self):
        return self.email
    def __str__(self):
        return self.username
```

Ispis 10: Model korisnika

I ostali modeli su izrađeni na sličan način. Za model nogometnog kluba (tima) su dodane statističke, povijesne i informacije karakteristične za taj tim (slike dresova, trener ili izbornik, logo itd.). Iako ovaj model ima jedinstveni broj (`id`), ime tima je također jedinstveno jer i u stvarnom svijetu ne postoje dva tima s istim imenom. Također, ime i sve ostale informacije o timu su promjenjive, osim spola koji je nepromjenjiv i oznake za valjanost tima koja se mijenja isključivo dodavanjem i brisanjem igrača. Model tima, kao i model korisnika, ne posjeduje ni jedan strani ključ, ali polja ovih modela služe kao strani ključevi drugim tablicama. Obavezne vrijednosti za kreiranje objekta klase tim su ime tima, ime izbornika, država i spol, a sve ostale vrijednosti mogu biti izostavljene i njima se dodjeljuju zadane vrijednosti.

Najvažniji model u aplikaciji je `Championship`, koji povezuje većinu drugih modela. Osim identifikacijskog broja, ima polje za ime, koje nije jedinstveno, jer više natjecanja može imati isti naziv. Sadrži i polje za datum početka, kojega unosi kreator prvenstva, za razliku od datuma kraja, koji se računa na osnovu datuma početka, broja sudionika, vrsti prvenstva (s kvalifikacijama ili bez) i vremenu potrebnom za odmor između pripadnih utakmica prvenstva. Statistički i opisni podaci (npr. maskota, otvaranje, posjećenost, troškovi organizacije) se mogu promijeniti, dok se podaci o kvalifikacijama, sudionicima, početku, kraju i spolu sudionika ne mogu mijenjati. Informacije o najboljim igračima, golmanima, medaljama, statusu prvenstva i sl. se mijenjaju pomoću aplikacije kada za to dođe vrijeme.

```
class Championship(models.Model):
    championshipid = models.AutoField(primary_key=True)
    name = models.TextField(null=False, blank=False)
    startDate = models.DateField(null=False, blank=False)
    endDate = models.DateField(null=False, blank=False)
    qualifications = models.BooleanField(null=False, blank=False,
default=False)
    host = models.TextField(null=False, blank=False)
    mascot = models.TextField(null=True, blank=True,
default='../../media/default_photo.jpeg')
    mascotdesc = models.TextField(null=True, blank=True, default="")
    mascotname = models.TextField(null=True, blank=True, default=None)
    winner = models.TextField(null=True, blank=True, default=None)
    secondplace = models.TextField(null=True, blank=True,
default=None)
    thirdplace = models.TextField(null=True, blank=True, default=None)
    openingphoto = models.TextField(blank=True,
default='../../media/default_photo.jpeg')
    cities = models.IntegerField(null=True, blank=True, default=None)
    matchesplayed = models.IntegerField(null=False, blank=False)
    status = models.TextField(null=True, blank=False)
    friendly = models.BooleanField(null=False, blank=False)
    bestplayername = models.TextField(null=True, blank=True)
    bestgkname = models.TextField(null=True, blank=True, default='')
    attendance = models.IntegerField(null=False, blank=True)
    goals = models.IntegerField(null=False, blank=True, default=0)
    organisationcost = models.IntegerField(null=False, blank=True)
    gender = models.TextField(null=False, blank=False,default='all')
    resttime = models.TextField(null=False, blank=False)
```

Ispis 11: Model nogometnog prvenstva

Među važnijim modelima su i `FootballMatch` i `Player`, za upravljanje nogometnim utakmicama i igračima. Model `FootballMatch` je sličan modelu `Championship`, jer sadrži vremenske elemente i statističke podatke, a model `Player` je sličan modelu `User` i ima slične informacije kao i za korisnika, što uključuje osobne podatke i fotografije, ali su dodane i neke specifične informacije svojstvene igraču (vrijednost na tržištu, tim za koji trenutno igra, pozicija i sl.). Nad modelom igrača se može obaviti mnogo radnji. Osim uređivanja informacija, moguće je napraviti transfer u novi klub, a isto tako i umiroviti igrača i maknuti njegovu vezu s timom. Tu se prvi put spominje postojanje stranog ključa koji može biti ništavan (engl. *nullable*), odnosno može poprimiti vrijednost polja neke druge tablice, ali može imati i vrijednost `NULL`. To je posebno korisno u slučaju brisanja tima jer je tada moguće sačuvati igrače umjesto da se oni i svi podaci o njima izbrišu.

Za ništavnost podataka postavlja se parametar `null=True` za pojedino svojstvo modela u zagradi. Model `FootballMatch` također ima jedno takvo polje. Riječ je o polju `groupid` koje definira je li utakmica u sklopu kvalifikacijske grupe. Kvalifikacijske grupe se izrađuju nakon što je kreirano prvenstvo, ako je odabrana opcija da je ono kvalifikacijsko. U tom slučaju broj sudionika mora biti djeljiv s 4 i prije eliminacijskih utakmica igraju se grupne, gdje se timovi dijele u grupe po 4 i igraju svaki po 3 utakmice, odnosno sveukupno se igra 6 utakmica u svakoj grupi. Ovakav način igranja utakmica naziva se Round Robin jer u njemu ciklički svaki tim igra protiv svakoga tima, za razliku od načina jednostruke eliminacije gdje se timovi uparuju i već nakon jedne utakmice se odlučuje koji tim ide dalje, a koji ispada te se broj timova smanjuje sa svakom odigranom utakmicom. U slučaju grupnih kvalifikacija tek na kraju svih odigranih utakmica slijedi odluka o prolasku i ispadanju timova. Od 4 tima, 2 ispadaju a 2 idu dalje. To znači da utakmice u sklopu grupe mogu završiti neriješenim rezultatom, a eliminacijske ne mogu.

Opisani su najvažniji modeli podataka u aplikaciji. Uz njih, tu je još 20 sporednih modela podataka koji pomažu u izgradnji gore navedenih. To su modeli za crvene i žute kartone, golove, kaznene udarce i formacije tijekom utakmice, modeli za omiljene timove i značke koje se dodjeljuju korisniku, za novosti, forum i komentare, za prijavljivanje neprimjerenog ponašanja korisnika administratorima, za sudioništvo i kvalifikacijske grupe tijekom prvenstva. Ovi modeli imaju manje polja u odnosu na glavne, a ako imaju vezu s modelom `Player`, posjeduju ništavne strane ključeve.

Na primjer, model `RedCard` sprema podatke o svim crvenim kartonima u aplikaciji, a može se pretraživati prema rednom broju prvenstva (`championshipid`), utakmice (`matchid`) ili igrača kojemu pripada (`playerid`). S obzirom da polje `playerid` može poprimiti vrijednost `NULL`, sprema se i ime igrača koje nije strani ključ već potpuno neovisno polje. Služi u slučaju da se igrač izbriše iz baze podataka (pošto tada polje `playerid` postaje `NULL`), podaci o crvenom kartonu ostaju, ne moraju se brisati.

```
class RedCard(models.Model):
    id = models.AutoField(primary_key=True, null=False, unique=True)
    playerid = models.IntegerField(blank=False, null=False)
    minute = models.IntegerField(blank=False, null=False)
    matchid = models.IntegerField(blank=False, null=False)
    team = models.IntegerField(blank=False, null=False)
    jerseyname = models.TextField(blank=False, null=False)
    championshipid = models.TextField(blank=False, null=False)
```

Ispis 12: Model crvenog kartona

4. 4. Serijalizatori

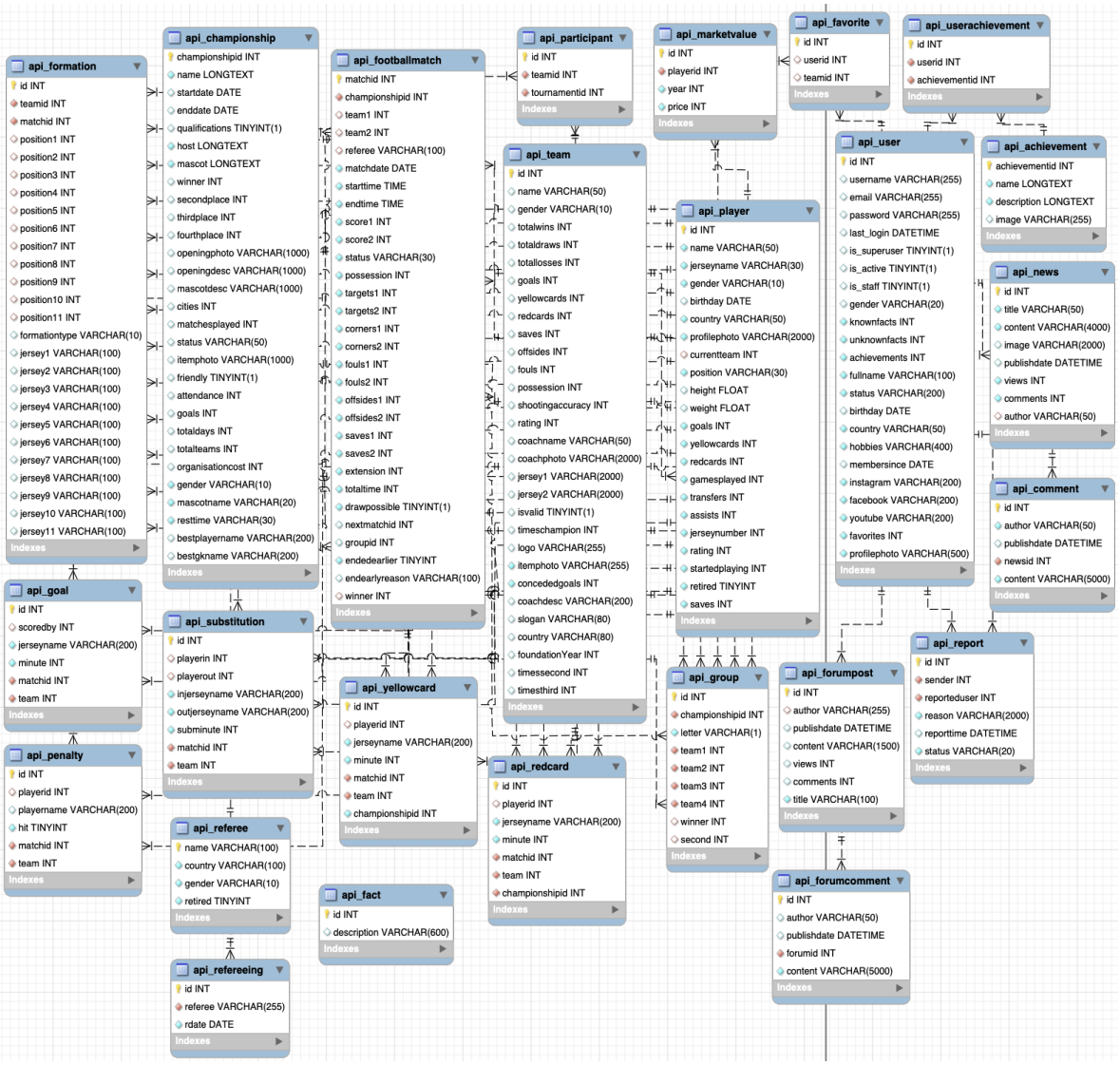
Serijalizatori su klase za prevođenje tipova podataka iz jednog oblika u drugi. Na primjer, u ovoj aplikaciji oni provode serijalizaciju modela podataka u format JSON i deserijalizaciju natrag. Koriste se u pogledima i pomoćnim funkcijama i njihova uloga je pripremanje podataka prema klijentskom dijelu aplikacije. Polja modela za serijalizaciju se mogu navoditi selektivno ili jednostavno poslati sva polja (`__all__`).

```
class ChampionshipSerializer(ModelSerializer):
    class Meta:
        model = Championship
        fields = '__all__'
User = get_user_model()
class UserCreateSerializer(UserCreateSerializer):
    class Meta(UserCreateSerializer.Meta):
        model = User
        fields = ('id', 'email', 'username', 'password', 'is_staff',
'is_active', 'knownfacts', 'unknownfacts', 'achievements', 'fullname',
'gender', 'status', 'birthday', 'country', 'hobbies', 'membersince',
'instagram', 'facebook', 'favorites', 'youtube', 'profilephoto')
```

Ispis 13: Primjer serijalizatora prvenstva i korisnika

4. 5. Baza podataka

Za potrebe ove aplikacije instaliran je alat za upravljanje relacijskim bazama podataka, MySQL, sa svim svojim značajkama (ljuska, grafičko sučelje, poslužitelj, usmjernici itd.). Ljuska (engl. *shell*) je korištena za strukturiranje baze, a grafičko sučelje samo za vizualizaciju veza među entitetima. Izrađeno je 25 entiteta, a svi osim jednoga su međusobno povezani. Slika 3 prikazuje strukturu baze podataka.



Slika 3: Dijagram entiteta i veza baze podataka

Kreiranjem modela u datoteci `models.py` modeli se ažuriraju naredbama:

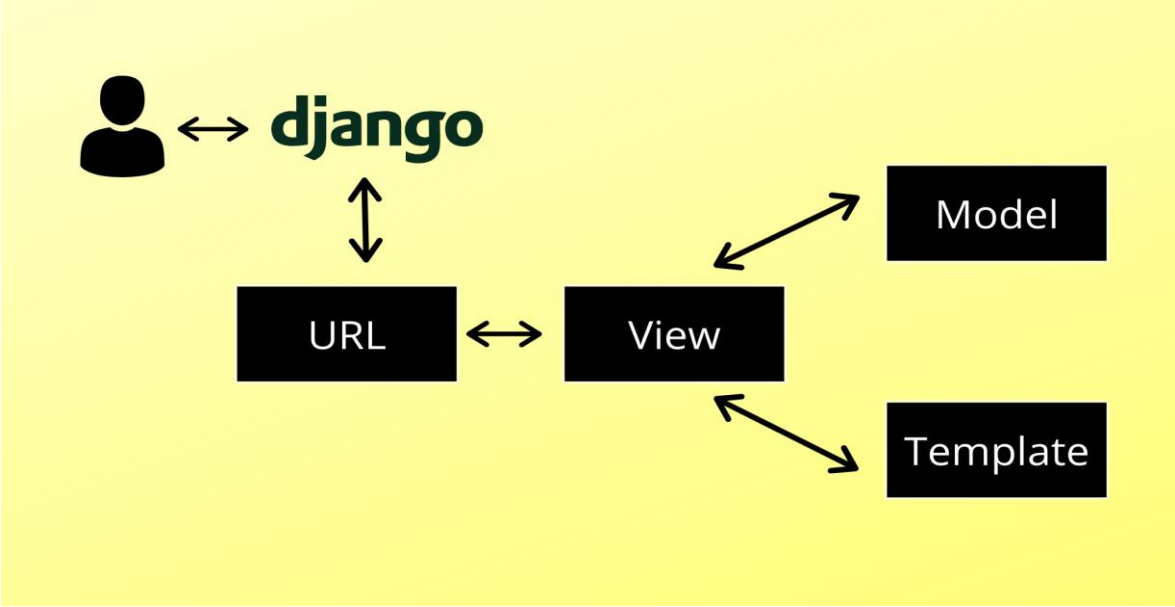
- `python manage.py makemigrations`
- `python manage.py migrate`

Nakon toga u MySQL ljusci se kreira baza podataka proizvoljnog imena (odabrano je `tournzilladb`). U njoj se onda kreiraju tablice čiji nazivi moraju sadržavati naziv direktorija u aplikaciji gdje se nalaze modeli (ovdje je to direktorij `api`) i naziv modela. Tako će tablica za prvenstvo imati naziv `api_championship` (direktorij `api` i model `Championship`). Spomenuta tablica je najvažnija u bazi podataka. Povezana je relacijom 1 prema više s tablicama `api_group`, `api_footballmatch`, `api_participant` i `api_redcard`. To znači da u jednom prvenstvu može biti više kvalifikacijskih grupa, utakmica, sudionika i crvenih kartona. Tablica `api_group` još ima vezu s tablicama `api_team` i `api_match`. U grupi se nalaze 4 tima pa tako ova tablica ima 4 polja od `team1` do `team4`. Unutar prvenstva tim može pripadati samo jednoj grupi. Također, u slučaju prvenstva s kvalifikacijama, prve utakmice su grupne i dodjeljuje im se broj grupe u polje `groupid`. Prvenstva su s timovima povezana poljima `winner` do `fourthplace`, ali i tablicom `api_participant` koja prikazuje sudioništvo timova na prvenstvu i služi za izbor dostupnih timova kada se kreira neko prvenstvo. Preciznije rečeno, kreiranjem prvenstva se provjerava s kojim od postojećih prvenstava se ono poklapa datumima i ako se nađu takva prvenstva, timovi koji se tu natječu odstranjeni su iz mogućih odabira. Tablice `api_goal`, `api_redcard`, `api_yellowcard`, `api_penalty`, `api_substitution` i `api_formation` su povezane vezama 1 prema 1 s tablicama `api_player`, `api_footballmatch` i `api_team`. Tako se za npr. gol bilježi igrač koji ga je ostvario, tim kojemu igrač pripada i utakmica koja se igra. Tablica `api_marketvalue` služi za praćenje vrijednosti igrača na tržištu kroz godine, tablica `api_formation` označava početnu postavu igrača u utakmici, a tablica `api_substitution` prati izmjene igrača tijekom utakmice. Tablica sa sucima je `api_referee` i povezana je s tablicom `api_refereeing`, koja sprema informacije o parovima datum-sudac kako bi se uoči utakmice moglo vidjeti koji suci su dostupni taj dan. Prijave neprimjerenog ponašanja korisnika spremaju se u tablicu `api_report` i njima upravlja administrator. Tablice za objave su `api_forum` i `api_news` te svaka od njih ima odgovarajuću tablicu za komentare (`api_comment` i `api_forumcomment`). Tablica `api_achievement` sadrži značke za aktivnost, a kada korisnik dobije značku to se bilježi u tablicu `api_userachievement` koja ima ulogu

posredničke tablice (engl. *junction table*). Tablica `api_fact` nema relacije jer zabavne činjenice ne ovise o drugim modelima, služe isključivo za zabavu korisnika.

4. 6. Pogledi i pomoćne funkcije

Pogledi definiraju povratnu vrijednost, odnosno sadržaj u pregledniku na putanji na kojoj se nalaze. Taj sadržaj može biti različitih vrsta, a u ovoj aplikaciji to je uglavnom tip podataka `HttpResponse` ili `JsonResponse`. Tu se može vidjeti odstupanje od uobičajenog korištenja pogleda i razilaženje poslužiteljskog dijela kôda od klijentskog, jer se u ovim funkcijama ne vraćaju nikakve komponente u HTML obliku. Pogledi su zaduženi isključivo za dohvaćanje i obradu podataka, odnosno za poslužiteljske zadatke.



Slika 4: Uobičajeni uzorak aplikacije razvojnog okvira Django

Slika 4 prikazuje strukturu samostalne aplikacije izrađene u razvojnom okviru Django. Ovakva arhitektura se naziva Model-Template-View (skraćeno MTV). Modeli su prikaz podataka iz baze kojima se upravlja pomoću pogleda, a predlošci oblikuju podatke u jeziku HTML ili nekom srodnom tako da se oni prikazuju krajnjem korisniku u pregledniku. S obzirom da ova aplikacija nije u potpunosti Django, izbačeni su predlošci. Uz URL se definira pogled koji rukuje modelima na način da kreira nove instance, dohvaća postojeće, uređuje ih, briše, sortira, filtrira i sl. (osnovne upravljačke operacije, engl. *CRUD – Create, Read, Update, Delete*). Pogledi su definirani u datoteci `views.py` i oni su svojevrsna

sučelja (engl. *interface*) između HTTP upita i pomoćnih funkcija. Te pomoćne funkcije sadrže implementaciju pogleda i logike rada s podacima (modelima) ovisno o vrsti primljenog HTTP upita.

```
@api_view(['GET', 'POST'])
@authentication_classes([])
@permission_classes([])
def getTeams(request: HttpRequest):
    if request.method == 'GET':
        return utils.getTeams(request)
    if request.method == 'POST':
        return utils.createTeam(request)


@api_view(['GET', 'PUT', 'DELETE'])
@authentication_classes([])
@permission_classes([])
def getTeam(request: HttpRequest, pk):
    if request.method == 'GET':
        return utils.getTeam(request, pk)
    if request.method == 'PUT':
        return utils.updateTeam(request, pk)
    if request.method == 'DELETE':
        return utils.deleteTeam(request, pk)
```

Ispis 14: Pogledi za timove

Pogled `getTeams` služi za dohvaćanje timova i kreiranje novog tima. Poziva dvije funkcije, `getTeams` (ako primi `GET` upit) i `createTeam` (ako primi `POST` upit), obje na istom URL-u. Za dohvaćanje pojedinog tima, uređivanje njegovih informacija i brisanje tog tima koristi se drugi pogled `getTeam`. U taj pogled je osim HTTP upita poslan i broj koji označava polje `id` u modelu `Team`, kako bi se znalo koji tim se dohvaća, uređuje ili briše. Na sve poglede se dodaju dekoratori. Dekorator `@api_view` je iz paketa Rest framework. On označava da se radi o pogledu i definira koji HTTP upiti su dozvoljeni na toj putanji. Dekorator `@authentication_classes` određuje koje autentikacijske klase su potrebne na tom pogledu. Prazna lista označava da nije potrebna autentikacija. Slično tome, dekorator `@permission_classes` provjerava autorizaciju, odnosno definira klase za dozvole koje su potrebne za provedbu metoda iz pogleda. Autentikacijske i autorizacijske klase za sve poglede su postavljene na prazne liste jer se autentikacija provjerava na drugi način, pomoću JWT tokena. Za svaki model su dodani odgovarajući pogledi na sličan

način kao za model tima. Na primjer, za nogometnu utakmicu napravljen je pogled koji dohvaća i briše utakmice te dodaje novu, ali samo za neko prvenstvo jer utakmice pripadaju prvenstvima i ne mogu postojati bez njih. Za rad s pojedinom utakmicom, osim broja za prvenstvo (`cid`) šalje se i broj za utakmicu (`mid`), a dozvoljene radnje su `GET` i `PUT`. Dakle, može se dohvatiti i uređivati utakmica (rezultati, kartoni, promjene igrača itd.).

```
@api_view(['GET', 'POST', 'DELETE'])
@authentication_classes([])
@permission_classes([])
def getMatches(request: HttpRequest, cid):
    if request.method == 'GET':
        return utils.getMatches(request, cid)
    if request.method == 'POST':
        return utils.createMatch(request, cid)
    if request.method == 'DELETE':
        return utils.deleteMatches(request, cid)
@api_view(['GET', 'PUT'])
@authentication_classes([])
@permission_classes([])
def getMatch(request: HttpRequest, mid, cid):
    if request.method == 'GET':
        return utils.getMatch(request, mid, cid)
    if request.method == 'PUT':
        return utils.editMatch(request, mid,cid)
```

Ispis 15: Pogledi za sve utakmice prvenstva i za pojedinu utakmicu

Pomoćne funkcije za `GET` upite su vrlo kratke i međusobno slične. Sastoje se od vraćanja podataka modela prevedenih u format JSON. Na primjer, kada korisnik pregleda listu svih postojećih timova to je omogućeno pomoćnom funkcijom `getTeams`, a kada posjeti stranicu profila nekog tima, koristi se funkcija `getTeam`.

```
def getTeams(request: HttpRequest):
    teams = Team.objects.all().exclude(id=500).order_by('-id')
    serializer = TeamSerializer(teams, many=True)
    return HttpResponse(json.dumps(serializer.data))

def getTeam(request: HttpRequest, pk):
    teams = Team.objects.get(id=pk)
    serializer = TeamSerializer(teams, many=False)
    return HttpResponse(json.dumps(serializer.data))
```

Ispis 16: Pomoćne funkcije za dohvaćanje svih i pojedinog tima

Kod dohvaćanja svih timova oni su poredani po identifikacijskom broju (`id`) kako bi oni koji su najnoviji bili prikazani prvi. Također, isključen je tim s identifikacijskim brojem 500 jer je to lažni tim imena „Unknown“. Taj tim služi za popunjavanje podataka utakmica koje se još trebaju igrati i kojima se ne znaju sudionici. Na primjer, u nekom prvenstvu se natječe 8 timova. Sudionici prve 4 utakmice su poznati, odnosno to su 4 para načinjena od tih timova. Međutim, kreiranjem prvenstva generira se struktura svih utakmica (engl. *bracket*) unaprijed jer je poznato koliko utakmica će se igrati (na osnovu broja timova i tipa prvenstva), ali nije poznato koji timovi će pobijediti u prvim utakmicama i ići dalje (slika 5). Zato se prilikom kreiranja utakmica s nepoznatim sudionicima koristi lažni tim jer polja `team1` i `team2` u tablici `api_footballmatch` ne mogu biti `NULL`.

Slika 5: Struktura utakmica prvenstva s poznatim i nepoznatim sudionicima

```
def getChampionships(request: HttpRequest):
    champ = Championship.objects.all().order_by('-championshipid')
    serializer = ChampionshipSerializer(champ, many=True)
    return HttpResponse(json.dumps(serializer.data))

def getUser(request: HttpRequest, pk):
    user = User.objects.get(id=pk)
    serializer = UserCreateSerializer(user, many=False)
    return HttpResponse(json.dumps(serializer.data))
```

Ispis 17: Pomoćne funkcije za dohvaćanje svih prvenstava i pojedinog korisnika

Datoteka za pomoćne funkcije se naziva `utils.py` (engl. *utility* – pomoćna funkcija). Funkcije koje rade s `GET` upitima za povratnu vrijednost imaju tip podataka `HttpResponse`. Taj format dopušta daljnju manipulaciju podacima, odnosno ručno postavljanje podataka na bilo koji drugi tip. Za pomoćne funkcije koje obrađuju `POST`, `PUT` i `DELETE` upite koristi se `JsonResponse`, potklasa klase `HttpResponse`. Kao što sam naziv kaže, vraća podatke u formatu JSON, odnosno na HTTP upit kojega obrađuje dodaje parametar `Content-Type: application/json`.

Jedna od najvažnijih pomoćnih funkcija kreiranja modela je `createChampionship`. Kreira prvenstvo na osnovu podataka koje primi u HTTP upitu s klijentskog dijela. Prvo instancira objekt klase `Championship` i pridruži mu svojstva, neka od njih iz upita, a neka zadana. Razlog tomu je ograničenost odabira na formi s klijentskog dijela. Tamo se može odabrati samo naziv prvenstva, broj timova, kvalifikacije (da ili ne), država domaćin, datum početka, je li prvenstvo prijateljsko, spol igrača i broj dana odmora između utakmica. Sva ta polja su obavezna pa se ovdje dohvaćaju iz HTTP upita pomoću objekta `request.POST`. Taj objekt se ponaša kao rječnik i njegovim vrijednostima se pristupa pomoću ključeva. Dakle, ime prvenstva se postavlja na `request.POST['name']` i tako redom. Ostale vrijednosti nije moguće unijeti prilikom kreiranja prvenstva (slika, maskota, troškovi organizacije itd.) nego tek nakon kreiranja, a neke vrijednosti se ne mogu uopće mijenjati. Dvije su vrste tih nepromjenjivih podataka. Prva vrsta su spomenuti obavezni podaci koji su uneseni prilikom kreiranja prvenstva (broj sudionika, spol, kvalifikacije itd.). Od početnih podataka samo ime se može promijeniti. Druga vrsta su podaci koji se automatski mijenjaju s vremenom i njima upravlja mehanizam odlučivanja (engl. *decider*) na klijentskom dijelu. To su podaci o pobjedniku, statusu prvenstva, najboljem igraču i sl. Polje `status` je uvijek postavljeno na „`Not started`“ jer se prvenstvo mora kreirati bar jedan dan prije početka. Polje `startDate` se odabire, ali polje `endDate` ne. Ono se računa formulama na klijentskom dijelu.

Važno je napomenuti da ova pomoćna funkcija ne kreira sudionike niti utakmice, a ne poziva ni metode koje ih kreiraju. Isključivo kreira model prvenstva, a za ostale komponente zaduženi su odgovarajući upiti na klijentskom dijelu (npr. za utakmice, nakon kreiranja prvenstva njegov identifikacijski broj se odmah dohvati i klijentski dio upućuje HTTP upit na putanju za kreiranje utakmica i prosljeđuje identifikacijski broj novokreiranog prvenstva). Kada su sva polja dodijeljena objektu prvenstva, ono se sprema

u bazu, serijalizira i vraća u obliku formata JSON za daljnji rad s njime.

Sljedeća pomoćna funkcija je kreiranje sudionika, prikazana u ispisu 18. Prima samo broj prvenstva kao parametar u putanji i broj tima kroz HTTP upit i kreira model sudionika. Koristi se za filtriranje dostupnih timova u slučaju preklapanja datuma prvenstava jer označava koji tim je sudionik kojeg prvenstva i od kada do kada je zauzet.

```
def createParticipant(request: HttpRequest, cid):
    partc = Participant()
    partc.tournamentid = cid
    partc.teamid = request.POST['teamid']
    partc.save()
    serializer = ParticipantSerializer(partc, many=False)
    return JsonResponse(serializer.data)
```

Ispis 18: Pomoćna funkcija kreiranja jednog sudionika

Važniju ulogu ima funkcija `createMatch`. Kreira utakmice u sklopu prvenstava, čiji se broj prosljeđuje kao dio putanje `/tournaments/<int:cid>/matches/` gdje je `cid` broj prvenstva. Može se pozvati 1 ili više puta jer prvenstvo može imati i samo jednu utakmicu. U HTTP upitu se moraju navesti timovi koji igraju (u slučaju da nisu poznati koristi se lažni tim), datum, vrijeme početka i kraja i mogućnost izjednačenog rezultata. Također se može, ali ne mora, navesti polje za suca (pošto nacionalnost suca ne smije biti ni jedna od država tih dvaju timova). Polje je ništavno jer u slučaju nepoznatih timova nije moguće znati koje nacionalnosti sudac ne smije biti pa se sve do početka utakmice stavlja `NULL`. I broj grupe se može i ne mora navesti, ovisno je li utakmica u sklopu grupe, a o tome također ovisi i polje `drawpossible` (indikator za moguć izjednačen završetak, za grupne utakmice je `True`, za eliminacijske `False`). Polje koje označava broj sljedeće utakmice je isto tako opcionalno. Prvenstva s dva tima imaju samo jednu utakmicu koja nema sljedeću, a za sve druge vrijedi veza sa sljedećom (osim u slučaju posljednje, odnosno finalne utakmice). Za sva polja osim gore opisanih koriste se zadane vrijednosti.

Opisane pomoćne funkcije rade s tekstualnim, brojčanim i vremenskim podacima, a funkcija `createTeam` uvodi novi tip podataka – fotografije. Za njih se također koristi jedan objekt poput rječnika, `request.FILES`. Služi za pristup datotekama koje se prilažu u HTTP upitu, a podržava razne vrste datoteka (fotografije, dokumente, zvučne zapise, video zapise, arhive, tekstualne datoteke itd.). Koristi se kao i `request.POST` (vrijednostima se

pristupa pomoću ključeva). Kada se šalje HTTP upit koji sadrži datoteke, tijelo takvog upita mora biti vrste `multipart/form-data`. Stoga se u upit dodaje zaglavlje `Content-Type` postavljeno na tu vrstu.

Pomoćna metoda kreiranja tima se sastoji od 3 dijela. U prvom dijelu se provjerava ispravnost poslanih podataka, jedinstvenost imena tima i izbornika (rijetko kada će dva ili više izbornika imati isto ime i prezime, a ako se to dogodi, dodaju im se nadimci).

```
def createTeam(request: HttpRequest):
    r = requests.get("http://localhost:8000/api/teams/")

    for x in json.loads(r.content):
        if (x['name'] == request.POST.get('name',False)):
            return JsonResponse({"message":"Team with chosen name
already exists."})
        elif (x['coachname'] == request.POST.get('coachname', False)):
            return JsonResponse({"message":"Team with this coach
already exists."})

if request.POST['name'] == '' or request.POST['coachname'] == '':
        return JsonResponse({"message":"Please fill in all required
fields."})
    if 'coachdesc' in request.POST and len(request.POST['coachdesc'])
> 200:
        return JsonResponse({"message":"Coach description too long. It
should be up to 200 characters."})
```

Ispis 19: Provjera ispravnosti podataka za kreiranje tima

Provjerava se i izostanak imena i je li opis izbornika duži od 200 znakova. U slučaju bilo koje neispravnosti vraća se poruka s objašnjenjem koji podatak je neispravan i tom porukom se dalje upravlja na klijentskom dijelu. Jedinstvenost podataka se provjerava slanjem upita na putanju za postojeće timove. Zatim se postavljaju svojstva modela na vrijednosti iz HTTP upita ili zadane ako ih u upitu nema. Svojstva tima obuhvaćaju mnogo informacija o njemu. Sljedeći korak je provjera fotografija u HTTP upitu. Pet polja ovog modela su fotografije. To su logo, slika izbornika, slika tima na listi svih timova i dva dresa (domaći i gostujući). Prilikom kreiranja tima moguće je priložiti sve (ili neke) fotografije, a nije obavezno priložiti nijednu. Za njih također postoje zadane vrijednosti.

```
if ('coachphoto' in request.POST):
    team.coachphoto = request.POST['coachphoto']
elif ('coachphoto' in request.FILES):
    now = datetime.now()
    img = Image.open(request.FILES['coachphoto'])
    coachphotoFileName = "media/"+now.strftime("%d-%m-%Y-%H-%M-
%S")+str(request.FILES['coachphoto'])
    img.save(coachphotoFileName)
    team.coachphoto = "../../" + coachphotoFileName
team.save()
    serializer = TeamSerializer(team, many=False)
    return JsonResponse(serializer.data)
```

Ispis 20: Postavljanje fotografije izbornika tima

Ako fotografija nije priložena u HTTP upitu, šalje se zadana vrijednost, a to je `../../media/default_photo.jpeg` (dodane su dvije oznake za roditeljski direktorij jer se fotografijama pristupa na klijentskom dijelu pa se sprema putanja iz perspektive klijentskog dijela). Ako je fotografija priložena, traži se u objektu `request.FILES`. Ona se zatim otvara i sprema u varijablu `img`. Daje joj se ime koje će sadržavati datum i vrijeme spremanja spojeno s imenom priložene fotografije (datum i vrijeme se dodaje u slučaju više fotografija s istim imenom). Potom se fotografija pohranjuje na navedenu lokaciju i podaci se zapisuju u bazu podataka, a objekt serijalizira i vraća u formatu JSON. Na sličnom principu se kreiraju i ostali modeli podataka. Moguće je kreirati igrača, ali samo unutar tima. Kasnije ga se može izbrisati ili umiroviti, ali prilikom kreiranja mora biti dio tima. Iako se igrač kreira na putanji koja uključuje broj tima, to je moguće promijeniti u formi i u tom slučaju postaviti tim iz objekta `request.POST`. Kreirati se mogu i objave, rasprave i komentari gdje se navodi autor i sadržaj (i naslov gdje to polje postoji). Za utakmice u tijeku kreiraju se golovi, kartoni i promjene igrača. Pomoćne funkcije koje ih kreiraju također utječu na svojstva modela `Player`, `Team` i `Championship`. Na primjer, kada se kreira gol (uobičajeni ili penal), dohvaća se igrač koji ga je postigao, tim kojemu pripada, protivnički tim koji je primio gol i prvenstvo kojemu utakmica pripada. Igraču i njegovu timu se sveukupni golovi (polje `goals`) uvećavaju, a također i prvenstvu (bilježi se broj svih postignutih golova unutar prvenstva), a protivničkom timu se broj primljenih golova uvećava za 1 (polje `concededgoals`). Isto vrijedi i za kartone, ali oni ne utječu na svojstva protivničkog tima. Gol se može i poništiti te se onda koristi pomoćna metoda `nullifyGoal` koja pronađe odgovarajući gol po broju i briše ga iz baze, a igraču i timu

smanji broj golova za 1, dok protivničkom timu broj primljenih golova smanji za 1.

```
def nullifyGoal(request: HttpRequest, mid, gid):
    goal = Goal.objects.get(matchid=mid, id=gid)
    goal.delete()
    player = Player.objects.get(id=goal.scoredby)
    player.goals = player.goals - 1
    player.save()
    tm = Team.objects.get(id=goal.team)
    tm.goals = tm.goals - 1
    tm.save()
    fmatch = FootballMatch.objects.get(matchid=mid)
    tm2 = Team.objects.get(fmatch.team1 if goal.team != fmatch.team1
else fmatch.team2)
    tm2.concededgoals = tm2.concededgoals - 1
    tm2.save()
```

Ispis 21: Poništavanje gola

Kao što je vidljivo u ispisu 21, Django pretragu nekog objekta obavlja pomoću funkcije `get` koja ima ulogu upita na bazu (engl. *query*), a poziva se nad atributom `objects` koji je ugrađen za svaki model. Funkcija `get` vraća samo jedan objekt tog modela (ukoliko je pronađen) i kao parametar prima neke uvjete kojih može biti jedan ili više. Ovdje se šalje broj utakmice (`mid`) i broj gola (`gid`). Kad je gol pronađen poništava se na razini igrača, tima i utakmice te konačno briše iz baze uz poruku potvrde u formatu JSON. Za razliku od gola, kartoni i penali su konačni i ne mogu se poništiti. Čak i da se sudac predomisli u vezi kartona ili postupi nepromišljeno, ne može povući odluku i u praksi treba dobro razmisliti o tome i konzultirati se s pomoćnim sucima. Zbog toga ne postoje funkcije brisanja jednog kartona ili penala. Međutim, ono što postoji i za kartone i golove je brisanje u množini, na razini tima ili prvenstva što dalje okida brisanje ovih objekata. Na primjer, brisanjem tima iz baze nema potrebe za pohranom njegovih žutih kartona pa se i oni brišu. Dohvate se metodom `filter`, također nad atributom `objects`, koja vraća listu.

```
def deleteRedCards(request: HttpRequest, mid):
    redcards = RedCard.objects.filter(matchid = mid)
    redcards.delete()
    return HttpResponse({"message":"Red cards of match are deleted"})
```

Ispis 22: Brisanje svih crvenih kartona utakmice

Uz funkcije `filter` i `get`, nad atributom `objects` se može izvršiti i metoda `all` koja vraća sve objekte tog modela, također kao listu. Ispis 23 prikazuje upotrebu te metode koja dohvaća sve forumske rasprave. Nad tom metodom se dalje izvršava metoda `order_by` koja prima atribut po kojemu će poredati objekte (za forum je to datum objave rasprave). Navođenjem minusa ispred riječi poredak će biti silazan, a njegovim izostavljanjem bit će uzlazan.

```
def getForum(request: HttpRequest):
    forumPosts = ForumPost.objects.all().order_by('-publishdate')
    serializer = ForumSerializer(forumPosts, many=True)
    return HttpResponse(json.dumps(serializer.data))
```

Ispis 23: Dohvaćanje svih rasprava foruma

Nazivanje pomoćnih funkcija je napravljeno na način da su funkcije dohvaćanja počinjale riječju „get“, funkcije kreiranja riječju „create“, funkcije uređivanja riječju „edit“ ili „update“, a funkcije brisanja riječju „delete“. U funkcijama uređivanja serijalizator se koristi za postavljanje novih vrijednosti na postojeću instancu nekog modela. U nastavku je naveden primjer za ažuriranje komentara na forumu. Dohvaćena je instanca komentara pomoću broja rasprave (`pid`) i broja komentara (`cid`). Od podataka primljenih s klijentskog dijela kreiran je rječnik, pomoću dekodiranja tijela HTTP upita. Dekodiranje je obavezno ako se podaci za ažuriranje dohvaćaju svojstvom `request.body` jer ono podatke sadrži u obliku niza bajtova. Koristi se uz zaglavlje `Content-Type: application/json`, a za tip `multipart/form-data` podacima se pristupa svojstvom `request.POST`.

```
def editForumComment(request: HttpRequest, pid, cid):
    comm = ForumComment.objects.get(forumid=pid, id=cid)
    somedata = json.loads(request.body.decode('utf-8'))
    serializer = ForumCommentSerializer(instance=comm, data=somedata)
    if serializer.is_valid():
        serializer.save()
    else:
        print(serializer.errors)
    return JsonResponse(serializer.data)
```

Ispis 24: Uređivanje postojećeg komentara na forumu

5. Implementacija klijentskog dijela

U ovom poglavlju je opisana izvedba klijentskog dijela aplikacije, razmještaj vizualnih komponenti, prikaz podataka različitim korisničkim ulogama, formule za donošenje odluka o rangiranju timova i igrača te vrednovanju rezultata. Autentikacija je također implementirana na klijentskoj strani te će se pružiti uvid u upravljanje stanjem uz pomoć alata Redux. Ovo poglavlje opisuje i oblikovanje aplikacije stilskim jezikom CSS.

5. 1. Početna stranica

Kao što je već spomenuto, početna stranica sadrži karticu s poveznicom na utakmice svih prvenstava koje su u tijeku, karticu sa zabavnim činjenicama o nogometu koje korisnik, neovisno o korisničkim ulogama, može označiti kao njemu poznate ili nepoznate i karticu koja vodi na listu svih korisnika kategoriziranih po korisničkim ulogama. Osim ovog sadržaja, početna stranica sadrži i naslov i podnaslov (naziv i opis aplikacije). Zabavne činjenice se dohvaćaju iz baze podataka pomoću modela `api_fact`. Osim dohvaćanja za prikaz korisniku, moguće ih je i dodavati, ali samo od strane korisnika s ulogom administratora. Za administratora na ovoj stranici postoji i dodatni gumb za kreiranje novih činjenica koji otvara modalni prozor za unos. Jedina provjera ispravnosti unosa sastoji se u tome da polje za unos ne smije biti prazno. Nakon što je činjenica uspješno unesena, pojavljuje se modalni prozor s potvrdom u trajanju od 2 sekunde. Prikaz činjenice je izveden pomoću metode `random` objekta `Math`, koja vraća pseudoslučajni broj u nekom rasponu.

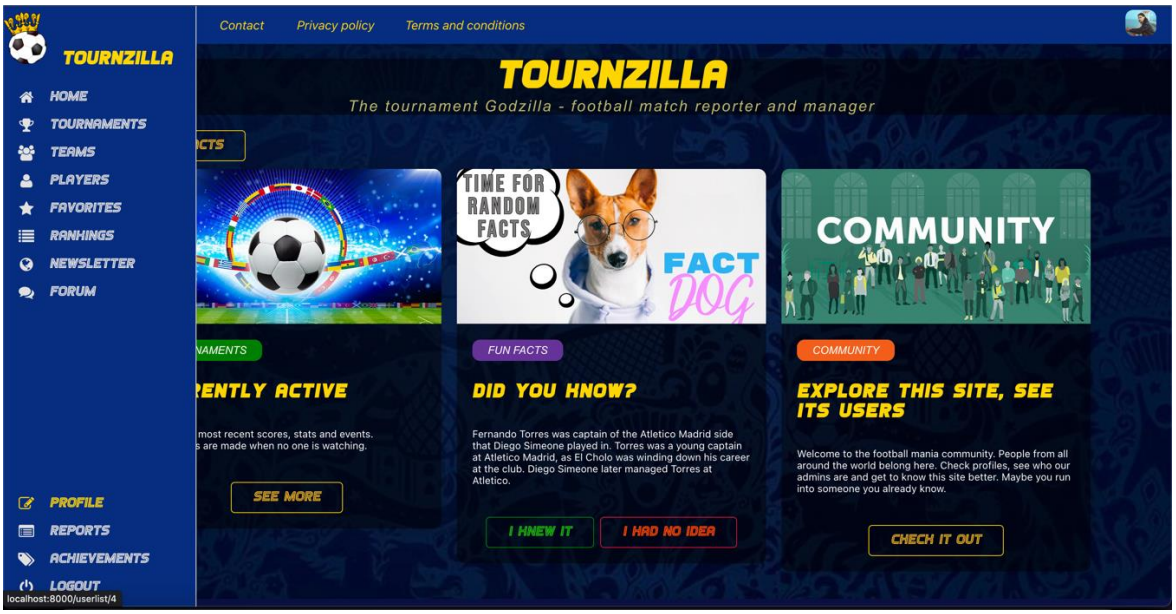
```
useEffect(() => {
    props.checkAuthenticated();
    props.loadUser();
    fetch('/api/facts/')
    .then((response) => response.json())
    .then(val => setFact(val[Math.floor(Math.random()*val.length)]))
  }, []);
```

Ispis 25: Dohvaćanje i prikaz zanimljivih nogometnih činjenica

5. 2. Navigacijske trake

U aplikaciji postoje dvije navigacijske trake za kretanje po različitim putanjama. Jedna se nalazi na vrhu stranice i sadrži uglavnom nepromjenjiv sadržaj, a to obuhvaća odredbe i

uvjete (engl. *terms and conditions*), politiku privatnosti (engl. *privacy policy*), informacije o stranici (engl. *about us*) i kontakt. Na ovoj traci se nalazi i gumb koji kontrolira temu na zahtjev korisnika, a također, ako je korisnik prijavljen na stranicu, na traci je i mala slika profila korisnika s desne strane, dok je ostali navedeni sadržaj s lijeve (slika 6).



Slika 6: Početna stranica za administratora, sadržaj navigacijskih traka

Što se tiče lijeve navigacijske trake, njen sadržaj je mnogo bogatiji i širina joj je prilagodljiva. Na njoj se nalaze poveznice na sve stranice aplikacije. Te poveznice su u obliku ikone i opisa. Opis se prikazuje samo prelaskom kursora preko površine trake, a u protivnom se vide samo ikone. Svi korisnici na traci vide poveznicu na početnu stranicu, prvenstva (turnire), timove, igrače, poredak timova i igrača, novosti, forum i prijavu, odnosno odjavu sa stranice. Prijavljeni korisnici imaju još i favorite, profil i značke, a administratori i listu prijava neprimjerenog ponašanja korisnika. I gornja i lijeva traka su prisutne na svim putanjama aplikacije. Na lijevoj traci se provjerava vrijednost teme koja je pohranjena u lokalnom spremniku (engl. *local storage*). Kada se tema dobije, ovisno o tekstu u lokalnom spremniku (može biti „dark“ i „light“), trenutna stranica se osvježava i ta tema dalje vrijedi na razini cijele aplikacije dok je korisnik ponovno ne promijeni. S ovom trakom, koja je komponenta naziva `NavBar`, povezuje se stanje aplikacije. Ono je proslijeđeno kao parametar u funkciju `mapStateToProps` koja dalje vraća objekt sa svojstvima kojima se može upravljati u komponentama aplikacije. Za potrebe upravljanja stanjem kreiraju se reduktori koji opisuju načine ažuriranja stanja. Reduktori ove aplikacije tiču se autentikacije korisnika i iz njih se dohvaćaju svi podaci koji razlikuju korisničke

uloge. S obzirom na to da se navigacijska traka nalazi na svakoj od stranica aplikacije, bilo je logično tu upravljati stanjem.

```
function setTheme () {
    window.localStorage.setItem('theme',
document.querySelector('#switch:checked') !== null ?
JSON.stringify("light") : JSON.stringify("dark"));
    window.location = window.location
}
```

Ispis 26: Promjena teme na navigacijskoj traci

```
  return (
    <div className="area">
    <nav className="main-menu">
        <ul>
          <li>
             <Link to={"/"}>
               <i className="fa fa-home fa-2x"></i>
               <span className="nav-text">Home</span>
             </Link>
          </li>
          <li className='has-subnav'> {props.isAuthenticated
             ?(<Link to={`/favorites/${props.user?.id}`}>
              <i className="fa fa-star"></i>
              <span className="nav-text">Favorites</span>
             </Link>) : null}
          </li>
          <li> props.isAuthenticated && props.user?.is_staff
            ?(<Link to={`/reports`} onClick={() => visit(`/reports`)}>
             <i className="fa fa-list-alt" aria-hidden="true"></i>
             <span className="nav-text">Reports</span>
            </Link>) : null}
          </li>
        </ul>
  )

const mapStateToProps = state => ({
    isAuthenticated: state.auth.isAuthenticated,
    user: state.auth.user
});
export default connect(mapStateToProps, {checkAuthenticated, loadUser,
logout})(NavBar);
```

Ispis 27: Mapiranje stanja, navigacija i provjera autentikacije korisnika

5. 3. Autentikacija

Početno stanje je organizirano kao objekt koji ima tokene za autentikaciju, svojstvo koje provjerava je li korisnik autenticiran te svojstva `message` i `user`. Tokeni za autentikaciju su `refresh token` (pohranjuje se u lokalni spremnik i može se koristiti za obnavljanje `access tokena`) i `access token` (obnovljiv, kratkog životnog vijeka, sadržan u zaglavlju HTTP upita). Svojstvo `isAuthenticated` je inicijalno postavljeno na vrijednost `null`. Svojstvo `message` će korisnika obavijestiti o procedurama autentikacije, najčešće poruke pogreške tijekom registracije i prijave, a svojstvo `user` dohvaća objekt autenticiranog korisnika.

```
const initialState = {
    access: localStorage.getItem('access'),
    refresh: localStorage.getItem('refresh'),
    isAuthenticated: null,
    message:null,
    user: null
};
```

Ispis 28: Početno stanje autentikacije

Napravljene su i neke konstante koje predstavljaju tip radnji (`type`) koje će ažurirati stanje. One obavljaju učitavanje korisnika, uspješnu prijavu, odjavu, registraciju, potvrdu e-maila, ponovno postavljanje lozinke i aktivaciju korisničkog računa. Na primjer, u slučaju uspješne registracije stanje se ažurira na način da je poruka o grešci postavljena na vrijednost `null`, a svojstvo `isAuthenticated` na vrijednost `true`. U slučaju uspješne prijave dobiveni `access token` se dohvaća iz objekta `payload` koji sadrži neke dodatne podatke o stanju vezanih za radnje. Između ostalog sadrži tokene, korisničke podatke za registraciju i prijavu, identifikacijske oznake za aktivaciju računa i sl. Odjava i neuspješna prijava su jednake po pitanju upravljanja stanjem. Uklanjaju tokene iz lokalnog spremnika i sve korisničke podatke postavljaju na zadano. Uspješna aktivacija računa i ponovno postavljanje lozinke ne mijenjaju stanje, a bezuspješne radnje samo dodaju poruku o pogrešci kako bi korisnik bolje razumio problem s autentikacijom. Postoji i konstanta `EMPTY_FIELDS` koja će provjeravati sva obavezna polja za unos, odnosno nedostatak podataka u njima. Kada se korisnik uspješno registrira, primit će poruku o registraciji i daljnjoj proceduri koja uključuje provjeru maila na koji se šalje poveznica za potvrdu računa, jer samom registracijom račun je kreiran, ali nije aktivan i nije se moguće prijaviti

sve dok se ne klikne na pristiglu poveznicu.

```
export default function(state=initialState, action) {
    const { type, payload } = action;
    switch(type) {
        case AUTHENTICATED_SUCCESS:
        case LOGIN_SUCCESS:
            return {
                ...state,
                message:null,
                isAuthenticated: true
            }
        case LOGIN_FAIL:
        case LOGOUT:
            localStorage.removeItem('access');
            localStorage.removeItem('refresh');
            return {
                ...state,
                access: null,
                refresh: null,
                message: payload?.detail ?? unknownErrorMsg,
                isAuthenticated: false,
                user: null
            }
        case REGISTER_FAIL:
            localStorage.removeItem('access');
            localStorage.removeItem('refresh');
            return {
                ...state,
                access: null,
                refresh: null,
                message: payload?.username
                    ? "Username: " + payload.username[0]
                    : payload?.password
                    ? "Password: " + payload?.password[0]
                    : payload?.email
                    ? "Email: " + payload.email[0]
                    : payload?.non_field_errors
                    ? "Re-password: " + payload.non_field_errors[0]
                    : unknownErrorMsg,
                isAuthenticated: false,
                user: null
            }
```

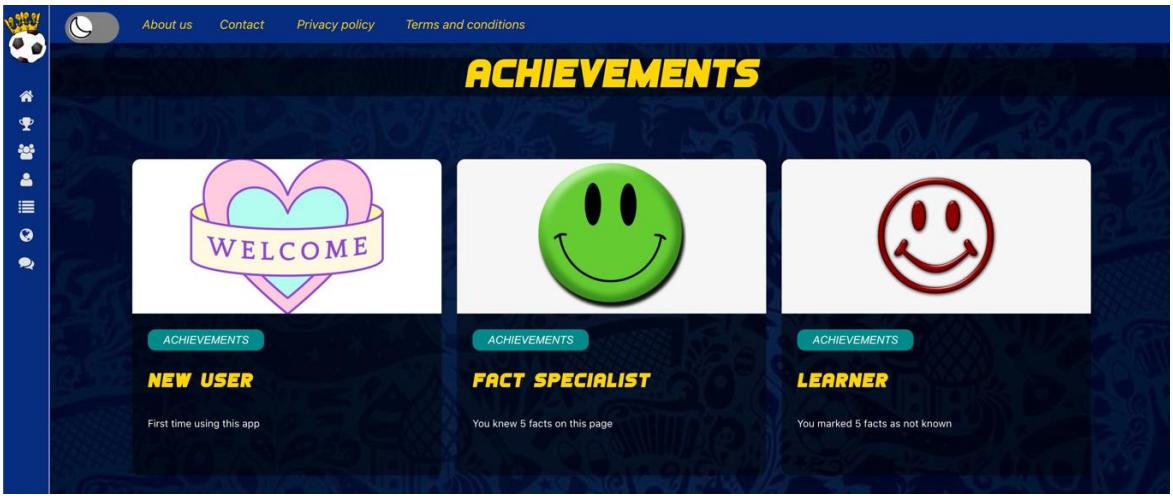
Ispis 29: Autentikacija i upravljanje stanjem

Za lakše upravljanje stanjem kreira se funkcija `combineReducers` čija je svrha kombiniranje reduktora u jednu funkciju, odnosno jedinstveni reduktor. Na taj način na

različitim dijelovima aplikacije pojedini reduktor može upravljati određenim sadržajem stanja.

```
const initialState = {
    access: localStorage.getItem('access'),
    refresh: localStorage.getItem('refresh'),
    isAuthenticated: null,
    message:null,
    user: null
};
```

Ispis 30: Kombiniranje reduktora



Slika 7: Korisničke značke za aktivnost

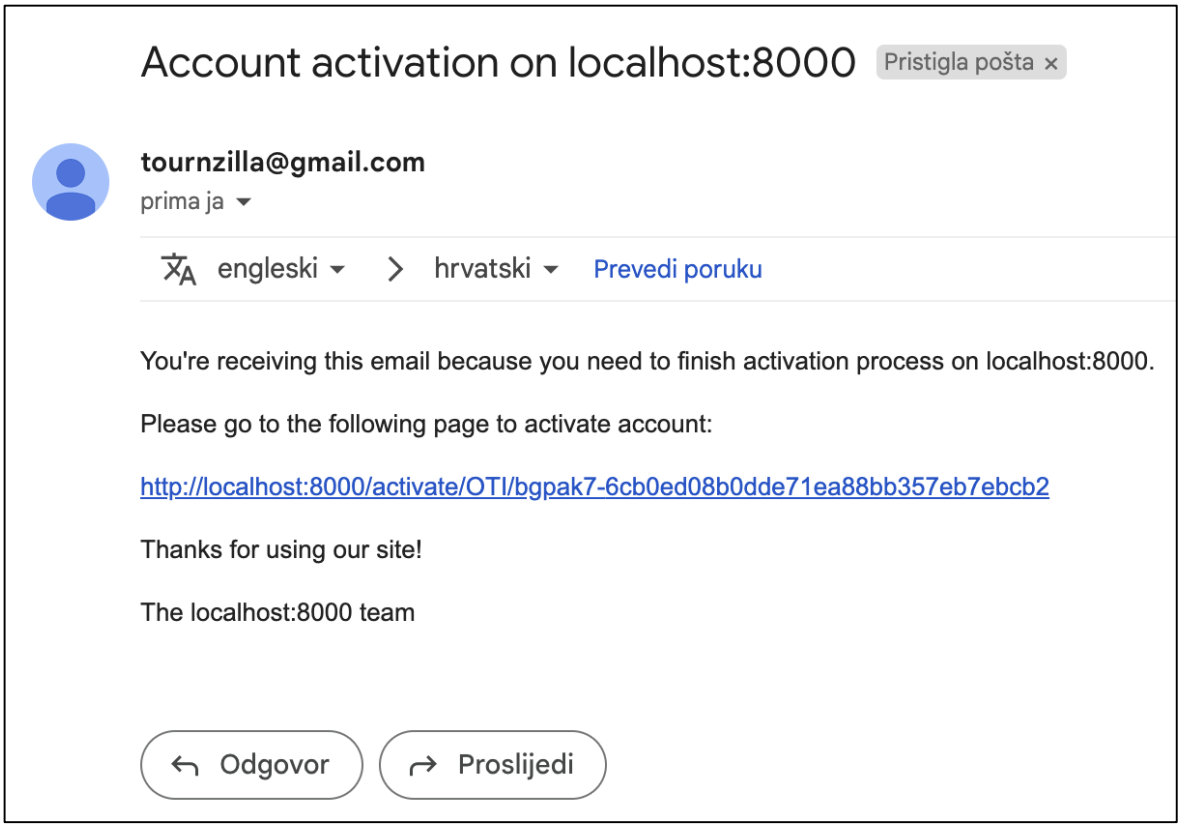
Za funkcionalnost autentikacije zadužene su radnje (engl. *actions*) koje obavljaju HTTP upite i komunikacijom s poslužiteljem upravljaju korisničkim podacima. Metoda `loadUser` dohvaća korisničke podatke i ovisno o vrsti odgovora kojeg primi s poslužitelja, otprema radnju `USER_LOADED_SUCCESS` ili `USER_LOADED_FAIL`. Putanje na kojima se ove radnje izvršavaju su iz biblioteke Djoser i već su automatski konfigurirane na poslužitelju samom instalacijom ove biblioteke. Učitavanje korisnika informacije dohvaća s putanje `/auth/users/me/` dodanu na osnovni URL (`http://localhost:8000`) koji je spremljen kao tzv. „proxy“ poslužitelj kako bi se upiti sa klijentskog dijela usmjeravali prema poslužiteljskom. Radnja učitavanja korisnika može rezultirati uspjehom i neuspjehom (ako nije pronađen odgovarajući korisnik, odnosno njegov `id`, uslijed greški poslužitelja, neispravnih tokena i sl.) i potrebno je ažurirati stanje sukladno tome. Također su potrebna i zaglavlja upita, autorizacija (`access token` iz lokalnog spremnika) i tip sadržaja (u ovom

slučaju format JSON).

Na sličnom principu funkcioniraju i ostale radnje te su one uglavnom asinkrone funkcije koje vraćaju druge funkcije (engl. *thunk functions*). Tako, na primjer, funkcija `checkAuthenticated` provjerava postojanje i ispravnost tokena u lokalnom spremniku. Putanja koju koristi je `/auth/jwt/verify/`, a ispravnost `access` tokena doznaje provjerom svojstva `code`. Samo ako je to svojstvo različito od vrijednosti `'token_not_valid'` token će biti prepoznat kao ispravan.

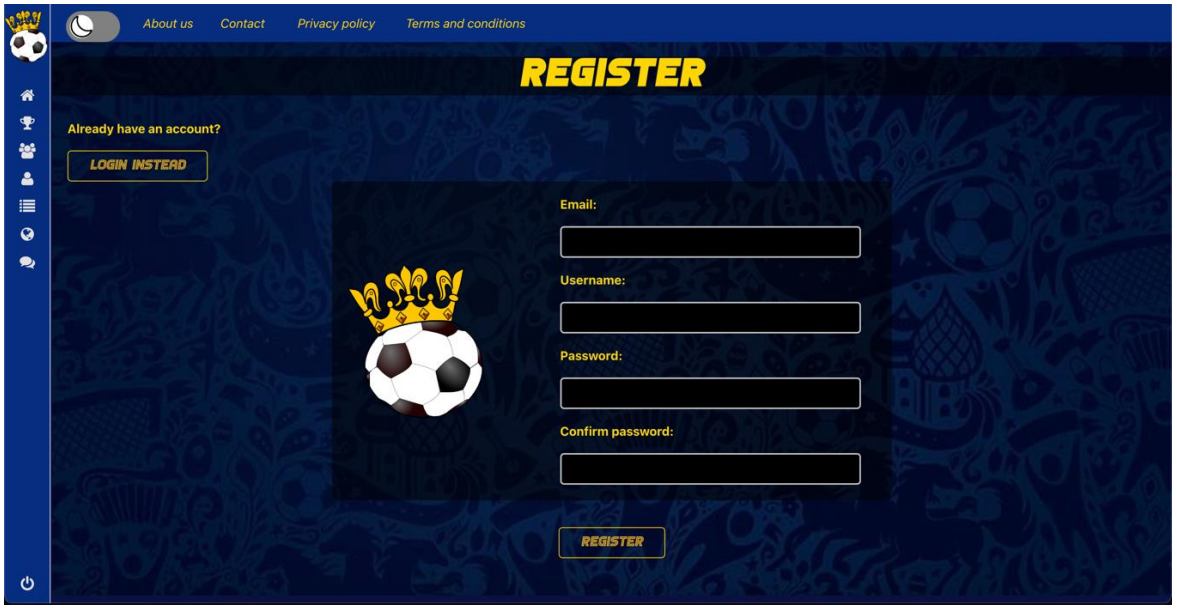
Radnja za prijavu korisnika prima uneseno korisničko ime i lozinku i šalje `POST` upit na putanju `/auth/jwt/create/`, ali ih prije toga uspoređuje s praznim vrijednostima (`EMPTY_FIELDS`) i ako se podudaraju, `payload` se postavlja na `null` što znači se korisnički podaci neće učitati i prijava je neuspješna. Ako ni jedan od unosa nije prazan, provodi se `POST` upit gdje su uneseni podaci tijelo upita, a zaglavlje je postavljeno da prihvaća bilo kakav oblik tekstualnog unosa. Ako je u odgovoru sadržan `access` token to znači da je prijava uspješna i otprema se radnja `loginUser`. U svim ostalim slučajevima došlo je do pogreške i korisnički podaci se resetiraju na zadane vrijednosti (`null`).

Za registraciju korisnika, slično kao za prijavu, je potreban rad s korisničkim unosom, ali ovoga puta s korisničkim imenom i emailom (koji moraju biti jedinstveni) i lozinkom unesenom dva puta. Ovisno o ispravnosti unesenim podataka i jedinstvenosti nekih od njih, korisniku se prikazuju poruke s obavijestima. Obavijest napisana zelenim slovima označava uspješnu registraciju, a poruke crvene boje neuspješnu uz razlog neuspjeha. Ako je prijava uspješna, korisnik na navedeni email dobiva poruku s poveznicom za aktivaciju računa, s obzirom da registracija samo dodaje korisnika u bazu, ali mu polje `is_active` postavlja na 0 sve dok ne aktivira računa klikom na poveznicu. To slanje poruka na mail je moguće zahvaljujući postavci `EMAIL_HOST` na poslužiteljskom dijelu u datoteci `settings.py` koje je postavljeno na vrijednost `tournzilla@gmail.com`. To je email s kojeg će pristizati sve poruke vezane za aplikaciju. Slika 8 prikazuje jednu takvu poruku.



Slika 8: Obavijest o registraciji i poveznica za aktivaciju

Razlozi neuspješne registracije su obično nepodudaranje lozinki, korisničko ime koje već postoji, email koji već postoji, ali i neki drugi razlozi koji su uključeni u biblioteku Djoser. To može biti prejednostavna lozinka, lozinka koja uključuje korisničko ime, potpuno numerička lozinka i sl. Također, za uspješnu registraciju se dobiva i prva značka aktivnosti korisnika. To je neka vrsta priznanja u obliku kartice na kojoj piše da se korisnik pridružio zajednici. Naziv značke je "New User" i dodjeljuje se novokreiranom korisniku putem polja `id`. To polje se dobiva slanjem `POST` upita na putanju `/auth/users/`, odnosno dohvaćanjem odgovora tog upita, a podaci značke se šalju na `/api/userachievements/` na što se još dodaje korisnikov `id`. Broj ove značke u bazi podataka je 3. Iako korisnički račun registracijom nije aktiviran, on se prikazuje na stranici svih korisnika i može se pregledati taj profil koji uključuje i značke. Iz tog razloga se ona odmah dodaje pa je drugi korisnici i gosti mogu vidjeti. Izgled značke za novog korisnika (uz još neke značke) se nalazi na slici 7, a forma za registraciju na slici 9.



Slika 9: Forma za registraciju novog korisnika

```
export const register = (email, uname, passwd, re_passwd) => async
dispatch => {if (email === "" || uname === "" || passwd === "" ||
re_passwd === "") {
        dispatch({
            type: EMPTY_FIELDS,
            payload: null
        }); else { try {
 await fetch (`${process.env.REACT_APP_API_URL}/auth/users/`, {
       method: 'post',
       body: JSON.stringify({
         "username":uname,
         "email" : email,
         "password":passwd,
         "re_password": re_passwd})
      }).then(data => data.json()).then(response => {
        const fd = new FormData()
        fd.append("achievementid",3)
        if (response.id) {
            fetch(`/api/userachievements/${response.id}/`, {
                method: 'post',
                body: fd
            })
            dispatch({
               type: REGISTER_SUCCESS,
               payload: response
            });
```

Ispis 31: Uspješna registracija korisnika i dodjela značke

Poveznica pristigla na mail sadrži putanju za potvrdu korisničkog računa, a ta putanja glasi `/activate/` na koju se dodaju slučajno generirani `uid` od 3 simbola i `token` od 39 simbola. Nakon što je korisnik aktivirao račun, taj token prestaje vrijediti i svakim novim klikom na poveznicu prikazuje se poruka da je token iskorišten. Osim ovih radnji implementirana je i radnja za ponovno postavljanje lozinke (prima email i šalje ga u `POST` upitu na `/auth/users/reset_password/`, a korisniku na taj mail šalje poveznicu koja sadrži putanju `/password_reset/confirm/` uz `uid` i `token`), radnja za potvrdu nove lozinke (kada korisnik klikne poveznicu na mailu, na nju se vrši `POST` upit s novom lozinkom unesenom dva puta i, ovisno o podudaranju i pravilima za lozinku, ažuriraju se novi podaci ili se prikazuje poruka s greškom).

5. 4. Tijek prvenstva

Turnir (prvenstvo) se može sastojati od jedne ili više utakmica. Može biti prijateljsko ili službeno te može imati grupne kvalifikacije. Kreiranje prvenstva izvodi se iz dva koraka. U prvom koraku se unose opće informacije o prvenstvu od kojih su sve obavezne i imaju zadane vrijednosti u slučaju izostavljanja unosa. Naziv prvenstva (nije jedinstven), broj timova koji se natječu, ima li grupnih kvalifikacija (omogućeno samo ako je broj timova djeljiv s 4), država domaćin, datum početka, je li prvenstvo prijateljsko ili ne, je li prvenstvo muško, žensko ili mješovito i vrijeme odmora između utakmica se navode ovdje. Nakon unosa potrebno je kliknuti na gumb za sljedeći korak, a ako je neki od unosa neispravan, prikazuje se obavijest. U protivnom se otvara stranica sa sljedećim korakom a to je odabir timova koji će se natjecati. Prikazuju se samo dostupni timovi, a oni se dobivaju usporedbom datuma početka i kraja svih postojećih prvenstava s datumima prvenstva koje se kreira. Kada se navede vrijeme početka prvenstva, metoda `calculateEndDate` na osnovu broja timova i vremena odmora generira datum kraja. Za prvenstva od samo 2 tima datum kraja jednak je datumu početka (jer se igra samo jedna utakmica). Za 3 tima se igraju dvije utakmice i datum kraja se računa dodavanjem broja dana odmora na datum početka na što se dodaje još jedan dan. To je zato jer utakmice koje ovise jedna o drugoj se ne smiju igrati isti dan, mora proći barem 24 sata između njih, a neovisne se mogu igrati isti dan. Prvenstvo može imati pravilnu ili nepravilnu strukturu. Ako je broj timova potencija broja 2, struktura je pravilna i timovi se samo uparuju te se kreiraju utakmice za svaki par za prvu rundu prvenstva. Sve te utakmice se igraju isti dan jer su neovisne, nijedna od njih nema zajedničkog sudionika. Sljedeća runda ima upola

manji broj utakmica i sudionici se dobivaju prolaskom timova iz prve runde. Sve dok se ne dođe do jedne utakmice, kreiraju se runde smanjivanjem broja timova za polovinu. Ta utakmica je za treće i četvrto mjesto pa se nakon nje kreira još jedna dodatna, odnosno finalna utakmica.

```
  function calculateEndDate(startDt, quals, size, teamRest) {
    let totalDays = 0; let restVar = 0; let tempEnd = new
Date(startDt);
    let tempNoOfTeams = size
    if (teamRest === '1 day') {restVar = 1}
    else if (teamRest === '2 days') {restVar = 2}
    else if (teamRest === '3 days') {restVar = 3}
    if (parseInt(size) === 2) {
      setTournamentData(prevState => ({...prevState, endDate:
startDt}));
    } else if (parseInt(size) === 3) {
      tempEnd.setDate(tempEnd.getDate() + restVar + 1);
      setTournamentData(prevState => ({...prevState, endDate:
formatDateUS(tempEnd)}));
    } else {
      if (quals === "Yes") {
        if (size === 4) {
          totalDays += (2 + 3 * restVar);
        }
        totalDays += (3 + 3 * restVar);
        tempNoOfTeams = tempNoOfTeams / 2;
      }
      if (!isPotentionOf2(parseInt(tempNoOfTeams))) {
        totalDays += restVar + 1
      }
      while(!isPotentionOf2(parseInt(tempNoOfTeams))) {
        tempNoOfTeams -= 1;
      }
      while (tempNoOfTeams > 2) {
        totalDays += 1 + restVar
        tempNoOfTeams /= 2
      }
      totalDays += 1
      tempEnd.setDate(tempEnd.getDate() + totalDays);
      setTournamentData(prevState => ({...prevState, endDate:
formatDateUS(tempEnd)}));}
  }
```

Ispis 32: Metoda računanja završetka prvenstva

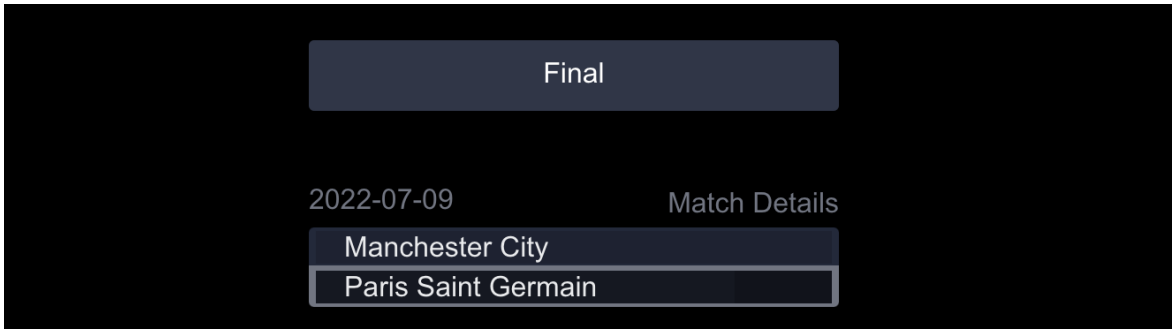
Prilikom kreiranja prvenstva moguće je vratiti se natrag i izmijeniti unos prvoga koraka te se svakim spremanjem podataka šalju odabrani parametri u putanju drugoga koraka. Ako

u drugome koraku nema dovoljno dostupnih timova, poruka o tome će se prikazati i upravitelj turnira ima mogućnost povratka na prethodni korak i izmjene unosa.

```
  function saveData() {
    tournamentData.wrongMsg = null;
    tournamentData.showWarning = false;
    if (tournamentData.name === '') {
      setWrongMsg("Tournament name cannot be empty.");
      setShowWarning(true);
    } else if (tournamentData.participantSize < 2 ||
tournamentData.participantSize > 32) {
      setWrongMsg("Incorrect number of teams. Must be between 2 and
32.");
      setShowWarning(true);
    } else if (new Date(tournamentData.startDate) <= new Date() || new
Date(tournamentData.startDate).toString() === 'Invalid Date') {
      setWrongMsg('Invalid date. Please select a valid day in the
future (tomorrow or later).');
      setShowWarning(true);
    } else {
      navigate(`/tournaments/create/add-
teams?tournamentData=${encodeURIComponent(JSON.stringify(tournamentDat
a))}`);
    }
  }
```

Ispis 33: Spremanje unosa prvog koraka kreiranja prvenstva

Najveći broj sudionika na prvenstvima je 32, a najmanji 2 (vidljivo na slici 10). Najraniji datum početka je sutrašnji dan. Ispravnim unosima svih polja i odabirom timova poziva se funkcija `processSave` koja kreira prvenstvo, provjerava pravilnost strukture te na osnovu toga kreira utakmice, sudionike, postave i suce. Za dva tima kreira jednu utakmicu (model `FootballMatch`), dva sudionika (model `Participant`), dvije postave (za svaki tim jedna, model `Formation`).



Slika 10: Izgled prvenstva s jednom utakmicom (dva tima)

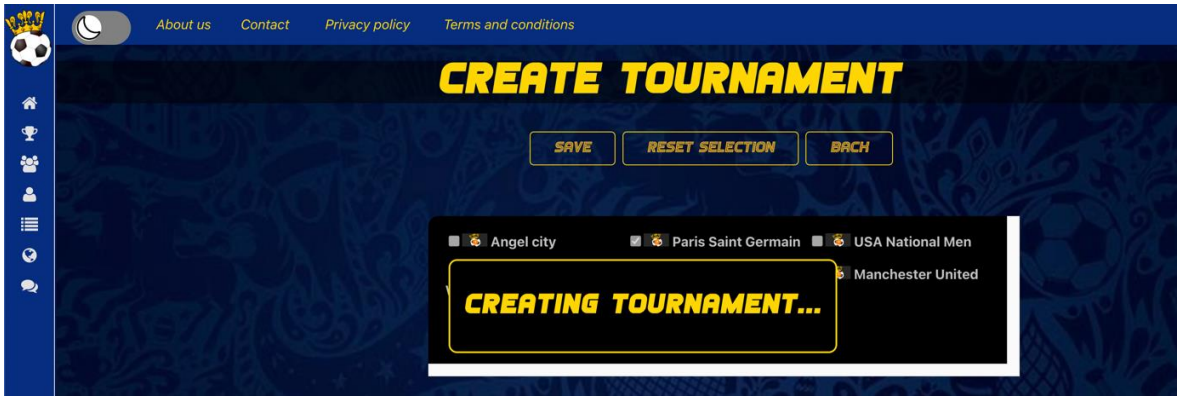
```
if (selectedTeams.length === 2) {
   const fData2 = new FormData()

   fData2.append("team1",selectedTeams[0])
   fData2.append("team2",selectedTeams[1])

   fData2.append("matchdate", tournamentData.startDate)
   fData2.append("starttime", `${startHour}:00:00`)
   fData2.append("endtime", formattedEndTime)
   fetch(`/api/tournaments/${t.championshipid}/matches/`, {
   method: 'POST',
   headers: {
     'Authorization': `JWT ${localStorage.getItem('access')}`
   },
   body: fData2
   })
}
```

Ispis 34: Kreiranje prvenstva za 2 tima

Objekt `tournamentData` se oblikuje pomoću primljenih vrijednosti iz prvog koraka, a varijabla `selectedTeams` od odabranih timova (u obliku komponente *checkbox*). To je niz u kojega se sprema `id` svakog tima. Duljina niza se uspoređuje s primljenim parametrom `totalteams` koji označava odabrani broj timova iz prvog koraka. Ako je duljina jednaka tom broju, može se dovršiti kreiranje turnira koje traje 8 sekundi. Tijekom tih 8 sekundi otvoren je modalni prozor koji obavještava da se proces kreiranja odvija (slika 11).



Slika 11: Obavijest o uspješnom kreiranju prvenstva

Tada se novo prvenstvo vidi na stranici `/tournaments` (datoteka `Tournaments.js`) gdje se nalaze sva prvenstva. Uz naziv se može vidjeti i status prvenstva. Za novokreirano prvenstvo je uvijek postavljen na „Not started“ jer se kreira barem dan ranije, a dalje te

statuse kontroliraju mehanizmi odlučivanja. Klikom na detalje nekog prvenstva otvara se stranica s općenitim podacima prvenstva, a na njoj se nalazi gumb za otvaranje stranice s utakmicama. Početak i kraj utakmica se postavljaju prilikom kreiranja prvenstva i dalje se ne mogu mijenjati, kao što se ni prvenstvo ne može odgoditi nego se mora izbrisati i ponovno kreirati. Za prvenstva s 3 tima je također situacija specifična. Prva dva tima igraju na dan početka, a nakon odmora pobjednik igra protiv trećega tima. Dakle, takvo prvenstvo ima 2 utakmice gdje su sudionici prve poznati, a jedan od sudionika druge je nepoznat (za nepoznate sudionike koristi se lažni tim čije je svojstvo `id` postavljeno na 500).

```
else if (selectedTeams.length === 3) {
    const nextStartDate = new Date(tournamentData.startDate);
    nextStartDate.setDate(nextStartDate.getDate() + restVar + 1)
    fData2.append("team1", selectedTeams[0])
    fData2.append("team2", selectedTeams[1])
    fData3.append("team1", selectedTeams[2])
    fData3.append("team2", 500)
    fData3.append("matchdate", formatDateUS(nextStartDate))
    fData2.append("starttime", `${startHour}:00:00`)
    fData2.append("endtime", formattedEndTime)
    fData3.append("starttime", `${startHour}:00:00`)
    fData3.append("endtime", formattedEndTime)
```

Ispis 35: Kreiranje prvenstva za 3 tima

Za slanje podataka u tijelo `POST` upita koristi se objekt `FormData` nalik rječniku. U njega se spremaju parovi ključ-vrijednost i on se koristi za modele podataka čije kreiranje uključuje i datoteke (u slučaju prvenstva slike maskote, otvaranja i sl.). Napravljene su i funkcije za pretvorbu prikaza datuma u američki i europski format jer baza podataka prima američki, a na stranici se uglavnom prikazuju u europskom. Za vrijeme igranja utakmica generira se slučajan broj između 0 i 23 (oba uključena).

```
const startHour = Math.floor(Math.random() * 24)
const start = new Date();
start.setHours(startHour, 0, 0);
const endTime = new Date(start.getTime() + 105 * 60000);
const endHours = endTime.getHours();
const endMinutes = endTime.getMinutes();
const formattedEndTime =
`${endHours}:${endMinutes.toString().padStart(2, '0')}:00`;
```

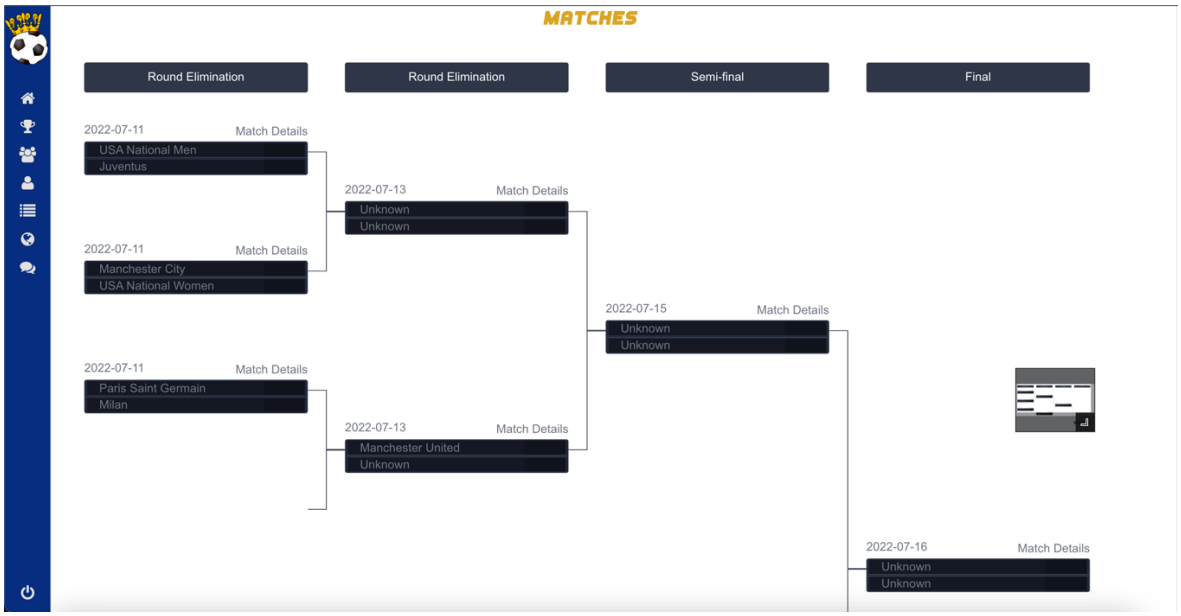
Ispis 36: Formatiranje vremena

Za sva prvenstva kojima je broj timova veći od 3, a ujedno i potencija broja 2, postupak je isti. Prva runda će sadržavati broj utakmica jednak broju timova podijeljen s 2 te će svaka runda raspolavljati broj utakmica. Na primjer, ako je broj timova 16, u prvoj rundi bit će 8 utakmica, u drugoj 4, u trećoj (polufinalu) dvije, te u borbi za 3. mjesto jedna uz još jednu koja je finalna. Po dvije utakmice iz svake runde imaju vezu na jednu utakmicu iz sljedeće runde. Postupak kreiranja utakmica pravilnih prvenstava je pokazan u ispisu 37.

```
const eliminationSize = parseInt(tournamentData.participantSize)
for (let mToCreate = matchId+1 ; mToCreate <= matchId +
eliminationSize - 1; mToCreate += 1 ) {
     if (loopStepCount === 3) { loopStepCount = 1 }
     const fdTmp = new FormData()
     if (opponents?.length > teamsCounter + 1 &&
formatDateUS(matchDateTmp) === tournamentData?.startDate) {
         fdTmp.append("team1", tournamentData?.qualifications ===
'Yes' ? 500 : opponents[teamsCounter])
         fdTmp.append("team2", tournamentData?.qualifications ===
'Yes' ? 500 : opponents[teamsCounter + 1])
     } else {fdTmp.append('team1', 500); fdTmp.append('team2', 500)}
     if (mToCreate === matchId + eliminationSize - 1) {
         fdTmp.append('nextmatchid', 0)
         fdTmp.append('matchdate', tournamentData.endDate)
     } else {
         fdTmp.append('nextmatchid', mToCreate + nextMatchIdCount)
         if (dateCounter > dateSetter) {
             dateSetter /= 2
             dateCounter = 1
             matchDateTmp.setDate(matchDateTmp.getDate() + restVar +
1);
         }
     fdTmp.append('matchdate', formatDateUS(matchDateTmp))
     }
     fdTmp.append("starttime", `${startHour}:00:00`)
     fdTmp.append("endtime", formattedEndTime)
fetch(`/api/tournaments/${t.championshipid}/matches/${mToCreate}/`, {
      method: 'PUT',
      headers: {'Authorization': `JWT
${localStorage.getItem('access')}`},
      body: fdTmp})
    if (loopStepCount === 2) { nextMatchIdCount -= 1 }
    loopStepCount += 1
    dateCounter += 1
    teamsCounter += 2
}
```

Ispis 37: Postavljanje datuma i sudionika utakmica pravilnih prvenstava

Datumi utakmica se mijenjaju po rundama. Sve utakmice prve runde se odvijaju isti dan (jer su svi timovi različiti) i to je dan početka prvenstva. U sljedećoj rundi se datum određuje dodavanjem dana odmora na taj datum i tako redom. Jedino se na finalnu utakmicu ne dodaju dani odmora nego se igra dan nakon utakmice za treće mjesto. Za prvenstva nepravilne strukture računa se koliko timova treba oduzeti da bi se dobila pravilna struktura. To znači računanje najbližeg manjeg broja koji je potencija broja 2. Na primjer, za prvenstvo sa 7 timova, najbliža manja potencija broja 2 je broj 4. Stoga od 7 treba oduzeti 3 tima da bi se dobilo njih 4. Tada se igraju 3 utakmice na početku prvenstva i tu sudjeluje 6 timova. Dakle, od njih 7 samo jedan ne igra u tim dodatnim utakmicama. Te utakmice su tzv. nepravilni dio prvenstva i uglavnom se ponašaju kao i pravilni uz neke razlike. Prva razlika je da nisu nužno potrebne dvije takve utakmice sa jednakim svojstvom `nextmatchid`. Nekad je dovoljna jedna, ovisno o broju timova. Također, sve utakmice nepravilnog dijela prvenstva se igraju u jednoj rundi i to na isti dan i sudionici tih utakmica su uvijek poznati. Slika 12 prikazuje nepravilno prvenstvo sa 7 timova.



Slika 12: Model nepravilnog prvenstva

Za dobivanje broja timova kojeg je potrebno oduzeti u slučaju pretvaranja nepravilnog prvenstva u pravilno implementirana je funkcija `extraPart` koja se poziva prilikom kreiranja takvih prvenstava. Toliko utakmica se kreira na dan početka prvenstva sa potrebnim brojem timova. Važno je napomenuti da prilikom odabira timova oni ne ostaju spremljeni u nizu onim redoslijedom kojim su odabrani već se nad nizom poziva funkcija

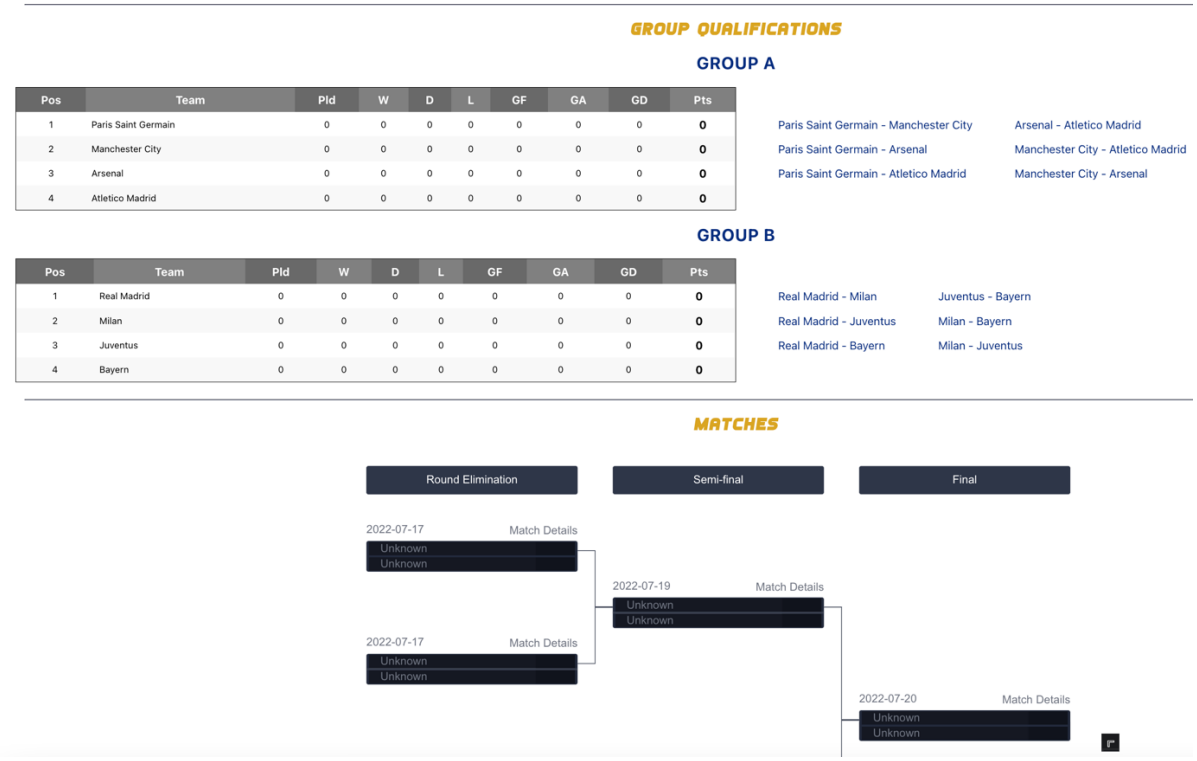
random koja ih pomiješa.

```
  function updateSelectedTeams(itemid) {
    setWrongMsg(null);
    setShowWarning(false);
    let count =
document.querySelectorAll('input[type="checkbox"]:checked');
    const checkboxes =
document.querySelectorAll('input[type="checkbox"]');
    if (count.length === parseInt(tournamentData?.participantSize)) {
      checkboxes.forEach((checkbox) => {
        checkbox.disabled = true;
      })
    }
    const checkbox =
document.querySelector(`#participantcheck${itemid}`);
    if (checkbox.checked) {
      count += 1;
      setSelectedTeams([...selectedTeams, itemid].sort(() =>
Math.random() - 0.5))
    } else {
      setSelectedTeams(selectedTeams.filter(x => x !== itemid));
      count -= 1;
    }
  }
```

Ispis 38: Validacija broja odabranih timova i postavljanje redoslijeda miješanjem

Navedene utakmice su eliminacijske, a postoje i grupne utakmice koje im mogu prethoditi. One služe za kvalifikacije timova na prvenstvo, najčešće u slučaju većeg broja timova, ali i neovisno o tome ako organizator prvenstva preferira kvalifikacije u prvenstvu. Kao što je već rečeno, sadrže po 4 tima gdje svaki igra protiv svakoga ciklički. To je sveukupno 6 utakmica po grupi. Završetkom grupnih utakmica broj timova se prepolovi i oni koji su prošli igraju eliminacijske utakmice (broj timova pa tako i utakmica može biti pravilan ili nepravilan). Stoga grupe služe za odmjeravanje snaga timova na drugi, možda čak i pošteniji način nego što je slučaj u eliminacijskim utakmicama, gdje nekad timovi imaju izravan prolazak u sljedeću rundu (u prvenstvima nepravilne strukture). Kod kreiranja prvenstva s grupnim kvalifikacijama svi timovi se podijele u grupe po 4 i kreiraju se grupe u kojima su navedena 4 tima, identifikacijska oznaka grupe (`id`) i slovo. Potom se grupe dohvaćaju, a utakmice koje se kreiraju imaju postavljeno svojstvo `groupid` na `id` odgovarajuće grupe. Ove utakmice nemaju poveznicu na sljedeću, sudionici svih njih su poznati i igraju se po dvije u istom danu. Dakle, prvi dan igraju prvi tim protiv drugoga, a

treći protiv četvrtoga. Zatim slijedi broj dana odmora između utakmica pa igraju prvi protiv trećega, drugi protiv četvrtoga. Nakon sljedećeg odmora igraju prvi protiv četvrtoga, a drugi protiv trećeg tima. Za svaku pobjedu u grupnim utakmicama tim dobiva 3 boda, za neriješen rezultat 1 bod i za gubitak 0 bodova. Bodovi se potom zbrajaju i prva dva tima po bodovima iz svake grupe ulaze u eliminacije. Slika 13 prikazuje grupne kvalifikacije.



Slika 13: Prvenstvo s grupnim kvalifikacijama, dvije grupe

```
fetch(`/api/groups/${props.id}/`)then(respo => respo.json())
.then(grps => { setGroups(grps) })
fetch(`/api/teams/`)then(tmResponse => tmResponse.json())
.then(tms => setTeams(tms))
```

Ispis 39: Dohvaćanje grupa za prikaz

```
let letterCount = 0;
for (let i = 0; i < selectedTeams.length; i += 4) {
    const letter = alphabet[letterCount]
    const grp = selectedTeams.slice(i, i + 4);
    const groupFD = new FormData();
    groupFD.append('championshipid', t.championshipid)
    groupFD.append('letter', letter);groupFD.append('team1', grp[0])
    groupFD.append('team2', grp[1]); groupFD.append('team3', grp[2])
    groupFD.append('team4', grp[3]); groupFD.append('winner', 'None')
    groupFD.append('second', 'None')
    fetch(`/api/groups/${t.championshipid}/`, {
        method: 'POST',
        headers: {
            'Authorization': `JWT ${localStorage.getItem('access')}`
        },
        body: groupFD
     })
     letterCount += 1
}
```

Ispis 40: Kreiranje grupa

Tijekom kreiranih prvenstava i njihovih utakmica dalje upravljaju mehanizmi odlučivanja koji na osnovu datuma i vremena prepoznaju o kojem dijelu utakmice se radi. Sve stranice na kojima se prikazuju prvenstva i utakmice pozivaju te mehanizme kako bi pružile ažurirane informacije. Nekad se pozivanje tih mehanizama obavlja samo prilikom učitavanja stranice, a nekad češće (npr. svakih 5 sekundi). Također, kada je posljednja utakmica prvenstva gotova, funkcije odlučivanja se pobrinu za proglašenje pobjednika i 3 mjesta ispod te najboljeg igrača i golmana.

5. 5. Mehanizmi odlučivanja

U datoteci `Deciders.js` se nalaze funkcije bitne za donošenje odluka o statusu prvenstva i utakmica. Služe za praćenje stanja u stvarnom vremenu i olakšavaju administratorima proces upravljanja prvenstvima, odnosno automatiziraju taj proces. U funkciji `tournamentDecider` provjerava se pojedino prvenstvo i usporedi s današnjim datumom. Ako je današnji datum između početka i kraja prvenstva, postavlja se polje `status` na vrijednost „In progress“, a ako je današnji datum kasniji u odnosu na kraj prvenstva postavlja se na „Finished“. Ova funkcija se najčešće poziva na listi prvenstava.

```
export function tournamentDecider (tournamentID) {
    const fd = new FormData()
    fetch(`/api/tournaments/${tournamentID}/`)
    .then(tRes => tRes.json())
    .then(tData => {
        if (new Date() >= new Date(tData.startDate) && new Date() <=
new Date(tData.endDate)) {
            fd.append('status', 'In progress')
            fetch(`/api/tournaments/${tournamentID}/`, {
                method: 'PUT',
                headers: { 'Authorization':
                           `JWT ${localStorage.getItem('access')}`
                },
                body: fd
            })
        }
    })
}
```

Ispis 41: Postavljanje statusa završenog prvenstva

Funkcija `singleMatchDecider` zadužena je za praćenje stanja pojedine utakmice, kao što su žuti kartoni, trenutna postava igrača, rezultati, statistike i promjene igrača. Isto tako služi za određivanje faze nogometne utakmice. Uspoređuje trenutno vrijeme s vremenom početka utakmice i vraća `string` čija vrijednost može biti `FIRST_HALF`, `HALVES_BREAK`, `SECOND_HALF`, `SECOND_BREAK`, `SECOND_EXTENSION`, `THIRD_BREAK`, `THIRD_EXTENSION` ili `PENALTIES`. Ove vrijednosti se postavljaju samo unutar 24 sata od početka utakmice, a ako je prošlo više, vraća se vrijednost „Played“ uz koji ide datum i vrijeme utakmice.

```
if (new Date() >= new Date(match.matchdate + "T" + match.starttime) &&
new Date() < new Date(new
Date(match.matchdate+"T"+match.starttime).getTime() +(45 * 60000))) {
        label = FIRST_HALF
        if (match.status === 'Not started') {
            const startFD = new FormData()
            startFD.append('status', 'In progress')
fetch(`/api/tournaments/${tournament.championshipid}/matches/${match.m
atchid}/`, {
        method:"put",
        headers: {
        'Authorization': `JWT ${localStorage.getItem('access')}`
    }, body: startFD
  })}
```

Ispis 42: Provjera prvog poluvremena

Za svaku utakmicu u slučaju prvog poluvremena status utakmice se sprema u bazu na vrijednost „In progress“ što označava da je utakmica počela. Pauza koja slijedi i drugo poluvrijeme samo vraćaju `string` koji kasnije služi kao indikator za mogućnost ažuriranja utakmice.

```
else if (new Date() >= new Date(new
Date(match.matchdate+"T"+match.starttime).getTime() +((105 +
parseInt(match.extension)) * 60000)) && new Date() < new Date(new
Date(match.matchdate+"T"+match.starttime).getTime() +((110 +
parseInt(match.extension)) * 60000))) {
        if (match.groupid) {
            let winner = null
            const groupEndFD = new FormData()
            if (match.score1 > match.score2) {
                winner = match.team1
                groupEndFD.append('winner', match.team1)
            } else if (match.score2 > match.score1) {
                winner = match.team2
                groupEndFD.append('winner', match.team2)
            }
            groupEndFD.append('status', 'Finished')
fetch(`/api/tournaments/${tournament.championshipid}/matches/${match.m
atchid}/`, {
     method:"put",
     headers: {
         'Authorization': `JWT ${localStorage.getItem('access')}`
     }, body: groupEndFD
})
    if (!tournament.friendly) {
        fetch(`/api/teams/${match.team1}/`)
        .then(res => res.json())
        .then(team1 => {
            fetch(`/api/teams/${match.team2}/`)
            .then(res2 => res2.json())
            .then(team2 => {
                winnerTeam = winner === match.team1 ? team1 : winner
=== match.team2 ? team2 : null;
                calculateEloRating(team1, team2, winner)
                calculateEloRating(team2, team1, winner)
        })
    })
 }
label = "Played " +  formatDateEU(match.matchdate) + " at " +
match.starttime + " GMT+0200"
}
```

Ispis 43: Ažuriranje utakmice nakon kraja drugog poluvremena

Ispis 43 prikazuje donošenje odluke za utakmicu nakon kraja drugog poluvremena i sudačke nadoknade. Ako je utakmica grupna, samo se završava. Ako je jedan od timova pobijedio, sprema se u bazu, a čak i ako ne, utakmica svejedno završava jer nije eliminacijska. Ako je utakmica eliminacijska provjerava se pobjednik na osnovu rezultata prvog i drugog tima. Ako su rezultati različiti pobjednik se bilježi u bazu i utakmica se završava. Nakon toga slijede provjera ima li utakmica sljedeću te je li sljedeća finalna, utakmica za treće mjesto ili obična eliminacijska utakmica i sukladno tome, pobjednički ili gubitnički tim se sprema kao jedan od sudionika sljedeće utakmice. Na primjer, ako je sljedeća utakmica za treće mjesto, onda tim koji je izgubio ide u tu utakmicu, a tim koji je pobijedio u finalnu. To je lako izvesti jer finalna utakmica uvijek ima polje `id` jednako polju `id` utakmice za treće mjesto uvećano za 1. To je zbog toga što se finalna utakmica kreira neposredno nakon utakmice za treće mjesto. Ako u eliminacijskoj utakmici nakon drugog poluvremena nema pobjednika, započinje pauza od 5 minuta

```
let winner = null
let secondplace = null
if (match.score1 > match.score2) {
    winner = match.team1
    secondplace = match.team2
} else if (match.score2 > match.score1) {
    winner = match.team2
    secondplace = match.team1
}
if (!winner) {
    label = SECOND_BREAK;
}
return label
```

Ispis 44: Postavljanje pauze nakon drugog poluvremena ako nema pobjednika

Nakon te pauze slijedi produžetak od 15 minuta pa još jedna pauza od 5 i još jedan produžetak od 15. Čak i ako u prvom produžetku dođe do promjene rezultata, drugi produžetak se igra svejedno. Ako ni nakon drugog produžetka nema pobjednika igraju se kazneni udarci, odnosno penali. Mogu trajati najdulje 24 sata, a ako ni tada nema pobjednika pobjeda se dodjeljuje prvome timu i uvećava mu se rezultat za 1 u bazi podataka. Prilikom izvođenja penala pojavljuje se modalni prozor koji je stalno aktivan sve do odluke o pobjedniku (ili do isteka 24 sata). Taj prozor sadrži jedan padajući izbornik za igrače prvom tima te jedan za igrače drugog tima. Uz njih su još dva padajuća izbornika u

kojima se za svaki penal označava je li pogodak ili promašaj.

```
fetch(`/api/penalties/${parseInt(props?.mid)}/`)
.then(res => res.json())
.then(data => {
  if (data.length >= 10) {
    const pens1 = data.filter(x => x.team === team1.id).filter(x => x.hit)
    const pens2 = data.filter(x => x.team === team2.id).filter(x => x.hit)
    const fd2 = new FormData()
    const fd3 = new FormData()
    if (pens1.length > pens2.length) {
       fd2.append('winner', team1.id)
       fd2.append('status', 'Finished')
       fetch(`/api/tournaments/${props.id}/matches/${props.mid}/`, {
         method: 'PUT',
         headers: {
           'Authorization': `JWT ${localStorage.getItem('access')}`
         },
         body: fd2
     }
);
```

Ispis 45: Odluka o pobjedniku nakon finala

Metoda `resolvePenalties` provjerava omjer pogodaka obaju timova i već nakon 10 penala (5 od svakoga tima) može donijeti odluku o pobjedniku. Ako je broj pogodaka za oba tima i dalje jednak, penali se nastavljaju. Slika 14 prikazuje formu za unos penala.

Slika 14: Izvođenje penala

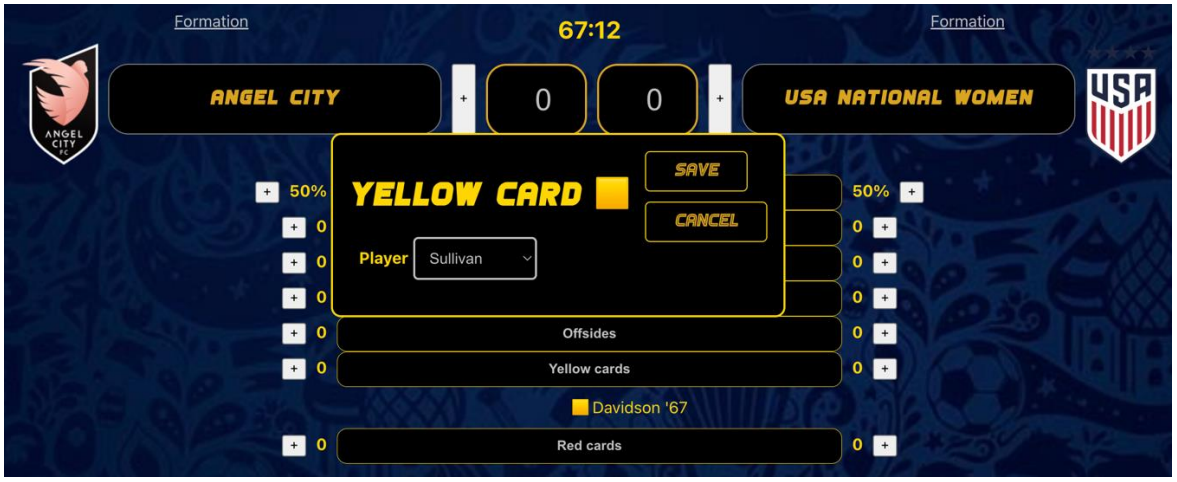
Slika 15: Prvo poluvrijeme

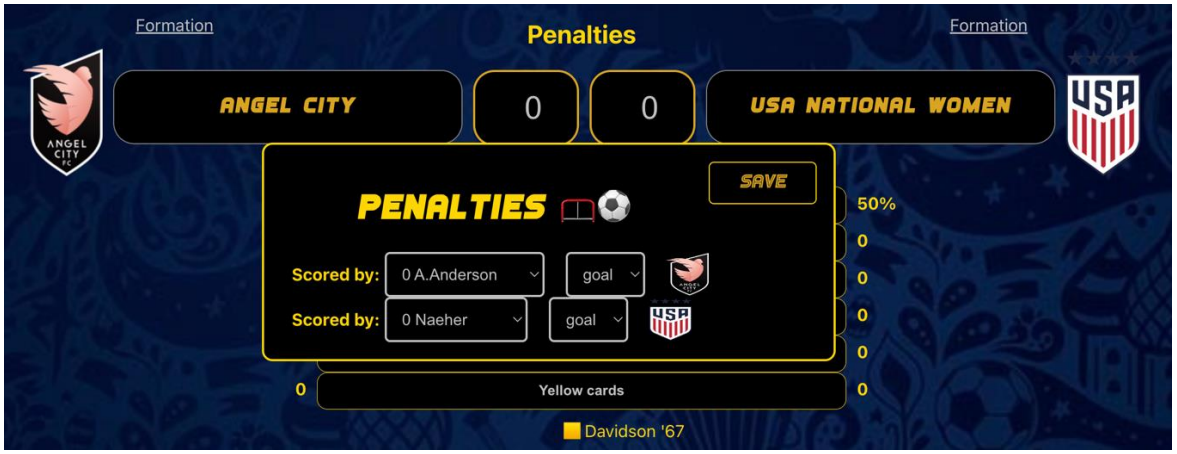
5. 6. Tijek utakmice

Tek kada je utakmica počela, administrator je može uređivati. To uključuje dodavanje i poništavanje golova, dodjeljivanje crvenih i žutih kartona, promjene igrača i dodavanje statistika (posjed, prekršaji, asistencije i sl.). U tu svrhu se pojavljuju modalni prozori gdje je to potrebno (gdje je moguć izbor) ili se na licu mjesta ažurira neko od polja (brojčane statistike). U poluvremenu i nadoknadama administrator može uređivati utakmicu (slike 15, 16 i 17), a u pauzama kada se ne igra svaki gumb za uređivanje nestane.

```
{isAdminLogged && !disable
? <button className={`${scoreChange} pointer-button`} id="plus1st"
onClick={() => opensModal ? openModalWithTitle(true) :
handleIncrement(true)}>+</button>
: null}
<label className={sideNumber}>{first}{isPercentage ? "%" :
null}</label>
<button className={matchSideStatsCard}>{title}</button>
<label className={sideNumber}>{isPercentage ? 100 - first :
second}{isPercentage ? "%" : null}</label>
{isAdminLogged && !disable ? <button className={`${scoreChange}
pointer-button`} id="plus2nd" onClick={() => {opensModal ?
openModalWithTitle(false) : handleIncrement(false)}}>+</button> :
null}
```

Ispis 46: Uvjetno prikazivanje gumba za uređivanje utakmice





Slike 16 i 17: Modalni prozori za žuti karton i izvođenje penala

5. 7. Elo rating

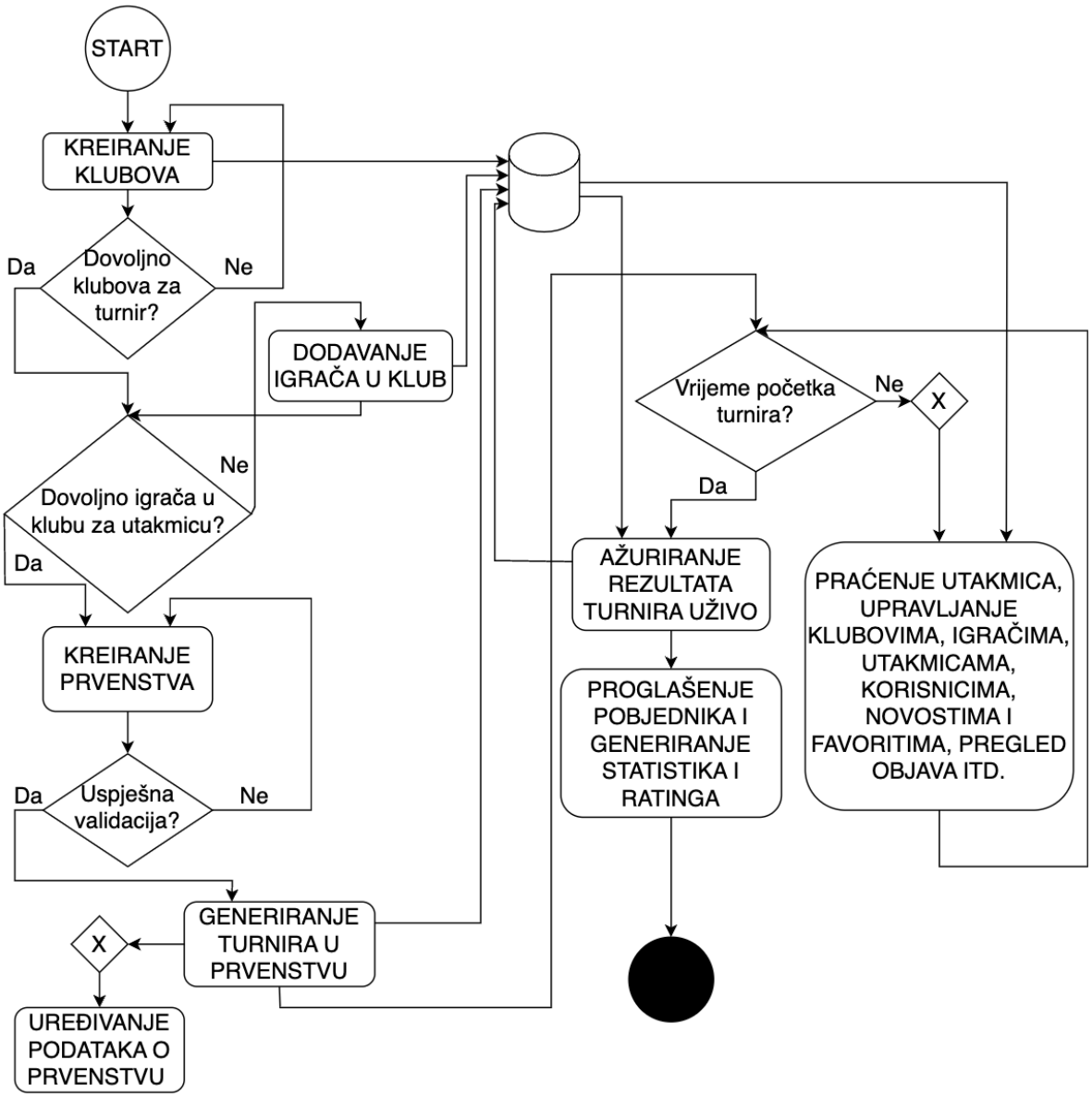
Sustav ocjenjivanja koji se koristi u igrama i sportovima naziva se Elo rating. Osmislio ga je mađarski fizičar Arpad Elo, po kojemu i nosi ime. Iako je sustav izvorno napravljen za šah, danas se najčešće koristi u video igrama i natjecateljskim sportovima. Ideja ovog sustava je da se na početku svakom timu dodijeli jednaka početna ocjena, koja se mijenja nakon svake utakmice na osnovu ishoda utakmice i postojećih ocjena timova. U obzir se uzimaju razni faktori, kao što su očekivani rezultat, stvarni rezultat i razlike u ocjenama između timova. Na taj način je omogućeno visoko pozicioniranje timova s niskim ocjenama u kratkom vremenu ako pobjede timove s višim ocjenama i obrnuto, brz pad timova s višim ocjenama ako izgube protiv tima s nižim. Prvo se očekivani rezultat tima računa po formuli $Ep = \frac{1}{1+10^{(Ro-Rp)/400}}$. Ep označava očekivani rezultat tima, Rp trenutnu ocjenu tima, a Ro trenutnu ocjenu protivničkog tima. Stvarni rezultat se dobiva ishodom utakmice sa stajališta promatranog tima (ako je pobijedio, stvarni rezultat je 1,

ako je neriješeno 0.5, a za gubitak je 0). Koristi se i koeficijent prilagodbe, tzv K-faktor, koji se proizvoljno određuje i konstantan je na razini cijele aplikacije. Ovdje je to 60 jer se ta vrijednost uglavnom koristi u sportovima. K-faktor se množi s razlikom stvarnog i očekivanog rezultata i dobivena vrijednost se naziva promjena ocjene, odnosno broj koji se dodaje na trenutnu ocjenu tima. Uvijek se dodaje, a nikad ne oduzima, jer je proces osmišljen da može vratiti pozitivne i negativne vrijednosti te tako prepoznaje oduzimanje. Na slici 18 prikazan je poredak timova i igrača po ocjenama.

Slika 18: Poredak timova i igrača

```
export function calculateEloRating(player, opponent, winner) {
    const playerRating = player.rating
    const opponentRating = opponent.rating
    const actualScore = winner === player ? 1 : winner === opponent ?
0 : 0.5
    const kFactor = 60
    const expectedScore = 1/(1 + (10**((opponentRating -
playerRating)/400)))
    const rateChange = kFactor * (actualScore - expectedScore)
    const newRating = playerRating + rateChange
    const fd = new FormData()
    fd.append('rating', parseInt(newRating))
    fetch(`/api/teams/${player.id}/`, {
        method: 'PUT',
        headers: {
            'Authorization': `JWT ${localStorage.getItem('access')}`
        },
        body: fd
    })
}
```

Ispis 47: Računanje ocjena timova metodom Elo rating



Slika 19: Dijagram toka prvenstva (*workflow*)

5. 8. Paket @g-loot

Prikaz tijeka eliminacijskih utakmica je omogućen paketom @g-loot brackets. Taj paket pruža vizualizaciju nogometnih utakmica u obliku stabla. Može se oblikovati po volji i ima mogućnost vezivanja utakmica ako se za svaku utakmicu navede sljedeća. Ovaj paket omogućuje i komponente kao što je `SingleEliminationBracket`, koja na osnovu slijeda utakmica kreira sliku u formatu SVG gdje su sve te utakmice povezane. Šalju joj se imena timova koji sudjeluju, vrijeme igranja utakmice, pobjednik (za završene utakmice) i nazivi rundi. Ako prvenstvo ima mnogo utakmica, omogućen je i smanjeni prikaz koji služi za povlačenje i pozicioniranje unutar slike. Za svaku utakmicu postoji i poveznica na detalje.

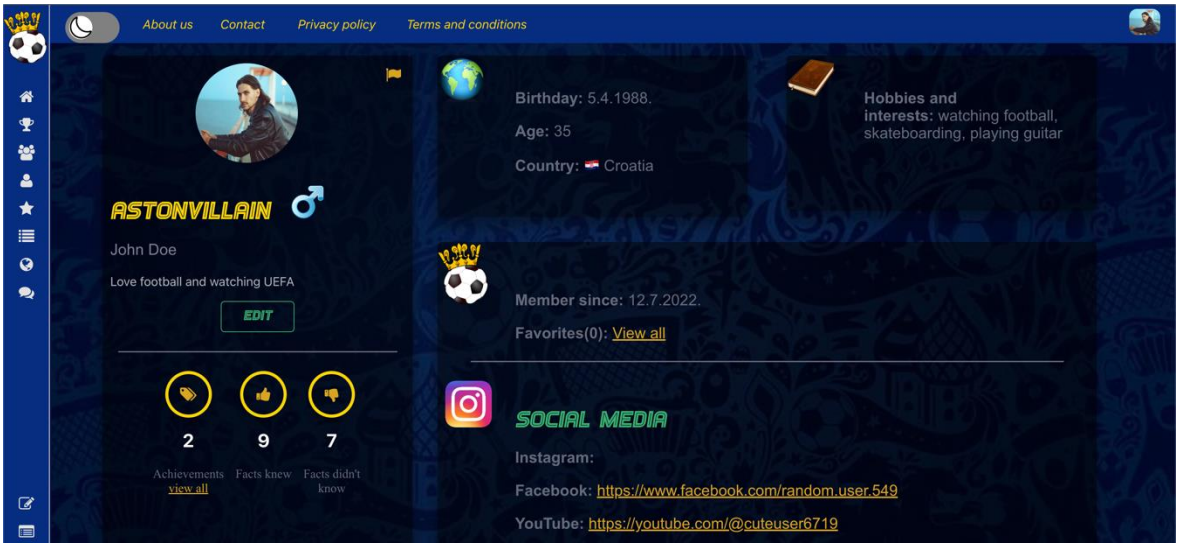
Slika 19 prikazuje tijek i uvjete za kreiranje utakmice te ostale aktivnosti na stranici.

5. 9. Profili

Detaljan prikaz informacija omogućuju profili. Ovdje postoje tri vrste profila:

- korisnički profil
- profil igrača
- profil tima

Korisnički i profil igrača su slični, sadrže važne informacije i sliku profila (slika 19). Profil igrača dodatno ima vrijednost igrača na tržištu koja se prikazuje grafički, po godinama. Korisnički profil omogućuje unos poveznica na društvene mreže tog korisnika, a timski profil je najdetaljniji i osim informacija i fotografija vezanih za tim, pruža i fotografije svih igrača koji su trenutno u tom timu. Te fotografije su i poveznice na profile igrača. Na slici 20 se može vidjeti profil korosnika.



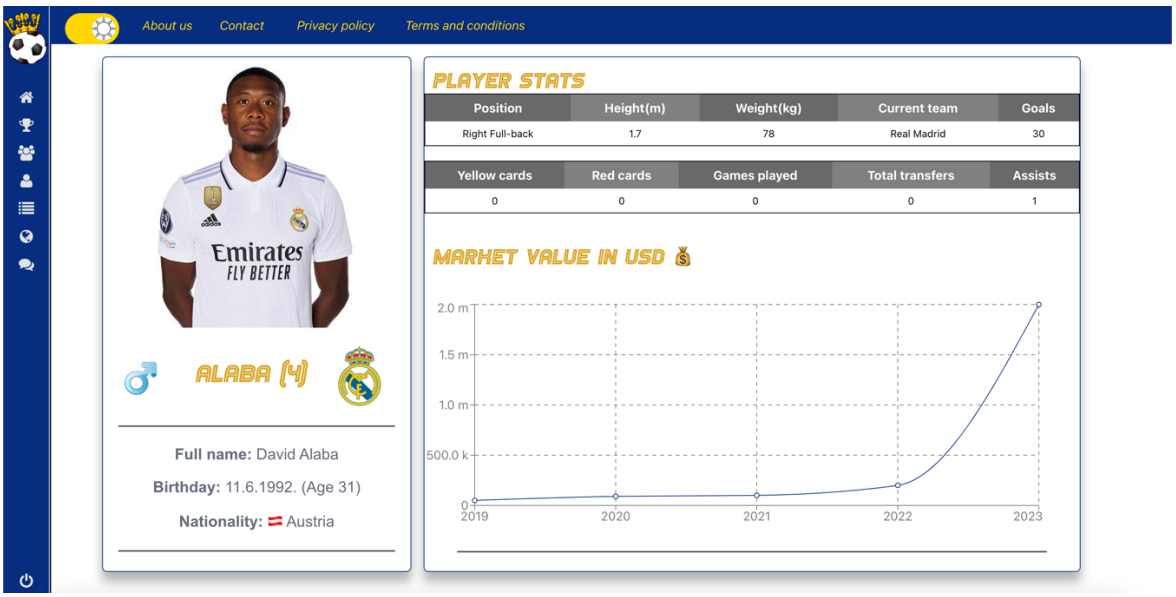
Slika 20: Korisnički profil

5. 10. Dizajn i teme

Glavna tema aplikacije je tamna i u njoj se koriste nijanse tamno plave boje. Postoji još i sporedna tema koja koristi bijelu (i u nekim slučajevima sivu) boju. Korišteni su fontovi Games (primarno), Times New Roman i sans-serif. Slova, simboli i ikone su obično zlatne ili srebrne boje, slično sportskim medaljama. Sveukupni dizajn je izrađen po uzoru na suvremene web stranice za videoigre (engl. *gaming sites*), čija su obilježja:

- poluprozirni tekstni okviri
- blago zaobljeni rubovi tekstnih okvira, gumba, fotografija i sl.
- naglašena interaktivnost grafičkih elemenata (promjena boje prelaskom preko njih)
- primarna tema je tamna
- pozadina tamne teme je fotografija ili uzorak, a ne boja
- obrub i/ili sjenčanje grafičkih elemenata
- minimalizam i simetričan razmještaj

Promjena teme obavlja se klikom na potvrdni okvir (engl. *checkbox*) u gornjem lijevom kutu. Slike 21 i 22 prikazuju izgled profila igrača u obje teme.



Slike 21 i 22: Profil igrača u svijetloj i tamnoj temi

6. Zaključak

Kroz ovaj rad je detaljno opisan postupak izrade web aplikacije za praćenje natjecateljskih turnira, a korištene tehnologije su se pokazale kao dobar izbor te su ga instalirani paketi za grafikone, fontove i strukture utakmica obogatili vizualno i funkcionalno. Dok je Django pogodan za izradu logike aplikacije i komunikaciju s bazom podataka, uz pomoć Reacta moguće je razviti dinamično i intuitivno korisničko sučelje.

Aplikacija pruža širok spektar mogućnosti raznim grupama korisnika, kao što su upravitelji nogometnih turnira, navijači i analitičari. Organizacija nogometnih turnira jako je intuitivna i pojednostavljena, a ažuriranje rezultata u stvarnom vremenu je dosta brzo. U ovoj aplikaciji registrirani korisnici mogu međusobno raspravljati o turnirima, kroz forum i komentiranje novosti, a omogućeno im je i uređivanje omiljenih klubova u svrhu selektivnog praćenja rezultata. Za korisnike tu su i profil i korisnička odlikovanja, a korisničke račune je moguće uklanjati ili ažurirati njihov status, ovisno o ponašanju i željama korisnika, što bi doprinijelo sigurnosti zajednice.

Kada je riječ o mogućim nadogradnjama aplikacije, jedna od najprikladnijih je mogućnost praćenja i drugih sportova osim nogometa, ali i drugih vrsta natjecanja (npr. znanstvenih, glazbenih, kuharskih, poljoprivrednih itd.). Nadalje, aplikacija bi se mogla nadograditi i dodavanjem proširenja za klađenje i predviđanje rezultata te dodavanjem detaljnijih mogućnosti statističkih analiza. Za korisnike bi se mogla izraditi i funkcionalnost izravnih poruka (engl. *direct messages, inbox*), no tada se nameće pitanje je li smisao aplikacije narušen i je li fokus pomaknut sa glavne svrhe aplikacije, pretvarajući je time u platformu društvenih mreža.

Literatura

[1] „Django documentation“, https://docs.djangoproject.com/en/4.1/ (posjećeno 28. 1. 2022.)

[2] „Django wiki“, https://en.wikipedia.org/wiki/Django_(web_framework) (posjećeno 1. 2.2022.)

[3] „React documentation“, https://reactjs.org/docs/getting-started.html (posjećeno 1. 2. 2022.)

[4] „Djoser docs“, https://djoser.readthedocs.io/en/latest/getting_started.html (posjećeno 1. 2. 2022.)

[5] „CSS resources“, https://developer.mozilla.org/en-US/docs/Web/CSS (posjećeno 1. 2. 2022.)

[6] „MySQL documentation“, https://dev.mysql.com/doc/ (posjećeno 1. 2. 2022.)

[7] „JWT introduction“, https://jwt.io/introduction/(posjećeno 2. 1. 2022.)

[8] „Redux documentation“, https://redux.js.org/tutorials/essentials/part-1-overview-concepts/(posjećeno 2. 1. 2022.)

[9] „Formation (association football)“, https://en.wikipedia.org/wiki/Formation_(association_football)(posjećeno 2. 1. 2022.)

[10] „Using React with Django“, https://blog.logrocket.com/using-react-django-create-app-tutorial/(posjećeno 2. 1. 2022.)

[11] „The commercialisation of football: Money Madness“, https://www.elartedf.com/commercialisation-football-money-madness/ (posjećeno 2. 1. 2022.)