# Web aplikacija za organiziranje aktivnosti na javnim sportskim terenima

**Kovačević, Luka**





**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**

**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni preddiplomski studij**

# WEB APLIKACIJA ZA ORGANIZIRANJE AKTIVNOSTI NA JAVNIM SPORTSKIM TERENIMA

**Završni rad**

**Luka Kovačević**

**Osijek, 2021.**

# SADRŽAJ

# 1. UVOD

U današnje vrijeme kada je migracija ljudi zbog posla sve prisutnija, mnogi se ljudi nalaze u mjestima stanovanja te nisu upoznati s javnim površinama za rekreativno bavljenje sportom. Svakodnevno stotine tisuća ljudi idu na sportske terene kako bi se rekreativno bavili nekim sportom. Mnogi od njih imaju problema s pronalaskom prikladnog javnog sportskog terena, kao i pronalaskom ljudi s kojima bi igrali timski sport. Lokalnom stanovništvu je također često problem među amaterskim igračima dogovoriti vrijeme i mjesto okupljanja. Trenutno se na tržištu ne nudi aplikacija koja može povezati ljude i istovremeno im ponuditi točne vremenske dogovore na točno određenim javnim sportskim terenima za više sportova.

Cilj ovog završnog rada je istražiti postojeća rješenja i usporediti ih pa sukladno tome napraviti vlastito rješenje koje će korisnicima ponuditi aplikaciju za snalaženje u novom naselju s ciljem organizacije lokalnih amaterskih sportskih okupljanja. Aplikacija će omogućiti registriranje i prijavu korisnika, nakon toga mogućnost odabira grada i sporta u kojemu želi pristupiti terenima, prijavljivanje na iste terene, prikaz dostupnosti terena i dodavanje drugih korisnika u svoju društvenu mrežu radi lakše organizacije. U drugom poglavlju su prikazane slične aplikacije i prikazano je trenutno stanje u području aplikacija koje nude prijavljivanje korisnika na sportske terene. Treće poglavlje prikazuje pozadinski dio aplikacije koji preko web sučelja dohvaća i šalje podatke nazad na korisnički dio web aplikacije. U četvrtom poglavlju su opisane programske tehnologije korištene prilikom izrade aplikacije, kao i prikaz najbitnijih dijelova programskog rješenja, dok je u petom poglavlju prikazano korištenje aplikacije i njeno testiranje.

## 1.1. Zadatak završnog rada

U završnom radu treba opisati mogućnost primjene web aplikacije za upravljanje slobodnim terminima javnih sportskih terena, te analizirati zahtjeve na sustav. Potrebno je predložiti model i arhitekturu web aplikacije, dijagram baze podataka, te adekvatno poslužiteljsko rješenje i projekcije maksimalnog opterećenja rada web aplikacije za organizaciju aktivnosti na javnim sportskim terenima, te predložiti metode poslužiteljskog skaliranja. Aplikacija treba imati mogućnost upravljanja rasporedom terena, dodavanja novih terena, priključivanja novih korisnika u aplikaciju prijavom na određeni događaj, pozivanja drugih korisnika na određeni događaj, notifikacije, administracije podataka unutar sustava. Potrebno je opisati korištene programske tehnologije i okolinu za razvoj web aplikacije, opisati koncept aplikacije, a u praktičnom dijelu rada programski ostvariti opisanu web aplikaciju s bazom podataka, te je

ispitati na dovoljnom skupu testnih korisnika, implementirati potrebne programske sigurnosne mehanizme, napraviti statičku analizu koda, te predložiti koncept mogućih budućih nadogradnji i daljnjeg razvoja.

# 2. PREGLED TRENUTNOG STANJA U PODRUČJU PRIJAVLJIVANJA NA JAVNE SPORTSKE TERENE

U ovom poglavlju su izložene sve postojeće aplikacije pretražene putem Interneta koje se bave istom problematikom kao što je zadatak ovog rada.

## 2.1. Pregled stanja u području

Organizacija bilo kakvog događaja, neovisno o broju ljudi, može iziskivati ponekad puno vremena i truda. Napretkom tehnologije, puno je lakše upravljati organizacijom događaja. Kada se govori o aplikacijama za organizaciju sportskih događanja, kakva se opisuje u ovome radu, na tržištu ne postoji puno opcija. Upravo iz tog razloga odlučeno je kreirati aplikaciju koja će pomoći korisnicima da na što lakši način dogovore sportsku aktivnost na javnom terenu u njihovom gradu. Nakon istraživanja, pronađeno je nekoliko aplikacija koje imaju djelomičnu funkcionalnost aplikacije opisane u ovom završnom radu. Prema tome, nije pronađena nijedna aplikacija koja ima punu funkcionalnost zadanog rada.

## 2.2. Postojeće slične aplikacije

### 2.2.1. Courtify

Prema [1], Courtify je jedina slična aplikacija za mobilne uređaje koja se trenutno nalazi na tržištu. U ovom trenutku ju je na Google Play Storeu preuzelo preko 1000+ korisnika.

Zaključno sa srpnjem 2021. godine, ona sadrži četiri terena za odabir, odnosno korisnici se mogu prijaviti na teren za nogomet, odbojku, glavomet, padel i tenis. Aplikacija nudi rezerviranje termina na jednom od igrališta unutar privatnih klubova s danom, vremenom početka, trajanjem rezervacije i željenog sportskog terena. Ako je željeni termin slobodan korisnik ga može rezervirati. Ono što ju bitno razlikuje od aplikacije o kojoj je ovaj rad jest to što Courtify svojim korisnicima omogućuje rezervaciju termina samo na terenima unutar privatnih klubova, dok se aplikacija iz rada temelji na rezerviranju termina na javnim sportskim površinama.

### 2.2.2. Facebook Events

Prema [2], Mark Zuckerberg je 2003. godine počeo s izradom Facebooka, web aplikacije koja će od 2004., nakon službenog osnivanja, postati jedna od najpoznatijih aplikacija na svijetu. Facebook mjesečno koristi 2.895 milijardi korisnika, a dnevno 1.95 milijardi. Mnogim tim korisnicima Facebook je koristan zbog brojnih alata koje nudi, među kojima su i Facebook

Events, koji na toj web aplikaciji postoje od 2005. godine. Facebook Events korisniku služi za kreiranje događaja, ali i za pregled događaja po njegovoj lokaciji ili interesu.

Pri kreiranju događaja na Facebooku, među kojima i sportskih, prvo je potrebno događaju dodijeliti naziv, datum i vrijeme početka. Korak kasnije, potrebno je napisati opis događaja i po želji dodijeliti mu naslovnu fotografiju. Nakon toga, potrebno je odrediti tko može vidjeti taj događaj te tko mu se kasnije može pridružiti. Prema tome, događaj je moguće podijeliti privatno, javno, među prijateljima i grupom. Privatno se odabire ako se želi da taj događaj vide samo osobe koje su pozvane. Kada se događaj postavi javno to znači da ga mogu vidjeti i pridružiti mu se svi korisnici Facebooka. Ako događaju trebaju prisustvovati samo prijatelji s korisničke liste prijatelja, onda se odabire opcija „prijatelji". Kada je događaj ciljan za određenu grupu koja postoji na Facebooku, onda se odabire opcija "grupa". Nakon kreiranja događaja, moguće je pozvati neke od korisnika za koje je organizator odlučio da želi da prisustvuju događaju. Osim što organizator može pozvati, on može dozvoliti i da pozvani korisnik može pozvati druge korisnike na taj isti događaj. Kada je korisnik pozvan na događaj može odabrati jednu od tri opcije, odnosno "dolazim", "možda" i "ne mogu doći". Organizator događaja će dobiti obavijest nakon što osoba koju je pozvao odabere jednu od te tri opcije. Osim toga, Facebook Events nudi i mogućnost da korisnik može vidjeti tko je odabrao jednu od tih opcija pa po tome može vidjeti tko dolazi na događaj, a tko ne.

### 2.2.3. Hamburg Active City

Prema [3], Hamburg Active City je inicijativa njemačkog grada Hamburga da se potakne ljude na sportske aktivnosti. Na stranicama grada ova inicijativa ima svoj dio gdje se mogu pogledati sva sportska događanja u gradu te u par klikova doći do informacija o klubovima i sportskim objektima. Trenutno sadrži interaktivnu kartu s 1566 sportskih objekata. Svaki objekt ima svoju ikonicu za sport kojim se moguće baviti na toj lokaciji kao i dodatne informacije o terenu (adresa, vlasnik, vrsta terena, tip podloge itd.). Hamburg Active city trenutno ne nudi korisniku prijave na terene već samo informacije o terenu gdje se nalazi i koje je vrste.

### 2.2.4. Timster

Prema [4], Timster je napravljen za lakše organiziranje malonogometnih utakmica putem android aplikacije. Napravljena je od strane srpske firme Solaris development & Design i dobitnik je druge nagrade *Smart City Challenge Serbia*. Aplikacija se fokusirala samo na privatne događaje za mali nogomet. Omogućuje sastavljanje timova, pronalaženje terena te ako

nema dovoljan broj igrača, pronalazak igrača u određenom krugu od postavljene lokacije kako bi ih se pozvalo na utakmicu.

### 2.2.5. SportEasy

Prema [5], aplikaciju SportEasy je napravila francuska firma SportEasy SaS. Predviđena je za korištenje amaterskim klubovima za lakše organiziranje treninga i utakmica. Pokriva većinu sportova i besplatna je za korištenje. Trenutno je koristi oko 10 tisuća klubova i 1.5 milijuna korisnika. Nudi jednostavna rješenja za kreiranje sportskih događaja i utakmica kao i pozivanje i potvrđivanje ljudi na dolazak. Također ova aplikacija nudi pregled statistike igrača ovisno o njihovom učinku i dolascima.

### 2.2.6. Premier Tennis Osijek

Prema [6], u Osijeku 2021. godine je otvoren prvi javni teniski teren na koji se može prijaviti bilo tko. Kako bi se prijavili na teren potrebno je otići na njihovu službenu stranicu i prijaviti se u željenom terminu. Da bi se ušlo na teren potrebno je prilikom dolaska na teren spojiti na bežičnu internetsku vezu kako bi se dobio kod, kojim se potom može ući na teniski teren.

# 3. IDEJNO RJEŠENJE I MODEL WEB APLIKACIJE

U ovom poglavlju opisan je dijagram rada aplikacije i arhitektura baze podataka.

## 3.1. Dijagram rada aplikacije

Dijagram rada aplikacije kreira se radi lakše izrade aplikacije, a omogućava tehničku strukturu za lakše shvaćanje cijelog procesa koji korisnik prolazi kroz aplikaciju. Prilikom otvaranja web aplikacije, korisniku se prikazuje sučelje za registraciju i prijavljivanje u aplikaciju. Nakon što se korisnik prijavio, ima mogućnost odabira grada i sporta u kojem želi provjeriti javne sportske terene, a može dodati i svoje „gradove favorite“. Ulaskom u jedan od ponuđenih gradova, korisnika dočekuje sučelje s listom javnih sportskih terena za određeni sport. Odabirom jednog od terena korisnika se vodi na izbor termina na željenom terenu. Ako se u tom terminu prijavilo više od željenog broja ljudi, korisnik će dobiti obavijest kako se ne bi stvarale gužve na određenim terenima pa korisnik može potražiti drugi teren ili termin. Na kraju, korisnik može preko svog korisničkog imena dodati druge korisnike u svoje favorite te, ako drugi korisnik prihvati, dobiti obavijesti da se osoba s njegove liste favorita prijavila u određenom terminu na javni sportski teren. Osim toga, korisnik može i prijaviti grešku na određenom javnom sportskom terenu. Za kraj, korisnik ima mogućnost pregleda svojih informacija, liste favorita te odjavu iz web aplikacije.

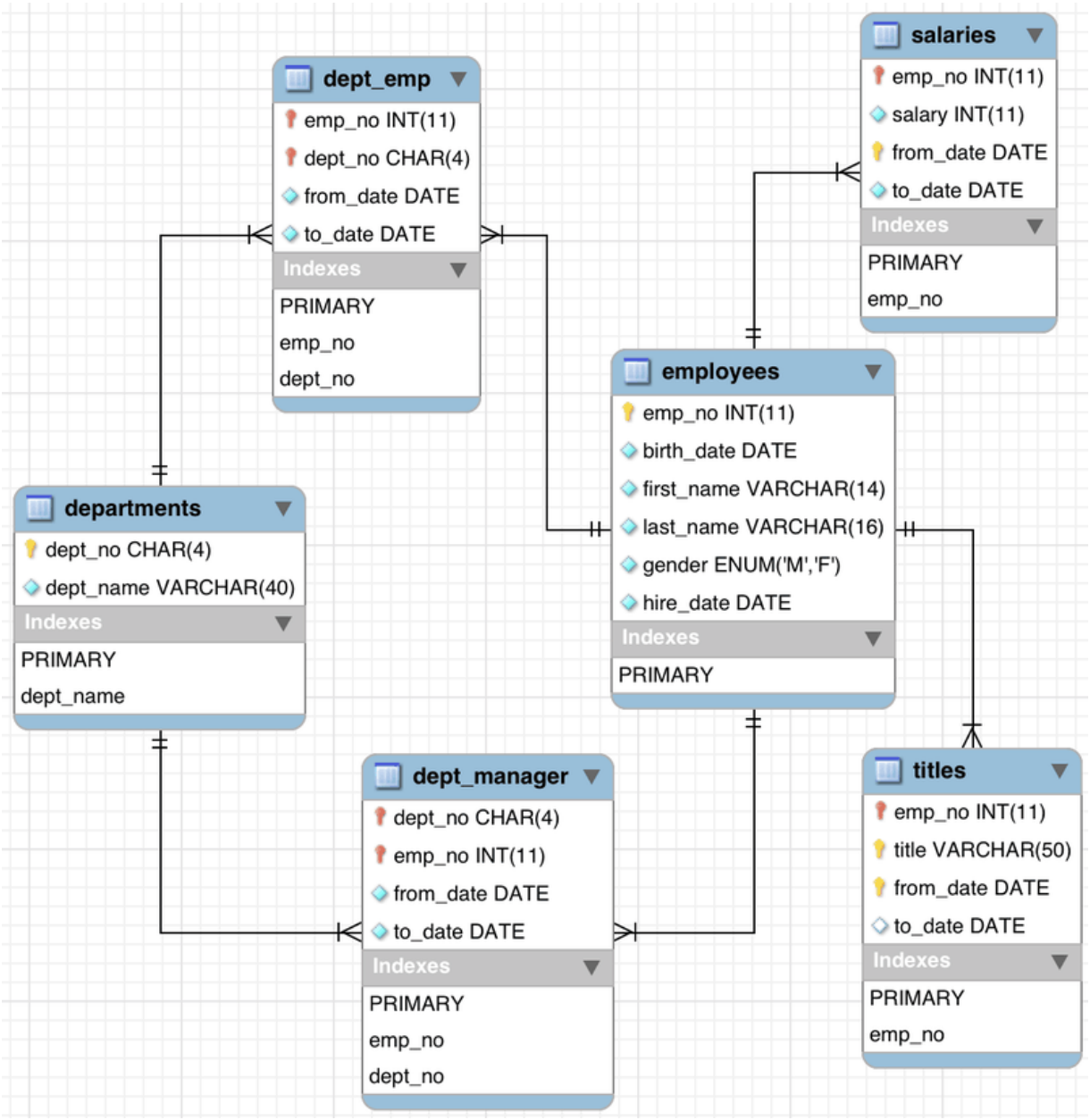
Prema slici 3.1 moguće je vidjeti da su funkcionalni zahtjevi za web aplikaciju sljedeći:

- Pri ulasku u aplikaciju korisnik se registrira ili prijavljuje
- Pretraga i pregled gradova i sportova
- Pregled javnih sportskih terena
- Prijavljivanje u termin na javnom sportskom terenu
- Dodavanje korisnika favorita
- Prijava oštećenja na javnom sportskom igralištu


**Slika 3.1** Dijagram rada aplikacije

## 3.2. Arhitektura baze podataka

Kako bi mogućnost spremanja gradova, javnih sportskih terena, korisnika, zakazanih termina korisnika i korisnikovih favorita bila ostvariva, potrebno je sve te podatke spremiti u bazu podataka. U ovom radu korištena je baza podataka MySQL (kratica od engl. *My Structured Querry Language*). Baza podataka MySQL je standardna SQL baza gdje se podaci spremaju u tablice. Ovakva baza koristi relacijski model povezivanja podataka. Prema slici 3.2 u ovome primjeru je korištena jedna baza podataka s više tablica u koje se spremaju potrebni podaci.




**Slika 3.2** Primjer izgleda baze podataka

# 4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

Za izradu pozadinskog dijela ove aplikacije odabran je jezik PHP uz programski okvir Laravel. U ovom poglavlju su objašnjene sve tehnologije i programski jezici korišteni prilikom izrade aplikacije završnog rada. Također su prikazani najbitniji dijelovi koda iz aplikacije i objašnjeni njeni elementi. Dodatno je objašnjena moguća nadogradnja aplikacije.

## 4.1. Korištene programske tehnologije

### 4.1.1. PHP

Prema [7], PHP je programski jezik koji je orijentiran prema C i Pearl sintaksi, a prvenstveno je namijenjen programiranju web stranica. PHP (kratica od engl. *Hypertext Prepocessor*) se ističe širokom podrškom baza podataka i internet protokola. Trenutno se PHP nalazi u preko 75% web aplikacija na Internetu pa ga to čini najrasprostranjenijim jezikom. Glavni konkurent PHP-u je Node.js jezik i okolina koja u posljednje vrijeme dobiva sve više na popularnosti. Prvotni PHP je nastao 1995. godine i zvao se PHP/FI, a napravio ga je Rasmus Lerdorf. U početku je PHP bio skup Pearl skripti za brojanje posjeta na njegovoj web stranici. Kada je nastala potreba za više funkcija, razvio je PHP u programskom jeziku C, a ova inačica je mogla raditi s bazama podataka i omogućila je korisnicima jednostavno kreiranje dinamičkih web stranica. PHP je program otvorenog koda i može ga bilo tko razvijati. Od svog nastanka, PHP je prošao kroz nekoliko inačica te je zadnja trenutna inačica PHP 7, dok je PHP 8 još u fazi testiranju.

### 4.1.2.Laravel

Prema [8], Laravel je slobodan programski okvir za programski jezik PHP koji prati model-pogled-upravitelj (engl. *MVC, model-view-controller*) arhitekturu, a baziran je na programskom okviru Symfony. Prvi put u beta verziji je izdan 2011. godine od strane Taylora Otwella. On omogućava pisanje lako čitljivog i jednostavnog koda. U startu nudi pakete koji značajno olakšavaju i skraćuju početak projekta. Sa svojim mnogim značajkama olakšava upravljanje bazama i modelima, obradu podataka, UNIT testiranje itd. Kroz svoj životni vijek je prošao kroz nekoliko inačica te je sada dostupan u verziji Laravel 8.

### 4.1.3. Javascript

Prema [9], JavaScript je najpopularniji skriptni jezik na svijetu kojeg podržavaju svi internetski preglednici. Za korištenje JavaScripta nije potrebna licenca. Cilj kreiranja JavaScripta je bio

dodati interaktivnosti u HTML stranice. JavaScript je interpreter što znači da se skripta izvršava naredbu po naredbu. Godine 1996. nastaje prva inačica JavaScripta koju je kreirao Brenden Eich. Kako bi se jezik što više koristio morao je biti standardiziran pa je 1998. godine odobren od strane međunarodnog ISO standarda. Trenutna inačica JavaScripta koji se koristi je ES6.

### 4.1.4. MySQL

Prema [10], MySQL je besplatan i otvoren sustav upravljanja relacijskom bazom podataka. MySQL je jedna od najčešćih izbora za bazu podataka prilikom izrade projekta. Relacijskog je tipa što omogućava strukturiran način pohranjivanja i pretraživanja velikih količina podataka. Napravljena je tako da bude brza što je dovelo do manjka funkcionalnosti. Vrlo je stabilna i ima dobro dokumentirane module i ekstenzije. Baza podataka MySQL ima podršku brojnih programskih jezika. Napravljena je od strane švedske kompanije MYSQL AB 1995. godine.

### 4.1.5. NGINX

Prema [11], NGINX je besplatan i otvoren web poslužitelj koji se može koristiti kao *reverse proxy*, HTTP *cache* i upravitelj opterećenjem (engl. *load balancer*). Dizajniran je za malu potrošnju memorije i veliku konkurentnost konekcija. Prilikom zahtjeva na web stranicu ne kreira se proces za svaki zahtjev već se zahtjevi odrađuju asinkrono. NGINX je kreirao Igor Sysoev 2004. godine te je do sada našao svoju uporabu kod mnogih velikih tvrtki kao što su Microsoft, IBM, Google itd.

### 4.1.6. Composer

Prema [12], Composer je računalni alat za upravljanje paketima u programskom jeziku PHP. Prilikom instalacije paketa za projekt, ne deklarira paket globalno nego ga specifično instalira od projekta do projekta, ovisno na kojem projektu ga se želi instalirati. Također, Composer ažurira pakete posebno za projekt kojem je potrebno ažuriranje. Razvili su ga 2012. godine Nils Adermann i Jordi Boggiano.

### 4.1.7. JSON

Prema [13], JSON je otvoreni standardizirani transportni format za dijeljenje podataka koji koristi čitljiv i razumljiv tekstualni zapis. Neovisan je o programskom jeziku te ga time može koristiti bilo koji programski jezik za obradu podataka. Format JSON polako zamjenjuje XML, a nastao je iz JavaScripta.

### 4.1.8. REST API

Prema [14], REST API je programsko sučelje koje daje vanjskim aplikacijama mogućnost obavljanja upita i ažuriranja aplikacijskih podataka. Oslanja se na komunikacijski protokol koji je bez stanja (engl. *stateless*), klijentsko-poslužiteljski, te uz mogućnost spremanja *cache* memoriju, a u praksi se uvijek koristi kroz protokol HTTP.

### 4.1.9. HTML

Prema [15], HTML (engl. *Hypertext Markup Language*) je prezentacijski jezik koji se koristi prilikom izrade web stranica. Njime se oblikuje izgled stranice te se stvaraju hiperveze na dokumente i druge stranice. Prvotno je korišten kao jezik za semantičko opisivanje znanstvenih dokumenata, te je bio vrlo ograničen. U kasnijim inačicama HTML-a (trenutna inačica HTML5) su proširene njegove mogućnosti kao što su dodavanje slika, prikaz videa i brojni drugi uređivački alati. Zato što je lagan za učenje je opće prihvaćen i postao standardom prilikom izrade web stranica.

### 4.1.10. CSS

Prema [15], CSS (engl. *Cacade Style Sheet*) je stilski jezik za dodatno uređivanje stranica koje su napisane u HTML prezentacijskom jeziku. Pomoću CSS-a stranice je moguće dodatno urediti te ih prilagoditi svim vrstama ekrana. CSS je nezavisan od HTML-a, ali se najčešće koriste zajedno. Razdvajanjem HTML-a i CSS-a se dobiva lakše održavanje stranica, dijeljenje stilova na više stranica i prilagođavanje stranica drugim okolinama.

### 4.1.11. Ajax

Prema [16], Ajax je skup tehnologija za izradu web stranica kako bi se mogao ostvariti asinkroni rad web stranica. Ajax koristi HTML i CSS elemente uz JavaScript programski jezik kako bi stranica mogla biti dinamička.

## 4.2. Najbitniji dijelovi programskog koda

### 4.2.1. Postavljanje korisnika

Ako prilikom izrade aplikacije nije postavljen sustav za registraciju korisnika, sustav ne bi mogao spremati personalizirane podatke koje korisnik unosi te bi se oni obrisali jednom kada korisnik napusti web lokaciju. Za neke web aplikacije nije potrebno imati korisnika, no u ovom

slučaju. kako bi korisniku pružili što više mogućnosti i jednostavnije korištenje. je potrebno spremiti njegove podatke i odabire.

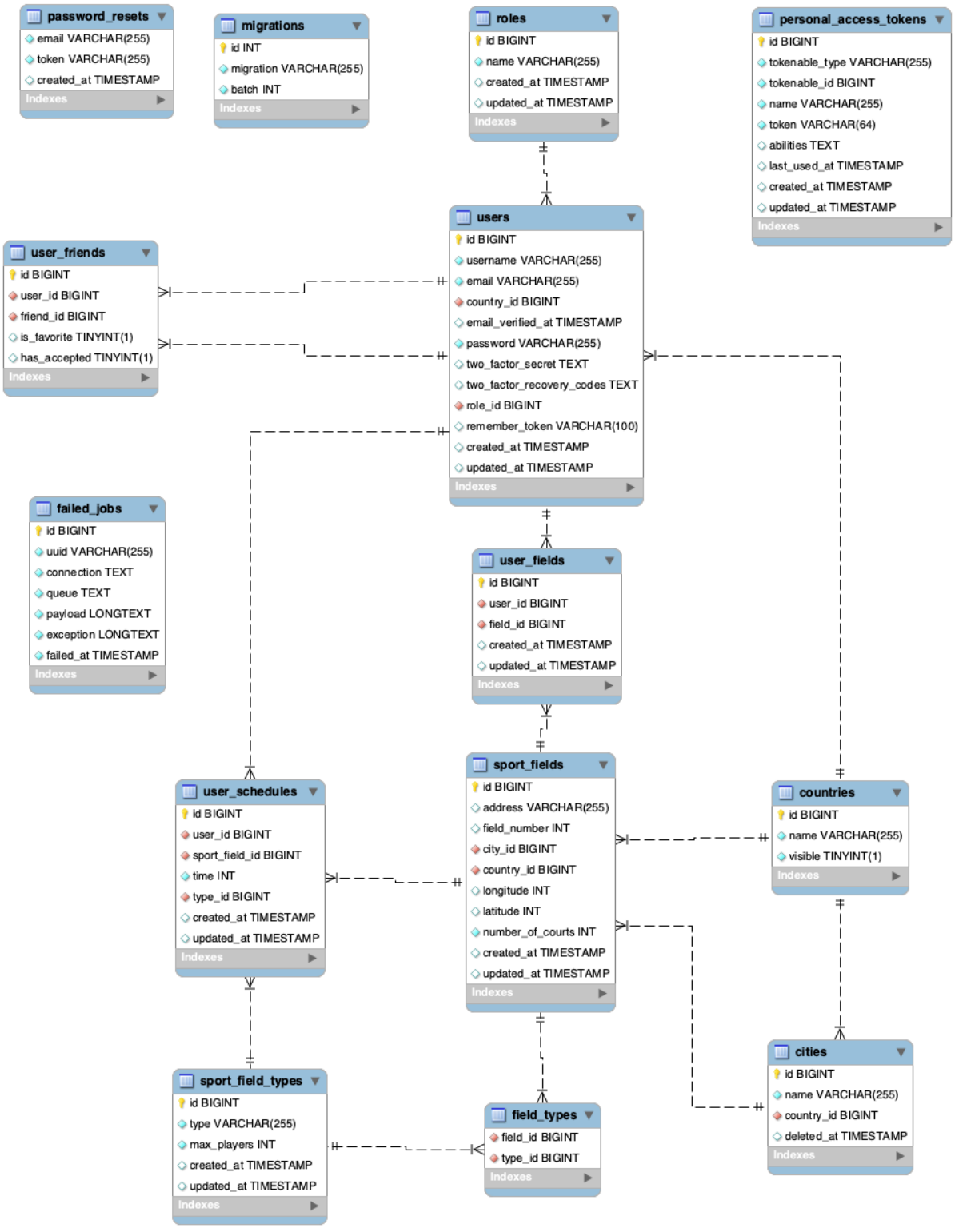
Kako bi korisnik mogao koristiti web aplikaciju mora se prvo registrirati. Prilikom registracije unosi ime, prezime, elektroničku adresu i korisničko ime. U ovom slučaju koristi se korisničko ime kako bi korisnik što lakše mogao pronaći druge korisnike te je ono jedinstveno na *web* aplikaciji. Aplikacija daje do znanja da korisnik mora ispuniti sva potrebna polja kako bi se registrirao i imao omogućen ulazak u aplikaciju.

Ako je ispunio sve potrebne podatke, njegovi podaci se spremaju u bazu podataka MySQL i korisniku se omogućava ulazak u aplikaciju.

### 4.2.2. Baza podataka

Obrada podataka pomoću PHP i MySQL-a se provodi korištenjem web sučelja i nekoliko vrsta ruta (prethodno opisano u potpoglavlju 4.8.3), uz korištenje transportnog formata JSON. Prema slici 4.1 je prikazana struktura baze podataka korištena u radu.

Baza podataka „sportcourt“ se sastoji od 14 tablica (Slika 4.1). Od tih 14 tablica, 10 se međusobno povezane stranim *id*-evima i aplikacija ih aktivno koristi tijekom svog rada. Preostale četiri tablice se koriste povremeno prilikom rada u okviru Laravel.




**Slika 4.1** Prikaz hijerarhije baze podataka *sportcourt*

Laravel za izradu tablica unutar baze podataka koristi migracije (engl. *migrations*). Unutar migracije se definiraju polja koja će određena tablica imati kao i njihovu vrstu. Migracije nude puno vrsta tipova polja koje su prilagođene Laravel frameworku kako bi olakšale njihovu izradu, npr. *id*, *foreingId*, *uuid*, *json*, *string*, *text* itd.. Za svako polje je moguće dodati i određene

zahtjeve koje ona trebaju ispunjavati, npr. *nullable*, *required*, *unique*, *constrained* (Slika 4.2). Migracija *users* sadri funkciju koja pravi tablicu baze podataka i automatski kreirana korisnike potrebne za testiranje aplikacije.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('username')->unique();
            $table->string('email')->unique();
            $table->foreignId('country_id')->constrained('countries');
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->foreignId('role_id')->constrained('roles');
            $table->rememberToken();
            $table->timestamps();
        });

        DB::table('users')->insert([
            'username' => 'admin',
            'country_id' => 1,
            'email' => 'admin@admin.com',
            'password' => Hash::make('test1234'),
            'role_id' => 1
        ]);
        DB::table('users')->insert([
            'username' => 'prvi',
            'country_id' => 1,
            'email' => 'prvi@prvi.com',
            'password' => Hash::make('test1234'),
            'role_id' => 2
        ]);
        DB::table('users')->insert([
            'username' => 'drugi',
            'country_id' => 1,
            'email' => 'drugi@drugi.com',
            'password' => Hash::make('test1234'),
            'role_id' => 2
        ]);
        DB::table('users')->insert([
            'username' => 'treci',
            'country_id' => 1,
            'email' => 'treci@treci.com',
            'password' => Hash::make('test1234'),
            'role_id' => 2
        ]);

    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

**Slika 4.2** Prikaz migracije *users*

### 4.2.3. Model-pogled-upravitelj

Laravel programski okvir prati arhitekturu model-pogled-upravitelj (engl. *Model-View-Controller*). *Model* je središnji dio programa. On direktno upravlja logikom, podacima i pravilima aplikacije. Sljedećim primjerom koda, prema slici 4.3, je prikazan model *User* iz aplikacije.

```php
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;

    protected $fillable = [
        'email',
        'username',
        'password',
        'country_id',
        'role_id'
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];

    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function friend()
    {
        return $this->hasMany(UserFriends::class);
    }

    public function field()
    {
        return $this->hasMany(SportField::class);
    }

    public function role()
    {
        return $this->belongsTo(Role::class);
    }
}
```

**Slika 4.3** Prikaz modela *Users*

*Fillable* je područje gdje se u modelu određuje koja se polja tablice *„users“* u bazi mogu popuniti s korisničke strane aplikacije. *Hidden* polja određuju koja polja će biti sakrivena prilikom poziva „User“ modela kako ne bi bila vidljiva unutar aplikacije. Funkcije *friend* i *field* su relacijske funkcije (specifične za Laravel okruženje) koje olakšavaju dohvaćanje prijatelja i terena povezanih s korisnikom (npr. kada se dohvati određeni korisnik, pozivanjem funkcije *korisnik->friend()* dohvaćamo preko *UserFriends* modela sve korisnike koje je korisnik dodao u svoju listu prijatelja). Model *UserFields* služi za određivanje korisnikovih omiljenih terena koje dodaje na listu omiljenih terena (Slika 4.4).

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class UserFields extends Model
{
    public $timestamps = false;

    protected $fillable = ['user_id', 'field_id'];

    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function field()
    {
        return $this->belongsTo(SportField::class);
    }
}
```

**Slika 4.4** Prikaz modela *UserFields*

Model *UserFields* služi za određivanje koja polja se mogu unijeti, u ovom slučaju su to *user_id* i *field_id*. U modelu se nalaze i dvije relacijske funkcije *user* i *field*. Pomoću ovih funkcija dohvaćaju se svi podaci od korisnika i terena koje je korisnik dodao na svoju listu omiljenih terena. Varijablom *timestamps* definirano je da prilikom zapisivanja u bazu neće imati zapis kada je dodavanje polja napravljeno. Prema slici 4.5 se vidi model *UserFriends* koji služi za određivanje korisnikovih prijatelja koje dodaje na svoju listu prijatelja.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class UserFriends extends Model
{
    public $timestamps = false;

    protected $fillable = ['user_id', 'friend_id', 'is_favorite', 'has_accepted'];

    public function friend()
    {
        return $this->belongsTo(User::class, 'friend_id', 'id');
    }

    public function user()
    {
        return $this->belongsTo(User::class, 'user_id', 'id');
    }

}
```

**Slika 4.5** Prikaz modela *UserFriends*

Model *UserFriends* sadrži četiri moguća polja za popunjavanje. U trenutnoj fazi projekta se koriste *user_id* i *friend_id* koji su oboje *id*-evi preuzeti iz users tablice. Polja *is_favorite* i *has_accepted* su polja koja koriste korisnicima da prihvate ili odbiju neželjeno dodavanje u listu prijatelja. Relacijska funkcija *friend* služi za dohvaćanje prijateljevih podataka. Model *SportField* služi kako bi administrator mogao kreirati sportski teren kojeg korisnik naknadno može dodati u svoje omiljene terene (Slika 4.6).

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class SportField extends Model
{
    protected $fillable = [
        'address',
        'field_number',
        'city_id',
        'country_id',
        'longitude',
        'latitude',
        'number_of_courts'
    ];

    public function city()
    {
        return $this->belongsTo(City::class);
    }

    public function country()
    {
        return $this->belongsTo(Country::class);
    }

    public function schedule()
    {
        return $this->belongsToMany(UserSchedule::class);
    }

    public function type()
    {
        return $this->hasManyThrough(SportFieldType::class, FieldTypes::class, 'field_id', 'id', 'id', 'type_id', );
    }

}
```

**Slika 4.6** Prikaz modela *SportField*

U ovom modelu se nalazi sljedećih šest polja:

- *address* - adresa igrališta i moguć njegov naziv
- *city_id* - povezani *id* grada iz *cities* tablice
- *country_id* - povezani *id* iz *countries* tablice
- *longitude* – geografska širina na kojoj se nalazi igralište
- *latitude* – geografska dužina na kojoj se nalazi igralište
- *type_id* – povezani *id* igrališta iz *types* tablice(nogometno, košarkaško, odbojkaško itd.)
- *number_of_courts* – broj terena koji se nalaze u sklopu igrališta

Varijable *longitude* i *latitude* se u trenutnoj inačici programa ne koristi. Oni su potrebni prilikom kasnije izrade Android i iOS aplikacija kako bi korisnik mogao pronaći najbliža igrališta u svojoj okolini.

Model također sadrži sljedeće relacijske funkcije:

- *city* – dohvaća sve podatke o gradu kojem teren pripada
- *country* – dohvaća sve podatke o državi kojoj teren pripada
- *type* – dohvaća naziv vrste terena
- *schedule* – dohvaća sve zapisane termine od korisnika

Pogled prikazuje korisniku podatke koje zatraži iz baze podataka, a omogućeno je programom. Pogled *fields/show* prikazuje korisniku odabrani teren s vremenima u koje se može prijaviti na njih, koje vrste sporta je moguće igrati na tom terenu, te broj ljudi koji je prijavljen u određeno vrijeme i za koji sport (Slika 4.7).

```
@extends('layouts.app')

@section('content')

<body>
    <a href="{{route('home')}}">Back</a>
    <div>
        @if ($errors->any())
        @foreach ($errors->all() as $error)
        <div class="alert alert-danger" role="alert">{{$error}}</div>
        @endforeach
        @endif
        <h2>Court {{$field->address}}</h2>
    </div>
    <table class="table table-striped">
        <thead>

            <tr>
                <td>Court Hours </td>
                @foreach($types as $type)
                <td>Playing {{ $type->type}}</td>
                @endforeach
                <td>{{$now_time}}</td>
            </tr>
        </thead>
        <tbody>
            @for($i=6; $i <= 23; $i++) <tr>
                <td>Hours {{ $i }}</td>
                @foreach($types as $type)
                <td>{{count(App\Models\UserSchedule::where('sport_field_id', $field->id)->where('time', $i)-
>where('type_id', $type->id)->whereDay('created_at', now()->day)->get())}}
                </td>
                @endforeach
```

**Slika 4.7** Prikaz pogleda *fields/show* (1. dio)

```
                @foreach($user_times as $user_time)
                @if($user_time->time === $i)

                <td>You have signed to play in this time</td>
                </tr>
                @break
                @endif
                @endforeach
                @endfor
        </tbody>
    </table>

    <form role="form" action="{{ route('store_schedule', $field->id) }}" method="POST">
        {{ csrf_field() }}
        <div class="form-group">
            <label>Chose time to come to court</label>
            <select style="width: 200px" class="form-control" id="time" name="time">
                @for($i=6; $i <= 23; $i++) <option value="{{$i}}">{{$i}}</option>
                    @endfor
            </select>
        </div>
        <div>
            <label>Pick sport which you would like to play</label>
            <select style="width: 200px" class="form-control" name="type_id" id="type_id">
                @foreach($types as $type)
                <option value="{{$type->id}}">{{$type->type}}</option>
                @endforeach
            </select>
        </div>


        <button type="submit" class="btn btn-primary">Save</button>
    </form>

</body>
@endsection
```

**Slika 4.7** Prikaz pogleda *fields/show* (2. dio)

Unutar pogleda *fields/show* su korišteni HTML i CSS elementi za strukturiranje podataka. Korisniku se prikazuje lista podataka pomoću tablice, a podaci su dohvaćeni iz kontrolera pomoću ruta. Slanje korisnikovog odabranog vremena i sporta se radi pomoću *select* polja za odabir vremena i sportova. Kako bi prepoznao da se koristi PHP programski jezik potrebno ga je staviti u vitičaste zagrade.

Pogled *users/friends_list* prikazuje korisniku sve dodane prijatelje, prijatelje koji su mu poslali zahtjev za dodavanjem u listu prijatelja, kao i mogućnost slanja zahtjeva za prijateljstvom nekom drugom korisniku (Slika 4.8).

```
@extends('layouts.app')

@section('content')
<body>
    <div class="header">
        <a href="{{route('home')}}" class="link-btn back-btn">Back</a>
    </div>
    <div class="page-title">
        <h2>
        Friends List
        </h2>

    </div>

    <table class="table table-striped">
        <thead>
            <tr>
                <th scope="col">
                Friend Username
                </th>

            </tr>

        </thead>
        <tbody>
            @foreach($user_friends as $user)
            <tr>
                <td>{{ $user->friend->username}}</td>
                <td>
                    <a href="{{ route('delete_friend', [$user->id]) }}" class="link-btn btn-delete">Delete</a>
                </td>
            </tr>
            @endforeach

            @foreach($to_accept as $not_accepted)
                <tr>
                    <td>
                        <label for=""> User with username {{ $not_accepted['username']}} has sent you friend
request</label>
                    </td>
                    <td>
                    <td>
                        <a href="{{ route('accept_friend', [$not_accepted['username']]) }}" class="link-btn btn-
delete">Accept</a>
                    </td>
                    <td>
                        <a href="{{ route('delete_friend', [$not_accepted['to_accept_id']]) }}" class="link-btn btn-
delete">Refuse</a>
                    </td>
                    </td>

                </tr>

            @endforeach
        </tbody>
    </table>

    <div>
        <h5>Add Friend</h5>
        <form role="form" action="{{ route('add_friend') }}" method="POST" class="cities-create-form">
            {{ csrf_field() }}

            <div class="input-group w-25 mb-3">
                <input type="text" class="form-control" name="friend_username"  placeholder="Enter friend username"
aria-label="Recipient's username" aria-describedby="basic-addon2">
                <div class="input-group-append">
                <button class="btn btn-outline-secondary" type="submit">Button</button>
                </div>
            </div>
            @if ($errors->any())
            @foreach ($errors->all() as $error)
            <div class="error-msg">{{$error}}</div>
            @endforeach
            @endif
        </form>

</body>

@endsection
```

Slika 4.8 Prikaz pogleda *users/friends_list*

Za prikaz podataka korisniku unutar pogleda *users/friends_list* su korišteni HTML i CSS elementi. Korisniku se prikazuje lista prijatelja pomoću tablice. Svaki prijatelj se postavlja u novi redak te mu se ispisuje korisničko ime prijatelja kao i mogućnost njegovog brisanja iz liste

prijatelja. Za dodavanje novog prijatelja se koristi *input* element kako bi se unijelo ime korisnika kojeg se dodaje u listu prijatelja te se šalje upravljaču pomoću rute.

Pogled *fields/create* prikazuje podatke potrebne za kreiranje novog terena (Slika 4.9). Ovu mogućnost ima samo korisnik s administratorskim ovlastima.

```
@extends('layouts.app')

@section('content')

<body>
    <div class="header">
        <a href="{{route('home')}}" class="link-btn back-btn">Back</a>
    </div>
    <div>
    <h3>Add Court</h3>
    </div>

    <form role="form" action="{{ route('create_field') }}" method="POST" class="cities-create-form">
        {{ csrf_field() }}
        <div class="form-group w-25">
            <label class="form-group-items">Name & Address</label>
            <input class="form-group-items form-control" name="address" placeholder="Address">
        </div>
        <div class="form-group w-25">
            <label class="form-group-items">Field number</label>
            <input class="form-group-items form-control" name="field_number" placeholder="field_number">
        </div>
        <div class="form-group w-25">
            <label class="form-group-items">Longitude</label>
            <input class="form-group-items form-control" name="longitude" placeholder="Longitude">
        </div>
        <div class="form-group w-25">
            <label class="form-group-items">Latitude</label>
            <input class="form-group-items form-control" name="latitude" placeholder="latitude">
        </div>
        <div class="form-group w-25">
            <label class="form-group-items">Number of courts</label>
            <input class="form-group-items form-control" name="number_of_courts" placeholder="Courts Nb.">
        </div>


        <div class="form-group">
            <label>Select Types:</label>
            <select id="types" name="types[]" multiple class="form-control">
                @foreach($types as $type)
                <option value="{{$type->id}}">{{$type->type}}</option>
                @endforeach
            </select>
        </div>


        <div class="form-group">
            <label class="form-group-items">Country</label>
            <select style="width: 200px" class="form-group-items form-control" id="country_id" name="country_id">
                <option value="N/A">--SELECT--</option>
                @foreach($countries as $country)
                <option value="{{$country->id}}">{{$country->name}}</option>
                @endforeach
            </select>
        </div>
        <div class="form-group">
            <label class="form-group-items">City</label>
            <select style="width: 200px" class="form-group-items form-control" id="city_id" name="city_id">
                <option value="N/A">--SELECT--</option>
            </select>
        </div>
        <div class="button-container">
            <button type="submit" class="btn btn-primary">Save</button>
        </div>
        @if ($errors->any())
        @foreach ($errors->all() as $error)
        <div class="error-msg">{{$error}}</div>
        @endforeach
        @endif
    </form>
</body>
```

**Slika 4.9** Prikaz pogleda *fields/create* (1. dio)

```
</html>
<script>
var selectElement = document.getElementById('country_id');
var citiesList = [];
selectElement.addEventListener('change', (e) => {

    var id = document.getElementById('country_id').value;
    var stateUrl = "{{ url('cities/getbycountry/')}}" + `/${id}`;

    console.log(stateUrl);
    async function getCities() {
        try {
            var response = await fetch(stateUrl, {
                method: 'GET'
            });
            var responseJson = await response.json();
            citiesList = responseJson;
            var selectCities = document.getElementById('city_id');
            selectCities.textContent = '';
            citiesList.forEach(element => {
                var option = document.createElement('option');
                option.value = element.id;
                option.innerHTML = element.name;
                selectCities.appendChild(option);
            });
        } catch (error) {
            console.log(error)
        }

    }
    getCities();

})
</script>
<script>
$(document).ready(function() {
    $('#types').multiselect({
        nonSelectedText: 'Select types',
        enableFiltering: true,
        enableCaseInsensitiveFiltering: true,
        buttonWidth: '400px'
    });
    $('#types_form').on('submit', function(event) {
        event.preventDefault();
        var form_data = $(this).serialize();
        $.ajaxSetup({
            headers: {
                'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
            }
        });
        $.ajax({
            url: "{{ url('create_field') }}",
            method: "POST",
            data: form_data,
            success: function(data) {
                $('#types option:selected').each(function() {
                    $(this).prop('selected', false);
                });
                $('#types').multiselect('refresh');
                alert(data['success']);
            }
        });
    });
});
</script>
@endsection
```

**Slika 4.9** Prikaz pogleda *fields/create* (2. dio)

Pogled za izradu terena *fields/create* koristi HTML i CSS elemente. Također sadrži pet polja za popunjavanje podataka, dva polja za odabir, s jednim mogućim odabirom, za odabir pripadnosti gradu i državi, te jedno polje s mogućnošću odabira više ponuđenih vrsti terena. Kako bi se odabrali gradovi koji pripadaju samo odabranoj državi potrebno je ispisati samo pripadajuće gradove odabranoj državi. PHP programski jezik se ne može izvršavati unutar internet preglednika te je za dinamičan prikaz gradova je potrebno koristiti JavaScript programski jezik kako bi se na odabiru države popunila lista gradova. Pošto se na nekim terenima može igrati više

sportova potrebno je postaviti listu s višestrukim odabirom sportova. Višestruki odabir je napravljen uz pomoć AJAX i jQuery-a.

Upravitelj prihvaća podatke koji su mu poslani s pogleda ili modela te ih obrađuje i šalje natrag u pogled ili model. Prema slici 4.10 pomoću upravitelja *UserFieldsController* obrađujemo podatke za korisnikove omiljene terene.

```php
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreUserFieldRequest;
use App\Models\City;
use App\Models\FieldTypes;
use App\Models\SportField;
use App\Models\UserFields;

class UserFieldsController extends Controller
{
    public function index()
    {
        $sport_fields = UserFields::with('field')->where('user_id', auth()->user()->id)->get();
        $cities = City::where('country_id', auth()->user()->country_id)->get();

        return view('userfields/userfields', compact(['sport_fields', 'cities']));
    }

    public function store(StoreUserFieldRequest $request)
    {
        UserFields::create([
            'user_id' => auth()->user()->id,
            'field_id' => $request['field_id'],
        ]);
        return redirect()->route('user_fields');
    }

    public function destroy($id)
    {
        UserFields::where('user_id', auth()->user()->id)->where('field_id', $id)->delete();
        return redirect()->route('user_fields');
    }

    public function getFieldsByCity($id)
    {
        return SportField::where('city_id', $id)->get();
    }

    public function getTypeByField($id)
    {
        return FieldTypes::with('type')->where('field_id', $id)->get()->pluck('type');
    }
}
```

**Slika 4.10** Prikaz upravitelja *UserFieldsControllera*

Unutar upravitelja *UserFieldsController* postoji sljedećih pet funkcija:

- *index* – funkcija koja dohvaća sve omiljene korisnikove terene i sve gradove te ih šalje na pogled *userfields* kako bi se prikazali svi željeni podaci
- *store* – funkcija kojom pohranjujemo korisnikova omiljena igrališta. Iz zahtjeva s pogleda dobivaju se podaci za spremanje koji se zatim šalju modelu *UserFields* kako bi bili pohranjeni u bazu

- *destroy* – funkcija kojom se brišu željeni podaci iz baze. Prima *id* terena koje korisnik želi maknuti sa svoje liste
- *getFieldsByCity* – funkcija koja dohvaća sve terene koji se nalaze u određenom gradu. Prima iz zahtjeva *id* grada po kojem se pretražuju tereni
- *getTypeByField* – funkcija koja dohvaća sve vrste sportova kojima se moguće baviti na terenu

Pomoću upravitelja *UserFriendsController* obrađuju se podaci za korisnikove dodane prijatelje (Slika 4.11).

```php
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreFriendRequest;
use App\Models\User;
use App\Models\UserFriends;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;

class UserFriendsController extends Controller
{
    public function listUserFriends(Request $request)
    {
    $user_friends = UserFriends::where('user_id', auth()->user()->id)->get();
    $to_accept = array();
    $sent_request = UserFriends::where('friend_id', auth()->user()->id)->where('has_accepted', false)->get();
    foreach($sent_request as $sent){
        $not_accepted_friend = UserFriends::where('user_id', auth()->user()->id)->where('friend_id', $sent-
>user_id)->first();
        if(!$not_accepted_friend){
            $to_accept[] = [
                'username' => $sent->user->username,
                'to_accept_id' => $sent->id
            ];
        }
    }

    return view('users/friends_list', compact(['user_friends', 'to_accept']));
    }

    public function store(StoreFriendRequest $request)
    {
        $friend = User::where('username', $request['friend_username'])->first();
        $friend_id = $friend->id;
        if(UserFriends::where('user_id', auth()->user()->id)->where('friend_id', $friend_id)->first()){
            return Redirect::back()->withErrors('Friend already added');
        }
        UserFriends::create([
            'user_id' => auth()->user()->id,
            'friend_id' => $friend_id,
            'has_accepted' => false
        ]);

        return redirect()->route('friends_list');
    }
```

**Slika 4.11** Prikaz upravitelja *UserFriendsController* (1. dio)

```
    public function acceptFriend($username)
    {
        $user = User::where('username', $username)->first();
        $accepted_friend = UserFriends::where('user_id', $user->id)->where('friend_id', auth()->user()->id);
        $accepted_friend->update([
            'has_accepted' => 1
        ]);
        UserFriends::create([
            'user_id' => auth()->user()->id,
            'friend_id' => $user->id,
            'has_accepted' => 1
        ]);

        return redirect()->route('friends_list')->with('success', 'Friend accepted successfully');
    }

    public function destroy($id)
    {
        $userfriend = UserFriends::where('id', $id)->first();
        UserFriends::where('user_id', $userfriend->user_id)->where('friend_id', $userfriend->friend_id)->delete();
        UserFriends::where('user_id', $userfriend->friend_id)->where('friend_id', $userfriend->user_id)->delete();
        return redirect()->route('friends_list')->with('success', 'Deleted successfully');
    }
}
```

**Slika 4.11** Prikaz upravitelja *UserFriendsController* (2. dio)

Unutar upravitelja postoje sljedeće tri funkcije:

- *listUserFriends* – funkcija kojom se dohvaćaju svi korisnikovi prijatelji
- *store* – funkcija kojom se sprema novog prijatelja kojeg je korisnik odabrao
- *acceptFriend* – funkcija kojom se prihvaća poslano prijateljstvo
- *destroy* – funkcija kojom se briše prethodno dodanog prijatelja iz liste najdražih prijatelja ili odbija poslano prijateljstvo

### 4.2.4. Zahtjevi

Zahtjevi (engl. *requests*) služe za obradu podatka nakon što su poslani s pogleda i prije nego što su došli do upravitelja. Služe kako bi se što više smanjilo opterećenje na bazu i poslužitelje, te isto tako smanjila mogućnost grešaka u bazi. U zahtjevima je definirano što korisnik može poslati, a da ne dođe do daljnje greške u programu. Definira se polje koje može biti broj, tekstualni ili *boolean* tip.

Zahtjev *StoreFriendRequest* služi kako bi se poslano polje *friend_username* provjerilo prije nego što dođe do upravitelja (Slika 4.12).

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreFriendRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'friend_username' => 'string|exists:users,username'
        ];
    }
}
```

**Slika 4.12** Prikaz zahtjeva *StoreFriendsRequesta*

Prvi zahtjev za poslano polje *friend_username* je provjera je li ono znakovnog tipa, a drugi zahtjev je provjera da poslana vrijednost postoji u bazi podataka. Tek nakon što su zadovoljena oba kriterija zahtjev dolazi do upravitelja.

### 4.2.5. Rute

Rute (engl. *routes*) se izrađuju kao most između pregleda i upravitelja. Rutama se definira na kojem pregledu ćemo imati mogućnost pozivanja funkcija iz upravitelja. Kako bi ruta bila funkcionalna prvo se definira metoda rute, ona može biti:

- *GET* – dohvaćanje podataka
- *POST* – slanje podataka
- *DELETE* – brisanje podataka

- *PATCH* – ažuriranje podataka

Nakon toga definira se mjesto na kojem se nalazi željeni pregled iz kojeg će dobivati ili slati podatke. Potom se određuje na kojeg se upravitelja odnosi ruta i funkciju s kojom se želi upravljati na toj ruti. Zadnja stavka nije obavezna i njome se imenuje naziv rute kako bi joj se lakše moglo pristupiti.

Prema slici 4.13, unutar web ruta se nalazi 28 ruta koje se koriste između pregleda i upravitelja.

```php
<?php

use App\Http\Controllers\CitiesController;
use App\Http\Controllers\CountriesController;
use App\Http\Controllers\SportFieldsController;
use App\Http\Controllers\UserFieldsController;
use App\Http\Controllers\UserFriendsController;
use App\Http\Controllers\UserScheduleController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return view('welcome');
});

Route::view('home', 'home')->name('home');
Route::get('/users/friends_list', [UserFriendsController::class, 'listUserFriends'])->name('friends_list');
Route::post('/users/friends_list', [UserFriendsController::class, 'store'])->name('add_friend');
Route::get('/users/delete/{id}', [UserFriendsController::class, 'destroy'])->name('delete_friend');
Route::get('/users/accept_friend/{username}', [UserFriendsController::class, 'acceptFriend'])-
>name('accept_friend');

Route::post('/users/fields_list', [UserFieldsController::class, 'store'])->name('add_field');

Route::get('/countries/list', [CountriesController::class, 'index'])->name('countries_list');
Route::get('/countries/create', [CountriesController::class, 'create'])->name('create_country');
Route::post('/countries/create', [CountriesController::class, 'store'])->name('store_country');
Route::delete('/countries/delete/{id}', [CountriesController::class, 'destroy'])->name('delete_country');

Route::get('/cities/list', [CitiesController::class, 'index'])->name('cities_list');
Route::get('/cities/getbycountry/{id}', [CitiesController::class, 'getCitiesByCountry'])->name('country_city');
Route::get('/cities/create', [CitiesController::class, 'create'])->name('create_city');
Route::post('/cities/create', [CitiesController::class, 'store'])->name('store_city');
Route::delete('/cities/delete/{id}', [CitiesController::class, 'destroy'])->name('delete_city');

Route::get('/fields/list', [SportFieldsController::class, 'index'])->name('fields_list');
Route::get('/fields/create', [SportFieldsController::class, 'create'])->name('create_field');
Route::post('/fields/create', [SportFieldsController::class, 'store'])->name('store_field');
Route::delete('/fields/delete/{id}', [SportFieldsController::class, 'destroy'])->name('delete_field');
Route::get('/fields/show/{id}', [SportFieldsController::class, 'show'])->name('show_field');

Route::get('/users/userfields', [UserFieldsController::class, 'index'])->name('user_fields');
Route::post('/users/userfields', [UserFieldsController::class, 'store'])->name('add_user_field');
Route::get('/users/userfield/delete/{id}', [UserFieldsController::class, 'destroy'])->name('delete_user_field');
Route::get('/users/getfieldbycity/{id}', [UserFieldsController::class, 'getFieldsByCity'])->name('city_fields');
Route::get('/users/gettypebyfield/{id}', [UserFieldsController::class, 'getTypeByField'])->name('field_types');

Route::post('/users/schedule/{id}', [UserScheduleController::class, 'store'])->name('store_schedule');
```

**Slika 4.13** Prikaz web ruta

### 4.2.6. E-poruka

Prilikom prijave korisnika na omiljeni teren, aktivira se slanje e-poruke (engl. *electronic mail* ili kratica *email*) svim korisnicima koji su ga dodali u listu prijatelja. Ovim načinom se javlja korisniku da će mu u određenom trenutku prijatelj biti na terenu i pozvati ga da mu se pridruži. Klasa *SportActivityMail* služi kako bi se iz upravitelja dohvatili potrebni podaci i poslali ih i pogled pomoću kojeg se uređuje izgled e-poruke koju primaju korisnici (Slika 4.14).

```php
use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class SportActivityMail extends Mailable
{
    use Queueable, SerializesModels;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct($username, $field_address)
    {
        $this->username = $username;
        $this->field_address = $field_address;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->from('luka.kovacevic23@gmail.com')
            ->markdown('emails.message')
            ->with([
                'username' => $this->username,
                'address' => $this->field_address
            ]);
    }
}
```

**Slika 4.14** Prikaz klase za obradu e-pošte

## 4.3. Poslužitelji

Prema [17], programski jezik PHP, zajedno sa okvirom Laravel, može obraditi oko 600 zahtjeva i 4000 odziva u sekundi. Za kompliciranije funkcije se taj broj smanjuje što za programsko rješenje rada ne predstavlja problem jer se koriste jednostavne funkcije.

### 4.3.1. Projekcija opterećenja

Prema [18], u Hrvatskoj se 38% ljudi bavi nekom tjelesnom aktivnošću, a tek 21% ljudi nekoliko puta tjedno, što je 840 tisuća ljudi od projiciranih četiri milijuna stanovnika. Prilikom ovog izračuna se odabire 21% ljudi koji se tjedno bave nekom tjelesnom aktivnošću. Prema [19], ako se uspoređuje statistika bavljenja fizičkom aktivnošću u Republici Hrvatskoj sa statistikom bavljenja fizičkom aktivnošću u Sjedinjenim Američkim Državama, u nedostatku analize bavljenja timskim sportom u Republici Hrvatskoj, onda se dobiva da tek 10% od 21% ljudi koji se bave tjelesnom aktivnošću se bavi nekim timskim sportom. Jednadžba 4-1 dovodi do broja od 84 000 ljudi. S obzirom na to da je aplikacija spremna obraditi 600 zahtjeva u sekundi, što je 36000 zahtjeva u minuti, može se zaključiti da neće biti nikakvih problema sa strane korisnika za korištenje aplikacije.

$$4000000 * (0{,}21 * 0{,}1) = 84000 \tag{2-1}$$

### 4.3.2. Poslužiteljsko rješenje i skaliranje

Za poslužiteljsko (engl. *server*) rješenje se predlaže korištenje usluga u oblaku računala (engl. *Cloud Computing Services*) jednog od poslužitelja Microsoftov Azure ili Amazon Web Services. Kako aplikacija ne bi koristila previše ili premalo poslužiteljske snage, potrebno ju je skalirati prema njenim potrebama na strani poslužitelja. Servisi u oblaku računala nude brza i jednostavna rješenja povećanjem ili smanjenjem virtualnih strojeva (engl. *Virtual Machines*) ako dođe do povećanog opterećenja ili smanjenog opterećenja na poslužitelja. Za parametre koji definiraju potrebu za dodatnim skaliranjem usluga u oblaku, moguće je pratiti odziv zahtjeva prema korisničkom web klijentu, promjenu broja zahtjeva u sekundi prema poslužitelju, pojavu prvih grešaka ili kratkotrajne nedostupnosti odgovora poslužitelja i slične parametre i prema tome podešavati poslužiteljsku infrastrukturu i usluge.

Skalirati je moguće na više načina. Prema [20], prvi pristup je da se prema detektiranim prethodno navedenim parametrima porasta ili smanjenja skalira snaga virtualnog stroja. Pri tome se usluga kratkotrajno obustavlja ako ne postoji redundantni virtualni stroj u oblaku koji bi u

potpunosti preuzeo funkciju poslužitelja, dok prethodna virtualna instanca ne završi skaliranje, zatim se skalira redundantna a glavnu ulogu preuzima skalirana. Ovo nije najbolja metoda zbog vršnih opterećenja i potencijalnih nedostupnosti ili potpune nedostupnosti. Ako se koristi ovakva metoda potrebno je pravovremeno uočiti promjene kako sustav ne bi završio potpuno nedostupan u vremenu koje je potrebno da se skaliranje čvorova završi.

Drugi način skaliranja je podizanje dodatnih redundantnih virtualnih strojeva koji bi preko upravitelja opterećenjem upravljali dolaznim zahtjevima od web klijenata prema poslužitelju. Time se ravnomjerno raspoređuje opterećenje svakog od poslužitelja, te nakon registracije novog čvora na upravitelju opterećenjem, rasterećuju postojeći poslužitelji. Nakon smanjenja vršnog opterećenja prema promatranim parametrima, višak redundantnih čvorova je moguće ukloniti po potrebi. Dodatan značaj ovakvog načina je da se vrijeme potrebno za pokretanje i usklađivanje novog čvora, odnosno virtualnog stroja ne osjeti u samom radu.

Treći način, koji se ne smatra klasičnim skaliranjem, nego je standard u web razvoju i razvojnim operacijama upravljanja infrastrukturom je postavljanje predmemorijskog (engl. *cache*) sustava i poslužitelja. Svaki moderni web sustav sadrži neku vrstu predmemorije, no unutar oblaka računala moguće je dodati zasebne instance usluga koje služe isključivo predmemoriranju podataka te se same virtualne strojeve štiti od nepotrebnih i stalnih dohvaćanja statičnih podataka. To je posebno bitno ako je sustav loše složen i predmemorija na samom web programu nije dobro implementirana, te za svaku informaciju je potrebno pozivati metode dohvaćanja iz baze.

Četvrti način je raščlanjivanje dijelova sustava u mikrousluge (engl. *microservices*) gdje manje dijelove koda izvode posebni čvorovi koji su specifične namjene, odnosno koriste se za dio funkcionalnosti ukupnog sustava. Time je moguće postići veću učinkovitost i smanjenje cijene korištenjem manjih instanci, rasteretiti glavne strojeve i usluge, te ih specifično koristiti, kao primjerice za prikupljanje podataka putem Internet stvari (engl. *Internet of Things*), dok korisnički poslužiteljski dio je u potpunosti odvojen od zahtjeva za dostavom mjerenja. U slučaju zadatka rada metoda mikrousluga bi bila nepotrebna iz razloga sažetosti samih funkcionalnosti, a dovelo bi do veće kompleksnosti sustava. Svaki od mikrouslužnih čvorova može imati istu strukturu i skalirati se slično kao i cjeloviti glavni sustav (opisno engl. *monolith*) opisan prethodno.

Peti način je korištenje funkcija kao servisa (engl. *Function as a Service*), gdje se pojedine funkcionalnosti izvršavaju kao specifične funkcije koje se mogu ugraditi direktno u sustavu

usluga pružatelja usluga u oblaku kroz Lambda funkcije. Stoga glavni sustav poziva određene funkcije isključivo po potrebi i na skalabilnoj infrastrukturi pružatelja *cloud* usluga, gdje niti programer, niti krajnji korisnik ne moraju direktno brinuti o dostupnosti i skaliranju, nego je ona određena maksimalnim kvotama dozvoljenim u paketima pružatelja usluga i gotovo u potpunosti dovoljna svim zahtjevima modernih web aplikacija. U današnje vrijeme je najpovoljniji model skaliranja hibridni model i ovisi o samom poslovnom modelu i financijskoj računici, gdje se pojedine funkcionalnosti drže u Lambdama. Dio infrastrukture je složena kao mikrouslužna struktura, a većinom zbog jednostavnosti održavanja i dalje glavni poslužitelji budu kao klasični virtualni strojevi koje programeri mogu samostalno postaviti i upravljati, te tako kontrolirati samostalno skaliranje nekim od prva dva modela glavni dio sustava.

Za ovu aplikaciju najprikladniji je drugi način skaliranja poslužitelja kako bi korisnikovo iskustvo bilo neometano. Također, rasterećuje se poslužitelj i ubrzano je vrijeme potrebno za pokretanje i usklađivanje novog čvora.

## 4.4. Daljnji razvoj aplikacije

Aplikacije je napravljena pomoću PHP programskog jezika, Laravel okruženja i njegovih komponenti za lagan prikaz podataka preko model-pogled-upravitelj sustava. Takav sustav nije najbolje dugoročno rješenje za aplikaciju te je unaprijed pripremljena za prijelaz na adekvatnije sustave. Trenutno web rješenje i prikaz može biti  pretvoreno u administracijsko sučelje za upravljanjem aplikacijom.

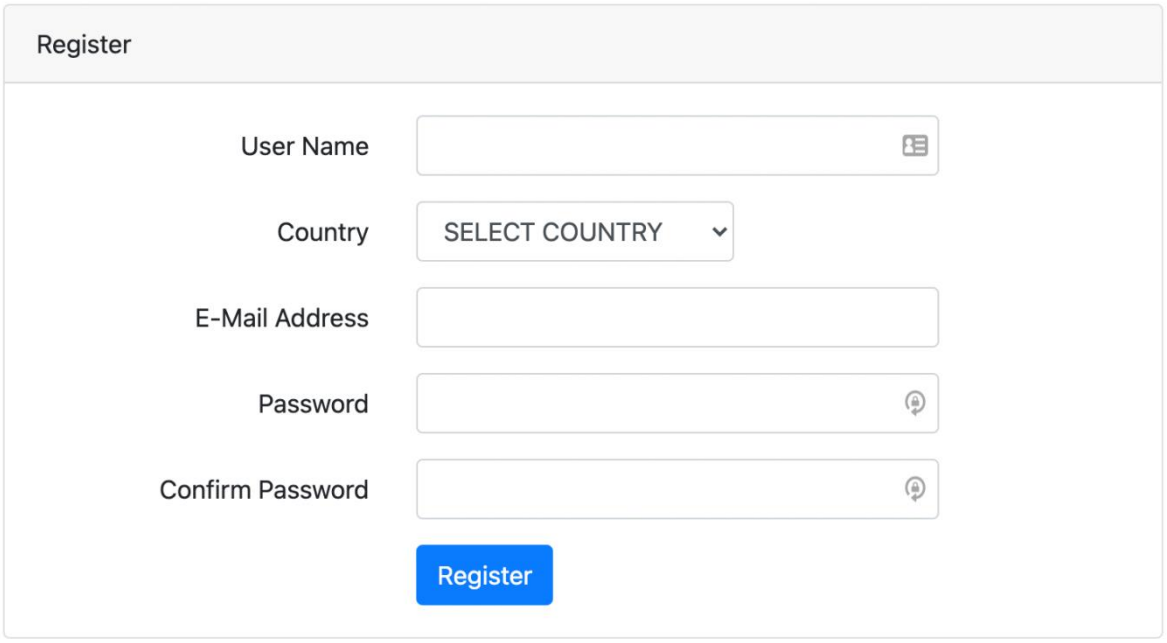
Kako bi druge aplikacije mogle pristupiti podacima potrebno je napraviti API sučelje koje će one koristiti. API sučelja se koriste kako bi se drugim aplikacijama ili programskim sučeljima dopustilo pristup trenutnom sustavu i ograničila količinu podataka koje mogu dobiti od aplikacije. Korisničko sučelje aplikacije, koja će se prikazivati na web pregledniku, može se izraditi u JavaScriptu i ReactJS programskim jezicima kao zasebna web komponenta aplikacije. Također, za lakši pristup moguće je napraviti Android i iOS aplikacije kako bi korisnik preko mobitela mogao pristupiti bilo gdje. Aplikaciju je moguće dalje nadograđivati funkcionalno i u smjeru dodavanja jednostavnog sklopovlja koje će komunicirati s aplikacijom, te upravljati rasvjetom terena, otključavanjem prostorija, upravljati grijanjem i slično. Aplikaciju je moguće proširivati s dodatnim privatnim terenima i mogućnošću plaćene rezervacije.

# 5. NAČIN RADA I ISPITIVANJE APLIKACIJE

U ovom poglavlju je prikazano korištenje aplikacije od strane korisnika i korisnika s administratorskim pravima, ispitivanje rada aplikacije, kao i statička analiza koda.

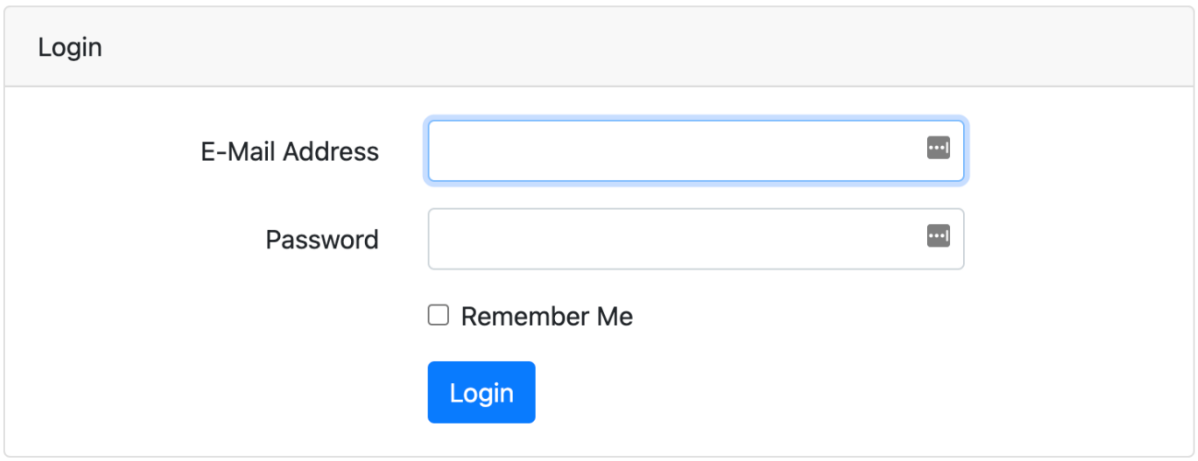
## 5.1. Korištenje web sučelja aplikacije

Otvaranjem web aplikacije se korisniku otvara početni prikaz gdje se može prijaviti u aplikaciju. Prema slici 5.1, na prikazu za prijavljivanje mu se prikazuje unos elektroničke pošte i lozinke. Ako korisnik nema napravljen korisnički račun, odabirom „registriraj“ vodi ga se na prikaz za registraciju gdje može ispuniti svoje podatke. Za registraciju na web aplikaciju su potrebni ime, prezime, korisničko ime, lozinka i država (Slika 5.2).



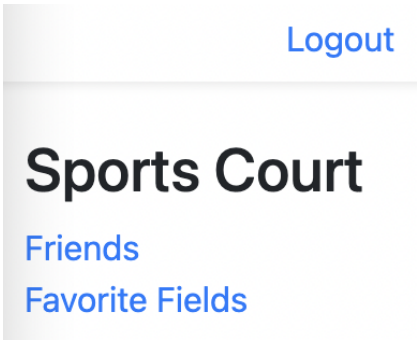

**Slika 5.1** Prikaz pregleda za registraciju u aplikaciju

Korisničko ime i elektronička pošta su jedinstveni u bazi podataka i ako već postoji, korisnik će biti obaviješten da proba s drugim podacima. Državu odabire kako bi suzili broj terena koji mu se može prikazati. Za prijavu korisnik mora unijeti e-poštu i lozinku (Slika 5.2).



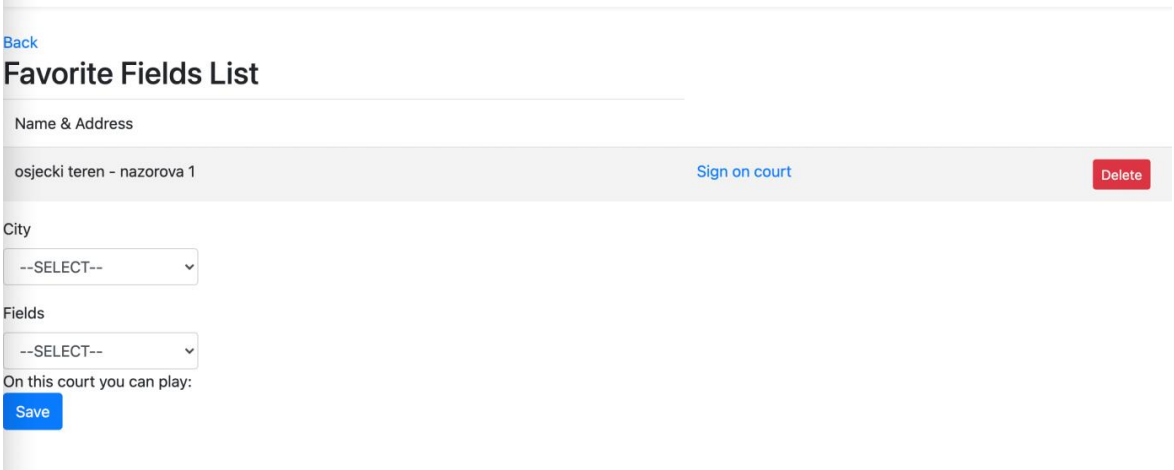

**Slika 5.2** Prikaz pregleda za prijavljivanje u aplikaciju

Nakon prijave se korisnika vodi na glavni prikaz gdje može odabrati daljnje opcije za korištenje aplikacije (Slika 5.3).



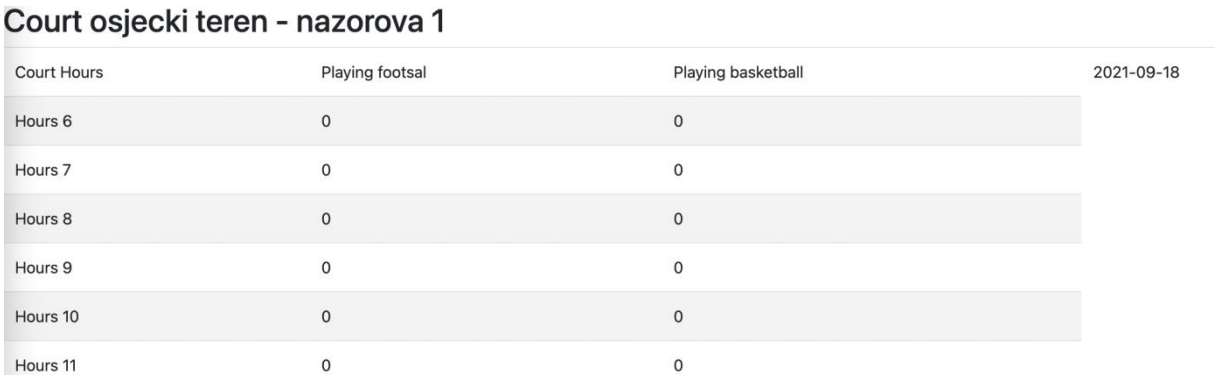

**Slika 5.3** Prikaz početnog pregleda izbornika za korisnika

Prema slici 5.4, odabirom opcije *Favorite Fields* vodi ga se na prikaz korisnikovih najdražih igrališta. Ako nema dodanih igrališta može odabirom grada vidjeti prikaz svih igrališta te s padajuće liste odabrati igralište koje želi dodati.




**Slika 5.4** Prikaz pregleda omiljenih terena

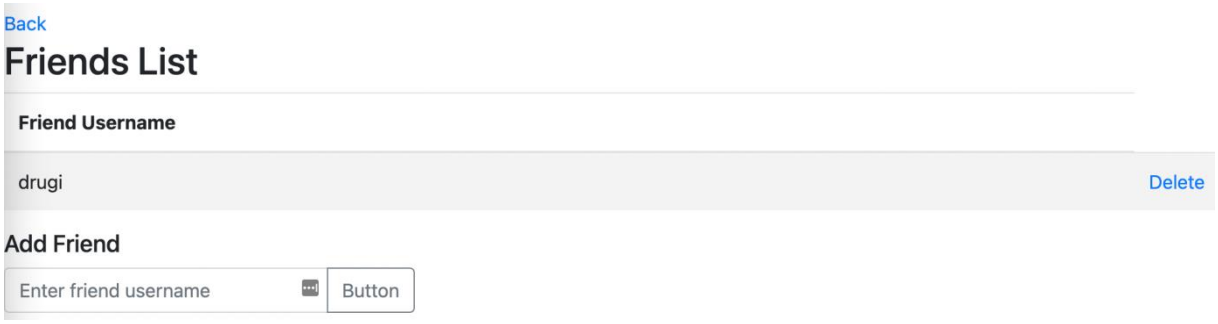
Pored svakog igrališta korisnik ima gumb koji ga vodi na prikaz određenog igrališta. Na prikazu igrališta korisnik može vidjeti koliko je ljudi prijavljeno na određeno igralište u određenom trenutku (Slika 5.5).




**Slika 5.5** Djelomični prikaz vremena i sportova na terenu

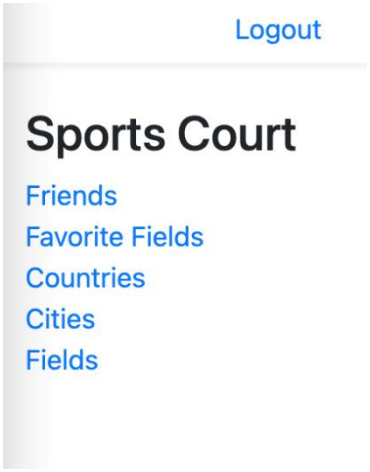
Prema slici 5.4, na dnu prikaza igrališta korisnik može odabrati vrijeme u koje se želi prijaviti na igralište. Nakon što odradi prijavu na prikazu terena, uz odabrani sat prijave, prikazuje mu se podsjetnik kada se prijavio.

Odabirom opcije *Friends* korisnika se vodi na prikaz njegovih prijatelja (Slika 5.6). Na ovom prikazu korisnik može provjeriti koji su njegovi dodani prijatelji.




**Slika 5.6** Prikaz liste prijatelja

Dodani prijatelji služe kako bi korisniku olakšali informiranje kada se njegov prijatelj prijavi na određeni teren. Kada se korisnik prijavi na teren svi korisnici koji su ga dodali na listu prijatelja će dobiti obavijest da je korisnik prijavio dolazak na određeni terena putem e-pošte. Prema slici 5.7, prilikom prijave korisnika s administratorskim pravima početno sučelje ima dodatne opcije za izradu država, gradova i terena, za razliku od normalnog korisničkog sučelja.




**Slika 5.7** Prikaz početnog sučelja administratorskog korisnika

U lijevom izborniku nalaze se:

*Favorite Fields* – tereni koje je korisnik dodao radi lakšeg snalaženja i kasniju prijavu na njih

*Friends* – igrači koje korisnik može dodati i dobiti obavijest kada se prijave u određeni termin na nekom terenu kako bi im se mogao pridružiti

*Countries* - polje vidljivo samo za administrator korisnika. Ovdje administrator može vidjeti sve države i ima mogućnost dodavanja novih država

*Cities* – polje vidljivo jedino za administrator korisnika. Ovdje administrator može vidjeti sve gradove i ima mogućnost dodavanja novih gradova

*Fields* – polje vidljivo samo za administrator korisnika. Ovdje administrator ima izlistanje svih polja i mogućnost kreiranja novog polja

## 5.2. Ispitivanje aplikacije

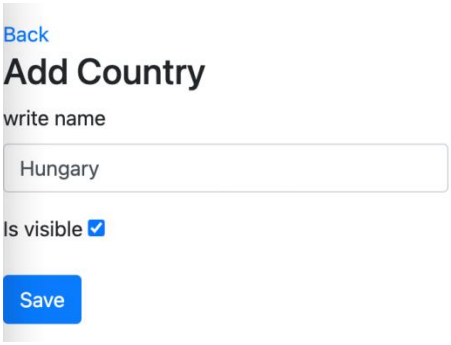
Prilikom ispitivanja aplikacije odabrano je ispitivanje glavnih značajki aplikacije. Pravljenje država, gradova i terena je odabrano kako bi se mogli izraditi novi tereni na koje će se korisnik dalje moći prijavljivati. Dodavanje prijatelja na listu prijatelja te dobivanje obavijesti kada se korisnikov prijatelj prijavio na teren. Dodavanjem terena na listu omiljenih terena provjerava se kako korisnik ne bi imao nikakvih poteškoća prilikom dodavanja terena. Izmjene na pregledu terena ispitane su kako bi se potvrdilo da se korisnik ispravno prijavio i ako se drugi korisnik prijavi na teren je li mu ispravno prikazan broj korisnika prijavljenih na teren.

### 5.2.1. Ispitivanje izrade država, gradova i terena

Izrada države, gradova i terena je ispitana s administrator korisnikom koji jedini ima prava pristupa i s korisnikom koji nema upravljačke ovlasti.


**Slika 5.8** Početno stanje na listi država

Prema slici 5.9, korisnik upravitelj može pristupiti stranici za izradu država i uspješno smo odradili izradu nove države (Slika 5.10).



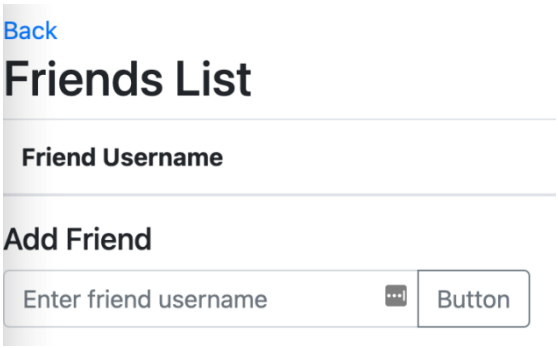

**Slika 5.9** Prikaz kreiranja Države


**Slika 5.10** Prikaz dodane države

Korisnik upravitelj može pristupiti stranici za izradu gradova i uspješno je s liste odabrao novo napravljenu državu, te je napravio novi grad koji će pripadati toj državi. Korisnik upravitelj može pristupiti stranici za izradu terena te je uspješno dodao novi teren novo napravljenom gradu. Običan korisnik na pregledu ne vidi u izborniku  polja koja vode do izrade država, gradova i terena (Slika 5.3). Prilikom pokušaja da otvori stranicu, na koja nema dopuštenja, korisnika se preusmjerava na početnu stranicu

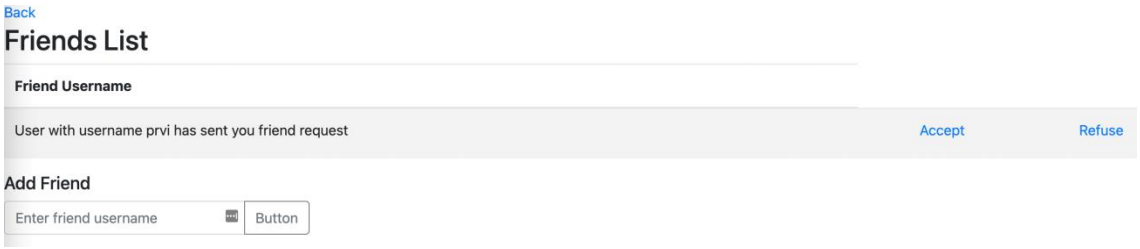
### 5.2.2. Ispitivanje dodavanja drugog korisnika u listu prijatelja

Ispitivanje dodavanja korisnika na listu prijatelja izvršeno je tako da su registrirana dva nova korisnika. „Prvi“ korisnik nije imao ni jednog prijatelja na listi prijatelja (Slika 5.11).



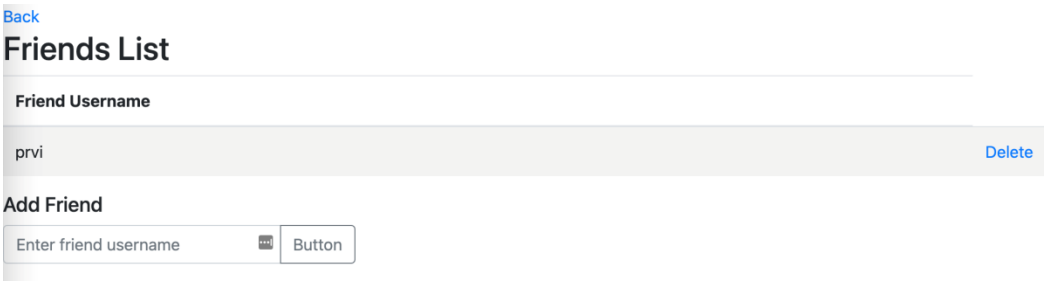

**Slika 5.11** Početno stanje liste prijatelja „prvi“ korisnika

„Prvi“ korisnik je poslao zahtjev za prijateljstvo „drugi“ korisniku. Prema slici 5.12, „drugi“ korisnik dobiva zahtjev za prijateljstvo.




**Slika 5.12** Prikaz zahtjev za prijateljstvo kod „drugi“ korisnik

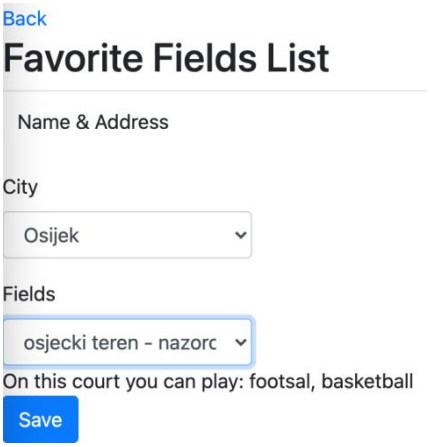
„Drugi“ korisnik prihvaća zahtjev te mu se dodaje korisnik „prvi“ u listu prijatelja (Slika 5.13).




**Slika 5.13** Prikaz dodanog prijateljstva

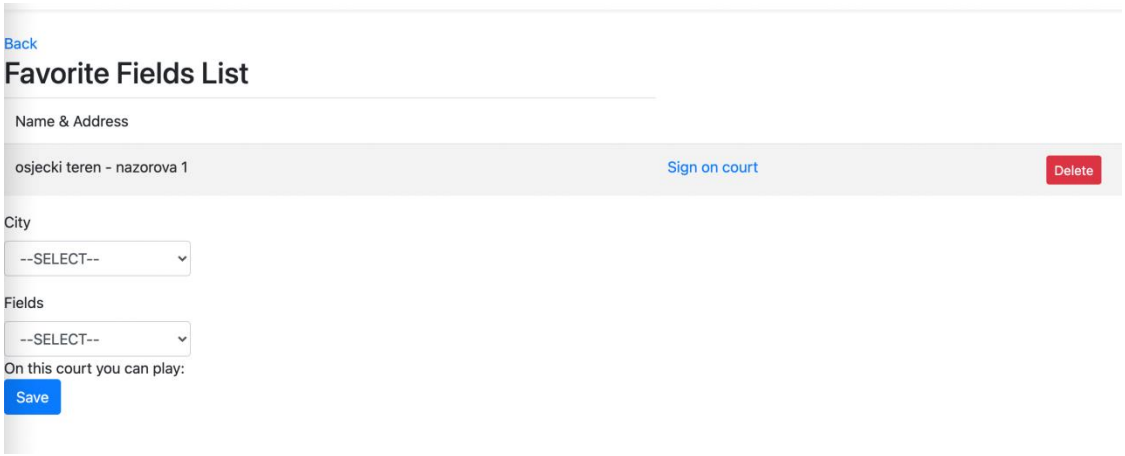
### 5.2.3. Ispitivanje dodavanja terena u listu omiljenih terena

Kako bi se utvrdila ispravnost dodavanja terena na listu omiljenih terena korisnik „prvi“ se prijavio u aplikaciju i odabrao prikaz liste omiljenih terena. Prema slici 5.14, iz izbornika je odabrao željeni grad i teren, nakon čega mu je prikazano kojim sportovima se moguće baviti na tom terenu.




**Slika 5.14** Prikaz odabira „prvi“ korisnika

Nakon odabira je spremio željeni teren i prikazan mu je na listi omiljenih terena (Slika 5.15).

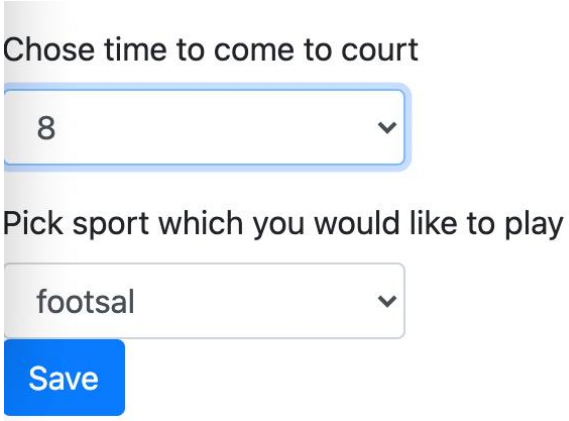



**Slika 5.15** Prikaz dodanog terena „prvi“ korisnika

Ovo ispitivanje je uspješno napravljeno i prikazano korisniku.

### 5.2.4. Ispitivanje izmjena na pregledu terena

Kako bi se ispitale izmjene na pregledu terena potrebno je bilo provesti dvije akcije. Prema slici 5.16, prvo se korisnik „prvi“ prijavljuje na teren te mu se ispravno moralo prikazati u koje vrijeme se prijavio (Slika 5.17).




**Slika 5.16** Vrijeme prijavljivanja „prvi“ korisnika

Back

Court osjecki teren - nazorova 1

| Court Hours | Playing footsal | Playing basketball | 2021-09-18 |
|---|---|---|---|
| Hours 6 | 0 | 0 | |
| Hours 7 | 0 | 0 | |
| Hours 8 | 1 | 0 | You have signed to play in this time |
| Hours 9 | 0 | 0 | |
| Hours 10 | 0 | 0 | |

**Slika 5.17** Prikaz prijave na teren „prvi korisnika“

Nakon toga je „drugi“ korisnik prijavljen i prijavljen je njegov dolazak u određeno vrijeme na isti teren. Kada je potvrđena uspješna prijava na teren s „drugi “ korisnikom ukupan broj igrača na terenu u osam sati za mali nogomet je porastao na dva igrača (Slika 5.18).

Back

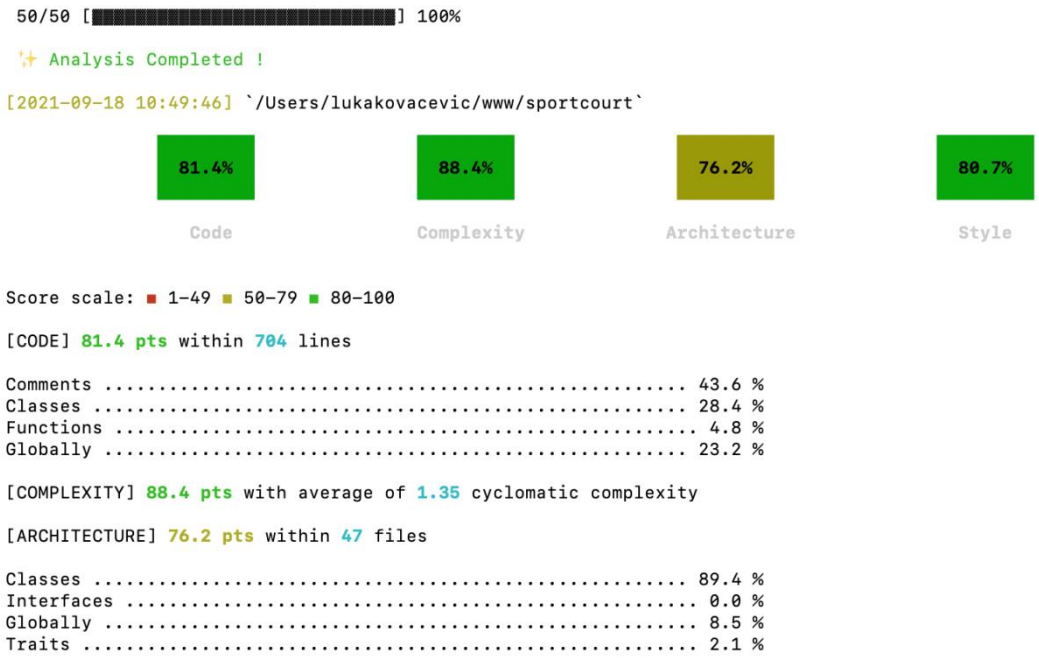
Court osjecki teren - nazorova 1

| Court Hours | Playing footsal | Playing basketball | 2021-09-18 |
|---|---|---|---|
| Hours 6 | 0 | 0 | |
| Hours 7 | 0 | 0 | |
| Hours 8 | 2 | 0 | You have signed to play in this time |
| Hours 9 | 0 | 0 | |
| Hours 10 | 0 | 0 | |

**Slika 5.18** Prikaz terena korisnika „drugi“ nakon prijave

### 5.2.5. Statička analiza koda

Statička analiza koda se radi kako bi se utvrdilo je li kod napisan ispravno, da nema sigurnosnih propusta i kako bi se stilski kod ispravnije napisao. Prilikom analize koda aplikacije *Sport Courts* se koristi Laravel paket za statičku analizu koda „PHP Insights“. Ovaj paket provjerava kvalitetu koda, stil, arhitekturu i kompleksnost. Također na vrlo čitljiv način prikazuje greške u kodu i postotak uspješnosti za svaku od kategorija. Prema slici 5.19, prva statička analiza koda je dala zadovoljavajuće rezultate gdje je jedino arhitektura bila ispod izvrsnih 80%.




**Slika 5.19** Prikaz rezultata prve statičke analize koda

Ova statička analiza također prijavljuje moguće sigurnosne greške kako bi ih se što lakše moglo ispraviti. Prema slici 5.20, se vidi da su najveće greške bile zbog ne korištenih elemenata koda, kojih je vrlo malo dok su ostale greške stilske prirode. Većina grešaka koje su prijavljene su zbog standardnih podataka koje nastanu prilikom pokretanja Laravel instalacije na koje se ne može utjecati. Nakon proučene statičke analize programskog koda i ispravljenih pogrešaka, uspješnost koda je poboljšana tako da su sva četiri parametra (*code, complexity, arhitecture* i *style*) iznad 80% uspješnosti izrade (Slika 5.21). Uklanjanjem nekorištenih elemenata koda i ispravljanjem operatora jednakosti dovelo je do izvrsne ocjene prilikom statičke analize programskog koda.

```
• [Code] Forbidden public property:
  app/Models/SportFieldType.php:9: Do not use public properties. Use method access instead.
  app/Models/UserFields.php:9: Do not use public properties. Use method access instead.
  app/Models/UserFriends.php:9: Do not use public properties. Use method access instead.
  +3 issues omitted

• [Code] Unused variable:
  app/Http/Controllers/SportFieldsController.php:45: Unused variable $all_times.

• [Code] Useless variable:
  app/Http/Controllers/CitiesController.php:49: Useless variable $cit.
  app/Http/Controllers/UserFieldsController.php:38: Useless variable $fields.
  app/Http/Controllers/UserFieldsController.php:45: Useless variable $types.

• [Code] Array indent:
  app/Http/Controllers/SportFieldsController.php:47: Array key not indented correctly; expected 16 spaces but found 24
  app/Http/Controllers/UserFriendsController.php:24: Array close brace not indented correctly; expected 12 spaces but found 8
  app/Models/SportField.php:16: Closing brace of array declaration must be on a new line

• [Code] Disallow implicit array creation:
  app/Http/Controllers/SportFieldsController.php:43: Implicit array creation is disallowed.
  app/Http/Controllers/SportFieldsController.php:46: Implicit array creation is disallowed.

• [Code] Disallow empty:
  app/Http/Middleware/RedirectIfAuthenticated.php:22: Use of empty() is disallowed.

• [Code] Disallow short ternary operator:
  app/Providers/RouteServiceProvider.php:60: Use of short ternary operator is disallowed.

• [Code] Disallow equal operators:
  app/Http/Controllers/SportFieldsController.php:27: Operator == is disallowed, use === instead.
  app/Http/Controllers/SportFieldsController.php:77: Operator == is disallowed, use === instead.
  app/Http/Controllers/UserScheduleController.php:28: Operator == is disallowed, use === instead.
  +7 issues omitted

• [Code] Declare strict types:
  routes/channels.php:1: Missing declare(strict_types=1).
  routes/console.php:1: Missing declare(strict_types=1).
  routes/web.php:1: Missing declare(strict_types=1).
  +44 issues omitted

• [Code] New with braces:
  app/Actions/Fortify/PasswordValidationRules.php:
@@ -15,3 +15,3 @@
     {
-        return ['required', 'string', new Password, 'confirmed'];
+        return ['required', 'string', new Password(), 'confirmed'];
     }

• [Code] Disallow mixed type hint:
  app/Actions/Fortify/UpdateUserProfileInformation.php:15: Usage of "mixed" type hint is disallowed.
  app/Actions/Fortify/UpdateUserProfileInformation.php:47: Usage of "mixed" type hint is disallowed.
  app/Http/Middleware/RedirectIfAuthenticated.php:18: Usage of "mixed" type hint is disallowed.
  +2 issues omitted
```

**Slika 5.20** Prikaz grešaka prve statičke analize

```
 50/50 [============================] 100%

 ✨ Analysis Completed !

[2021-09-18 11:11:48] `/Users/lukakovacevic/www/sportcourt`

   86.6%          88.4%          81.0%          81.9%
   Code           Complexity     Architecture   Style

Score scale: ■ 1-49 ■ 50-79 ■ 80-100

[CODE] 86.6 pts within 701 lines

Comments ..................................................... 43.9 %
Classes ...................................................... 28.0 %
Functions ..................................................... 4.9 %
Globally ..................................................... 23.3 %

[COMPLEXITY] 88.4 pts with average of 1.35 cyclomatic complexity

[ARCHITECTURE] 81 pts within 47 files

Classes ...................................................... 89.4 %
Interfaces ..................................................... 0.0 %
Globally ...................................................... 8.5 %
Traits ........................................................ 2.1 %
```

**Slika 5.21** Prikaz krajnje statičke analize

# 6. ZAKLJUČAK

Zbog stalnih migracija ljudi zbog posla i lakše organizacije sportskih amaterskih aktivnosti dolazi do težeg pronalaženja aktivnih ljudi. Cilj ovog završnog rada je olakšati u bilo kojem gradu na svijetu ljudima pronalazak javnih sportskih terena kao i organiziranje aktivnosti na njima. Trenutno se na tržištu ne nudi kombinacija ove dvije radnje. Korisniku je omogućeno da pretraži javne sportske terene po gradu i priključi aktivnostima na njima. Dodavanje drugih korisnika u favorite omogućuje lakšu organizaciju aktivnosti. Za izradu aplikacije je korišten programski jezik PHP zajedno s okvirom Laravel. Za izradu korisničkog sučelja korišteni su HTML i CSS, dok su PHP i JavaScript korišteni za dodatne funkcionalnosti. Kako bi se spremili podaci korisnika i sportskih terena, koristila se baza podataka MySQL. Prijedlog modela web aplikacije je ostvaren i ispitan za granične slučajeve kako bi se pokazao ispravnim i točnim. Statička analiza koda je pokazala izvrsne rezultate nakon manjih ispravaka gdje su svi parametri postigli izvrsnu uspješnost ispitivanja. Time je dokazana i kvaliteta programskog rješenja. Nakon ispitivanja, web aplikacija se pokazala ispravnom za sve krajnje slučajeve korištenja.

Web aplikacija bi se mogla dodatno unaprijediti s porukama koje korisnici mogu izmijeniti, izradom mobilne aplikacije radi još lakšeg korištenja, dodavanjem organizacije turnira i vođenjem statistike pojedinog igrača. Dodatna nadogradnja aplikacije bi se mogla izvršiti dodavanjem jednostavnog sklopovlja koje bi na terenima otvorenog tipa moglo upravljati rasvjetom terena. Kod terena zatvorenog tipa, sklopovlje bi moglo upravljati otključavanjem terena korisnicima koji su ga rezervirali.

## POPIS LITERATURE


[1] Courtify, Palma de Mallorca, 2021., dostupno na: www.courtifyapp.com [21.9.2021.]

[2] Encyclopedia Britannica, Inc., Chicago, 2021., dostupno na: https://www.britannica.com/topic/Facebook [21.9. 2021.]

[3] Hamburg.de, Hamburg, 2021., dostupno na: https://www.hamburg.de/active-city-map/ [21.9. 2021.]

[4] APKPure, San Francisco Bay Area, 2014.-2021., dostupno na: https://apkpure.com/timster-na%C4%91i-igra%C4%8De-za-fudbal/com.solaris.timster [21.9. 2021.]

[5] SportEasy SAS, Pariz, 2021., dostupno na: https://www.sporteasy.net/en/home/ [21.9. 2021.]

[6] Donna Tenis Team, Osijek, 2021. dostupno na: https://premiertennisosijek.com/en [21.9. 2021.]

[7] L. Welling, L. Thomson, PHP and MySQL Web Development, Addison-Wesley, Boston, 2017.

[8] M. Beam, Laravel 5 Essentials, Packt Publishing Ltd, Birmingham, 2015.

[9] Ecma Interenationl, Ženeva, 2020., dostupno na: https://262.ecma-international.org/11.0/ [21.9. 2021.]

[10] P. DuBois, MySQL, Pearson Education, London, 2008.

[11] C. L. Nedelcu, Nginx HTTP Server, Pack Publishing Ltd, Birmingham, 2015.

[12] MIT License, Massachusetts Institute of Technology, Boston, 2012., dostupno na: https://getcomposer.org/ [21.9. 2021.]

[13] F. Pezoa, J.L. Reutter, F. Suarez, M. Ugarte, D. Vrgoč, ACM Digital Library, Foundations of JSON Schema, New York, 2016.

[14] M. Masse, REST API Design Rulebook, O'Reilly Media, Newton, 2012.

[15] J. Niederst Robbins, Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics, O'Reilly Media, Newton, 2012.

[16] E. Woychowsky, Ajax: Creating Web Pages With Asynchronous JavaScript and XML, Pearson Education, Inc., London, 2007.

[17] Sportska Hrvatska, Zagreb, 2014, dostupno na: https://www.hoo.hr/hr/olimpizam/olimpijske-vijesti/4848-cak-62-hrvata-nije-tjelesno-aktivno-u-nedovoljnom-kretanju-prednjace-stariji [24.9.2021.]

[18] U.S Bureau of Labor Statistics, Washington, 2017, dostupno na: https://www.bls.gov/spotlight/2017/sports-and-exercise/home.htm [24.9.2021.]

[19] TechEmpower, Inc., Los Angeles, 2021., dostupno na: https://www.techempower.com/benchmarks/#section=data-r20&hw=cl&test=fortune [24.9.2021.]

[20] R. Patil, R.K.Singh, Scaling in Cloud Computing, International Journal of Advance Research, IJOAR.org, (1/1) (1/1), str. 21-27, siječanj, 2013.

# SAŽETAK

Zadatak ovog završnog rada je izrada web aplikacije za prijavu korisnika na javne sportske terene. U mjestima stanovanja ljudi su uglavnom povezani na Internet, imaju pristup javnim sportskim terenima i treba im dati što više alata da bi se potaknulo bavljenje sportom. S obzirom da programski jezik PHP, zajedno s okvirom Laravel, nudi brz i jednostavan način izrade web aplikacije, oni su odabrani za izradu. Korištenjem baze MySQL, omogućeno je spremanje podataka o korisnicima i sportskim terenima. Web aplikacija korisniku omogućuje registraciju i prijavu u sustav, pretraživanje javnih sportskih terena, dodavanje termina dolaska na njih, dodavanje drugih korisnika u svoje kontakte te obavještavanje kada se prijatelj prijavio na određeni teren. Statička analiza programskog koda pokazala je visoku razinu kvalitete rješenja za sva četiri parametra analize (*code*, *complexity*, *arhitecture*, *style*). Ispitivanje aplikacije pokazalo je da korisnik može odraditi sve zadatke zadane u ovom završnom radu.

**Ključne riječi:** javni sportski tereni, Laravel, MySQL, PHP, web aplikacija.

## ABSTRACT

The task of this paper is to create a web application for a user to check in on public sport courts. In inhabited areas, people are mostly connected to the Internet and have access to public sport courts. To actively participate in sports events we need to give them as many tools as we can. Since programming language PHP with framework Laravel offers to create web applications fast and easily, it was chosen for this application. Database MySQL provides a fast and reliable database to store user and sport courts data. The web application allows users to register and log in to the application, search through cities and public sports courts, check-in on desirable date and time on the sport court, add favorite other users, as well as notifying friends when a user applies to court. Static analysis has shown that all four parameters (code, complexity, architecture, style) are excellently written. While testing this web application it was proven that is functional for all edge cases.



**Keywords:** Laravel, MySQL, PHP, public sport courts, web application.

# ŽIVOTOPIS

Luka Kovačević rođen je 15. svibnja 1990. godine u Našicama. Nakon pohađanja Osnovne škole August Harambašić u Donjem Miholjcu, upisuje srednju školu u Valpovu smjer elektrotehničar. Po završetku srednje škole upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo. Od 2013. do 2015. godine radi u tvornici Thermia d.o.o. u uredu nabave materijala. 2017. godine se zaposlio u tvrtki Marrow Labs d.o.o. kao programer web aplikacija gdje radi i danas. Od programskih jezika poznaje PHP i JavaScript, a od programskih okolina Laravel, Node.js i React.

# PRILOZI

**Prilog 1.** Završni rad u datoteci docx

**Prilog 2.** Završni radu u datoteci pdf

**Prilog 3.** Programski kod projekta web aplikacije