IZRADA WEB APLIKACIJE ZA NOGOMETNE TURNIRE

Peranni, Luciano

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

LUCIANO PERANNI

Z A V R Š N I R A D

IZRADA WEB APLIKACIJE ZA NOGOMETNE TURNIRE

Split, rujan 2021.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Mrežne usluge i programiranje

Z A V R Š N I R A D

Kandidat: Luciano Peranni

Naslov rada: Izrada web aplikacije za nogometne turnire

Mentor: Haidi Božiković, viši predavač

Split, rujan 2021.

Sadržaj

Sažetak

Tema ovog završnog rada je izrada responzivne web aplikacije za organiziranje nogometnih turnira ili utakmica korištenjem modernih programskih alata *Spring boot*, *Vue* i *Postgre*. Izrađena web aplikacija bi pojednostavnila i ubrzala organizaciju turnira i utakmica.

Aplikacija je implementirana i na klijentskoj i na poslužiteljskoj strani te pruža mogućnosti izrade korisničkog računa za pristupanje i sudjelovanje u nogometnim događajima. Nudi se također i administratorsko sučelje za upravljanje bazom korisnika te kreiranje, pretraživanje i uređivanje nogometnih događaja. Gostima aplikacije omogućen je pregled događaja prije odluke o registraciji.

Ključne riječi: *Spring boot, Vue, Postgre*, responzivnost, nogomet

Summary

Development of a Web application for football tournaments

The goal of this undergraduate thesis is the development of a responsive web application for organizing football tournaments or matches using modern programming tools like *Spring boot*, *Vue* and *Postgre*. The created application would make it easier and faster to organize football tournaments and matches.

The application was implemented both on the client side and server side and it offers a possibility of creating a user account for accessing and participating in football events. An admin interface is also included, it enables administrators to manage the users' base as well as create, browse, and edit football events. Application guests are allowed to browse events before committing to register.

Keywords: *Spring boot, Vue, Postgre*, Responsiveness, *football*

1. Uvod

Osnovni motiv stvaranja i oblikovanja web aplikacije je nemogućnost redovitog okupljanja dovoljnog broja suigrača za igranje nogometnih utakmica. Današnji život je ubrzan i odrastanjem nadolazi sve više obaveza što zapravo rastužuje strastvene ljubitelje nogometa jer praktički postaje nemoguće se redovito nalaziti sa kolegama i prijateljima. Aplikacija predstavlja najbolje i najučinkovitije rješenje za redovitu i pravovremenu organizaciju utakmica.

Život u vrijeme stalne povezanosti na internet pruža mogućnost da se putem aplikacije koja bi omogućila spontano organiziranje i sudjelovanje u nogometnim turnirima stvore nove prilike za bavljenje društvenim aktivnostima kao što je amaterski, rekreacijski nogomet. Aplikacije poput ove razvijene za praktični dio ovog završnog rada, čine se kao dobro rješenje. Služeći se *mobile-first* pristupom, za izradu aplikacijskog korisničkog sučelja (eng. *Frontend*) korišten je *Vue,* jedna od najpoznatijih i podržanijih biblioteka temeljenih na *Javascript* programskom jeziku. Za poslužiteljski dio aplikacije (eng. *Backend*) odabran je razvojni okvir (eng. *Framework*) *Spring boot* temeljen na *Java* programskom jeziku. *Spring boot* je dio krovne *Spring* platforme te je napravljena kao odgovor na kompleksnost konfiguracije iste, pa kao takva nudi lakši i brži način rada uz neke unaprijed konfigurirane česte postavke.

2. Korištene tehnologije

2.1. Visual Studio Code

Microsoftov *Visual Studio Code* odabran je za potrebe razvoja korisničkog sučelja aplikacije. *Visual Studio Code* je iznimno popularan i lagan uređivač teksta koji podržava razna proširenja za pomoć pri svakodnevnome radu [1]. Uz namještanje teme samog editora, nude se i sustavi pametnog popunjavanja kôda, alati za pristup bazi podataka, podrške za razne programske jezike i platforme i mnogo više.

2.2. Intellij Idea

Intellij Idea specijalizirano je razvojno okruženje za rad na programima temeljenim na programskom jeziku *Java* [2]. *Intellij Idea* pruža pregršt korisnih funkcionalnosti koje značajno ubrzavaju rad poput istovremenog preimenovanja varijable na svim korištenim mjestima, brzo nalaženje i nadomještanje traženih pojmova u cijelom projektu itd.

Uz funkcionalnosti vezane za samo pisanje kôda, nudi i podršku oko samog pokretanja i testiranja kôda, automatskog pakiranja u *jar* arhive, pristupa bazi podataka te povezivanja na sustav verzioniranja.

2.3. Vue

Vue je moderna biblioteka za razvoj korisničkih sučelja aplikacija tipa SPA (eng. *Single Page Applications*) [3]. Veličina mu je praktički neznatna (otpr. 20 kilobajta) i jednostavno je naučiti njome rukovati. Razvoj *Vuea* započeo je njen izumitelj Evan You 2013. godine, motiviran svojim tadašnjim korištenjem *Angular* biblioteke na svome radnome mjestu u Googleu. Cilj je bio izvući iz *Angulara* sve ono što ga čini jednostavnim za korištenje u jednu kompaktnu i lako proširivu biblioteku [4]. Primjeri stranica razvijenih *Vue*-om su web stranice proizvođača video igara Nintendo te kompanije za verzioniranje programskog kôda Gitlab.

Temelj funkcionalnosti *Vuea* je sustav koji omogućava deklarativno prikazivanje korištenjem kratke i jasno razumljive sintakse. Primjer je dan u ispisu (Ispis 1**)**.

```
<div id="app">
  {{ message }}
</div>
     var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

Ispis 1: Izgled osnovne sintakse *Vue* aplikacije

Korištenjem ovakvog načina upravljanja sadržajem na stranici nema više potrebe za izravnim pristupom HTML (eng. *HyperText Markup Language*) dokumentu, već se promjene na istom vrše direktno *Javascript* jezikom. Deklaracijom varijable unutar *data* objekta ostvaruje se pristup istoj u vizualnom dijelu aplikacije (eng. *Template*). Komponente jedne *Vue* aplikacije se sastoje od tri dijela. To su, već spomenuti, vizualni dio, programski kôd u *Javascriptu* te stilski dio, koji upravlja samim stilom *Template*-a i koji najčešće bude pisan u CSS-u (eng. *Cascading Style Sheets*), ali podržava i druge jezike poput LESS-a (eng. *Leaner Style Sheets*) i SASS-a (eng. *Syntactically Awesome Style Sheets*).

Vue također nudi i razne atribute koji služe deklarativnom prikazivanju sadržaja. U primjeru (Ispis 2) je prikazana funkcija atributa *v-if* koji elementu kojem pripada omogućuje da bude prisutan samo ako varijabla na koju je postavljen ima vrijednost *true*.

```
<div id="app-3">
  <span v-if="seen">Now you see me</span>
</div>
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

Ispis 2: Korištenje *Vue* atributa

Postoje mnogi različiti atributi koji imaju funkcionalnosti poput selektivnog prikazivanja, reagiranja na korisničke postupke i akcije, jednostavno prikazivanje listi itd.

Jedna bitna stavka po kojoj se *Vue* razlikuje od svojih konkurenata poput *Reacta* i *Angulara* je dvosmjerno povezivanje (eng. *two-way binding*). Varijable u programskom kôdu te njihovo korištenje u predlošku reagiraju na promjene te ih propagiraju povezanoj inačici u drugom dijelu komponente. To se postiže *v-model* atributom (Slika 1).

HTML

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

JS

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

Primjer za završni

Primjer za završni

Slika 1: Dvosmjerna povezanost

Razvidno je u primjeru da upisivanje teksta u polje sa *v-model* atributom ujedno i mijenja vrijednost varijable u programskom dijelu te se onda ista ta vrijednost i u stvarnom vremenu prikaže paragrafu koji koristi istu tu vrijednost.

Uz same varijable koje predstavljaju stanje aplikacije ili komponente, *Vue* nudi i pisanje metoda i izračunatih vrijednosti (eng. *computed variables*).

U primjeru (Slika 2) prikazana je metoda koja usmjerava korisnika na stranicu s adresom danom kao parametar te izračunata vrijednost koja daje informaciju o tome koja je

tema boja trenutno odabrana u aplikaciji. Osnovna razlika između metoda i izračunatih varijabli je to što izračunate varijable moraju vraćati podatak dok su metode malo slobodnije po pitanju što mogu raditi. Razvojni programer je onaj koji će odlučiti što će koristiti od ovih opcija ovisno o potrebama koje program mora zadovoljiti.

Slika 2: Metode i izračunate varijable

Moguće je i kreiranje *Vue* komponenti u datotekama sa ekstenzijom *.vue* (Ispis 3). Takve datoteke značajno olakšavaju razvoj jer u sebi sadrže sva tri dijela komponente (predložak, kôd i stil).

```
<template>
  <h1 class="date-header">{{ date }}</h1>
</template>
<script>
export default {
  data() { return { date: "" }}};
</script>
<style lang="css" scoped>
  .date-header {
    font-size: 20px;
  }
</style>
```

Ispis 3: Prikaz *.vue* datoteke

2.3.1. Vue CLI

Vue CLI najkorišteniji je način rada na *Vue* aplikacijama. Radi se o konzolnome programu koji omogućava brzo i uniformno kreiranje novih projekata. Program nudi različite opcije prilikom pokretanja poput odabira programskog jezika (*Javascript* ili *Typescript*), instalacije često korištenih paketa, postavljanje sustava testiranja te spremanja odabranih opcija u predložak za ponovno korištenje.

2.3.2. Vuex

Vuex je biblioteka koja dolazi skupa sa instalacijom *Vuea*. Slično *Reactovoj Redux* biblioteci ona služi za pohranu stanja na razini cijele aplikacije. Podatci koji su često potrebni poput informacije o tome je li korisnik trenutno prijavljen te njegov token za autentikaciju upita na poslužiteljski dio programa, najbolje je čuvati u *Vuex* objektu (Ispis 4). Temelji se na *Flux* biblioteci za *Javascript* te radi na principu statičkih varijabli koje predstavljaju danu informaciju, na principu mutacija koje mijenjaju vrijednost tih varijabli te na principu akcija koje izvode mutaciju.

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  },
  actions: {
    increment (context) {
      context.commit('increment')
    }
  }
})
```

Ispis 4: Primjer *Vuex* objekta

2.3.3. Vuetify

Vuetify je zbirka gotovih *Vue* komponenti temeljenih na *Googleovom Material Designu*. Osim što je ova biblioteka vrlo podržana, ona je i jako popularna jer uz već spomenute komponente poput polja za unos, tablica i kartica, nudi također i vlastiti sustav za

implementaciju responzivnih stranica. Taj je sustav izveden kroz specijalne atribute koji mijenjaju i prilagođavaju veličine i udaljenosti između elemenata ovisno o veličini zaslona.

2.4. Spring boot

Spring boot programska je platforma temeljena na *Java* programskom jeziku. Inačica je krovne *Spring* platforme i namijenjena je bržem kreiranju, konfiguriranju i pokretanju *Spring* aplikacija [5]. Umjesto standardne XML konfiguracije omogućava konfiguriranje aplikacije kroz tzv. *Bean* klase. *Spring boot* pruža mnoge module često korištenih funkcionalnosti, među njima primjerice module za autentikaciju, testiranje, povezivanje s bazom podataka kroz *Spring Data JPA* biblioteku, kreiranje servisa na *REST* (eng. *Representational State Transfer*) principu, spremanje aplikacije u *Docker* spremnike itd.

U odnosu na konvencionalni *Spring*, značajna je prednost činjenica što *Spring boot* aplikacije imaju uključeni poslužitelj, najčešće *Tomcat* ili *Jetty*, što znači da je pokretanje same aplikacije moguće jednostavnim stiskom na gumb *pokreni* u razvojnome okruženju.

2.5. Baza podataka

Za bazu podataka aplikacije odabran je *Postgre*. *Postgre* je moćan sustav relacijskih baza podataka sa preko 30 godina aktivnog razvoja [6]. Temeljen na podršci za *SQL* (eng. *Structured Query Language*) jezik, drugi je najkorišteniji sustav za relacijske baze podataka iza *MySQLa*. Jako je fleksibilan te podržava mnoge bitne funkcionalnosti u radu s bazama podataka. Uz osnovne upite i kreaciju tablica omogućava spremanje čestih upita u *viewove* i procedure za ponovno korištenje. Bitna značajka *Postgrea* je njegova mogućnost proširivanja dodacima poput spremanja podataka iz tablica u međuspremnike, odnosno, *cachiranje* i replikacije cijelih baza podataka zbog potencijalne potrebe za *backup*-ovima u svrhu prevencije gubitka podataka.

Za integriranje s bazom podataka poslužiteljski dio aplikacije koristi *Hibernate* biblioteku. *Hibernate* spada u skupinu *ORM*-ova (eng. *Object Relational Mapper*). *ORM* biblioteke rade na principu mapiranja tablica iz baze podataka u objekte klasa programskog jezika i mapiranja upita na te tablice u programske metode. *Spring boot* u svojoj glavnoj

biblioteci nudi zgodno sučelje za korištenje *Hibernatea,* radi se o sučelju imena *Spring Data JPA. Spring Data JPA* nije ništa drugo nego kolekcija anotacija i metoda koje omogućavaju automatsko generiranje klasa za pristup baza (eng. *Repository classes*) [7]. U primjeru (Ispis 5) prikazan je izgled modela koji odgovara stvarnoj tablici u bazi podataka te pristupne klase.

```
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "comment")
@Data
public class Comment {

    @Id
    @GeneratedValue
    private long id;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Lob
    @Type(type = "text")
    private String text;

    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name = "event_id", referencedColumnName = "id")
    @JsonIgnore
    private Event event;
}
```

Ispis 5: Model *Comment* tablice

Može se primijetiti da uz korištenje anotacija poput *@Entity* i *@Table* se definira klasa koja predstavlja tablicu *Comment* koja služi za pohranu komentara korisnika na događaje. Anotacijama se također definiraju i relacije među objektima. U ovom konkretnom primjeru postavljene su relacije tipa *više na jedan* između komentara, te korisnika i događaja, korištenjem anotacije *@ManyToOne*. Jedan komentar pripada jednome događaju i korisniku ali događaji i korisnici mogu imati više komentara. *Spring Data JPA* automatski mapira tipove polja klase u prikladne *SQL* tipove ali po potrebi moguće je ručno definirati tip anotacijama. Za tekst komentara je odabran tip *text* koji odgovara *SQL* tipu *TEXT*. S obzirom da se radi o tipu koji može poprimiti dosta veliku veličinu, polje je dodatno označeno anotacijom *@Lob* koja optimizira dohvaćanje većih podataka.

U narednom ispisu (Ispis 6) prikazan je izgled *interface* klase koja generira pristupnu klasu. One se kreiraju nasljeđivanjem *Spring*-ovog JpaRepository *interfacea* te anotacijom

@Repository. Prilikom pokretanja aplikacije, *Spring* na temelju definicije *interfacea* generira stvarnu implementaciju s metodama kreiranim na temelju naziva metoda u *interfaceu* koje moraju biti precizno napisane. Tako će se za *interface CommentRepository* generirati klasa s metodom za dohvaćanje svih komentara po identifikacijskom ključu događaja (eng. *eventId*). Uz metode definirane od strane programera se također generiraju česte metode poput metoda za spremanje, brisanje po ključu te nalaženja po ključu samog komentara.

```
@Repository
public interface CommentRepository extends JpaRepository<Comment, Long>
{
    List<Comment> findAllByEventId(long eventId);
}
```

Ispis 6: *Repository* sučelje za pristup bazi

U slučaju potrebe za kompliciranijim upitima *Spring Data JPA* nudi i vlastoručno pisanje upita korištenjem *JPQL* jezika (eng. *Jakarta Persistence Query Language*) [8]. *JPQL* omogućava korištenje *Java* klasa pri pisanju upita te se koristi pomoću anotacije *@Query* (Ispis 7). Upit iz ispisa generira *SQL* upit koji dohvaća mail adresu, korisničko ime, lozinku i *rolu* iz tablice korisnika te ih premapira u model potreban za Autentikaciju korisnika.

```
@Query("Select new
me.lperanni.exam.security.model.AuthUserModel(u.email,
u.userName, u.password, u.role) " +
        "from User u where u.userName = :username")
Optional<AuthUserModel> findAuthByUserName(String
username);
```

Ispis 7: *JPQL* upit

3. Opis praktičnog rada

Praktični dio ovoga rada sastoji se od klijentske i poslužiteljske strane web aplikacije za organizaciju i pronalaženje nogometnih utakmica/turnira. U narednim će cjelinama svaka funkcionalnost biti opisana s obje strane.

3.1. Navigacijska traka

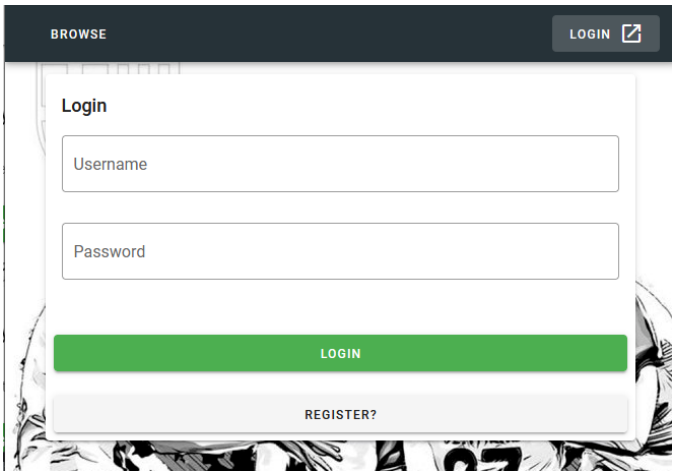
Navigacijska traka (Slika 3) predstavlja primarni način pristupa raznim izbornicima aplikacije. S obzirom na to je li korisnik ulogiran ili ne, te je li ulogirani korisnik ujedno i administrator aplikacije, navigacijska traka može imati tri različita oblika.

Slika 3: Različite izvedbe navigacijske trake

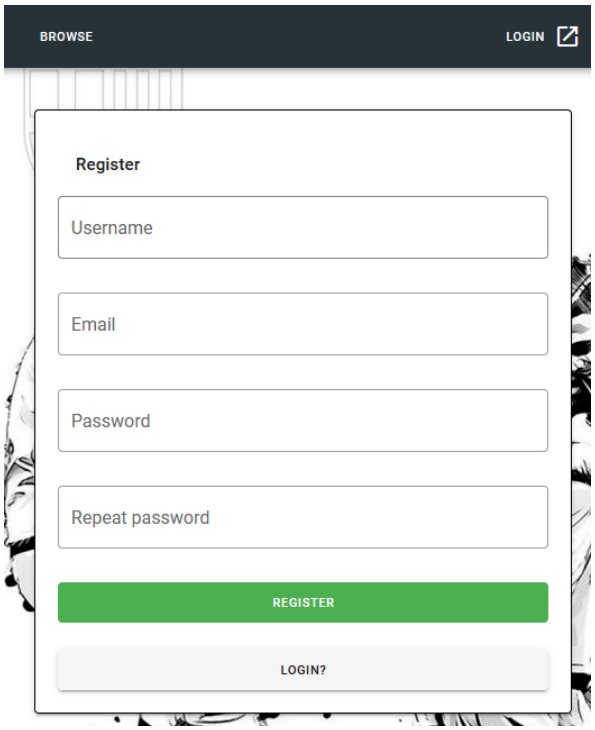
Prva inačica predstavlja neprijavljenog korisnika kojemu je omogućeno da kroz gumb *Browse* pregleda trenutno aktivne turnire. Druga inačica predstavlja standardnog prijavljenog korisnika kojemu je omogućeno da kroz gumb *Home* pristupa glavnom izborniku aplikacije te da se koristeći opciju *Logout* odjavi iz iste. Treća verzija namijenjena je administratorskim korisnicima koji uz sve opcije koje ima standardni korisnik imaju i opciju ulaska u administratorski izbornik.

3.2. Autentikacija i autorizacija

Prijava i registracija u aplikaciju omogućena je kroz *Login* i *Register* forme (Slika 4, Slika 5). Za registriranje u sustav potrebno je upisati korisničko ime, mail adresu i željenu lozinku

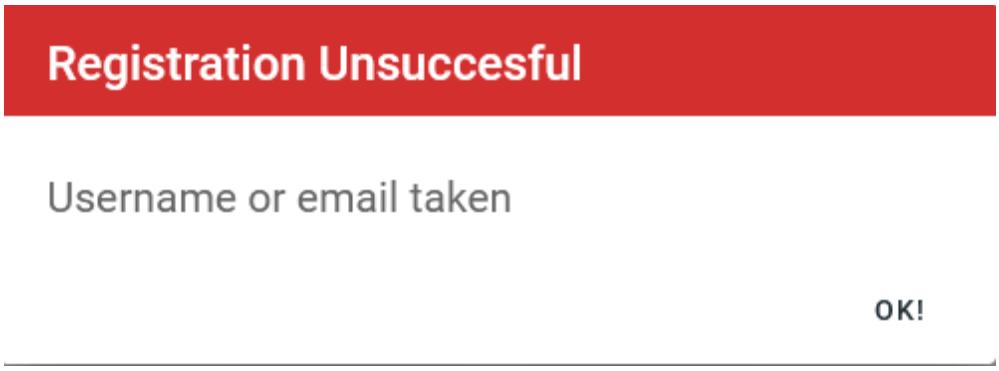


Slika 4: Forma za prijavu



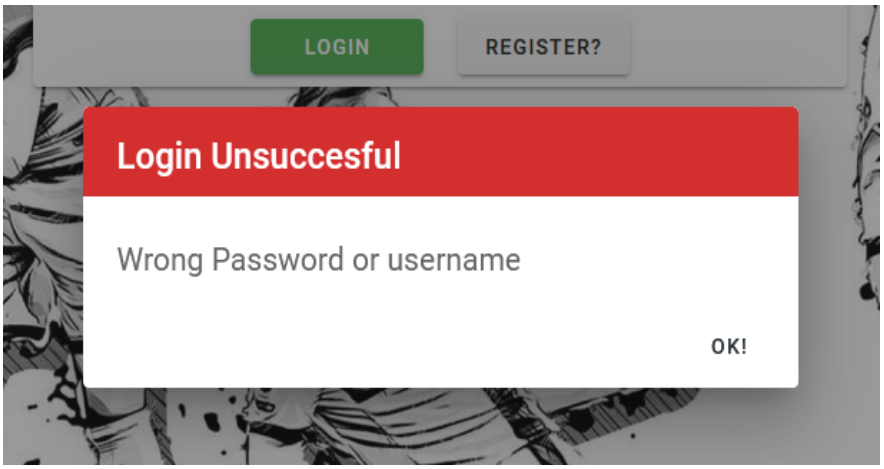
Slika 5: Forma za registraciju

Prilikom registracije kroz navedene forme se pošalje *POST* upit na poslužiteljski dio aplikacije. Ako je upit uspješno proveden, korisnika će aplikacija preusmjeriti na zaslon za prijavu, a u slučaju neuspjeha (primjerice korisnik sa istim korisničkim imenom već postoji), korisniku će se prikazati dijalog s porukom o neuspjeloj registraciji (Slika 6).



Slika 6: Dijalog pri neuspjeloj registraciji

Isto vrijedi i za prijavu u sustav. Ako korisnik unese krivu lozinku pojavit će mu se poruka o neuspjeloj prijavi u obliku dijaloga (Slika 7).



Slika 7: Dijalog pri neuspjeloj prijavi

HTTP upiti (eng. *Hypertext Transfer Protocol*) se na poslužiteljskoj strani obrađuju pomoću kontroler klasa. Registracija je implementirana uz pomoć metode u takvoj kontroler

klasi kao i *Spring boot* modula za sigurnost (eng. *spring-boot-security-starter*). Kontroler metoda koja je odgovorna za primanje zahtjeva za registraciju prikazana je na ispisu.

Uočljivo je da je anotacijom *@PostMapping* označena metoda (Ispis 8) koja prima objekt klase *AuthUserModel* koja u kontekstu aplikacije predstavlja model s podacima za autentikaciju. Sama metoda poziva servisnu klasu koja sadrži daljnji slijed logike registracije unutar aplikacije.

```
@PostMapping("/register")
public ResponseEntity<User> signUp(@RequestBody AuthUserModel user) {
    try {
        User newUser = service.register(user);
        return new ResponseEntity<>(newUser, HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
}
```

Ispis 8: Primjer metode kontroler klase

Servisna klasa (Ispis 9) potom kreira novi objekt klase *User* koja predstavlja osnovnu klasu korisnika aplikacije. Potom se novom korisniku pridijeli rolu *USER*, što je oznaka za standardnog korisnika aplikacije, te enkôdira lozinku prije samog spremanja u bazu.

```
public me.lperanni.exam.persistence.model.User
register(AuthUserModel authUser) throws Exception {

    var user = new me.lperanni.exam.persistence.model.User();
    user.setUserName(authUser.getUserName());
    user.setEmail(authUser.getEmail());
    user.setRole(UserRole.USER);
    user.setPassword(encoder.encode(authUser.getPassword()));
    try {
        return repository.save(user);
    } catch (Exception e) {
        System.out.println(e.getMessage());
        throw new Exception("User could not be registered");
    }
```

Ispis 9: Servisna klasa registracije

Prijava u aplikaciju se također obavlja HTTP upitom. Ako je uspješna prijava u aplikaciju, ona će klijentu vratiti *JWT* token. Token predstavlja jedinstvenu *hashiranu* vrijednost koja u sebi sadrži dovoljnu količinu informacija za identifikaciju korisnika. Taj proces identifikacije prilikom slanja upita se naziva autorizacijom.

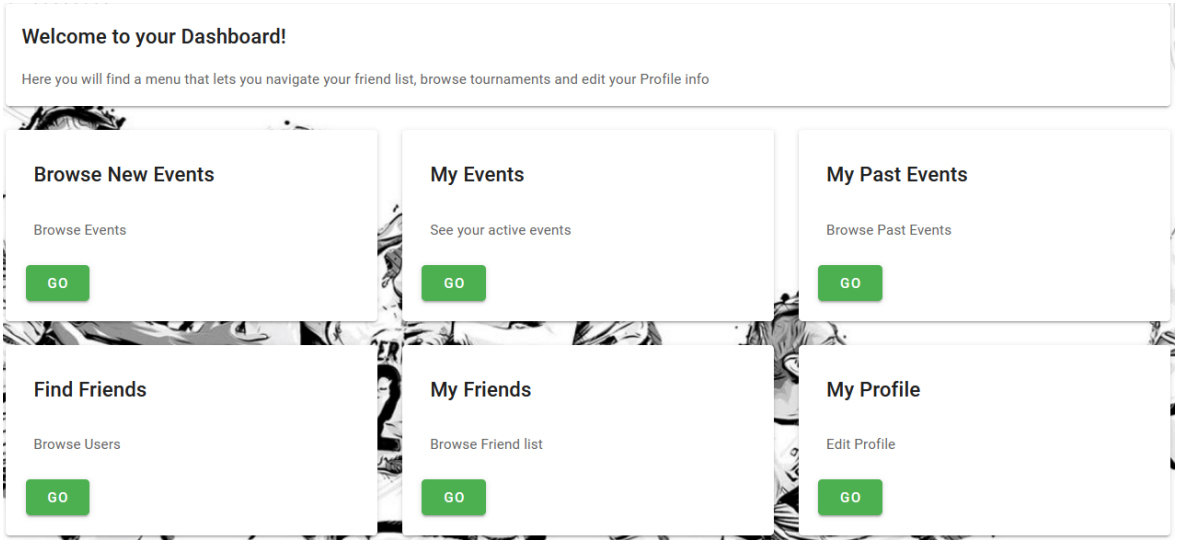
U konkretnom slučaju što se tiče ove aplikacije (Ispis 10), u token se spremaju korisničko ime, njegova *rola* te vrijeme isteka i tajna kao dodatna sigurnosna mjera. Isti taj token će biti potreban za upite na zaštićene krajnje točke (eng. *endpoints*) programa, primjerice krajnja točka za dohvaćanje podataka o korisničkom profilu.

```
var claimsMap = new HashMap<String, Object>();
claimsMap.put("role", ((User)
auth.getPrincipal()).getAuthorities().toArray()[0]);

String token = Jwts.builder()
        .setSubject(((User) auth.getPrincipal()).getUsername())
        .setExpiration(new Date(System.currentTimeMillis() +
EXPIRATION_TIME))
        .addClaims(claimsMap)
        .signWith(SignatureAlgorithm.HS256,
SECRET.getBytes(StandardCharsets.UTF_8))
        .compact();
```

Ispis 10: Sastavljanje *JWT* tokena

3.3. Glavni izbornik

Glavni izbornik aplikacije izveden je u obliku kartica koje predstavljaju sve funkcionalnosti aplikacije koje korisniku stoje na raspolaganju (Slika 8).



Slika 8: Prikaz glavnog izbornika

Kartice se sastoje od naslova, kratkog opisa stranice te gumba za odlazak na odgovarajuću stranicu. Implementirane su koristeći *v-card* komponentu iz *Vuetify* biblioteke. Na ispisu (Ispis 11) prikazan je kôd za prikaz jedne kartice unutar *Dashboard* komponente, koja predstavlja cijeli izbornik. Može se uočiti korištenje atributa *v-for* koji dozvoljava iteriranje po listi karata umjesto ponavljanog tipkanja istog kôda.

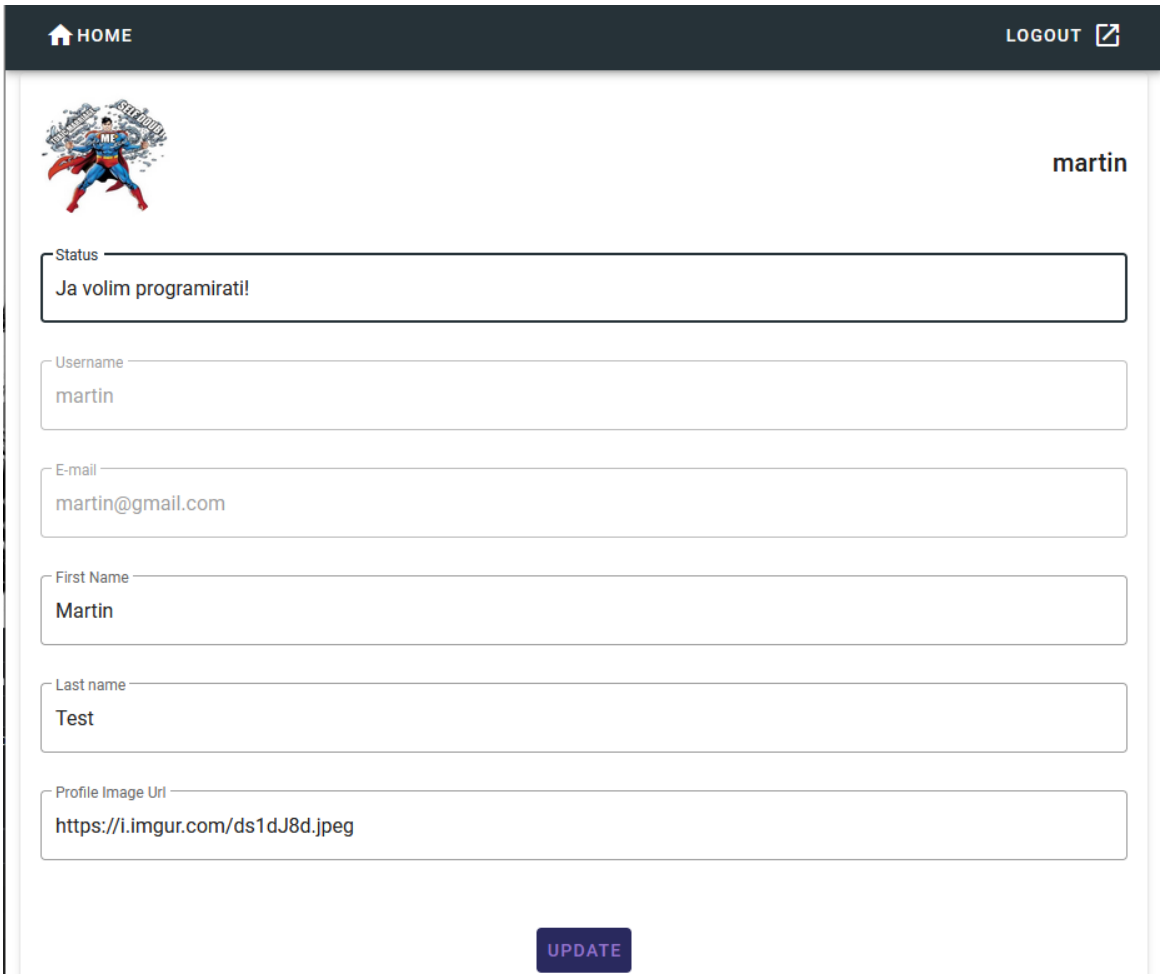
```
<v-card :card="card" class="pa-3" hover>

   <v-card-title class="mb-3">{{ card.title }}</v-card-title>
       <v-card-text>{{ card.text }}</v-card-text>
       <v-card-actions>
          <v-btn left class="success" router :to="card.link">Go</v-btn>
       </v-card-actions>
</v-card>
```

Ispis 11: Primjer *v-card* komponente

3.4. Uređenje profila

Stranica za uređenje korisničkog profila (Slika 9) sadrži osnovne informacije o korisniku. Omogućeno je dodavanje informacija o punom imenu, postavljanje profilne slike

pomoću poveznice te postavljanje korisničkog statusa. Korisniku nije omogućeno mijenjanje korisničkog imena ni mail adrese, već to obavlja administrator. Klikom na gumb *Update*, spremaju se trenutni podatci u odgovarajućim poljima u bazu podataka preko HTTP upita na poslužitelju. U slučaju uspjeha, prikaže se dijalog potvrde.

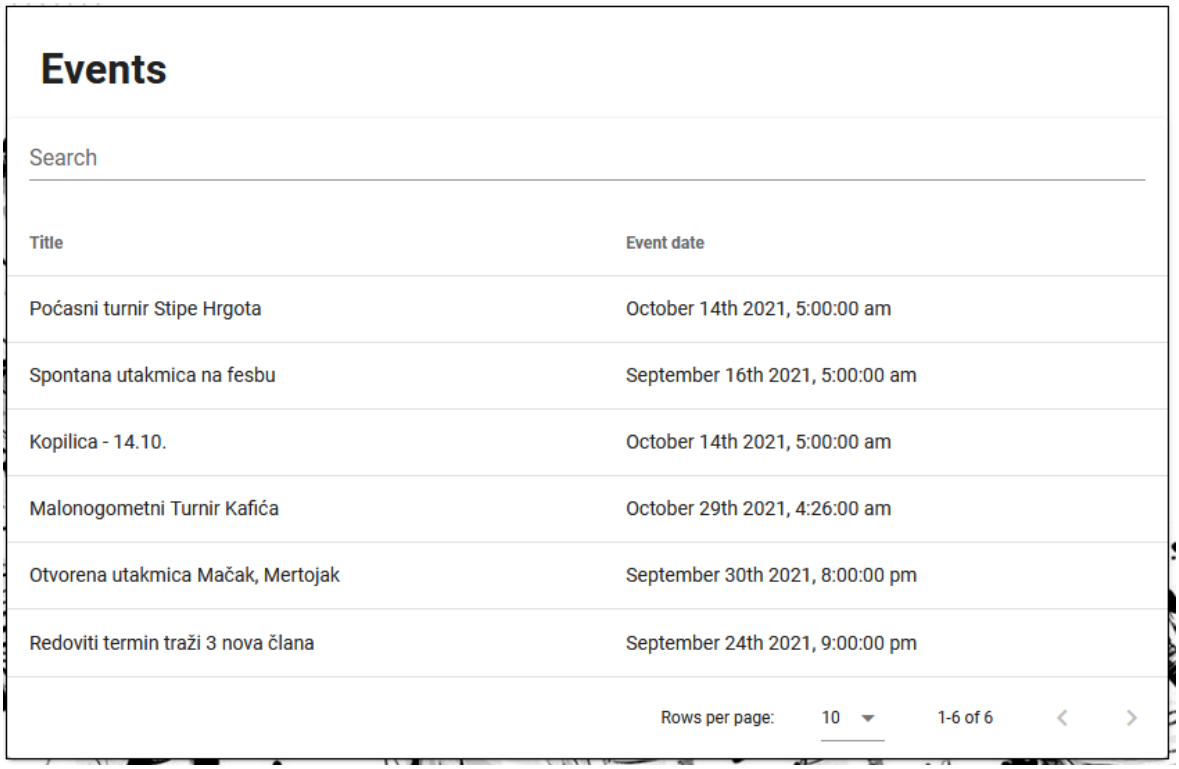


Slika 9: Prikaz stranice za uređenje profila

3.5. Popis događaja

Pretraživanje događaja u glavnome izborniku ima više opcija. Moguće je pretražiti nadolazeće događaje, događaje kojima je već ostvaren pristup te prethodno posjećene događaje. Ovisno o odabiru, učitava se stranica sa željenim setom (Slika 10). Upite prema poslužitelju sastavlja se pomoću *Javascript* biblioteke *thwack*. *Thwack* je biblioteka koja nudi jednostavan način slanja zahtjeva na poslužitelje preko *Javascript fetch* API-a (eng.

Application Programming Interface). Na primjeru (Ispis 12) su prikazani konfiguracija i funkcije za slanje zahtjeva preko *thwacka*. *Thwack* biblioteka nudi mogućnost automatskog dodavanja dobivenog *JWT* tokena na svaki upit koji se šalje put poslužitelja.



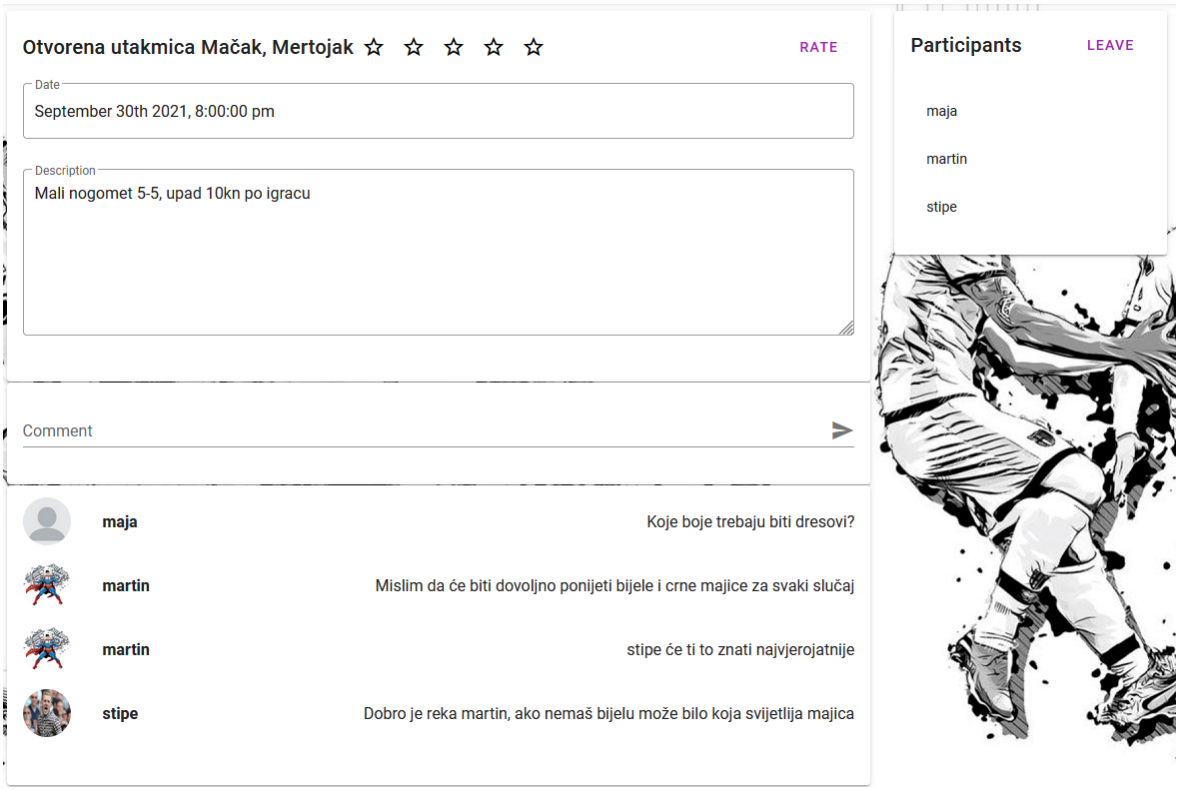
Slika 10: Pregled događaja

```
const configureThwack = () => {
  thwack.defaults.baseURL = process.env.VUE_APP_BASE_URL
  thwack.addEventListener('request', event => {
    const token = store.state.auth.token
    if (token) {
      event.options.headers = {
        ...event.options.headers,
        Authorization: 'Bearer ' + token
      }
    }
  })
configureThwack()
export function all() {
  return thwack.get(endpoints.event.all+"?active=true").then(({ data })
=> data);
}
export function me() {
  return thwack.get(endpoints.user.me).then(({ data }) => data);
}
export function get(username) {
  return thwack.get(endpoints.user.get(username))
       .then(({ data }) => data);
}
```

Ispis 12: Korištenje Thwack biblioteke

3.6. Opisna stranica događaja

Klikom na redak željenog događaja navigira se do stranice koja predstavlja glavni pregled istog (Slika 11). Omogućava se pristupanje klikom na gumb *join* u gornjoj desnoj kartici koja sadrži i trenutni popis sudionika. Uz same informacije o turniru moguće je i ostaviti komentar u za to predviđenoj sekciji ispod opisne. Ova stranica također nudi i mogućnost ocjenjivanja prošlih turnira.

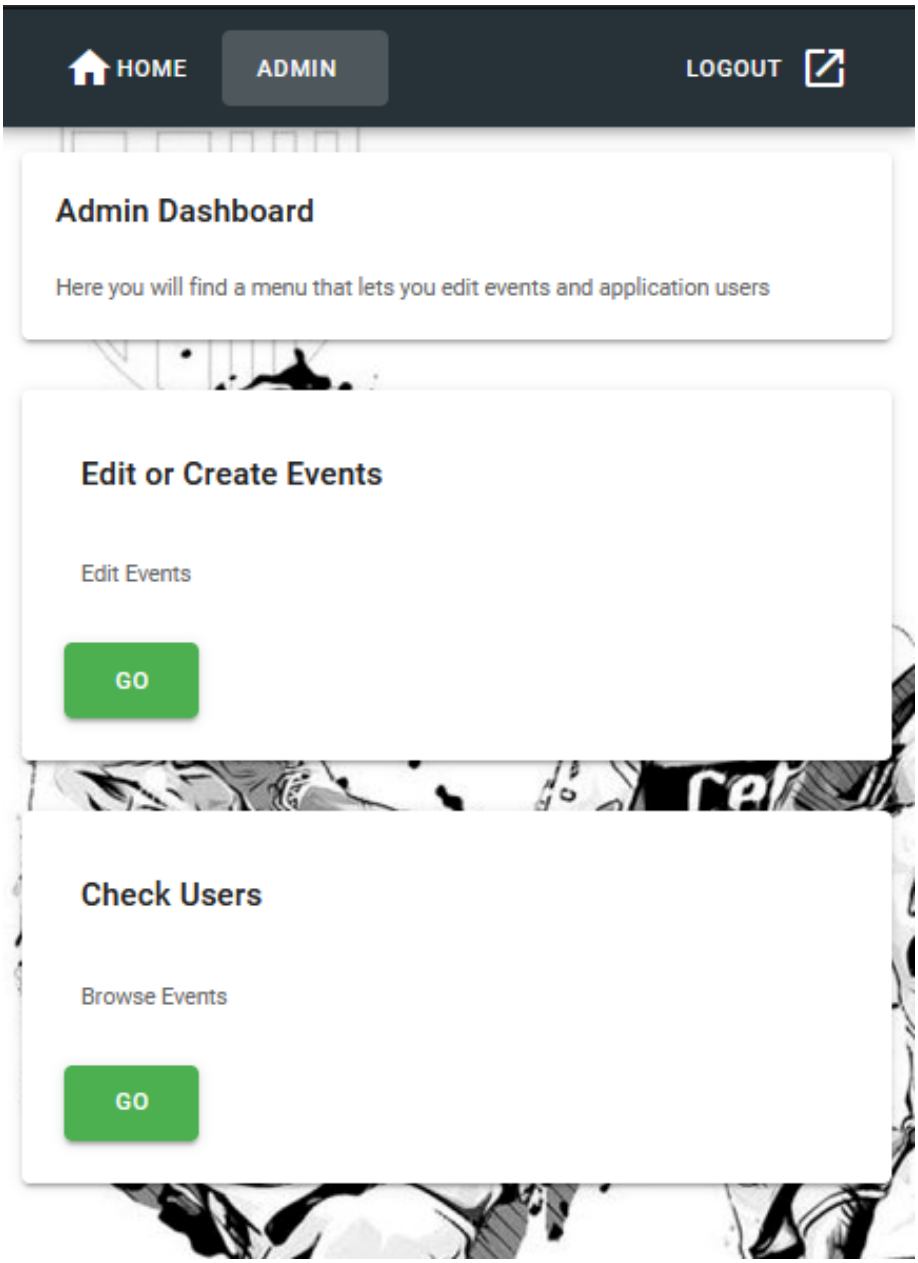


Slika 11: Stranica pojedinog događaja

Klikom na gumb *rate* otvara se dijalog za ocjenjivanje. Ocjene svih korisnika za pojedini događaj se onda kroz upit na bazu agregiraju te se njihova prosječna vrijednost prikazuje pokraj naziva samog događaja.

3.7. Administratorski izbornik

Administratorski korisnici aplikacije imaju pristup izborniku za uređivanje korisnika i turnira. Na slici (Slika 12) je prikazana mobilna verzija tog izbornika. Uočljivo je da se veličina elemenata koje sačinjavaju stranicu prilagođava veličini samog zaslona. Ponuđene su dvije opcije: Kreiranje ili uređivanje događaja/turnira te upravljanje korisnicima aplikacije.

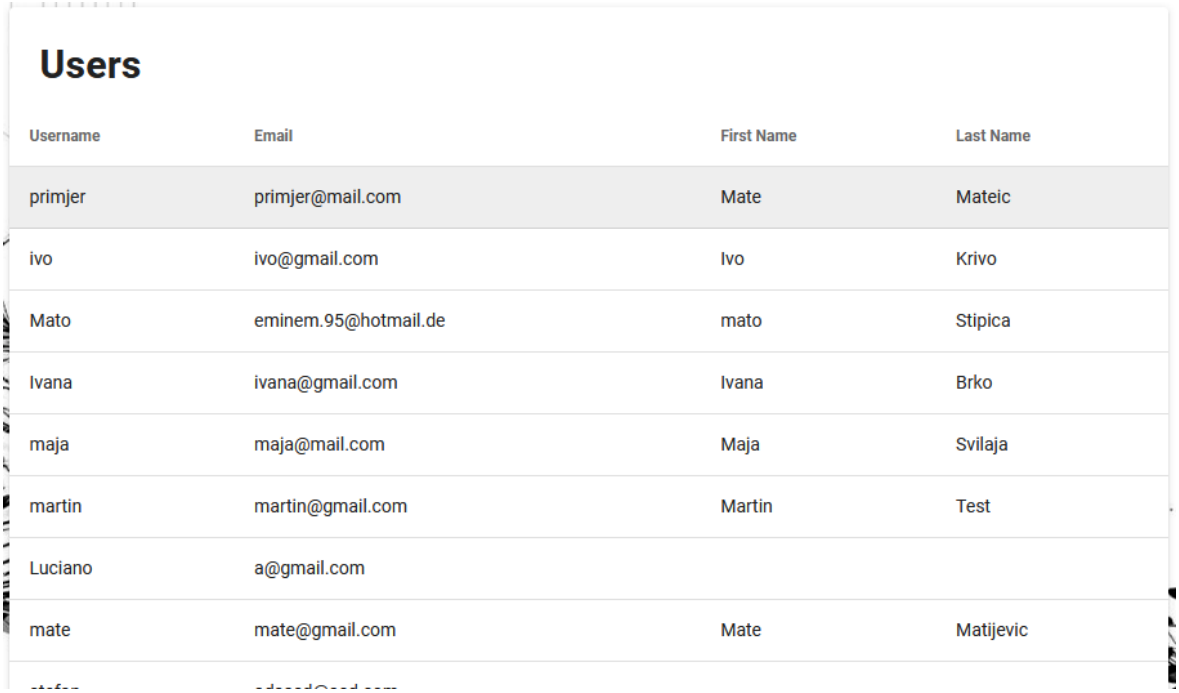


Slika 12: Izbornik s opcijama za administratore

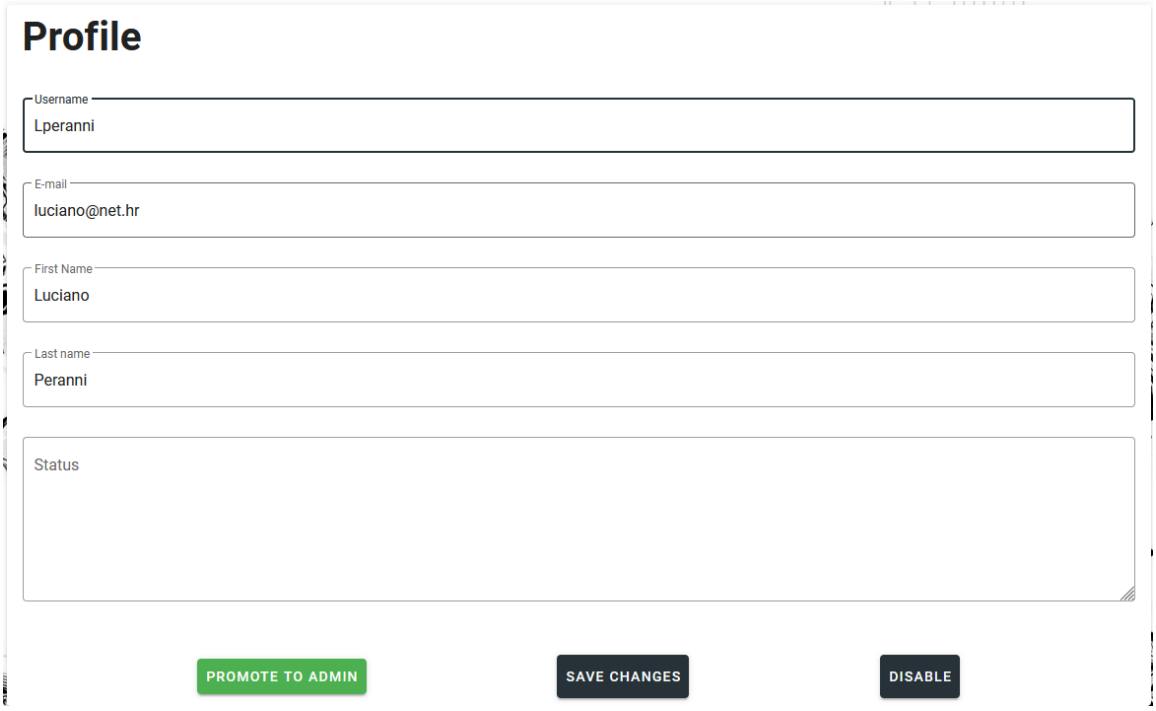
3.7.1. Upravljanje korisnicima

Stranica za upravljanje korisnicima nudi tablicu svih korisnika s opcijom pretrage po korisničkom imenu, te stvarnome imenu ako ga je korisnik odlučio unijeti (Slika 13). Klikom na redak u kojem se nalazi traženi korisnik pristupa se stranici njegovog profila s posebnim ovlastima uređivanja (Slika 14). Tu se nude opcije mijenjanja pojedinih stavki računa poput korisničkog imena i e-mail adrese. Moguće je imenovati druge korisnike administratorima

te isto tako im oduzeti ta prava. U slučaju potrebe za brisanjem, odnosno deaktiviranjem korisnika klikom na gumb *disable* onemogućava se korisniku prijava u aplikaciju.



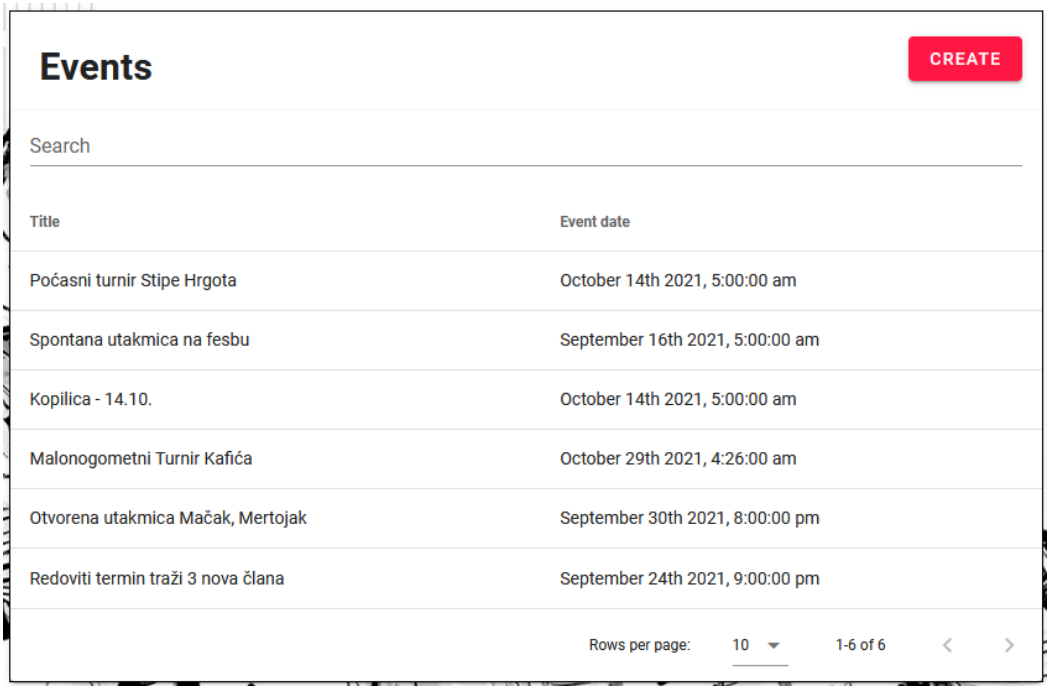
Slika 13: Administratorski pregled korisnika aplikacije



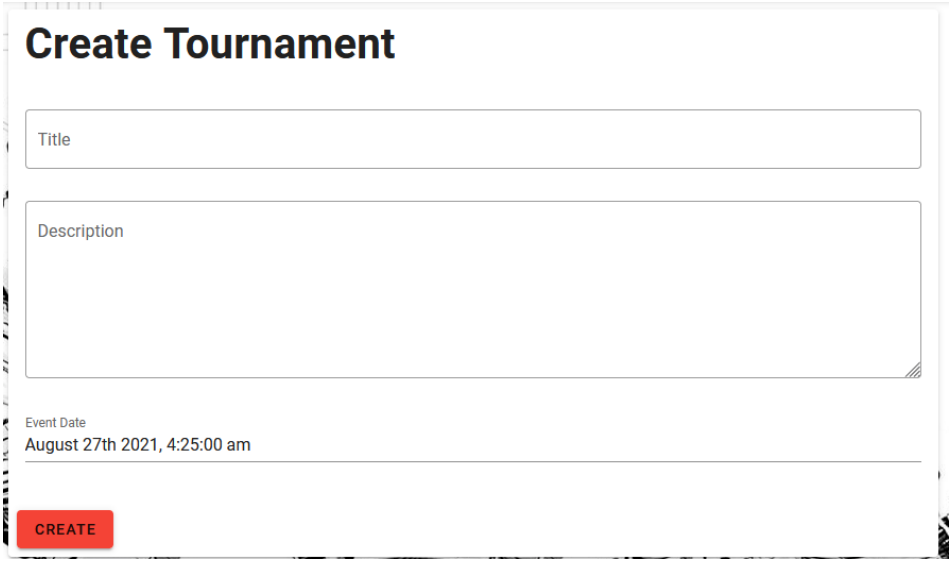
Slika 14: Administratorska stranica za uređivanje korisničkih profila

3.7.2. Upravljanje događajima

Navigacijom na izbornik (Slika 15) i odabirom događaja također se nude mogućnosti upravljanja osnovnim informacijama o postojećim događajima te kreiranje novih (Slika 16) klikom na gumb *Add Event* u gornjem desnom dijelu aplikacije.



Slika 15: Administratorski pregled za uređivanje i kreiranje novih događaja



Slika 16: Stranica za kreiranje novih događaja

Sa poslužiteljske strane svi upiti za kreiranje novih događaja se obrađuju u odgovarajućoj kontroler klasi naziva *EventController*. U sljedećem ispisu (Ispis 13) prikazuje se jedna od metoda kontroler klase. Metoda u primjeru odgovorna je za obradu upita za promjenu podataka o pojedinom događaju.

```
@PutMapping("/{eventId}")
public ResponseEntity<Event> updateEvent(@PathVariable long
eventId, @RequestBody Event eventDetails,

@RequestHeader("Authorization") String token) {

    var role = RequestUtils.extractRoleFromToken(token);
    if (!ADMIN.equals(UserRole.valueOf(role))) {
        return ResponseEntity.badRequest().build();
    }
    var eventOptional = repository.findById(eventId);
    if (eventOptional.isEmpty()) {
        return ResponseEntity.notFound().build();
    }
    var event = eventOptional.get();
    event.setEventDate(eventDetails.getEventDate());
    event.setTitle(eventDetails.getTitle());
    event.setDescription(eventDetails.getDescription());

    var response = repository.save(event);
    return ResponseEntity.of(Optional.of(response));
}
```

Ispis 13: Kontroler metoda za upit promjene vrijednosti događaja

S obzirom da se radi o metodi rezerviranoj za administratore aplikacije, prije same obrade zahtjeva provjerava se *rola* korisnika koristeći identifikacijski token (eng. *JWT*) proslijeđen upitom poslužitelju. Ako se ustanovi da korisnik ima pravo korištenja metode, onda se, po parametru *eventId* samog upita, izvlači događaj povezan s tim ključem te se nad njime obavljaju tražene promjene proslijeđene kroz tijelo (eng. *body*) upita. Ukoliko nema događaja s tim ključem u bazi podataka, metoda će klijentu poslati odgovor sa statusnim kôdom 404 *Not Found*.

3.8. Automatski mailovi

Aplikacija ima mogućnost periodičkog slanja e-mail poruka na mail adrese korisnika korištenjem *JavaMailer* integracije za *Spring boot*. Za pokretanje periodičkih zadataka korištena je *Spring bootova Scheduler* funkcionalnost (Ispis 14).

```
@Scheduled(fixedRate = 60_000 * 60 * 24) // Svaka 24 sata
void sendMail() {
    log.info("Sending mail");
    eventRepository.findAllActiveEventsWithParticipants()
            .stream()
            .filter(e ->
e.getEventDate().isBefore(Instant.now().plus(1, ChronoUnit.DAYS)))
            .forEach(event -> event.getParticipants()
                    .forEach(p ->
emailService.sendEventReminderMail(p.getEmail(), p.getUserName(),
event.getTitle(), event.getEventDate())));
}
```

Ispis 14: Periodički zadatak slanja e-mail poruka

Periodički zadatci se definiraju korištenjem *@Scheduled* anotacije. Anotacija kao parametar prima razne načine definiranja perioda, primjerice *UNIX cron* izraz ili broj koji predstavlja milisekunde između svakog izvršavanja zadatka. U ovom konkretnom slučaju korišten je period od 24 sata. Sama metoda u svome toku izvlači sve nadolazeće događaje skupa sa prijavljenim korisnicima te iterirajući kroz njih pošalje e-mail poruku svim korisnicima koji imaju događaj unutar sljedećih 24 sata.

Konfiguracija slanja e-mail poruka nalazi se u *application.yml* datoteci u poslužiteljskom dijelu aplikacije (Ispis 15).

```
clientAddress: http://localhost:3000
  mail:
    sender: e62b419679-fe07ab@inbox.mailtrap.io
    password: f1f5bddb8b65a7
    host: smtp.mailtrap.io
    port: 587
    username: ea3f0e2256bb35
    auth: PLAIN
    tls: Optional (STARTTLS on all ports)
```

Ispis 15: Konfiguracija servisa za slanje e-mail poruka

Potrebni su podaci o e-mail poslužitelju putem kojih će se slati poruke. *Host* označava adresu poslužitelja te protokol koji se koristi, u ovome slučaju *SMTP* (eng. *Simple Mail Transfer Protocol*). Također je potrebno unijeti podatke za autentikaciju na servis, to su korisničko ime, lozinka, dodjeljeni račun za slanje te sami način autentikacije, konkretno u ovome slučaju *PLAIN* za jednostavnu autentikaciju korisničkim imenom i lozinkom.

Za potrebe ovog završnog rada odabran je besplatan servis simulacije slanja e-mail poruka *Mailtrap* koji se pokazao jako korisnim alatom prilikom razvoja aplikacije. *Mailtrap*

presretne poslane poruke te ih prikazuje u preglednom korisničkom sučelju što omogućava testiranje slanja poruka bez samog slanja i plaćanja skupih usluga poput Googleovog Gmail-a i Microsoftovog Outlook-a [9].

3.9. Responzivnost i izgled

Responzivnost označava smjer razmišljanja gdje se aplikacija prilagođava (eng. *responds*) korisničkom ponašanju i okolini na temelju veličine, platforme i orijentacije [10].

Na primjer, na ispisu je prikazan izgled sučelja na zaslonu veličine mobilnog uređaja. Za skaliranje fonta korišteni su CSS medijski upiti (eng. *media query*). Medijskim upitima omogućeno je postavljanje vrijednosti CSS atributa ovisno o količini piksela. Određenu točku pri kojoj se aktivira CSS kôd unutar medijskog upita naziva se *breakpointom.* Primjer je dan u sljedećem ispisu (Ispis 16).

```
:root {
  font-size: 18px;
}
@media only screen and (max-width: 600px) {
  :root {
    font-size: 12px;
  }
}
@media only screen and (min-width: 600px) {
  :root {
    font-size: 14px;
  }
}
@media only screen and (min-width: 768px) {
  :root {
    font-size: 16px;
  }
}
```

Ispis 16: Primjer medijskog upita

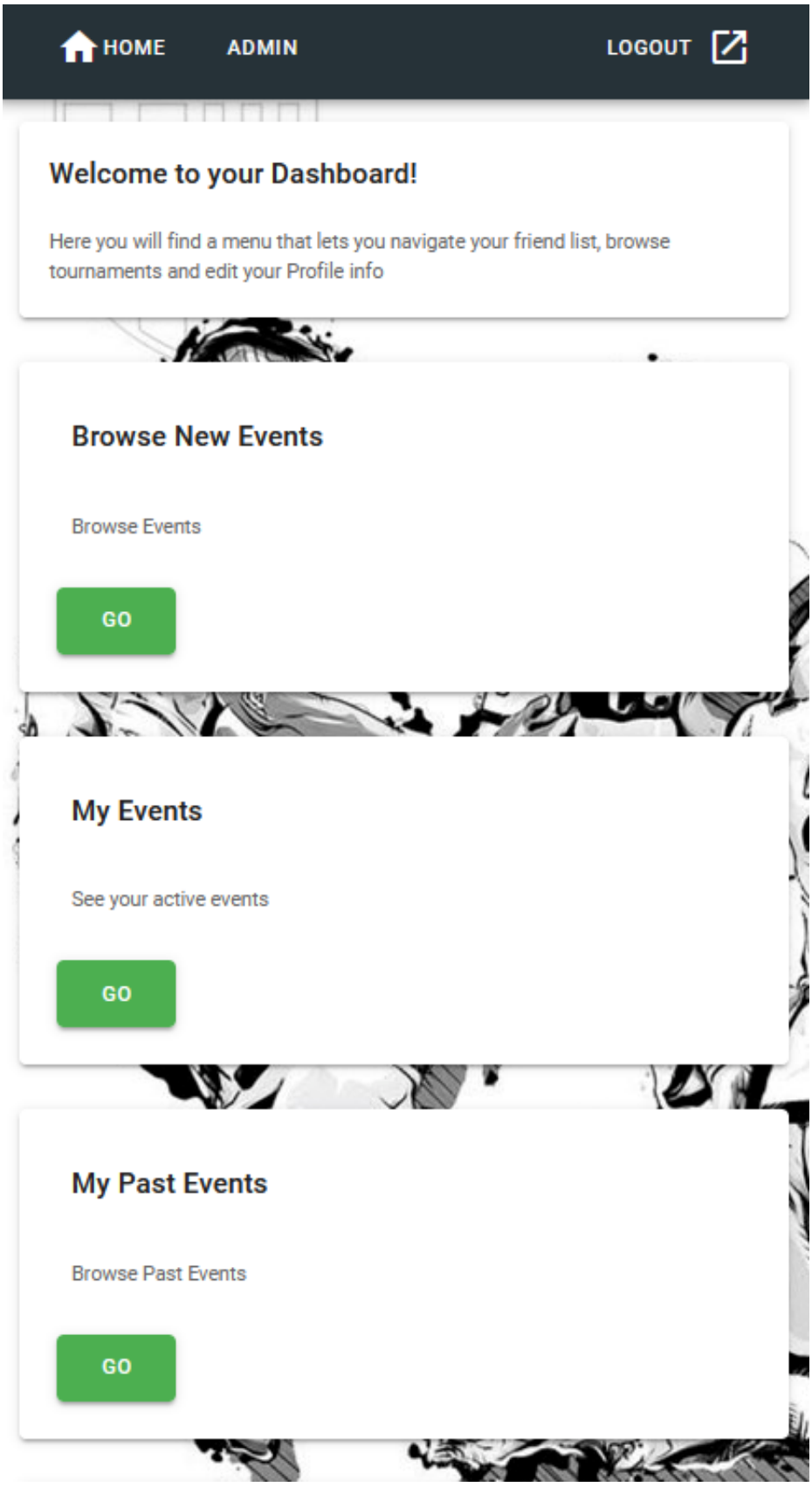
Za potrebe ove aplikacije postavljena je osnovna veličina fonta od 18 piksela. Smanjenjem zaslona na veličine definirane kôdom, font se postepeno smanjuje u koracima od 2 sve do 12 piksela za prikaz na manjim mobilnim uređajima. Pri tom svaka točka promjene odgovara standardnim veličinama često korištenih uređaja poput računala, manjih laptopa, tableta i mobilnih uređaja. U izvedbi ove aplikacije zanemareno je prilagođavanje izgleda za pametne narukvice jer takve zahtijevaju zaseban dizajn i ne spadaju u ciljanu skupinu uređaja koje ova aplikacija podržava.

Koristeći biblioteku *Vuetify* implementiran je željeni stil aplikacije po Googleovom *Material Design* dizajnerskom jeziku. Značajke *Material Designa* su ugodan izgled prilagođen *mobile-first* pristupu u izradi korisničkih sučelja. Stoga je izgled aplikacije modeliran u obliku kartica koje lako skaliraju u veličini ovisno o veličini uređaja na kojemu se prikazuju. *Vuetify* također nudi posebne atribute za postavljanje veličine *Vue* komponenti temeljene na CSS *breakpointovima*. Primjer toga je vidljiv u idućem ispisu (Ispis 17).

```
<v-row>
    <v-col cols="12" lg="4" sm="12" xs="12"
    v-for="card in cards" :key="card.link">
     <v-card :card="card" class="pa-3" hover>
       <v-card-title class="mb-3">{{ card.title }}</v-card-title>
       <v-card-text>{{ card.text }}</v-card-text>
       <v-card-actions>
         <v-btn left class="success" router :to="card.link">Go</v-btn>
       </v-card-actions>
     </v-card>
   </v-col>
 </v-row>
```

Ispis 17: Primjer pomoćnih atributa za postizanje responzivnosti

Na slici (Slika 17) je prikazan izgled glavnog izbornika na zaslonu veličine prosječnog mobilnog uređaja. Responzivnost je postignuta koristeći spomenute *breakpointove*.



Slika 17: Mobilna verzija glavnog izbornika

4. Zaključak

Web aplikacijom opisanom u ovome radu omogućeno je spontano i jednostavno povezivanje ljudi s interesom igranja nogometa. *Mobile-first* pristup u izradi iste se pokazao jako korisnim kao način za efektivno skaliranje i prilagođavanje korisničkog sučelja raznim veličinama zaslona. Moderne tehnologije poput *Vuea* i njezine biblioteke *Vuetify* uvelike olakšavaju takav pristup rada, a sam kôd je jako organiziran i čitljiv. Rad sa *Vueom* je iznimno brz i efektivan što ga čini idealnim rješenjem za razvoj aplikacija ovog tipa i slično.

Sa poslužiteljske se strane *Spring boot,* kao već provjereno rješenje s velikom podrškom i pregrštima informacija i materijala dostupnim na mreži, pokazao kao izvrstan alat za izradu pozadinske strane aplikacije. Funkcionalnosti poput spajanja i izvršavanja upita nad bazom, namještanja automatskih periodičkih zadataka, slanja mailova i mnogih drugih se vrlo brzo i kvalitetno daju izraditi *Spring bootom*.

S obzirom na veliku količinu funkcionalnosti koje još mogu biti dodane u aplikaciju, njen razvoj će se nastaviti. Planovi za budućnost rada na aplikaciji uključuju proširenje samog komunikativnog dijela koji se trenutno sastoji od slanja mailova i pisanja komentara dodavanjem integracije društvenih medija poput Facebooka i Googlea sa svrhom omogućavanja dijeljenja sadržaja na istim.

Literatura

[1] Visual Studio Code, „Documentation“, https://code.visualstudio.com/docs - Datum zadnjeg pristupa 27.08.2021

[2] Intellij Idea, „What is Intellij Idea?“, https://www.jetbrains.com/idea/features/ - Datum zadnjeg pristupa 27.08.2021

[3] VueJs, „Getting started“, https://vuejs.org/ - Datum zadnjeg pristupa 27.08.2021

[4] Evan You, „The story of Evan You, the father of Vue js“, https://www.ateamsoftsolutions.com/the-story-of-evan-you-the-father-of-vue-js/ - Datum zadnjeg pristupa 27.08.2021

[5] Spring vs Spring boot, „A comparison between Spring and Spring boot“, https://www.baeldung.com/spring-vs-spring-boot - Datum zadnjeg pristupa 27.08.2021

[6] PostgreSQL, „What use PostgreSQL“, https://www.postgresql.org/about/ - Datum zadnjeg pristupa 27.08.2021

[7] Spring Data JPA, „Documentation“, https://spring.io/projects/spring-data-jpa - Datum zadnjeg pristupa 27.08.2021

[8] JPQL, „JPA JQPL Introduction“, https://www.javatpoint.com/jpa-jpql-introduction - Datum zadnjeg pristupa 27.08.2021

[9] Mailtrap, „Mailtrap by Railsware“, https://devcenter.heroku.com/articles/mailtrap - Datum zadnjeg pristupa 27.08.2021

[10] Responsive Web Design, „Responsive Web Design – What it is and how to use it“, https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/ - Datum zadnjeg pristupa 27.08.2021