Razvoj web aplikacije za malonogometne turnire pomoću MERN Stack razvojnog okvira

Plišić, Matteo

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski studij računarstva

Završni rad

Razvoj web aplikacije za malonogometne turnire pomoću MERN Stack razvojnog okvira

Rijeka, rujan 2023.

Matteo Plišić
0069089404

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Prijediplomski studij računarstva

Završni rad

Razvoj web aplikacije za malonogometne turnire pomoću MERN Stack razvojnog okvira

Mentor: Doc.dr.sc. Marko Gulić

Rijeka, rujan 2023.

Matteo Plišić
0069089404

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____

Matteo Plišić

Zahvala

Zahvaljujem mentoru doc. dr. sc. Marku Guliću na podršci tijekom pisanja ovoga rada i korisnim raspravama i savjetima. Također zahvaljujem svim ostalim profesorima zavoda za računarstvo Tehničkog fakulteta na pruženim znanjima i iskustvima kao i mojoj obitelji na kontinuiranoj podršci, strpljenju i dodatnoj motivaciji na tijekom studiranja.

Sadržaj

Popis slika **vii**

1 1. Uvod **1**

1.1 Opis aplikacije 1

2 2. Opis tehnologija **3**

2.1 React 3

2.2 Material UI 5

2.3 Node.js 8

2.4 ExpressJS 9

2.5 Mongodb 11

2.6 Zašto MERN 13

2.7 Zašto full stack razvoj 14

3 Opis funkcionalnosti **16**

3.1 Kreiranje React aplikacije 16

3.2 Korisnički pogled aplikacije 19

3.3 *Backend* aplikacije 38

3.3.1 Izgled direktorija 38

3.3.2 Osnovna logika 38

3.3.3 Modeli 41
3.3.4 Kontroleri 42
3.3.5 Logika kontrolera korisnika 42
3.3.6 Logika kreiranja turnira 47

4 Zaključak **59**

Bibliografija **60**

Pojmovnik **61**

Sažetak **61**

Popis slika

2.1 Razlika između DOM-a i virtualnog DOM-a [1] 4
2.2 Izgled mui gumbi 6
2.3 Kod za generiranje mui gumbi 6
2.4 Izgled mui forme 6
2.5 Kod za generiranje mui forme 7
2.6 Primjer osnovnog Node.js Servera 8
2.7 Primjer funkcije koja obavlja autorizaciju 10
2.8 Primjer zapisa u MongoDB baz podataka 12
2.9 Model korisnika 13

3.1 Početna React aplikacija 17
3.2 Direktorij klijentske strane aplikacije 19
3.3 Početni prikaz 20
3.4 Pogled sa formom za registraciju 20
3.5 Primjer validacijske poruke za prazna polja 21
3.6 Primjer validacijske poruke email kada se email već koristi 21
3.7 Primjer validacijske poruke neispravan email 22
3.8 Primjer validacijske poruke za nedovoljno dugu lozinku 22
3.9 Pogled sa obrascom za prijavu 23
3.10 Pogled sa obrascom za prijavu 23

3.11 Početni pogled autentificiranog korisnika 24
3.12 Pogled s formom za stvaranje turnira 25
3.13 Validacijska poruka za kreiranje turnira s neispravnim datumom . . 25
3.14 Validacijska poruka za kreiranje turnira sa krivim brojem timova . . 26
3.15 Validacijska poruka za kreiranje turnira s praznim poljem 26
3.16 Validacijska poruka za uspješno kreiranje turnira 27
3.17 Prikaz turnira korisnika 27
3.18 Prikaz izmjene turnira 28
3.19 Grupe turnira 29
3.20 Raspored utakmica turnira 30
3.21 Pogled upisivanja rezultata utakmica 31
3.22 Pogled svih turnira 32
3.23 Pogled svih turnira 32
3.24 Pogled svih turnira 33
3.25 Pogled svih turnira 33
3.26 Prikaz administracije ekipa 34
3.27 Izmjena ekipe 34
3.28 Prikaz administracije igrača 35
3.29 Padajući izbornik ekipa 35
3.30 Prikaz izmjene igrača 36
3.31 Prikaz administracije svih korisnika 36
3.32 Prikaz izmjene korisnika 37
3.33 struktura direktorija backend 38
3.34 Datoteka server.js 40
3.35 Datoteka server.js 41
3.36 Model korisnika 43

3.37 funkcija signup 44
3.38 registracija korisnika 45
3.39 Prikaz zapisa korisnika u bazi 45
3.40 Login funkcija 46
3.41 Login funkcija 46
3.42 Kreiranje turnira 47
3.43 Kreiranje turnira 48
3.44 Model turnira 49
3.45 Funkcija kreiranja turnira, 2. dio 50
3.46 Funkcija kreiranja rasporeda, 1. dio 51
3.47 Funkcija kreiranja rasporeda, 2. dio 53
3.48 Funkcija kreiranja korisnika 54
3.49 Funkcija dohvaćanja korisnika 55
3.50 Funkcija brisanja korisnika 56
3.51 Funkcija dohvaćanja pojedinog korisnika 57
3.52 Funkcija ažuriranja korisnika 58

Poglavlje 1

1. Uvod

1.1 Opis aplikacije

U suvremeno doba, kada je digitalizacija neizbježna u svakom aspektu naših života, ni sportski događaji poput različitih liga i turnira nisu izuzetak. Cilj organizatora takvih događaja je putem interneta približiti te događaje svima zainteresiranima za praćenje. U skladu s tom tendencijom, ovaj završni rad predstavlja razvoj web aplikacije namijenjene organizaciji i praćenju sportskih događaja, posebno malonogometnih turnira.

Svrha ove aplikacije je omogućiti pratiteljima malonogometnih turnira jednostavan i učinkovit način praćenja rezultata utakmica i turnira koji ih zanimaju, dok organizatorima pruža jednostavan način vođenja tih turnira i njihovih utakmica.

U samoj aplikaciji korisnici se mogu registrirati i prijaviti nakon čega mogu kreirati i voditi svoj turnir. Pri kreiranju turnira korisnik odlučuje vrijeme, mjesto i naziv turnira te odabire ekipe koje će se u njemu natjecati. Pri izradi turnira, automatski se generiraju dvije grupe od četiri ekipe čije prvoplasirane ekipe igraju utakmicu za pobjednika turnira dok drugoplasirane ekipe u grupama igraju utakmicu za treće mjesto. Nakon kreiranja turnira korisnik može upisivati rezultate utakmica te tako voditi turnir. Dodatno, korisnicima koji nisu prijavljeni ni registrirani mogu pratiti rezultate svih odigranih utakmica i turnira kao i vidjeti sve nadolazeće turnire. Osim običnih korisnika postoje i administratori koji mogu dodavati i izmijeniti

podatke o korisnicima,igračima i ekipama ili brisati iste. Za izradu aplikacije korišten je MERN Stack razvojni okvir koji se sastoji od četiri glavne tehnologije: MongoDB, React.js, Node.js i Express.js.

Na strani poslužitelja koristi se Node.js, zajedno s njegovim okvirom Express.js, popularnim serverskim okvirom temeljenim na asinkronim događajima. To omogućuje izgradnju skalabilnih web aplikacija te brz i učinkovit razvoj serverske logike. Za upravljanje bazom podataka koristi se MongoDB, sve popularnija skalabilna baza podataka temeljena na dokumentima, razvijena za velike količine podataka.

Na strani klijenta koristi se React.js, priznata *open-source* knjižnica koja služi za izgradnju korisničkih sučelja ili komponenata za web aplikacije. Razvijen od strane Facebooka, React.js se usredotočuje na učinkovito i dinamično upravljanje korisničkim sučeljima putem koncepta nazvanog "komponente".

Također, uz React.js, na klijentskoj strani se koristi Material-UI, istaknuta *open-source* biblioteka koja pruža napredne komponente i stilizaciju sučelja za *web* aplikacije. Ova biblioteka temelji se na "Material Design" dizajnerskim smjernicama, koje je razvio Google, te omogućuje brzu izgradnju modernih i atraktivnih korisničkih sučelja. Material-UI nudi širok spektar unaprijed definiranih komponenata kao što su tipke, forme, izbornici i mnoge druge, s već stiliziranim sučeljima koja olakšavaju konzistentan izgled.

Poglavlje 2

2. Opis tehnologija

Za potrebe aplikacije ovog završnog rada, kako je prethodno navedeno, korištene su sljedeće tehnologije: React.js, Node.js, Express.js i MongoDB. Integracija ovih tehnologija rezultira MERN tehnološkim stogom, koji omogućuje razvoj složenih aplikacija punog stoga. Ova aplikacija uključuje korisničko sučelje i poslužiteljsku logiku te će ova logika biti detaljnije objašnjena u nastavku.

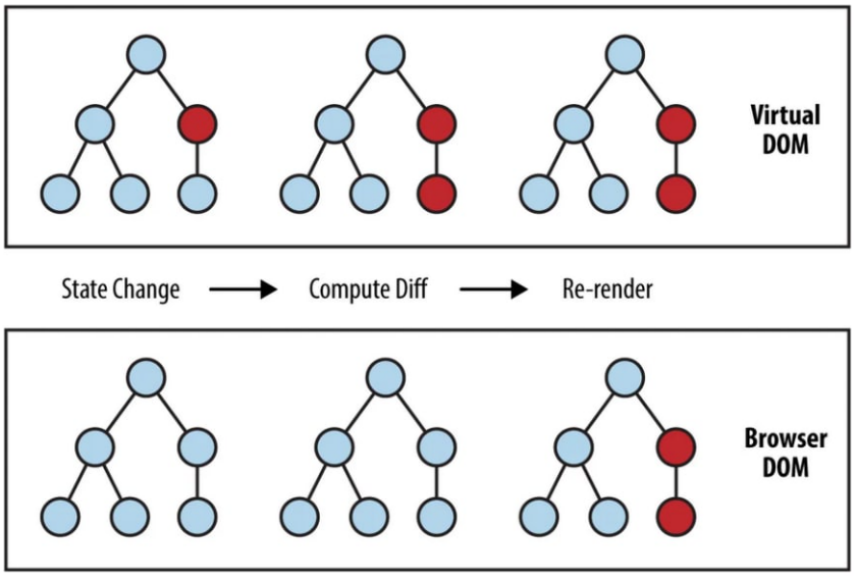
2.1 React

Korisničko sučelje predstavlja prvi dojam koji korisnik stječe prilikom otvaranja internetske stranice. Osnovne tehnologije koje se koriste za njegov dizajn obuhvaćaju prezentacijski jezik HTML, koji stranici pruža strukturu, jezik za stiliziranje CSS, koji je zadužen za vizualno uljepšavanje stranice, te konačno JavaScript kao jezik koji pridodaje funkcionalnosti stranici. Da bi proces razvoja bio olakšan i ubrzan, koriste se različite knjižnice i radni okviri. Jedna od takvih knjižnica je React.js, koji je upotrijebljen u ovom radu.

React.js je komponentno orijentirana i otvorena JavaScript knjižnica koja se koristi za izradu korisničkih sučelja na deklarativan način. Bitno je naglasiti da React.js nije radni okvir, iako mnogi to pogrešno interpretiraju. On je odgovoran isključivo za sloj pogleda unutar arhitekture Model-Pogled-Kontroler eng. Model-View-Controller, skraćeno MVC. Deklarativni pristup omogućava aplikacijama da budu

istovremeno efikasne i fleksibilne. Stvaranje jednostavnih prikaza za svako stanje aplikacije te ažuriranje pojedinačne komponente i renderiranje samo te komponente, dok druge ostaju nepromijenjene, znatno poboljšava optimizaciju.

Sve u svemu, React.JS je izvrstan izbor za razvoj korisničkog sučelja iz više razloga. Njegova efikasnost, koja proizlazi iz upotrebe virtualnog DOM-a, apstraktne kopije DOM-a koji je stvarna struktura elemenata internetske stranice. Tijekom promjena na stranici i njihovih međukoraka, stvara se virtualni DOM, a zatim se uspoređuje virtualni DOM sa pravim DOM-om kako bi se pronašle razlike te se samo razlike primjenjuju na stvarni DOM što osigurava brže ažuriranje i bolje performanse.Na slici 2.1 prikazan je tijek stanja virtualnog DOM-a i pravog DOM-a pri nekim promjenama.



Slika 2.1 Razlika između DOM-a i virtualnog DOM-a [1]

Modularnost komponentne arhitekture olakšava ponovnu upotrebu i održavanje koda. Deklarativni pristup pojednostavljuje razvoj jer se fokusira na rezultat, a ne na detalje implementacije. Aktivna zajednica i bogat ekosustav alata pružaju podršku iskusnim razvijateljima. Sposobnost ReactJS-a da optimizira ažuriranje samo promijenjenih dijelova sučelja dodatno unapređuje performanse i korisničko iskustvo. Ukratko, React.js nudi dinamičnost, modularnost i učinkovitost, čineći ga

izvrsnim izborom za moderna i funkcionalna korisnička sučelja.[2]

2.2 Material UI

Material-UI je popularna i široko korištena *open-source* knjižnica za izradu korisničkih sučelja na webu s modernim i vizualno privlačnim dizajnom. Razvijena za React.js, Material-UI pruža mnogo komponenata, stilova i smjernica temeljenih na principima Googleovog Material Designa. Knjižnica omogućuje programerima jednostavno stvaranje dosljednih, odzivnih i interaktivnih korisničkih sučelja. Predefinirane komponente Material-UI-a obuhvaćaju širok raspon UI(korisničko sučelje, eng. user interaface) elemenata, kao što su gumbi, obrasci, navigacijske trake i mnogi drugi, olakšavajući brz razvoj uz održavanje dosljednog dizajnerskog jezika.

Zahvaljujući doprinosima preko 2.500 pojedinaca, ova knjižnica omogućava ubrzan razvoj. Programeri se mogu usredotočiti na osnovnu poslovnu logiku, jer opsežne komponente Material UI-a pokrivaju različite potrebe korisničkog sučelja.

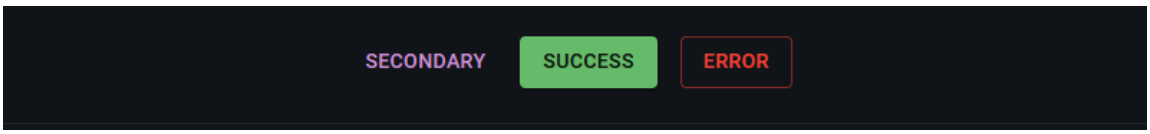
Implementacija Materijalnog dizajna pridržava se visokih standarda estetike i funkcionalnosti. Komponente su pažljivo izrađene kako bi osigurale ljepotu i praktičnost, uz fleksibilnost da se odstupi od službenih specifikacija pružajući više iznimnih opcija.

Prilagodljivost je ključna značajka koja pruža niz intuitivnih opcija za prilagođavanje komponenata specifičnim potrebama. Predlošci u knjižnici demonstriraju obim prilagodbe koji se može postići.

Korisničko iskustvo koje nudi Material UI potiče suradnju među različitim timovima. Razvojnim inženjerima koji kreiraju poslužiteljski dio aplikacije kao i manje tehnički usmjerenim dizajnerima ova knjižnica olakšava suradnju, jer pojednostavljuje njihovu interakciju.

Povjerenje tisuća organizacija ukazuje na to da Material UI ima najveću zajednicu korisničkog sučelja unutar React ekosustava. S obzirom na svoju povijest koja seže unatrag do 2014. godine, ova knjižnica je izdržala test vremena i zadržava svoju posvećenost. To jamči korisnicima podršku zajednice i u godinama koje dolaze, kao što se vidi na platformama poput Stack Overflowa i drugih.

Na slici 2.2 može se vidjeti kako se vrlo lako i efikasno, sa samo nekoliko linija koda, mogu definirati estetski ugodni gumbi kao gotove komponente koje bi koristili u aplikaciji.[3]



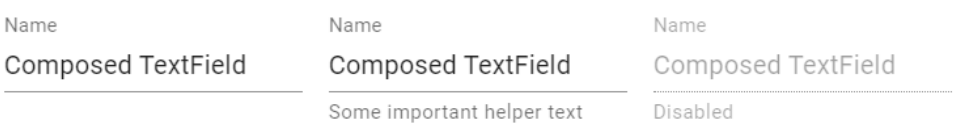
Slika 2.2 Izgled mui gumbi

Na slici 2.3 može se vidjeti programski kod koji definira tri gumba prikazana na slici 2.2.

```
<Button color="secondary">Secondary</Button>
<Button variant="contained" color="success">
  Success
</Button>
<Button variant="outlined" color="error">
  Error
</Button>
```

Slika 2.3 Kod za generiranje mui gumbi

Atribut *variant* određuje stilski izgled gumba, to jest ima li gumb punu pozadinsku boju kao što je to slučaj ako je vrijednost varijable *contained* ili da gumb ima samo boju okvira, dok je ostatak gumba transparentan za slučaj da je vrijednost varijable *outlined*. Na slici 2.4 nalazi se još jedan primjer jednostavnog kreiranja estetskih ugodnih komponenti, u ovom slučaju formi.



Slika 2.4 Izgled mui forme

Na slici 2.5 se nalazi kod koji definira formu prikazanu na slici 2.4.

```
<FormControl variant="standard">
  <InputLabel htmlFor="component-simple">Name</InputLabel>
  <Input id="component-simple" defaultValue="Composed TextField" />
</FormControl>
<FormControl variant="standard">
  <InputLabel htmlFor="component-helper">Name</InputLabel>
  <Input
    id="component-helper"
    defaultValue="Composed TextField"
    aria-describedby="component-helper-text"
  />
  <FormHelperText id="component-helper-text">
    Some important helper text
  </FormHelperText>
</FormControl>
<FormControl disabled variant="standard">
  <InputLabel htmlFor="component-disabled">Name</InputLabel>
  <Input id="component-disabled" defaultValue="Composed TextField" />
  <FormHelperText>Disabled</FormHelperText>
</FormControl>
```

Slika 2.5 Kod za generiranje mui forme

Kod prikazan na slici 2.5 koristi više komponenti forme s različitim poljima za unos teksta. Svako polje za unos je unutar *FormControl* komponente koja omogućuje prilagođavanje izgleda forme dok vrijednost *standard* atributa *variant* označava standardni izgled komponente. Prva dva polja su obična tekstualna polja za unos za koje se koristi komponenta *Input*. Atribut id označava identifikator komponente dok *defaultValue* označava inicijalnu vrijednost komponente. Atribut *aria-describedby* se koristi u HTML-u kako bi se pridružile opisne poruke elementima, obično elementima koji imaju ulogu ulaznih polja ili drugih interaktivnih elemenata Svako polje ima svoju oznaku *InputLabel* koja označava što korisnik treba unijeti. Atribut *htmlFor* se koristi u HTML-u kako bi se uspostavila veza između oznake i odgovarajućeg

elementa za unos koji je u ovom slučaju *Input*. Treće polje je onemogućeno za unos atributom *disabled*. To znači da korisnik ne može mijenjati sadržaj polja, ali se polje ipak prikazuje. Također ima svoj tekst koji ukazuje da je polje onemogućeno. Komponenta *FormHeperText* služi kao opisni tekst polja za unos.

2.3 Node.js

Node.js je inovativno okruženje za izvođenje JavaScripta na poslužiteljskoj strani, poznato po svojoj asinkronoj, događajem vođenoj arhitekturi, optimizirano za izradu skalabilnih mrežnih aplikacija. Za razliku od konvencionalnih pristupa, Node.js učinkovito rukuje s brojnim vezama istodobno, bez blokiranja izvođenja. Priloženi primjer na slici 2.6 *hello world* demonstrira njegovu sposobnost istovremenog procesiranja. Upotrebljavajući modul 'http', Node.js gradi server koji odgovara na dolazne zahtjeve slanjem poruke *"Hello World"*. To ilustrira njegovu nenametljivu narav te ističe odsutnost učinkovitosti temeljenih na nitima. Node.js oslobađa programere

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Slika 2.6 Primjer osnovnog Node.js Servera

od brige o zastojima zahvaljujući svojoj nenametljivoj naravi i odsustvu tradicionalnih zaključavanja. Većina funkcija u Node.js izbjegava izravne I/O operacije, osim prilikom upotrebe sinkronih metoda iz standardne knjižnice Node.js, osiguravajući

nenametljiv tijek rada. Ova jedinstvena karakteristika otvara put razvoju skalabilnih sustava u Node.js, jer izbjegava uska grla u resursima.

Node.js usvaja jedinstveni model konkurentnosti koji ga razlikuje od tradicionalnih paradigmi koje koriste OS niti. Sliči dizajnima iz sustava poput Rubyjeve Event Machine i Pythonovog Twisteda, ali ide korak dalje. Umjesto da tretira petlju događaja kao knjižnicu, Node.js je uključuje kao konstrukciju za izvođenje. Ovaj pristup pojednostavljuje proces, budući da Node.js ulazi u petlju događaja nakon izvršavanja ulaznog skripta i izlazi kada su ispunjeni svi povratni pozivi.

Node.js se ističe u rukovanju HTTP zahtjevima, što ga čini idealnim za razvoj web okvira zbog usmjerenosti na streaming i nisku latenciju. Njegova sposobnost generiranja podprocesa putem *child_process.fork* Aplikacijskog programskog sučelja (eng. application programming interface) pruža mogućnost iskorištavanja više jezgri, omogućavajući učinkovito paralelno procesiranje. To se dalje proširuje modulom za klasteriranje, olakšavajući dijeljenje utičnica među procesima za optimizirano ravnoteženje opterećenja preko jezgri. [4]

2.4 ExpressJS

Express.js je kompaktan okvir koji nadograđuje funkcionalnosti Node.js web poslužitelja kako bi pojednostavio API-je i dodao korisne nove značajke. Olakšava organizaciju funkcionalnosti aplikacije putem *middlewarea* i ruta. Express.js omogućuje brz i jednostavan razvoj Node.js web aplikacija. Njegova jednostavna postavka i prilagodba čine ga popularnim izborom. Omogućuje definiranje ruta aplikacije pomoću HTTP metoda kao što su post, get, put i delete, kao i URL-ova, osiguravajući logičan protok zahtjeva. Express.js dolazi s raznim modulima za middleware koji omogućuju izvođenje dodatnih aktivnosti zahtjeva i odgovora. To omogućuje proširenje mogućnosti aplikacije bez ponovnog pisanja već postojećeg koda i smanjuje redundanciju. Jednostavno se integrira s različitim sustavima predložaka poput Jade i Vash, što pojednostavljuje generiranje dinamičkog sadržaja. Također omogućuje definiranje middlewarea za upravljanje greškama i iznimkama, osiguravajući pouzdanu obradu neočekivanih situacija. Express.js je postao vodeći web okvir za Node.js s

ciljem izgradnje web aplikacija i aplikacijskog programskog sučelja, te je često nazivan standardnim okvirom za poslužitelje u Node.js ekosustavu. Korištenje Express.js omogućuje programerima uštedu vremena i fokusiranje na bitne zadatke umjesto na izradu osnovnih komponenti aplikacije. Jedan od takvih zadataka je autentikacija korisnika koja se može vidjeti na slici 2.7.

```
async function login(req, res) {
  try {

    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) return res.sendStatus(401);

    const passwordMatch = bcrypt.compareSync(password, user.password);
    if (!passwordMatch) return res.sendStatus(401);


    const exp = Date.now() + 1000 * 60 * 60 * 24 * 30;
    const token = jwt.sign({ sub: user._id, exp, name: user.name, superUser: user.superUser }, process.env.SECRET,);


    res.cookie("Authentication", token, {
      expires: new Date(exp),
      httpOnly: true,
      sameSite: "lax",
      secure: process.env.NODE_ENV === "production",
    });


    res.json({ token })
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 2.7 Primjer funkcije koja obavlja autorizaciju

U funkciji prikazanoj na slici 2.7 autentikacija se radi u više koraka. Prvo se traži korisnik sa emailom poslanom u zahtjevu, nakon ćega se provjerava je li poslana i zatim kriptirana lozinka jednaka zapisu u bazi podataka. U slučaju da je korisnik pronađen i lozinka je ispravna kreira se JWT kao token za autorizaciju i autentifikaciju pri budućim zahtjevima koji će se slati sa klijentske strane. U tokenu se nalaze informacije o korisniku kao što su id korisnika, njegovo ime i atribut *superUser* koja označava je li korisnik adminstrator ili nije. Token se kriptira sa *process.env.SECRET* koji služi kao tajni ključ za enkripciju. U slučaju nepostojećeg emaila ili krive lozinke klijentskoj strani se vraća statusni kod 401 koji označava pokušaj neautoriziranog

pristupa, dok u slučaju greške u radu se vraća statusni kod 400.

2.5 Mongodb

MongoDB je dokumentna baza podataka koja se temelji na strukturi dokumenata, koja je sastavljena od parova polje-vrijednost. Dokumenti u MongoDB-u slični su JSON objektima. Vrijednosti polja mogu uključivati druge dokumente, nizove i nizove dokumenata.

Prednosti korištenja dokumenata su:

- Dokumenti se podudaraju s izvornim tipovima podataka u mnogim programskim jezicima.
- Ugrađeni dokumenti i nizovi smanjuju potrebu za skupim spajanjima (joinovima).
- Dinamična shema podržava tečni polimorfizam.

MongoDB pohranjuje dokumente u kolekcijama, koje su analogne tablicama u relacijskim bazama podataka kao što su SQL Server,PostgreSQL i druge. Osim kolekcija, MongoDB podržava čitanje samo prikaza (pogleda) i materijaliziranih prikaza na zahtjev. Ključne značajke uključuju visoku performansu, podršku za upite putem MongoDB upita, podršku za agregaciju podataka, tekstualno pretraživanje i geoprostorne upite. MongoDB također pruža visoku dostupnost putem mehanizma replikacije, poznatog kao *replica set*, koji osigurava automatsku zamjenu i redundanciju podataka. Horizontalno skaliranje omogućava distribuciju podataka preko skupa strojeva, a tehnika *sharding* omogućava raspodjelu podataka preko klastera strojeva. Podržava više strojeva za pohranu, kao što su WiredTiger i In-Memory, a također pruža aplikacijsko programsko sučelje za povezivanje vanjskih strojeva za pohranu. MongoDB skladišti podatke u fleksibilnim dokumentima sličnim JSON-u, što omogućava raznolikost polja i promjenu strukture podataka tijekom vremena. Model dokumenata se povezuje s objektima u aplikacijskom kodu, što olakšava rad s podacima. Ad hoc upiti, indeksiranje i agregacija u stvarnom vremenu pružaju moćne načine pristupa i analize podataka. Uz svoju jednostavnost za razvojne programere

i sposobnost ispunjavanja kompleksnih zahtjeva na svakoj skali, MongoDB pruža upravljački program za više od 10 jezika, a zajednica je izgradila još desetke. MongoDB je baza podataka s dokumentima koja kombinira skalabilnost i fleksibilnost s moćnim mogućnostima upita i indeksiranja. Vlastitu MongoDB bazu podataka moguće je podesiti lokalno na svom računalu ili poslužitelju ili na oblaku pomoću atlasa dostupnog na službenoj stranici MongoDB-a.[5] Za svrhe ovog rada korištena je lokalna MongoDB baza podataka te *desktop* program MongoDB Compass[6] radi lakše provjere stanja baze podataka i pronalaženja grešaka prilikom programiranja.

Na slici 2.8 prikazano je kako su dokumenti korisnika zapisani u bazi podataka nalik na JSON objekte.

Slika 2.8 Primjer zapisa u MongoDB baz podataka

Na slici 2.9 može se vidjeti model korisnika na poslužiteljskoj strani koji odgovara zapisima u bazi prikazanima na slici 2.8.

```
const mongoose = require("mongoose")
const userSchema = new mongoose.Schema({
    name:{
        type:String,
        required:true,
    },
    email: {
        type:String,
        required:true,
        unique:true,
        lowercase:true,
        index:true
    },
    password: {
        type:String,
        require: true,

    },
    superUser:{
        type:Boolean,
        default:false
    }

});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

Slika 2.9 Model korisnika

Svaki korisnik mora imati svoje ime, email i lozinku, te ako je korisnik administrator vrijednost atributa *superUser* će biti *true*, a u suprotnom *false*. Za svaki se atribut mora definirati tip podatka te po potrebi se mogu definirati ostale vrijednosti kao što su *required* koja označava je li podatak obavezan za zapis u bazu ili *default* koji označava vrijednost koja će se spremiti u bazu ako druga vrijednost nije dana.

2.6 Zašto MERN

MERN, već prethodno spomenuti tehnološki stog, u današnje vrijeme postaje sve popularniji iz brojnih razloga koji će biti objašnjeni u nastavku. Ideja tehnološkog stoga je da jedini programski jezik korišten za izradu web aplikacija bude JavaScript, koji je bez sumnje najpopularniji programski jezik za web aplikacije, posebno u dijelu

izrade korisničkog sučelja. U tom dijelu razvoja, React.js dominira nad svim ostalim knjižnicama i radnim okvirima, stoga ne čudi što je on primarni odabir. Također se koristi i radni okvir Angular, čime dobivamo MEAN stog.

Node.js i Express.js su preferirani odabiri za izradu poslužiteljske strane u MERN stogu (MongoDB, Express.js, React, Node.js) iz više razloga. Prvo, oslanjanje stoga na JavaScript tijekom cijelog razvojnog procesa promiče dosljednost i ponovnu uporabu koda, sjedinjujući razvoj i klijentske i poslužiteljske strane. Asinkrona i događajem vođena arhitektura Node.js-a osigurava neblokirajuće izvršavanje, što je idealno za upravljanje brojnim istodobnim vezama bez smanjenja performansi. Opsežni ekosustav otvorenog koda putem npm paketa pojednostavljuje zadatke razvoja suradnjom između programera, od integracije baze podataka do autentifikacije.

Nadalje, Express.js, široko prihvaćeni okvir za web aplikacije za Node.js, pruža minimalistički, ali snažan alatni set za izgradnju API-ja i upravljanje logikom na strani poslužitelja. Zahvaljujući svojoj fleksibilnoj strukturi, moguće je definirati rute za različite HTTP metode i URL-ove, što pojednostavljuje organizaciju i čitljivost krajnjih točaka.

I Node.js i Express.js imaju izrazito aktivne zajednice, što osigurava pregršt tutoriala, dokumentacije i modula koje je zajednica doprinijela za učenje i rješavanje problema.

2.7 Zašto full stack razvoj

Prije nego li objasnimo prednosti i mane full stack razvoja, potrebno je objasniti što uopće je full stack razvoj. On uključuje razvoj i upravljanje *frontend* i *backend* aspektima aplikacija što je svakako zahtijevnije od posvećivanja samo jednom dijelu razvoja, ali zato donosi brojne prednosti te je s toga izrazito popularan u početnim tvrtkama. Neke od tih prednosti su:

- Razumiijevanje kompletne logike nekog procesa koji se odvija u aplikaciji
- Potrebno je manje vremena pošto jednu značajku u potpunosti može napraviti samo jedna osoba i ne gubi se vrijeme na prijenos informacija između ljudi

- Potreban je manji broj ljudi kako bi se napravio posao što znaći i manji troškovi poslovanja

Zaključno, radi navedenih prednosti i karakteristika full stack razvoja, jasno je kako je za svakog programera poželjno da se barem okuša u full stack razvoju i vidi je li to za njega ili bi se radije specijalizirao samo u jednom području, bilo to *frontend* ili *backend*.

Poglavlje 3

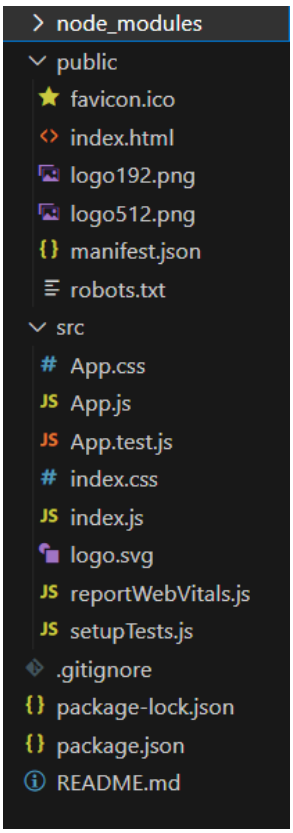
Opis funkcionalnosti

3.1 Kreiranje React aplikacije

Prvi dio aplikacije koji korisnik vidi je njen *frontend*, u našem slučaju je to ReactJS. React projekt se kreira naredbom

```
npx create-react-app my-app
```

kojom se kreira direktorij sa nazivom my-app koji ima strukturu kao na slici 3.1.



Slika 3.1 Početna React aplikacija

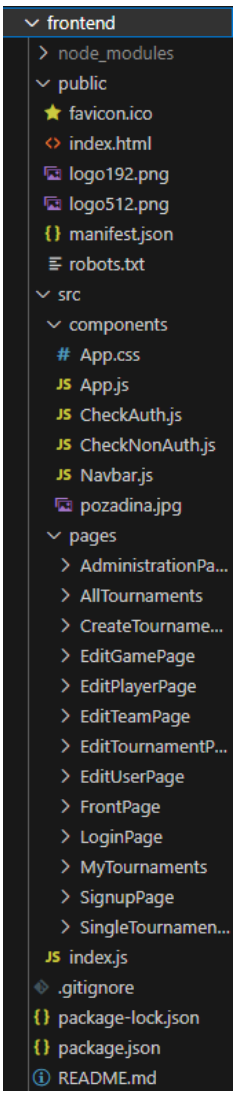
U direktoriju sa slike 3.1 se nalazi slijedeće:

- Direktorij node_modules u kojem se nalaze sve potrebne ovisnosti projekta, koje se automatski instaliraju prema specifikacijama definiranim u datoteci package.json.
- Direktorij public sadrži statičke resurse koji se poslužuju izravno pretraživaču. U njemu se nalazi index.html, temeljna točka ulaza aplikacije. Direktorij src sadrži izvorni kod aplikacije.
- Datoteka index.js predstavlja glavni ulaz u aplikaciju. Ona renderira glavnu React komponentu u root elementu unutar index.html.
- Datoteka App.js sadrži glavnu komponentu vaše aplikacije, koja je početna točka za izgradnju korisničkog sučelja.

- Datoteke App.css i index.css su CSS datoteke za stiliziranje vaših komponenata.
- Datoteka logo.svg je primjer SVG slike koji dolazi s projektom.
- Datoteka reportWebVitals.js koristi se za mjerenje performansi aplikacije.
- Datoteka package.json sadrži metapodatke o projektu, popis ovisnosti, skripti i druge konfiguracije.
- Datoteka package-lock.json služi za zaključavanje verzija ovisnosti kako bi se osigurala dosljednost među različitim instalacijama.
- Datoteka README.md sadrži informacije o projektu za njegovo lakše razumijevanje.
- Datoteka .gitignore navodi datoteke i direktorije koji se ne bi trebali pratiti u sustavu za verzioniranje kao što je Git.

3.2 Korisnički pogled aplikacije

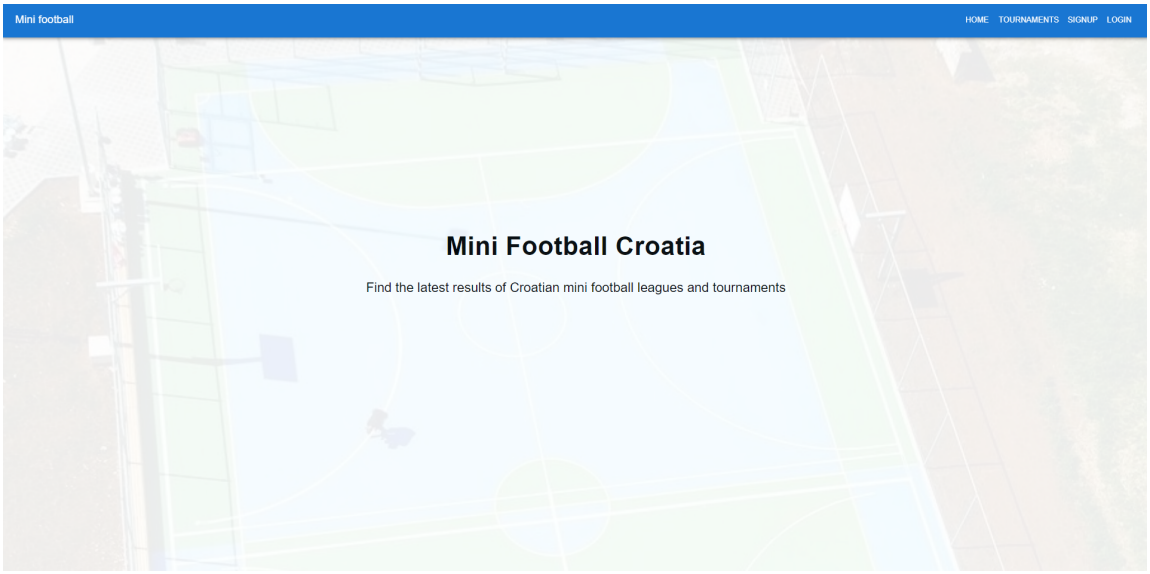
Aplikacija obrađena u ovom radu je podijeljena u 2 direktorija. U ovom poglavlju će biti obrađen direktorij koji sadrži React.js aplikaciju, čija je struktura prikazana na slici 3.2:



Slika 3.2 Direktorij klijentske strane aplikacije

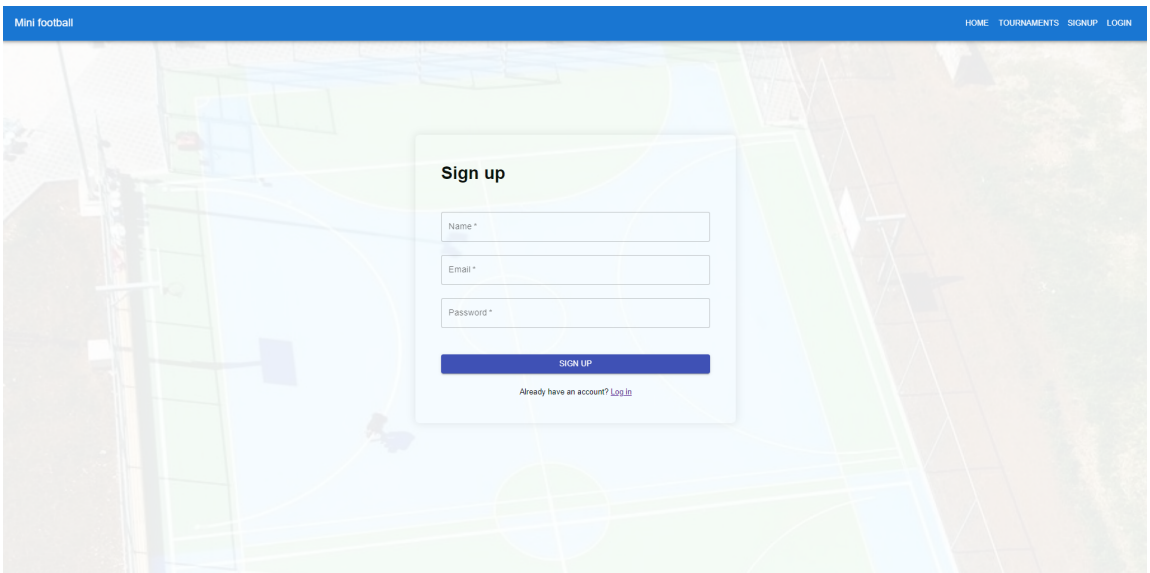
Svaki pogled ima svoj zaseban direktorij kako bi se lakše navigiralo po projektu te po potrebi dodavale datoteke za svaki zaseban pogled. Kada korisnik prvi put

posjeti stranicu, koja se nalazi na *portu* 3001, dočekat će ga prikaz kao na slici 3.3.



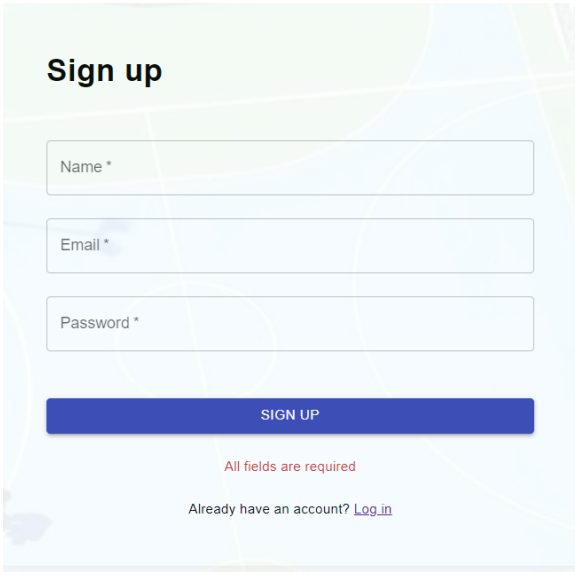
Slika 3.3 Početni prikaz

Klikom na karticu Signup korisnika se vodi na obrazac gdje se može registrirati koristeći svoje ime, email i lozinku kao što je prikazano na slici 3.4.



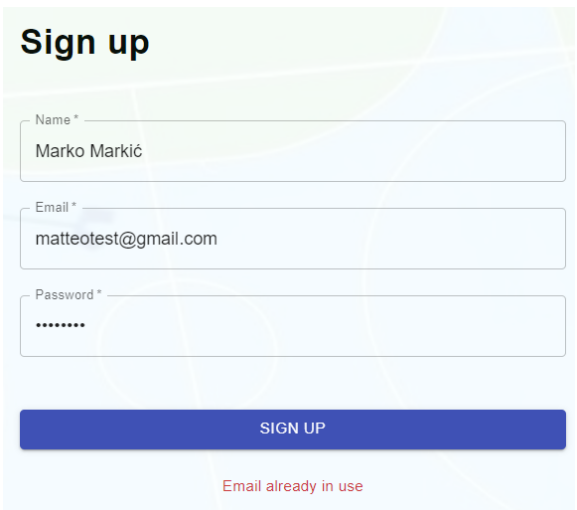
Slika 3.4 Pogled sa formom za registraciju

U slučaju da korisnik nije unio jedan ili više podataka prikazat će se validacijska poruka kao na slici 3.5.



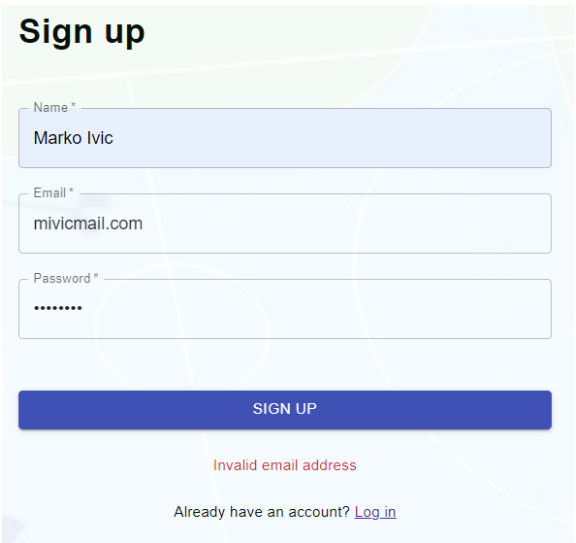
Slika 3.5 Primjer validacijske poruke za prazna polja

Dodatno, u slučaju da se korisnik pokuša registrirati sa emailom koji je već korišten pri registraciji, prikazat će se validacijska poruka kao na slici 3.6.



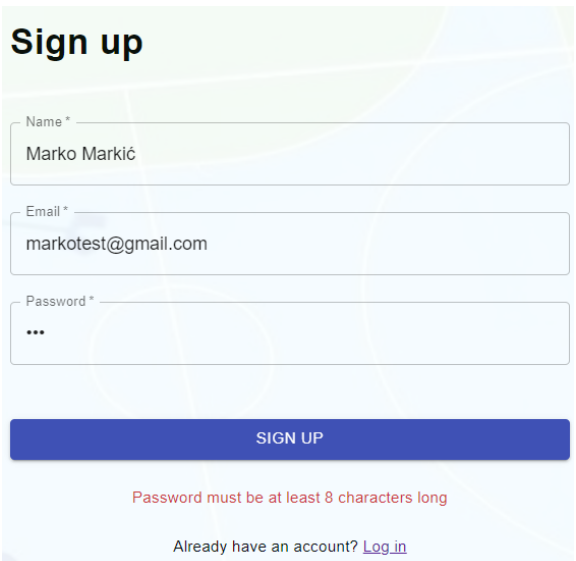
Slika 3.6 Primjer validacijske poruke email kada se email već koristi

Isto tako, ako korisnik upiše email neispravnog formata, javlja se validacijska poruka prikazana na slici 3.7.



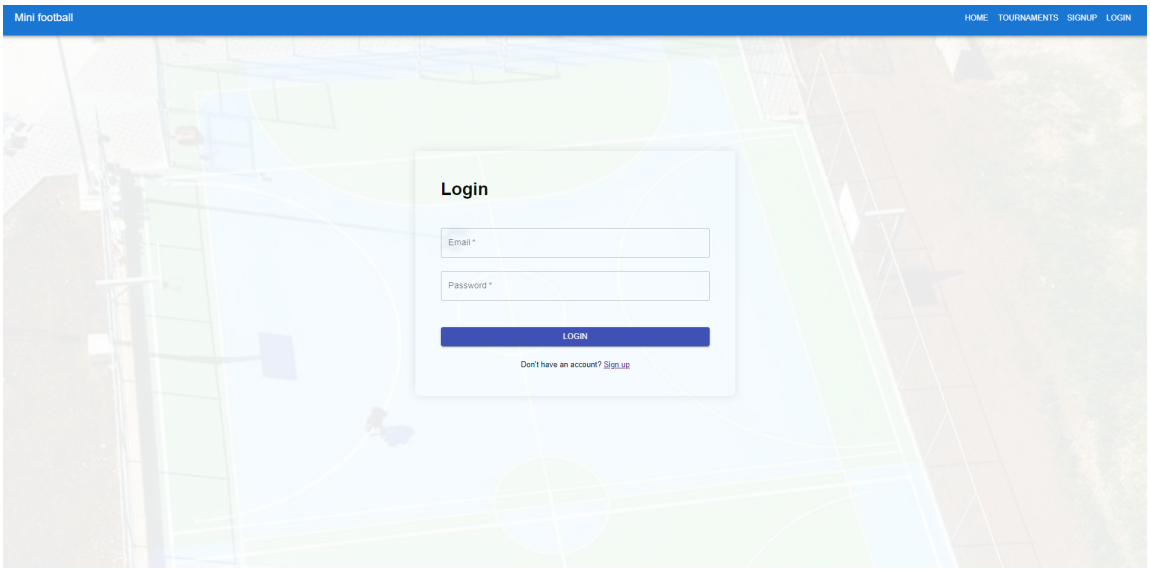
Slika 3.7 Primjer validacijske poruke neispravan email

Validacijska poruka vezana uz neispravnu lozinku se može prikazati u slučaju da korisnik koristi lozinku kraću od 8 znakova, pri ćemu se ispisuje validacijska poruka prikazana na slici 3.8.



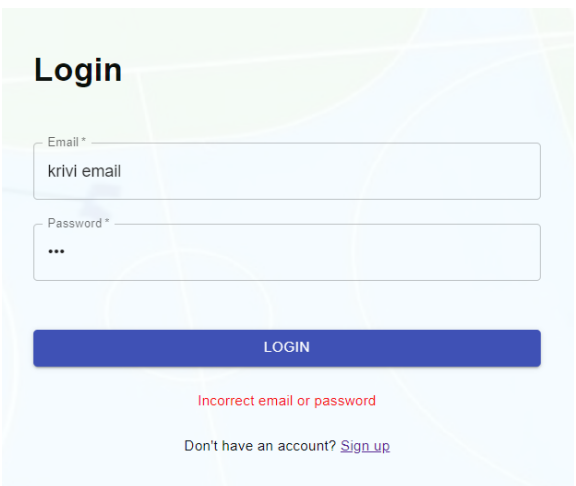
Slika 3.8 Primjer validacijske poruke za nedovoljno dugu lozinku

Nakon što se korisnik registrirao može se ulogirati klikom na Login karticu koja vodi do obrasca za prijavu prikazane na slici 3.9.



Slika 3.9 Pogled sa obrascom za prijavu

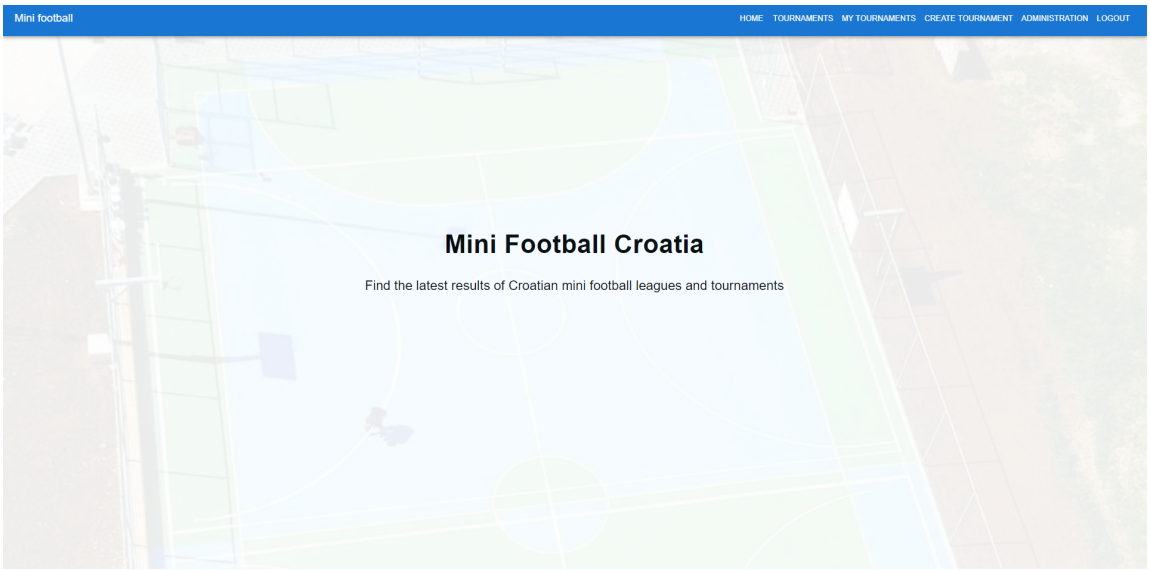
U slučaju unosa krivih podataka, to jest neispravne lozinke ili emaila, prikazuje se validacijska poruka prikazana na slici 3.10.



Slika 3.10 Pogled sa obrascom za prijavu

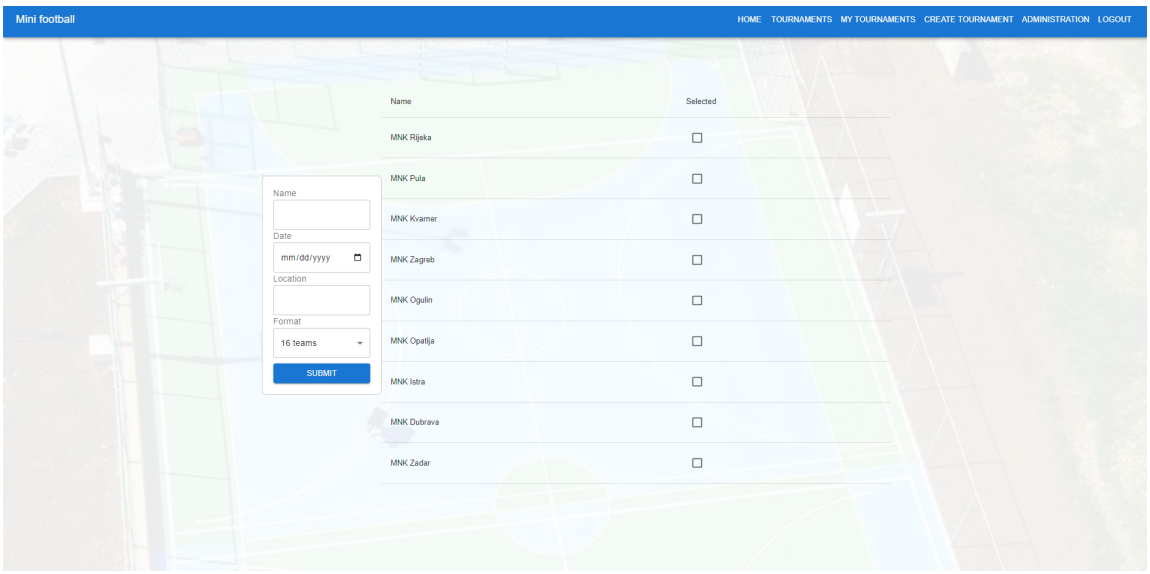
Za potrebe ovog rada, koristit će se administratorski korisnik koji ima dodatne

mogućnosti administracije, što će kasnije biti dodatno objašnjeno i prikazano. Nakon ispravnog unosa korisničkih podataka, korisnik će biti preusmjeren na pogled prikazan na slici 3.11.



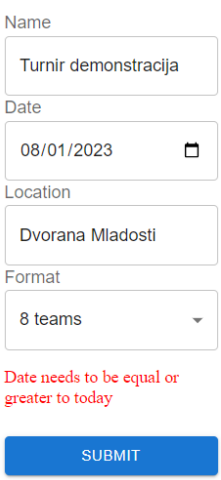
Slika 3.11 Početni pogled autentificiranog korisnika

U usporedbi s neautentificiranim korisnikom, dostupne su dvije ili tri kartice, ovisno o tome je li korisnik administrator ili nije. To su sljedeće kartice: *MY TOURNAMENTS*, *CREATE TOURNAMENT* i *ADMINISTRATION*. Klikom na karticu *CREATE TOURNAMENT*, korisnik će biti preusmjeren na pogled prikazan na slici 3.12.



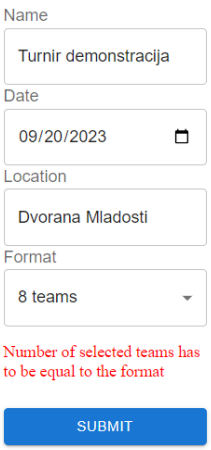
Slika 3.12 Pogled s formom za stvaranje turnira

Na ovom prikazu, korisnik odabire ekipe koje želi uključiti u turnir koji će stvoriti, zajedno s nazivom turnira, datumom i mjestom održavanja. Mjesto održavanja, odnosno lokacija, je proizvoljan tekst koji korisnik unosi, a nije prethodno zadana opcija. Taj tekst bi trebao odgovarati dvorani ili igralištu na kojem bi se takav turnir mogao održavati. U slučaju da korisnik unese datum koji je prije dana kreiranja turnira, prikazuje se validacijska poruka prikazana na slici 3.13.



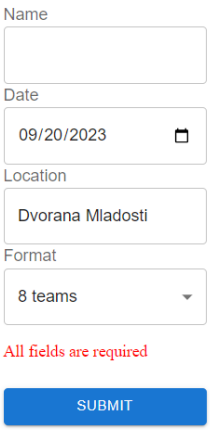
Slika 3.13 Validacijska poruka za kreiranje turnira s neispravnim datumom

U slučaju odabira broja ekipa koji ne odgovara odabranom formatu javlja se validacijska poruka prikazana na slici 3.14.



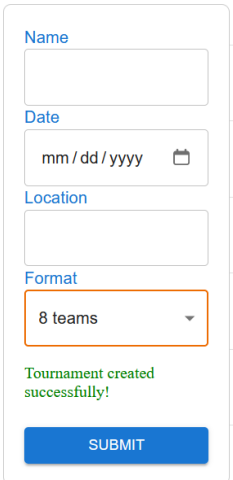
Slika 3.14 Validacijska poruka za kreiranje turnira sa krivim brojem timova

Dodatno, ako korisnik ostavi neko polje prazno javlja se validacijska poruka prikazana na slici 3.15.



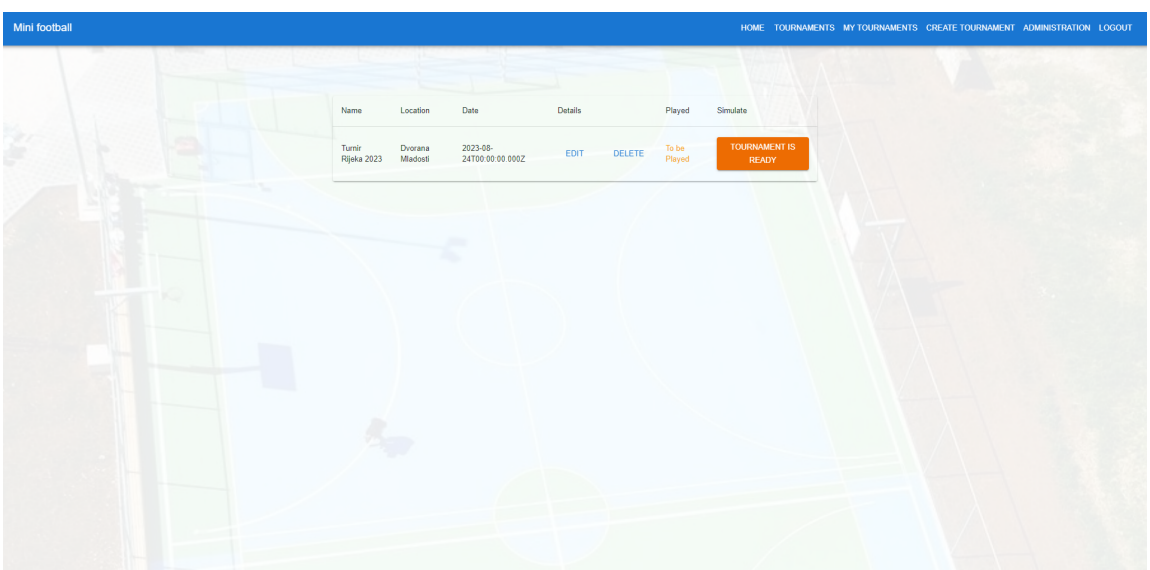
Slika 3.15 Validacijska poruka za kreiranje turnira s praznim poljem

Nakon uspješnog stvaranja turnira, korisnik će primiti obavijest o uspješnom stvaranju putem validacijske poruke, kako je prikazano na slici 3.16.



Slika 3.16 Validacijska poruka za uspješno kreiranje turnira

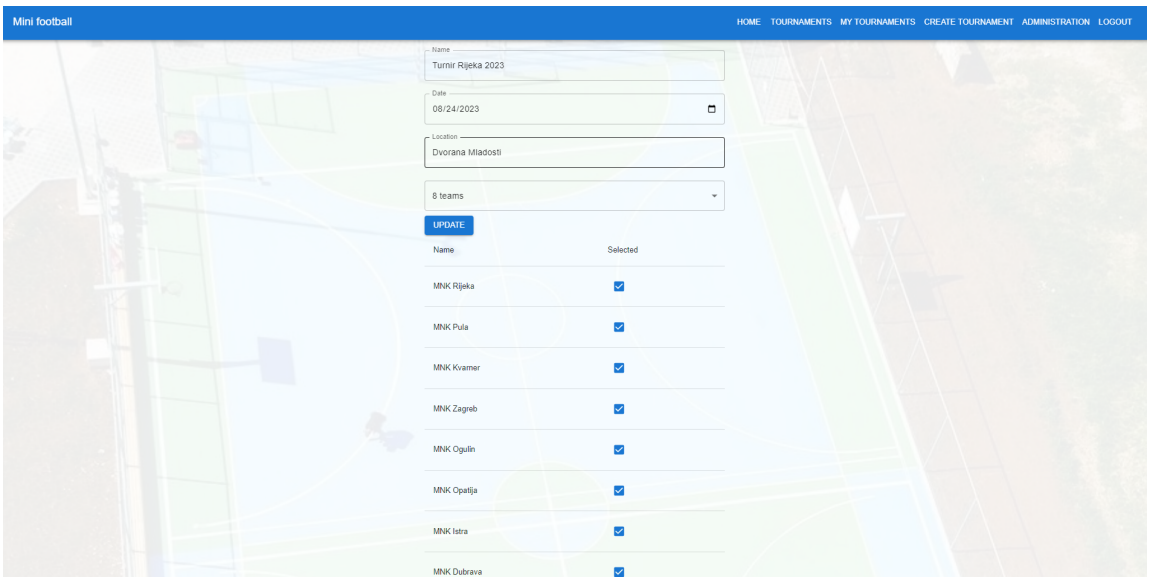
Nakon što je turnir kreiran, klikom na karticu *My tournaments* odvedeni smo na pogled prikazan na slici 3.17.



Slika 3.17 Prikaz turnira korisnika

U tablici su prikazani naziv turnira, njegova lokacija, datum te opcije za uređivanje i brisanje turnira. Također, tu je gumb koji označava da je turnir spreman za održavanje, nakon čega nije moguće izvršiti daljnje promjene. Dodatno, u tablici se

nalazi stupac *Played* koji označava je li turnir odigran. Ako turnir još nije odigran, u tom stupcu će pisati *To be played*, dok će u suprotnom pisati *Done*. Ako se klikne na gumb *Edit* prije nego što se turnir označi kao spreman za održavanje, korisniku će se prikazati pogled sa slike 3.18:

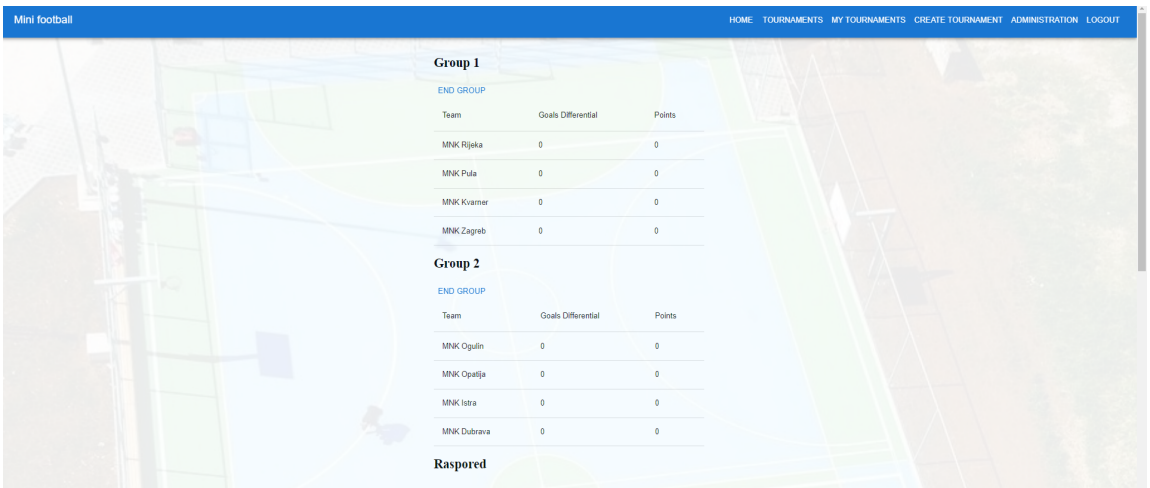


Slika 3.18 Prikaz izmjene turnira

Na pogledu sa slike 3.18, korisnik - odnosno organizator turnira - ima mogućnost promjene svih važnih podataka o turniru u slučaju pogreške ili izmjene planova. Klikom na odgovarajući gumb, korisnik može izvršiti promjene u detaljima turnira..

Nakon uspješne izmjene, korisnik se vraća na prikaz svojih turnira, gdje može vidjeti ažurirane informacije o turnirima, uključujući i turnir koji je upravo izmijenjen.

Klikom na gumb *Tournament is ready* označava se da je turnir spreman za početak. Nakon ovog klika, gumb nestaje. Ako korisnik ponovno klikne na gumb *Edit*, otvorit će se drugačiji prikaz, kao što je prikazano na slikama 3.19 i 3.20. Na slici 3.19 su prikazane grupe koje su generirane nasumično.



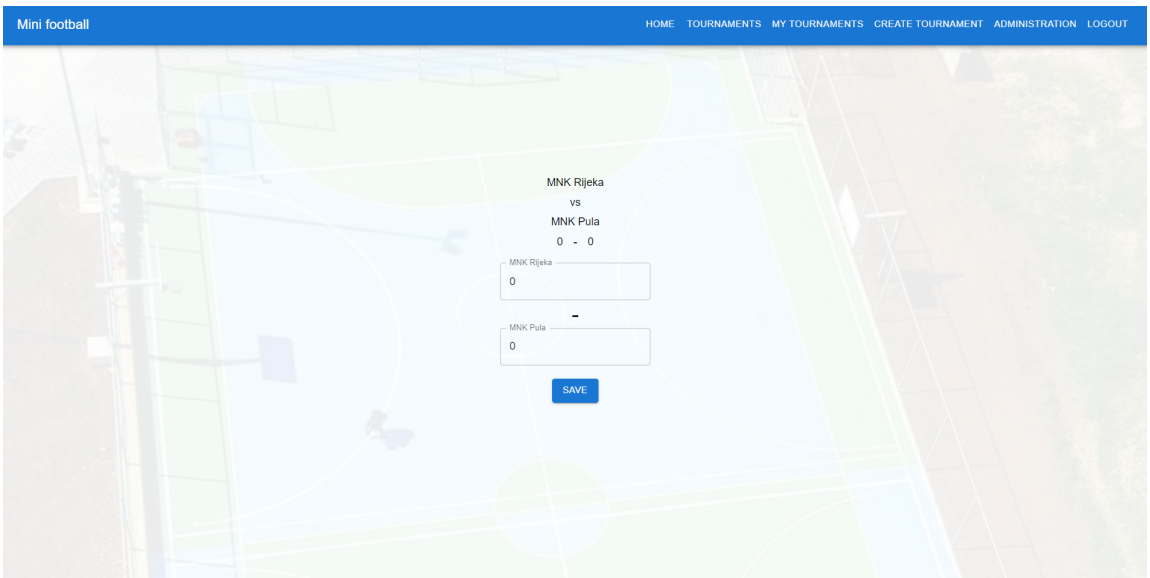
Slika 3.19 Grupe turnira

U svakoj grupi vidimo ime svake ekipe, njihov broj bodova, kao i gol razliku. Grupe su sortirane primarno po broju bodova, a u slučaju istog broja bodova prednost ima ekipa s boljom gol razlikom. Također, na tom prikazu postoji gumb *End group*. Kada se taj gumb klikne u obje grupe, to označava da su utakmice u toj grupi završene i generiraju se utakmice za 1. i 3. mjesto. Pobjednici skupina igraju finale, dok drugoplasirani igraju za 3. mjesto. Za svrhe ovog rada, izrađen je raspored utakmica na način da se prva polovica utakmica svake grupe igra prvog dana turnira, druga polovica se igra dan nakon, a finale i utakmica za 3. mjesto se igraju trećeg dana. Vrijeme odigravanja utakmica je postavljeno na termine u 10, 12, 14, 16, 18 i 20 sati. Ovo odražava situaciju na stvarnom turniru, uz pretpostavku da se sve utakmice održavaju na istom terenu. Raspored utakmica prikazan je na slici 3.20:

Phase	team1	team2	Date	Result
Group Stage	MNK Rijeka	MNK Pula	24-08-2023 at 10:00	EDIT
Group Stage	MNK Ogulin	MNK Opatija	24-08-2023 at 12:00	EDIT
Group Stage	MNK Pula	MNK Zagreb	24-08-2023 at 14:00	EDIT
Group Stage	MNK Opatija	MNK Dubrava	24-08-2023 at 16:00	EDIT
Group Stage	MNK Rijeka	MNK Zagreb	24-08-2023 at 18:00	EDIT
Group Stage	MNK Ogulin	MNK Dubrava	24-08-2023 at 20:00	EDIT
Group Stage	MNK Pula	MNK Kvarner	25-08-2023 at 10:00	EDIT
Group Stage	MNK Opatija	MNK Istra	25-08-2023 at 12:00	EDIT
Group Stage	MNK Rijeka	MNK Kvarner	25-08-2023 at 14:00	EDIT
Group Stage	MNK Ogulin	MNK Istra	25-08-2023 at 16:00	EDIT
Group Stage	MNK Kvarner	MNK Zagreb	25-08-2023 at 18:00	EDIT
Group Stage	MNK Istra	MNK Dubrava	25-08-2023 at 20:00	EDIT
3rd Place Game	To be determined	To be determined	26-08-2023 at 18:00	EDIT
Final	To be determined	To be determined	26-08-2023 at 20:00	EDIT

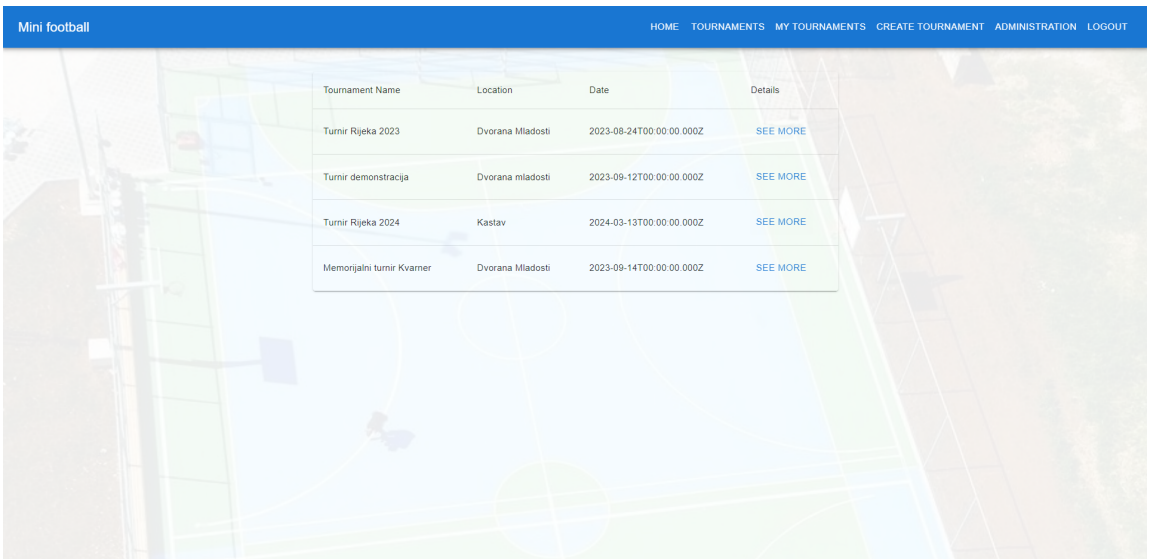
Slika 3.20 Raspored utakmica turnira

Klikom na gumb *edit* pored pojedine utakmice, korisnik je odveden na prikaz gdje može upisati rezultat što se može vidjeti na slici 3.21, dok klikom na gumb *save* se spašava rezultat utakmice u bazu podataka i vodi korisnika natrag na administraciju turnira.



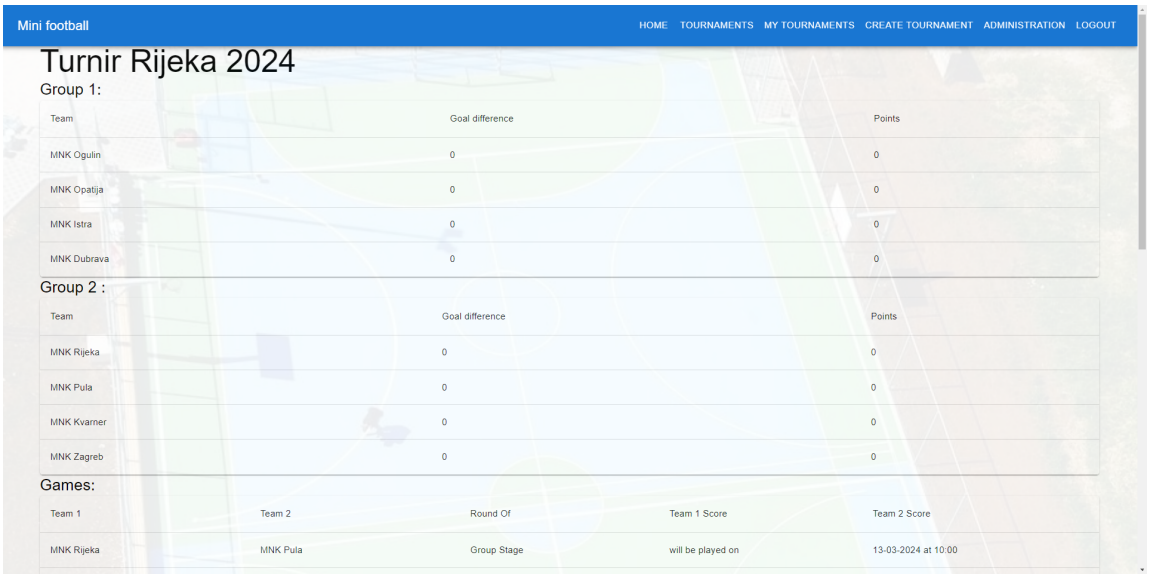
Slika 3.21 Pogled upisivanja rezultata utakmica

Klikom na karticu *TOURNAMENTS* korisnik je odveden na pogled na slici 3.22 gdje može vidjeti sve turnire. Također korisnik ne mora biti prijavljen kako bi mogao pristupidi pogledu sa slike 3.22.



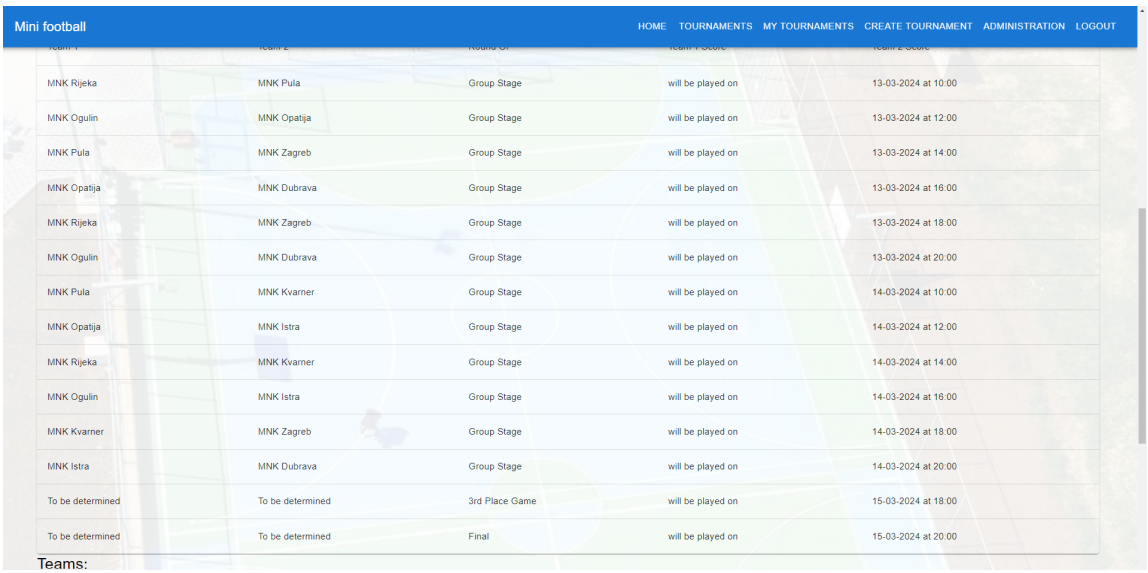
Slika 3.22 Pogled svih turnira

Klikom na gumb *SEE MORE* korisnik je odveden na pogled gdje može vidjeti sve informacije o turniru kao što je prikazano na slikama 3.23, 3.24 i 3.25.



Slika 3.23 Pogled svih turnira

Na slici 3.23 se nalaze grupe turnira dok se na slici 3.24 nalaze sve utakmice turnira.



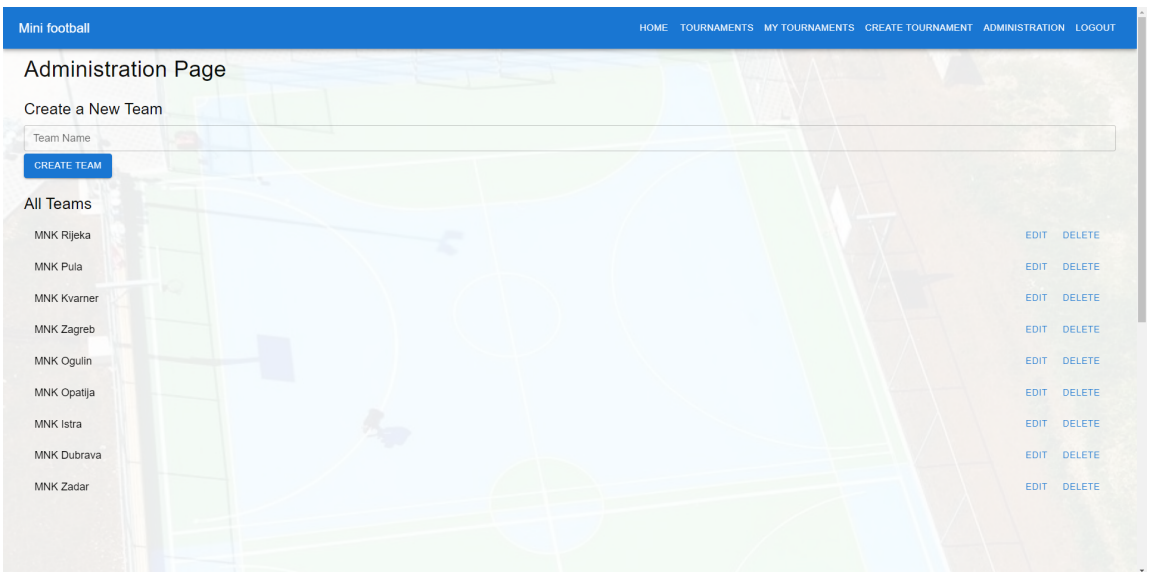
Slika 3.24 Pogled svih turnira

Kao zadnji dio pogleda,na slici 3.25 se nalazi popis svih ekipa koje se natječu u turniru.

Slika 3.25 Pogled svih turnira

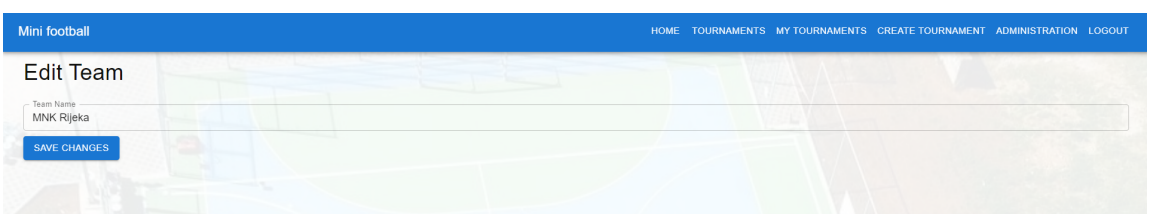
Za završetak opisa *frontenda*, biti će predstavljen administratorski segment odgovoran za stvaranje, uređivanje i brisanje natjecateljskih timova, pojedinačnih igrača unutar tih timova te administraciju korisnika aplikacije. Pristup ovom segmentu imaju isključivo korisnici kojima je varijabla *superUser* u bazi podataka postavljena na istinitu vrijednost. Administratorski pregled se sastoji od tri osnovne kompo-

nente. Prva komponenta je posvećena upravljanju timovima i prikazana je na slici 3.26.



Slika 3.26 Prikaz administracije ekipa

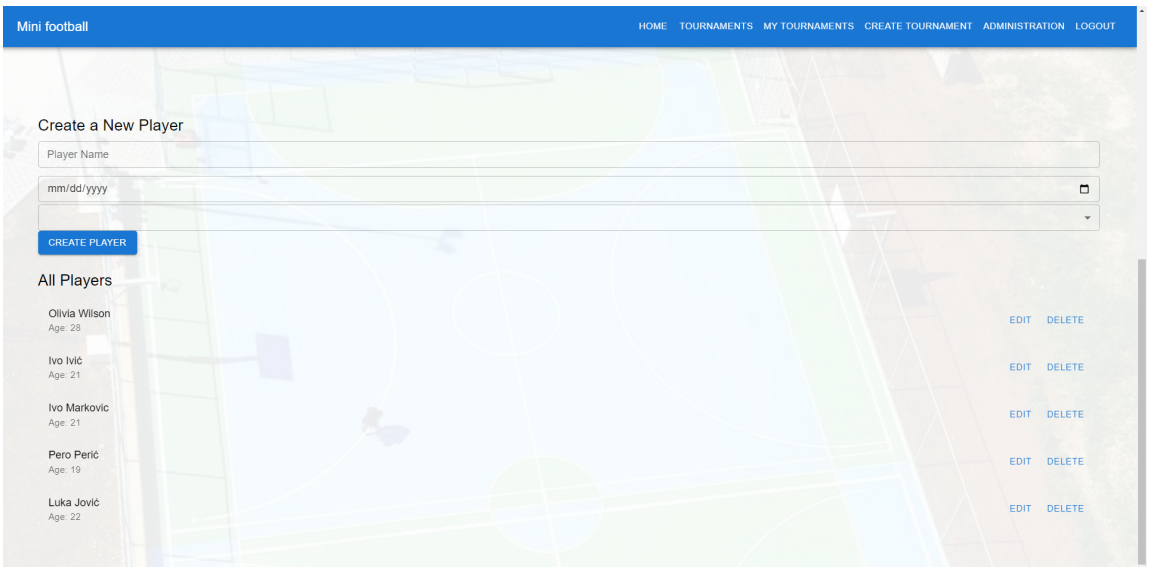
Na slici 3.26 je prikazana mogućnost stvaranja nove ekipe putem unosa imena tima koji želimo kreirati. Također su dostupne opcije brisanja tima putem gumba *Delete* i izmjene podataka putem gumba *Edit*. Klikom na gumb *Edit*, korisnik se preusmjerava na prikaz sa slike 3.27:



Slika 3.27 Izmjena ekipe

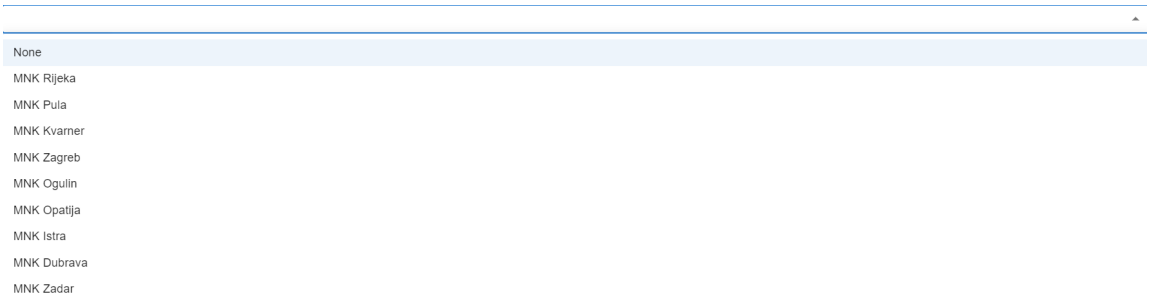
Izmjenom naziva ekipe i klikom na gumb *Save changes* pohranjuju se željene promjene, a aplikacija preusmjerava korisnika na administratorsku stranicu.

Na drugom dijelu administratorske stranice nalazi se segment posvećen administraciji igrača. Korisnicima je omogućeno stvaranje, brisanje i izmjena igrača na sličan način kao i za timove. To je prikazano na slici 3.28.



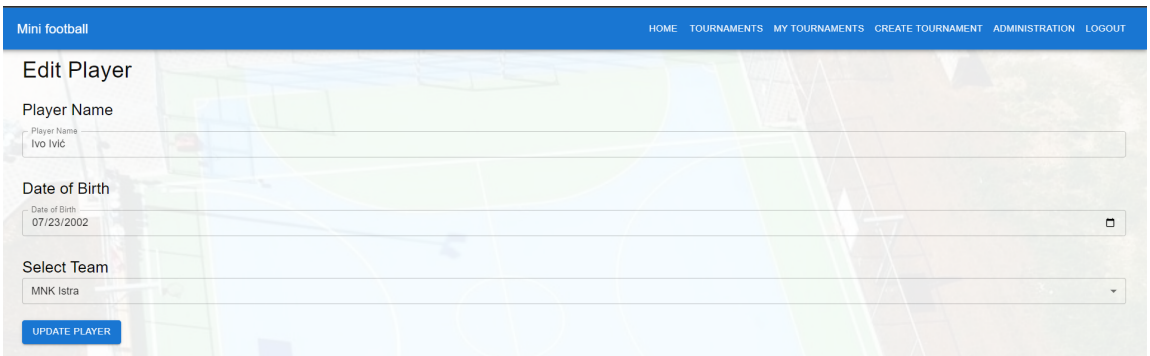
Slika 3.28 Prikaz administracije igrača

Unosom imena igrača, odabirom datuma rođenja te biranjem ekipe, omogućuje se stvaranje novog igrača. Za odabir ekipe, korisnik može kliknuti na donji dio trake što će otvoriti padajući izbornik, prikazan na slici 3.29.



Slika 3.29 Padajući izbornik ekipa

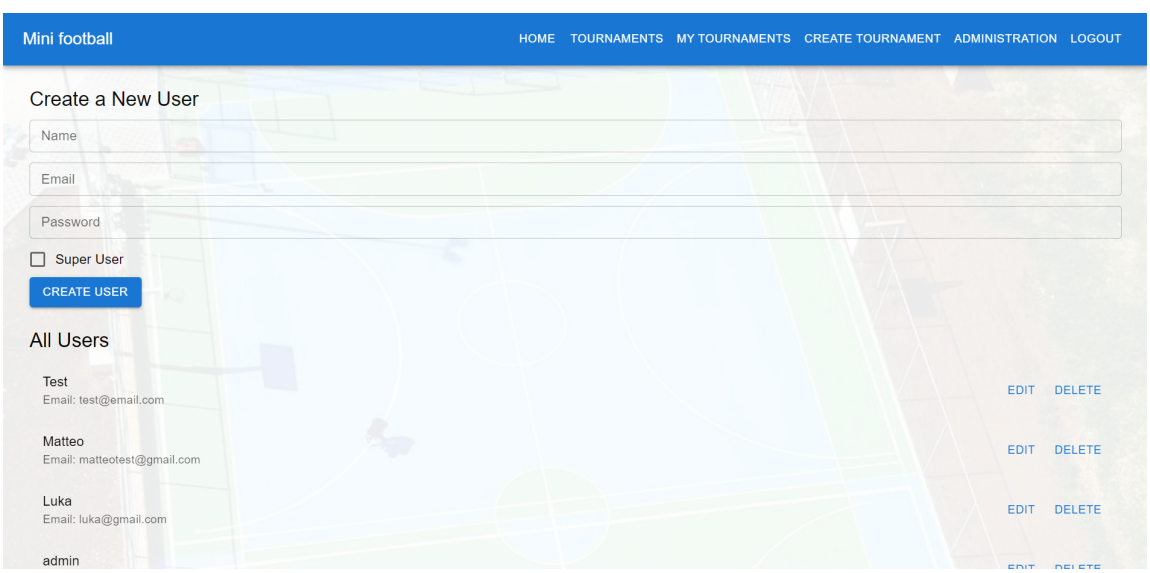
Kao i kod administracije timova, pritiskom na gumb *Edit*, korisnik će biti preusmjeren na stranicu koja mu omogućava uređivanje podataka igrača, kako je prikazano na slici 3.30.



Slika 3.30 Prikaz izmjene igrača

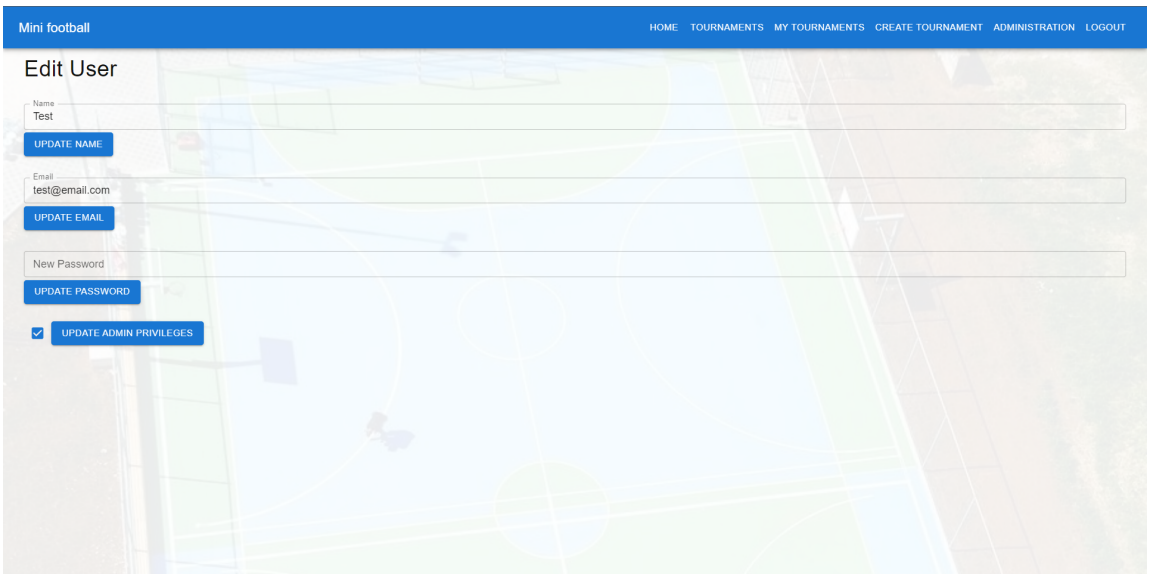
Nakon izmjene željenih podataka i pritiskom na gumb *Update player*, promjene će biti spremljene, a korisnik će ponovno biti preusmjeren na početnu stranicu administratorskog sučelja.

Kao zadnji segment stranice administracije biti će objašnjen dio odgovoran za administraciju korisnika čiji se izgled može vidjeti na slici 3.31.



Slika 3.31 Prikaz administracije svih korisnika

Klikom na gub *DELETE* briše se korisnik iz baze podataka, dok klikom na gumb *EDIT* korisnika se vodi na pogled prikazan na slici 3.32.



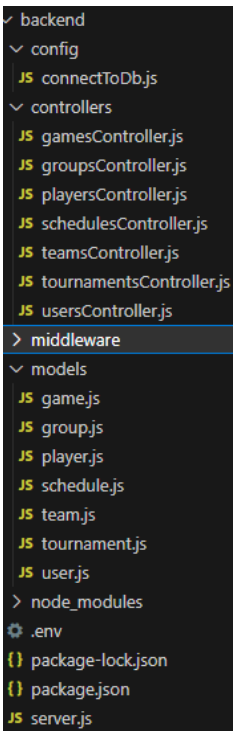
Slika 3.32 Prikaz izmjene korisnika

Izmjenom pojedinog podatka o korisniku i klikom na pripadajuću gumb *update* radi se izmjena te je korisnik preusmjeren na početnu stranicu administracije gdje se nalaze izmijenjeni podaci. Dodavanjem ili micanjem privilegija administratora, korisniku se daje ili miče mogučnost administracije timova, igrača i korisnika.

3.3 *Backend* aplikacije

3.3.1 Izgled direktorija

U ovom dijelu će biti opisana serverska logika Express.js backend dijela korištenog za ovaj projekt. Datoteke poslužitelja Express.js-a smještene su unutar direktorija *backend*, koji ima strukturu prikazanu na slici 3.33.



Slika 3.33 struktura direktorija backend

3.3.2 Osnovna logika

Osnovna logika Express.js servera je da se sastoji od modela, koji odgovaraju modelima u bazi podataka, i kotronolera, koji rade manipulaciju sa podacima i modelima. Dodatno postoji i takozvani *middleware* koji se koristi kao posrednik između različitih komponenti sustava ili kao sloj funkcionalnosti koji se primjenjuje na zahtjeve

ili podatke dok prolaze kroz sustav, ali za svrhe ovog rada, jedini *middleware* je korišten pri autentifikaciji korisnika. "Najvažnija" datoteka u cijelom *backendu* je server.js. Pomoću nje se se odrađuju sve bitne zadaće poslužitelja a u ovom slučaju to je sljedeće:

- Učitavanje varijabli okruženja: Provjerava se okruženje i učitavaju se varijable okruženja iz .env datoteke ako se aplikacija ne izvodi u produkcijskom okruženju.
- Uvoz zavisnosti: Uvoze se potrebne zavisnosti kao što su express, cors, cookie-parser, te lokalni moduli za kontrolere i middleware-ove.
- Kreiranje instance aplikacije: Kreira se instanca Express.js aplikacije.
- Konfiguracija aplikacije: Postavljaju se osnovne konfiguracije za aplikaciju, kao što su parsiranje JSON podataka, upotreba cookie-parser za rukovanje kolačićima i omogućavanje CORS (Cross-Origin Resource Sharing) za podršku zahtjevima s drugih domena.
- Spajanje na bazu podataka: Poziva se funkcija connectToDb kako bi se aplikacija povezala s bazom podataka.
- Rute: Definiraju se različite rute koje omogućuju komunikaciju s aplikacijom. Rute su podijeljene u logičke skupine, svaka skupina ima odgovarajući kontroler koji obrađuje logiku zahtjeva.
- Povezivanje ruta s kontrolerima: Rute su povezane s odgovarajućim kontrolerima koji sadrže logiku obrade zahtjeva.

Na slici 3.34 prikazan kod datoteke "server.js" koji je odgovoran za sve prethodno spomenute zadatke poslužitelja, osim definiranja ruta, njihovog povezivanja s kontrolerima i određivanja porta na kojem aplikacija "sluša":

```
if (process.env.NODE_ENV != "production") {
    require("dotenv").config();
}

const express = require("express");
const cors = require("cors");
const cookieParser = require("cookie-parser");
const connectToDb = require("./config/connectToDb");

//controllers
const usersController = require("./controllers/usersController");
const tournamentsController = require("./controllers/tournamentsController");
const teamsController = require("./controllers/teamsController");
const playersController = require("./controllers/playersController");
const requireAuth = require("./middleware/requireAuth");
const groupsController = require('./controllers/groupsController');
const schedulesController = require("./controllers/schedulesController")
const gamesController = require("./controllers/gamesController")

const app = express();


app.use(express.json());
app.use(cookieParser());
app.use(
  cors({
    origin: true,
    credentials: true,
  })
);

// Connect to database
connectToDb();
```

Slika 3.34 Datoteka server.js

Na slici 3.35 prikazan je kod koji ilustrira preostale funkcionalnosti iz datoteke "server.js" koje nisu bile obuhvaćene na slici 3.34.

```
  //groups
app.get('/groups', groupsController.getAllGroups);
app.post('/groups', groupsController.createGroup);
app.delete('/groups/:id', groupsController.deleteGroup);
app.get('/groups/:id', groupsController.getGroupById);
app.put('/groups/:id', groupsController.updateGroup);
app.get('/finish-group/:id',groupsController.finishGroup)
//schedules

app.get("/schedule",schedulesController.getSchedules)
app.post("/schedule",schedulesController.createSchedule)
app.get("/schedule/:id",schedulesController.getScheduleById)
app.get("/schedule-finals/:id",schedulesController.generateFinals)

// games
app.post('/game', gamesController.createGame);
app.get('/game', gamesController.getAllGames);
app.get('/game/:id', gamesController.getGameById);
app.put('/game/:id', gamesController.updateGameById);
app.delete('/game/:id', gamesController.deleteGameById);

  // Start server
  app.listen(process.env.PORT);
```

Slika 3.35 Datoteka server.js

3.3.3 Modeli

U Express.js, web aplikacijskom okviru za Node.js, modeli se odnose na način organizacije i upravljanja podacima. Iako sam Express.js ne pruža ugrađeni sloj za modeliranje podataka, u ovom radu se koristi Mongoose za MongoDB, što ispunjava tu potrebu.

Modeli obično predstavljaju strukturirane podatke kao što su informacije o korisnicima, unosima rezultata turnira itd. Oni omogućuju razdvajanje logike baze podataka od ostalih dijelova aplikacije, kao što su rute i kontroleri. Modeli olakšavaju komunikaciju s bazom podataka, izvođenjem upita za dohvaćanje, unos, ažuriranje ili brisanje podataka.

3.3.4 Kontroleri

Kontroler je sastavni dio arhitekture koji se koristi za organizaciju i upravljanje rutama i logikom aplikacije. Kontroleri su odgovorni za obradu zahtjeva koji dolaze od klijenta (pretraživača, mobilne aplikacije itd.) te pružaju odgovarajuće odgovore.

Kontroleri služe kako bi odvojili poslovnu logiku od rutiranja i pomažu u održavanju čiste i organizirane strukture aplikacije. Svaka ruta u Express.js aplikaciji može biti povezana s odgovarajućim kontrolerom. Kontroleri sadrže funkcije koje se izvršavaju kada dolazi zahtjev na određenu rutu.

Uobičajena struktura kontrolera u Express.js uključuje funkcije poput *get*, *post*, *put*, *delete*, koje odgovaraju HTTP metodama za rute. Ove funkcije obično obrađuju podatke dobivene iz zahtjeva, komuniciraju s modelima (ako se koristi odvojena slojevita arhitektura), izvršavaju potrebne operacije nad podacima i generiraju odgovor koji se šalje klijentu.

3.3.5 Logika kontrolera korisnika

U ovom potpoglavlju će biti objašnjeno nekoliko funkcionalnosti koje su povezane s korisnicima. Dok smo u dijelu za *frontend* objašnjavali izgled i opisivali dostupne funkcionalnosti ovog projekta, sada će detaljnije biti pojašnjen kako te funkcionalnosti zapravo funkcioniraju.

Na slici 3.36 se nalazi model korisnika koji prikazuje sve atribute koje korisnik posjeduje:

```
const mongoose = require("mongoose")
const userSchema = new mongoose.Schema({
    name:{
        type:String,
        required:true,
    },
    email: {
        type:String,
        required:true,
        unique:true,
        lowercase:true,
        index:true
    },
    password: {
        type:String,
        require: true,

    },
    superUser:{
        type:Boolean,
        default:false
    }

});

const User = mongoose.model('User', userSchema);

module.exports = User;
```

Slika 3.36 Model korisnika

Kao što je prikazano na slici 3.36, korisnik mora unijeti svoje ime, e-mail adresu i lozinku. Dodatno, postoji atribut *SuperUser* koji označava je li korisnik administrator, što zavisi od toga je li ta vrijednost istinita ili ne.

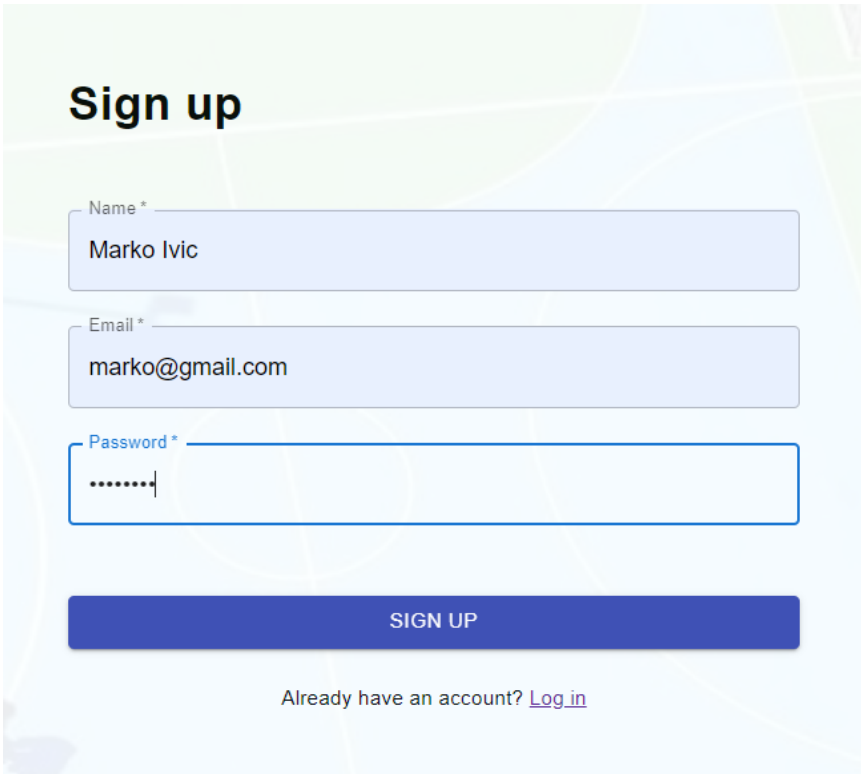
Prvo će biti objašnjen proces registracije korisnika i što se događa kada korisnik klikne na gumb na registracijskom obrascu.

Na slici 3.37 se nalazi funkcija *signup* koja obavlja proces kreiranja korisnika.

```
async function signup(req, res) {
  try {

    const { name, email, password, superUser } = req.body;


    const hashedPassword = bcrypt.hashSync(password, 8);


    await User.create({ name, email, password: hashedPassword, superUser });


    res.sendStatus(200);
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.37 funkcija signup

Prvo što se događa u funkciji *signup* jest da se iz tijela zahtjeva preuzimaju podaci o imenu, e-mailu i lozinki. Zatim se provodi kriptiranje, odnosno hashiranje lozinke. Nakon toga, kreira se korisnik s imenom, hashiranom lozinkom i e-mailom, te se takav korisnik pohranjuje u bazu podataka. Za ovaj primjer, biti će registriran korisnik "Marko Ivić" s e-mailom "marko@gmail.com" i lozinkom "12345678", kao što je prikazano na slici 3.38.



Slika 3.38 registracija korisnika

Klikom na gumb *SIGN UP* dobiva se zapis u bazi podataka prikazan na slici 3.39.


```
_id: ObjectId('64e14d9de807c5e3f51565ff')
name: "Marko Ivic"
email: "marko@gmail.com"
password: "$2a$08$U/MXsHbAwyrsEcy8Q10CT.2voggCtuliG1MUAXnecEPbIh6Et57va"
superUser: false
__v: 0
```

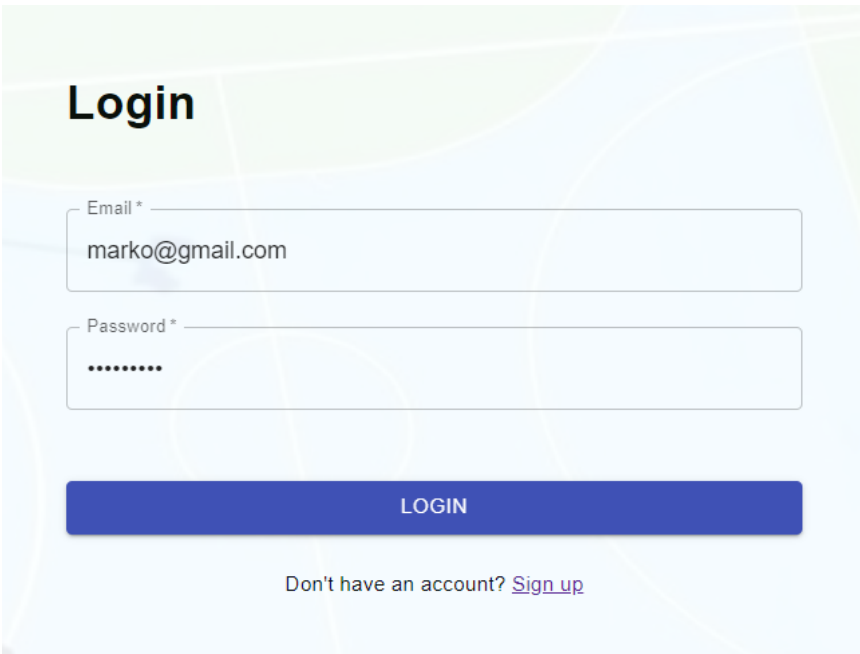

Slika 3.39 Prikaz zapisa korisnika u bazi

Nakon što je korisnik registriran, može se prijaviti putem funkcije za prijavu *login*.Funkcija je prikazana na slici 3.40.

```
async function login(req, res) {
  try {

    const { email, password } = req.body;

    const user = await User.findOne({ email });
    if (!user) return res.sendStatus(401);

    const passwordMatch = bcrypt.compareSync(password, user.password);
    if (!passwordMatch) return res.sendStatus(401);


    const exp = Date.now() + 1000 * 60 * 60 * 24 * 30;
    const token = jwt.sign({ sub: user._id, exp, name: user.name, superUser: user.superUser }, process.env.SECRET,);


    res.cookie("Authorization", token, {
      expires: new Date(exp),
      httpOnly: true,
      sameSite: "lax",
      secure: process.env.NODE_ENV === "production",
    });


    res.json({ token })
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.40 Login funkcija

Nakon što smo registrirali svog korisnika, sada se možemo sa istim podacima prijaviti. Kada kliknemo gumb login kao što je prikazano na slici 3.41:

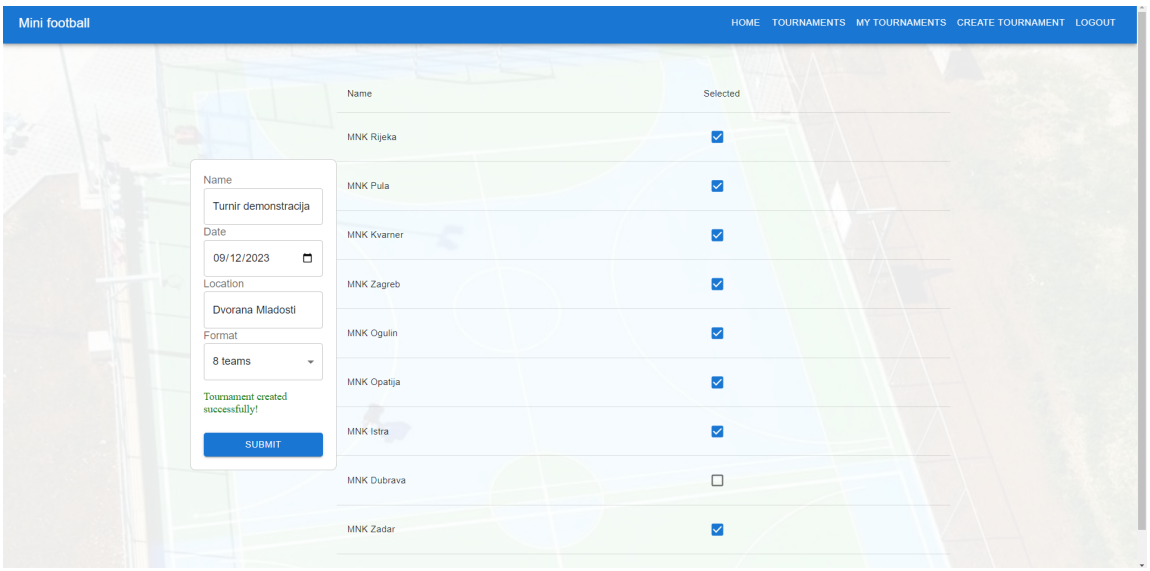


Slika 3.41 Login funkcija

pokrenut će se *login* funkcija. Na početku će se sa tijela zahtjeva uzeti e-mail i lozinka. Zatim će se pokušati naći korisnika sa tim e-mailom, ako ne uspije vratit će se korisniku povratna informacija sa statusnim kodom 401. Također će se poslati isti kod u slučaju da je korisnik upisao krivu lozinku.Funckija provjerava je li upisana lozinka jednaka onoj koja se nalazi u bazi podataka i ako nije vraća statusni kod 401. U slučaju da su jednake, kreira se token,tj. kolačić koji traje mjesec dana u koji se upisuju vrijednosti id-a korisnika, njegovo ime i vrijednost je li korisnik administrator ili nije i kao takav se natrag šalje korisniku. Korisnika se preusmjerava na početnu stranicu nakon koje korisnik može kreirati svoj turnir, pa će to biti slijedeći korak.

3.3.6 Logika kreiranja turnira

Biti će kreiran turnir s 8 ekipa, nazivom Turnir demonstracija, koji će se početi odvijati 12.9.2023. na lokaciji Dvorana mladosti kao što je slučaj na slici 3.42.



Slika 3.42 Kreiranje turnira

Nakon klika na gumb *Submit* poziva se poduža funkcija *createTournament* te će biti objašnjena dio po dio jer je previše duga da se prikaže u samo jednoj slici. Početak funkcije nalazi se na slici 3.43.

```
async function createTournament(req, res) {
  try {
    const token = req.cookies.Authorization;
    const decoded = jwt.verify(token, process.env.SECRET);
    const { name, date, location, format, selectedTeams,replace } = req.body;
    console.log("start" + replace)

    const existingTournament = await Tournament.findOne({ name });

    if(replace && existingTournament){
      await existingTournament.deleteOne()
    }

    else if (existingTournament) {

      return res.status(400).json({ error: 'Tournament name must be unique' });
    }

    const tournament = await Tournament.create({
      name,
      date,
      location,
      format,
      user: decoded.sub,
      teams: selectedTeams,
      schedule:null
    });

    const group1 = new Group({ tournament: tournament, teams: [],teamScores:[] });
    const group2 = new Group({ tournament: tournament, teams: [],teamScores:[] });
```

Slika 3.43 Kreiranje turnira

Ulaskom u funkciju, provodi se autentikacija korisnika putem JWT tokena. U slučaju da token nije poslan u zahtjevu, izazvat će se iznimka te neće biti moguće kreirati turnir. Također, ukoliko se zbog bilo kojeg drugog razloga pojavi iznimka, to će spriječiti kreiranje turnira, a vratit će se statusni kod 500 koji će kasnije biti prikazan.

Nakon uspješne autentikacije, iz tijela zahtjeva se preuzimaju podaci kao što su ime turnira, datum, lokacija, format i timovi koji sudjeluju. Također, postoji varijabla *replace* koja se koristi za označavanje zamjene ili izmjene turnira. Međutim, za ovaj trenutni slučaj to nije primjenjivo.

Nadalje, provjerava se postoji li već turnir s istim imenom. Ako postoji i varijabla *replace* je istinita, pronađeni turnir će biti obrisan. Ako je varijabla *replace* lažna, korisnik će dobiti povratnu informaciju da ime turnira mora biti jedinstveno, odnosno da već postoji turnir s tim imenom.

Poslije provedenih provjera, kreira se novi turnir s odgovarajućim imenom, datumom početka, lokacijom, formatom, korisnikom koji ga je stvorio, sudjelujućim timovima i privremeno postavljenim rasporedom (trenutno ima vrijednost *null*). Turnir je stvoren prema modelu prikazanom na slici 3.44.

```
const User = require("./user");
const mongoose = require("mongoose");

const tournamentSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,

  },
  date:{
    type: Date,
    required: true
  },
  location: {
    type: String
  },
  format: {
    type: String,
    required: true,
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
  teams: {
    type: [mongoose.Schema.Types.ObjectId],
    ref: 'Team',
    required: true,
  },
  isDone: {
    type: Boolean,
    required: true,
    default: false
  },
  schedule: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Schedule',
  },
  finalised:{
    type:Boolean,
    default:false
  }
});

const Tournament = mongoose.model('Tournament', tournamentSchema);
module.exports = Tournament;
```

Slika 3.44 Model turnira

Nakon što je turnir uspješno kreiran, slijedi korak stvaranja dviju grupa. U ovom radu je korišten format turnira s osam ekipa podijeljenih u dvije grupe. Za svaku grupu, koristi se prethodno kreirani turnir kao nadređeni element. Timovi i rezultati, uključujući bodove i gol-razliku (koji se prate putem varijable *goalsScored*), trenutno su postavljeni kao prazan niz.

Na slici 3.45 se nalazi ostatak funkcije *createTournament*:

```
    const numTeams = selectedTeams.length;
    for (let i = 0; i < numTeams; i++) {
      if (i < numTeams / 2) {
        group1.teams.push(selectedTeams[i]);
        group1.teamScores.push({ team:new mongoose.Types.ObjectId(selectedTeams[i]), goalsScored: 0, points: 0 })
      } else {
        group2.teams.push(selectedTeams[i]);
        group2.teamScores.push({ team:new mongoose.Types.ObjectId(selectedTeams[i]), goalsScored: 0, points: 0 })
      }
    }


    await group1.save();
    await group2.save();




    const schedule = await createSchedulelocal(tournament,group1,group2)
    tournament.schedule = schedule
    await tournament.save()

    const simplifiedTournament = {
      _id: tournament._id,
      name: tournament.name,
      date: tournament.date,
      location: tournament.location,
      format: tournament.format,
      user: tournament.user,
      teams: selectedTeams,

    };

    res.json(simplifiedTournament)


  } catch (error) {
    console.log(error);
    res.sendStatus(500).json(s);
  }
}
```

Slika 3.45 Funkcija kreiranja turnira, 2. dio

Ostatak funkcije djeluje na sljedeći način: koristeći *for* petlju ekipe se pseudo-nasumično dodjeljuju grupama i postavljaju im se početne vrijednosti *goalsScored* i *points* na 0. Nakon toga, grupe se spremaju u bazu podataka funkcijama *save*, a zatim se kreira raspored utakmica, o čemu će biti više objašnjeno nešto kasnije. Nakon što je raspored kreiran, isti se dodjeljuje turniru i taj turnir se zatim sprema u bazu. Na kraju, stvara se varijabla *simplifiedTournament* kako bi kreirani turnir mogao biti vraćen korisniku, odnosno frontend serveru, u JSON formatu. Na dnu funkcije nalazi se *catch* blok za hvatanje iznimki, što je ranije spomenuto.

Nadalje, sada će biti objašnjen proces kreiranja rasporeda putem funkcije *createSchedulelocal*, koja je također kompleksna i čiji prvi dio je prikazan na slici 3.46:

```
const createRoundRobinSchedule = async (group,groupNumber) => {
  const games = [];
  const numTeams = group.teams.length;

  const startDateX = new Date(group.tournament.date.getTime() +  24 * 60 * 60 * 1000)
  const startDateY = new Date(group.tournament.date.getTime())
  let gameTime = 0
  let gameDay = 0
  for (let i = 0; i < numTeams - 1; i++) {
    for (let j = i + 1; j < numTeams; j++) {

      const startDate = gameDay%2 ? startDateX : startDateY;
      gameDay++;
      let startHour;
      if (gameTime === 0) {
        startHour = groupNumber === 1 ? 10 : 12;
      } else if (gameTime === 1) {
        startHour = groupNumber === 1 ? 14 : 16;
      } else {
        startHour = groupNumber === 1 ? 18 : 20;
      }

      startDate.setHours(startHour);
      const newGame = new Game({
        team1: group.teams[i],
        team2: group.teams[j],
        roundOf: "Group Stage",
        tournament: group.tournament,
        startDate,
        group: group,
      });
      await newGame.save();
      games.push(newGame._id);
      gameTime++;

      if(gameTime === 3)
      gameTime = 0
    }
  }

  return games;
};
```

Slika 3.46 Funkcija kreiranja rasporeda, 1. dio

Prvi dio ove funkcije zapravo predstavlja pomoćnu asinkronu funkciju *CreateRoundRobinSchedule*, koja služi za stvaranje rasporeda za pojedinu grupu za dva dana kada se odigravaju utakmice te grupe. To se postiže tako da se odaberu dva datuma: jedan na dan početka turnira i drugi na dan nakon toga. Zatim, putem dvije *for* petlje, stvaraju se sve moguće kombinacije utakmica unutar te grupe, slično sustavu

natjecanja "svatko sa svakim", ali bez uzvratnih utakmica. U okviru petlji, koristi se varijabla *gameDay* kako bi se odredio dan na temelju njenog parnosti, nakon čega se ta varijabla povećava. Također, koristi se varijabla *gameTime* za određivanje vremena igranja utakmice. Ovisno o vrijednosti varijable *gameTime*, određuje se satnica utakmice: za *gameTime* vrijednosti 0, utakmica se igra u 10 sati za grupu 1 i u 12 sati za grupu 2; za *gameTime* vrijednosti 1, utakmica se igra u 14 sati za grupu 1 i 16 sati za grupu 2; te za *gameTime* vrijednosti 2, utakmica se igra u 18 sati za grupu 1 i 20 sati za grupu 2. Nakon određivanja vremena, sati se postavljaju u varijablu *startDate*, nakon čega se stvara utakmica s dva tima, fazom natjecanja kojoj utakmica pripada, turnirom, vremenom početka i grupom kojoj utakmica pripada. Na kraju svake petlje, varijabla *gameTime* se povećava, te ako je vrijednost 3, postavlja se natrag na 0 za sljedeću iteraciju petlje. Na kraju, pomoćna funkcija vraća utakmice te grupe putem varijable *games*. Sada možemo preći na drugi dio funkcije, koji je prikazan na slici 3.47:

```
    const group1Games = await createRoundRobinSchedule(group1,1);
    const group2Games = await createRoundRobinSchedule(group2,2);



    const thirdPlaceGame = new Game({
      team1: null,
      team2: null,
      roundOf: "3rd Place Game",
      tournament: tournament,
     startDate: new Date(tournament.date.getTime() + 2 * 24 * 60 * 60 * 1000).setHours(18),
    });
    await thirdPlaceGame.save();


    const finalGame = new Game({
      team1: null,
      team2: null,
      roundOf: "Final",
      tournament: tournament,
      startDate: new Date(tournament.date.getTime() + 2 * 24 * 60 * 60 * 1000).setHours(20),
    });
    await finalGame.save();

    const schedule = new Schedule({
      tournament,
      group1,
      group2,
      group1Games,
      group2Games,
      thirdPlaceGame: thirdPlaceGame._id,
      final: finalGame._id,
    });

    await schedule.save();

    return schedule
  } catch (error) {
    console.error('Error creating schedule:', error);

  }
}
```

Slika 3.47 Funkcija kreiranja rasporeda, 2. dio

U nastavku funkcije, prvo se kreiraju utakmice za dvije grupe putem prethodno opisane pomoćne funkcije. Grupa i broj grupe (1 ili 2) se šalju kao argumenti toj pomoćnoj funkciji, koja zatim vraća utakmice za tu grupu. Nakon što se taj korak obavi, kreiraju se finalne utakmice kao i utakmica za 3. mjesto. Vrijeme tih utakmica je postavljeno na 2 dana nakon početka turnira, pri čemu je za finalnu utakmicu vrijeme 20 sati, a za utakmicu za 3. mjesto je vrijeme 18 sati. Timovi su postavljeni na vrijednost *null* jer još nisu poznate ekipe koje će sudjelovati u tim utakmicama.

Naposljetku se stvara raspored povezan s turnirom kojem pripada, grupama, njihovim utakmicama, finalnom utakmicom i utakmicom za 3. mjesto. Nakon toga, raspored se sprema upotrebom funkcije *save* te se putem naredbe return vraća stvoren raspored. Također, ispod toga nalazi se *catch* blok za slučaj da dođe do iznimke,

pri čemu se nastala greška zapisuje u konzolu.

U administrativnom dijelu aplikacije, dio odgovoran za kreiranje, brisanje i izmjenu korisničkih podataka prikazanih na slici 3.31 ima svoju poslužiteljsku logiku sa funkcijama koje će biti objašnjene u nastavku. Prva takva je funkcija odgovorna za kreiranje novog korisnika, prikazana na slici 3.48

```
async function createUser(req, res) {
  try {

      const token = req.cookies.Authorization;
      const decoded = jwt.verify(token, process.env.SECRET);
      console.log(decoded)
      const admin = await User.findById(decoded.sub)
      console.log(admin)
      if (!admin.superUser)
        res.sendStatus(401)

    const { name, email, password, superUser } = req.body;
    const hashedPassword = bcrypt.hashSync(password, 8);
    const user = await User.create({ name, email, password: hashedPassword, superUser });
    res.status(201).json(user);
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.48 Funkcija kreiranja korisnika

U funkciji se prvo radi autentikacija i autorizacija korisnika, te ako korisnik ima odgovarajuće pravo, to jest administrator je, kreira se novi korisnik sa podacima poslanih sa klijentske strane te se klijentskoj strani vraća odgovor o uspješnoj kreaciji korisnika. Sve naredne funkcije imaju istu logiku autorizacije i autentikacije pa se u nastavku to neće ponovno objašnjavati.

Funkcija *getUsers* prikazana na slici 3.49 je odgovorna za dohvaćanje svih postojećih korisnika na stranicu administracije.

```
async function getUsers(req, res) {
  try {
      const token = req.cookies.Authorization;
      const decoded = jwt.verify(token, process.env.SECRET);
      console.log(decoded)
      const admin = await User.findById(decoded.sub)
      console.log(admin)
      if (!admin.superUser){
        res.sendStatus(401)
        return;
      }

    const users = await User.find();
    res.json(users);
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.49 Funkcija dohvaćanja korisnika

Funkcija *getUsers* ima jednostavnu zadaću dohvaćanja svih postojećih korisnika te slanju istih klijentskoj strani kako bi se mogli prikazati korisniku.

Funkcija *deleteUser* prikazana na slici 3.50 odgovorna je za brisanje pojedinog korisnika metodom *findByIdAndDelete* nakon čega je klijentska strana obaviještena o uspješnom brisanju.

```
async function deleteUser(req, res) {
  try {
    const token = req.cookies.Authorization;
    const decoded = jwt.verify(token, process.env.SECRET);
    console.log(decoded)
    const admin = await User.findById(decoded.sub)
    console.log(admin)
    if (!admin.superUser){
      res.sendStatus(401)
      return;
    }
    const userId = req.params.id;
    const user = await User.findByIdAndDelete(userId);

    if (!user) return res.sendStatus(404);
    user.deleteOne();
    res.sendStatus(204);
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.50 Funkcija brisanja korisnika

Klikom na gumb *Edit* pored pojedinog korisnika, prikazano na slici 3.31, korisnik je odveden na prikaz sa slike 3.32 gdje je funkcija *getUserById*, koja se nalazi na slici 3.51 odgovorna za dohvaćanje pojedinog korisnika pretražujući po id-u te slanja podataka o istom korisniku klijentskoj strani.

```
async function getUserById(req, res) {
  try {
    const token = req.cookies.Authorization;
    const decoded = jwt.verify(token, process.env.SECRET);
    console.log(decoded)
    const admin = await User.findById(decoded.sub)
    console.log(admin)
    if (!admin.superUser){
      res.sendStatus(401)
      return;
    }
    const userId = req.params.id;
    console.log("id "+ req.params.id)
    const user = await User.findById(req.params.id);
    if (!user) return res.sendStatus(404);
    res.json(user);
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.51 Funkcija dohvaćanja pojedinog korisnika

Posljednja opisana funkcija će biti *updateUser* prikazana na slici 3.52 koja je odgovorna za ažuriranje korisnika. Pošto je moguće ažurirati svaki podatak o korisniku posebno, tako je napravljena logika poslužiteljske strane.

```
async function updateUser(req, res) {
  try {
    const token = req.cookies.Authorization;
    const decoded = jwt.verify(token, process.env.SECRET);
    console.log(decoded)
    const admin = await User.findById(decoded.sub)
    console.log(admin)
    if (!admin.superUser){
      res.sendStatus(401)
      return;
    }
    const userId = req.params.id;
    const { name, email, password, superUser } = req.body;
    const user = await User.findById(userId);
    if(name !== null){
      user.name = name;
    }
    if(password !== null){
      user.password = password;
    }

    if(email !== null){
      user.email = email;
    }
    if(superUser !== null){
      user.superUser = superUser;
    }

   user.save()

    if (!user) return res.sendStatus(404);
    res.json(user);
  } catch (err) {
    console.log(err);
    res.sendStatus(400);
  }
}
```

Slika 3.52 Funkcija ažuriranja korisnika

U funkciji *updateUser* se pronalazi korisnik kojeg se želi ažurirati te se zatim provjerava koji podatak je potrebno ažurirati uspoređujući svaki mogući podatak sa vrijednosti *null* i u slučaju da podatak nije jednak istoj, on se ažurira. Naposlijetku se korisnik ažurira u bazi funkcijom *save*.

Poglavlje 4

Zaključak

U ovom radu cilj je bio napraviti aplikaciju za vođenje i praćenje malonogometnih turnira. Aplikacija omogućava jednostavno i intuitivno korištenje svih mogučnosti, od registracije i prijave korisnika, kreiranja, vođenja i praćenja rezultata turnira kao i administracija aplikacije putem administrativnog dijela. Aplikacija ima implementaciju turnira od 8 ekipa podijeljenih u dvije grupe čiji pobjednici igraju finale dok drugoplasirani igraju utakmicu za treće mjesto. Koristeći sve popularniji MERN stack razvojni okvir omogućena je jednostavna izgradnja efikasne i korisniku intuitivne aplikacije. Koristeći popularne React.js i Material-UI biblioteke kreira se estetski ugodno i privlačno korisničko sučelje te njihov bogati ekosustav i zajednica olakšavaju riješavanje svih potencijalnih problema do kojih dođe u razvoju. Node.js i Express.js, popularne tehnologije za razvoj poslužiteljske strane aplikacije omogućavaju laku kontrolu i rukovanje s podatcima te njihovu interakciju sa bazom podataka za potrebe aplikacije. Kao sama baza je korišten MongoDB, baza podataka temeljena na dokumentima i kreirana za efikasno rukovanje sa velikim količinama podataka. U budućem razvoju aplikacije može se dodati implementacija za formate turnira za ostali broj ekipa, primjerice 16 ili 32 te njihova podjela u grupe sa raznim brojem ekipa, a ne samo 4. Zaključno, u budućnosti se može iskoristiti postojanje samih igrača u sustavu kako bi se pratili strijelci zabijenih golova radi praćenja najboljih igrača i strijelaca pojedinih turnira pošto su to informacije za koje često postoji interes.

Bibliografija

[1] Real dom virtual dom. , s Interneta, https://dev.to/swarnaliroy94/introduction-to-react-real-dom-virtual-dom-363 Accessed on August 11, 2023.

[2] About react.js. , s Interneta, https://react.dev/ Accessed on August 11, 2023.

[3] Material-ui. , s Interneta, https://mui.com/ Accessed on August 11, 2023.

[4] About node.js. , s Interneta, https://nodejs.org/en/about Accessed on August 11, 2023.

[5] Mongodb documentation. , s Interneta, https://docs.mongodb.com/ Accessed on August 11, 2023.

[6] Mongodb compass. , s Interneta, https://www.mongodb.com/products/compass Accessed on August 11, 2023.

Sažetak

U ovom radu napravljena je aplikacija za kreiranje i vođenje malonogometnih turnira putem interneta. Glavne funkcionalnosti su kreiranje turnira i vođenje istih i njegovih utakmica. Korisnici se mogu registrirati i prijaviti kako bi kreirali svoje turnire, ali također mogu i kao neprijavljeni korisnici pristupiti informacijama o svim turnirima. Aplikacija također ima i administrativni dio putem kojeg administratori mogu dodavati, brisati i mijenjati igrače i timove kao i korisnike, za slučaj da korisnik želi promijeniti neke od svojih podataka. Korištene tehnologije su React.js i Material-UI za klijentski dio, popularne biblioteke koje osigravaju estetski ugodno, ali i efikasno korisničko sučelje, dok su za poslužiteljski dio korišteni su Node.js i Express.js zbog svoje efikasnosti i lakoće kreiranja poslužiteljske logike. Za bazu podataka korišten je MongoDB.

Ključne riječi — **MongoDB, Express.js, React.js, Node.js, Material-UI**

Abstract

For this thesis, an application has been created for creating and managing mini football tournaments online. The main functionalities include creating tournaments and managing them along with their matches. Users can register and log in to create their own tournaments and unregistered users can also access information about all tournaments without logging in. The application also features an administrative section where administrators can add, delete, and modify players, teams, as well as users in case a user wants to change some of their information. The technologies used are React.js and Material-UI for the client-side, popular libraries that provide aesthetically pleasing and efficient user interfaces. For the server-side, Node.js and Express.js were used due to their efficiency and ease of creating server-side logic. MongoDB was used as the database.

Keywords — **MongoDB, Express.js, React.js, Node.js, Material-UI**