Web aplikacija za objavu i prijavu na honorarne poslove

Živković, Ivan

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:607888

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-06**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

WEB APLIKACIJA ZA OBJAVU I PRIJAVU NA HONORARNE POSLOVE

Završni rad

Ivan Živković

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 22.11.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Ivan Živković
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4421, 20.09.2018.
OIB studenta:	47811779912
Mentor:	Doc.dr.sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Mirko Köhler
Član Povjerenstva:	Doc.dr.sc. Zdravko Krpić
Naslov završnog rada:	Web aplikacija za objavu i prijavu na honorarne poslove
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada	Napraviti aplikaciju koja će poslodavcima omogućiti postavljanje honorarnih poslova u različitim kategorijama. Posloprimci će moći postavljati svoje dosadašnje poslove i vjštine u svoj profil. Također će se moći prijaviti na honorarne poslova, a poslodavac će moći odabrati najboljeg posloprimca.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	22.11.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 07.01.2019.

Ime i prezime studenta:	Ivan Živković
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4421, 20.09.2018.
Ephorus podudaranje [%]:	3%

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za objavu i prijavu na honorarne poslove**

izrađen pod vodstvom mentora Doc.dr.sc. Ivica Lukić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
   1.1. Zadatak završnog rada 1
2. PREGLED KORIŠTENIH TEHNOLOGIJA I ALATA 2
   2.1. HTML5 2
   2.2. CSS3 3
   2.3. Bootstrap (v4.1.3) 4
   2.4. JavaScript 6
   2.5. AJAX 7
   2.6. Laravel (v.5.7.9) 8
   2.7. MySQL 9
   2.8. WAMP 10
   2.9. Visual Studio Code 11
3. STRUKTURA PROJEKTA 12
   3.1. Laravel predložak 12
   3.2. Modeli podataka 13
   3.3. Kontroleri 15
   3.4. Migracije 16
   3.5. *Seeder-i* 17
   3.6. Public mapa 18
   3.7. Resources mapa 19
   3.8. Rute 20
4. KORISNIČKA SUČELJA 21
   4.1. Navigacijska traka i podnožje 21
   4.2. Home 22
   4.3. Community 22
   4.4. Work 23
   4.5. Employ 27
   4.6. Posts 29
   4.7. Log in i Register 30
   4.8. Profil 31

5. PROGRAMSKO RJEŠENJE 33
5.1. Registracija korisnika 33
5.2. Kreiranje oglasa za posao 34
5.3. Prijava na oglas 36
5.4. Upravljanje prijavama 38
5.5. Pretraga oglasa 40
5.6. Praćenje korisnika 44
6. ZAKLJUČAK 48
LITERATURA 49
SAŽETAK 51
ABSTRACT 52
ŽIVOTOPIS 53

1. UVOD

Tema ovog završnog rada je izrada web aplikacije za objavu i prijavu na honorarne poslove. Korisnici se mogu registrirati i izraditi korisničke profile te se isti mogu uređivati i brisati. Korisnici imaju mogućnost objave oglasa za posao te uređivanje i brisanje istih, također sami mogu pretraživati i filtrirati poslove te se mogu prijaviti na tuđe poslove pri čemu su poslodavci obaviješteni e-poštom. Kao poslodavci, korisnici imaju mogućnosti pregleda svih prijava na njihove poslove te opcije prihvaćanja ili odbijanja prijava pri čemu su potencijali kandidati obaviješteni e-poštom. Kao korisnici koji su se prijavili na poslove, imaju uvid u svoje prijave te njihov status. Iste prijave mogu i otkazati. Nakon odrađenog posla, korisnici se mogu međusobno ocijeniti u smislu da poslodavci ocjenjuju korisnike koji su radili na poslu i obrnuto. Korisnici se međusobno mogu pratiti i prestati pratiti te mogu pregledavati tuđe profile i objave. Isto tako imaju uvid koga prate i tko njih prati. Korisnici se mogu kontaktirati putem forme na njihovim profilima. Također postoji mogućnost objave komentara na objavama i poslovima te mogućnosti sviđanja istih. U drugom poglavlju će se dati pregled korištenih tehnologija poput HTML5, CSS, Laravel i ostalih te alata. U trećem poglavlju će se dati pregled na strukturu projekta kroz opis datoteka, kontrolera, ruta, mapa i drugih. U četvrtom poglavlju prikazat će se korisnička sučelja uz kratak opis a neka od njih su stranica za prijavu, registraciju i pretragu poslova. U petom poglavlju će se objasniti programska rješenja za glavne funkcionalnosti poput registracije, objave oglasa za posao, praćenja korisnika i drugih. U praktičnom dijelu završnog rada izrađena je funkcionalna web aplikacija korištenjem opisanih tehnologija u drugom poglavlju.

1.1. Zadatak završnog rada

Zadatak završnog rada je izrada web aplikacije na kojoj korisnici mogu objavljivati svoje oglase za posao te se prijavljivati na tuđe. Potrebno je također implementirati osnovne funkcionalnosti koje bi takav sustav trebao podržavati. Kao dodatak, aplikacija treba sadržavati neke od značajki društvene mreže kao što su međusobno praćenje, objava komentara, sviđanja i slično.

2. PREGLED KORIŠTENIH TEHNOLOGIJA I ALATA

U ovom poglavlju će ukratko biti objašnjene tehnologije i alati koji su se koristili tijekom izrade praktičnog dijela završnog rada, tj. web aplikacije.

2.1. HTML5

HTML (engl. *HyperText Markup Language)* je opisni jezik koji služi za strukturiranje i opis web stranice te njenog sadržaja. *Opisni jezik* podrazumijeva da pomoću njega ne kreiramo poslovnu logiku web aplikacije korištenjem funkcija, varijabli, petlji i slično nego raspoređujemo sadržaj aplikacije prikazane u pregledniku korištenjem HTML elemenata koje preglednik interpretira i korisniku prikazuje kao web stranicu [1].

HTML5 je najnovija verzija HTML jezika u kojoj su predstavljeni novi elementi, tipovi za unos podataka u forme, atributi te druge stvari.

```
<!DOCTYPE html>
<html>
<head>
<title>Naslov stranice</title>
</head>
<body>

<h1>Ovo je naslov s veličinom fonta: 24px</h1>
<p>Ovo je paragraf.</p>
<p id="crveni_tekst">Ovo je dodatni tekst koji ima crvenu boju fonta.</p>

</body>
</html>
```

Sl. 2.1. Primjer jednostavnog HTML dokumenta

Kao što možemo vidjeti na slici 2.1., HTML dokument se sastoji od više različitih oznaka i elemenata. Osnovne oznake su <*html*> koja definira HTML dokument, <*head*> koja definira zaglavlje dokumenta u koje se unosi naslov stranice, meta podaci, poveznice na vanjske CSS ili JavaScript dokumente i slično te <*body*> oznaka koja definira tijelo dokumenta u kojemu navodimo sadržaj HTML dokumenta.

2.2. CSS3

CSS *(*engl. *Cascading Style Sheets)* je stilski jezik kojim se opisuje na koji način su HTML elementi a samim time i web stranica, prikazani u preglednicima te služi za kreiranje stilova. To znači da korištenjem CSS-a je moguće definirati poziciju elemenata, boje, margine, vrstu i veličinu fonta te mnoga druga svojstva.

CSS3 je najnovija verzija CSS-a koja je predstavila nova svojstva i mogućnosti kao što su višestruke pozadinske slike elementa, sjene, zakrivljenost kutova elementa te brojne druge.

Sintaksa se sastoji od selektora i deklaracijskog bloka. Deklaracijski blok se također sastoji od dva dijela a to su svojstvo i vrijednost/i [2].

Postoji više od dvadeset vrsta selektora u CSS-u a neki od njih su:

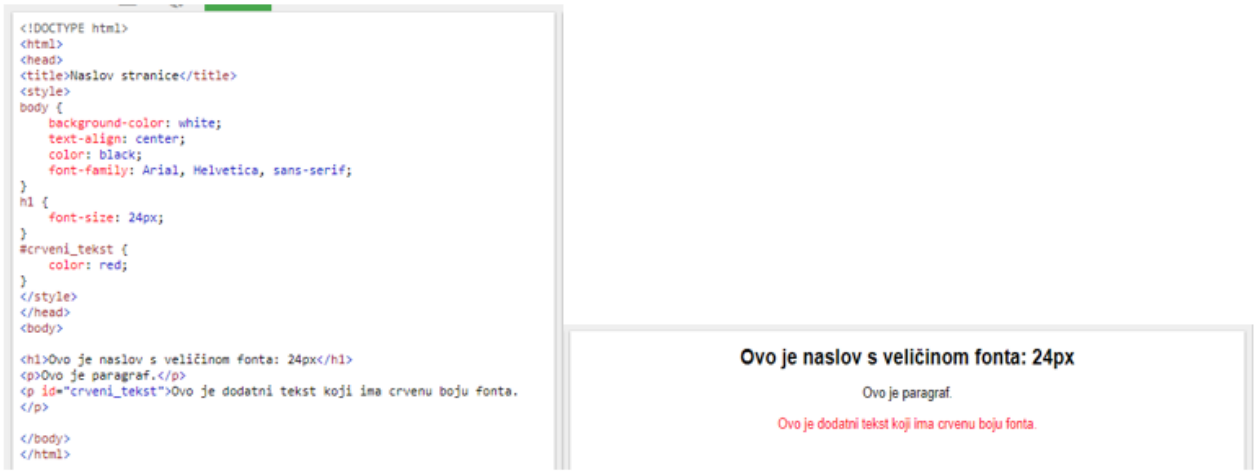
- Identifikacijski (#) – označava element čija vrijednost *id* atributa je jednaka sadržaju iza # znaka
- Klasni (.) – označava sve elemente čija vrijednost *class* (hrv. *klasa*) atributa je jednaka sadržaju iza točke
- Univerzalni (*) – označava sve elemente
- Selektor na razini elementa (npr. *div*) – označava sve <*div*> elemente

Sl. 2.2. Primjer CSS sintakse [2]

CSS povezujemo s HTML dokumentom na četiri načina:

- Pisanjem CSS koda unutar vrijednosti *style* (hrv. *stil*) atributa nad elementom
- Dodavanjem CSS koda u zaglavlje dokumenta unutar *<head></head>* oznaka
- Povezivanjem lokalne (*.css*) CSS datoteke korištenjem *<link href=" ">* oznake unutar zaglavlja dokumenta čija vrijednost *href* atributa je putanja do lokalne datoteke
- Povezivanjem vanjske (*.css*) CSS datoteke korištenjem *<link href=" ">* oznake unutar zaglavlja dokumenta čija vrijednost *href* atributa je internet adresa do vanjske datoteke

Preporučljivo je sve CSS datoteke imati u projektu lokalno jer u slučaju prekida internetske veze ili u slučaju da je adresa na kojoj se nalazi datoteka neaktivna, korisnik ih neće moći koristiti.



Sl. 2.3. Primjer korištenja CSS-a u kombinaciji s HTML kôdom

2.3. Bootstrap (v4.1.3)

Bootstrap je besplatni, *front-end* radni okvir (engl. *front-end framework*) otvorenog koda dizajniran za brži i lakši razvoj web stranica. Uključuje HTML, CSS i JavaScript tehnologije za kreiranje i dizajn gotovih komponenti kao što su predlošci za forme, tablice, dugmad, padajući izbornici i mnoge druge. Bootstrap nam omogućuje lako stvaranje prilagodljivog dizajna (engl. *Responsive Web Design*)

korištenjem gotovih CSS klasnih selektora koji imaju predefinirana svojstva te vrijednosti za određene točke loma (engl. *breakpoints*) i širine stranice što u prijevodu znači da će se web stranica i komponente prilagoditi veličini zaslona i izgledati prikladno na zaslonu mobitela, tableta ili većeg monitora [3]. Bootstrap verzija 4.1.3 nije podržana na svim preglednicima kao npr. na pregledniku Internet Explorer verzije 10 jer koristi određena CSS svojstva i JavaScript metode koje taj preglednik još nije usvojio ili ih ne podržava.

S HTML dokumentom se povezuje kao i bilo koja druga CSS datoteka. Može se besplatno preuzeti lokalno ili se povezati kao vanjska datoteka. S obzirom na to da koristi dodatne zavisnosti, moramo i njih uključiti u projekt kako bi radio ispravno.

```
<div class="container">
<div class="jumbotron">
  <h1>Pozdrav!</h1>
  <p>Ovo je primjer HTML dokumenta na čije su elemente primijenjene neke od Bootstrap klasa.</p>
  <hr>
    <a class="btn btn-primary btn-lg" href="#" role="button">Bootstrap</a>
</div>
</div>
```

Sl. 2.4. Primjer HTML elemenata na koje su primijenjene neke od Bootstrap klasa

Sl. 2.5. Rezultat slike 2.4.

2.4. JavaScript

JavaScript je skriptni jezik koji posjeduje značajke objektno-orijentiranog programskog jezika koji se izvodi na klijentskoj strani (engl. *client side*), tj. u pregledniku korisnika. Kada kažemo da posjeduje značajke objektno-orijentiranog programskog jezika mislimo na to da postoji određeni koncept klasa, objekata, enkapsulacije i nasljeđivanja. U JavaScript-u ne postoje klase u smislu kao kod drugih programskih jezika, nego princip prototipa. JavaScript je jezik koji ne zahtijeva kompiliranje nego se izvodi i interpretira u trenutku kada preglednik čita skriptu.

Svrha korištenja je ostvarivanje dinamičkih i interaktivnih web stranica jer pomoću JavaScripta možemo kreirati logiku, animirati elemente, pozivati određene metode prilikom klika na dugme, brisati i kreirati HTML elemente te nam pruža brojne druge mogućnosti.

JavaScript povezujemo s HTML dokumentom na tri načina [4]:

1. Pisanjem JavaScript koda unutar <*script*></*script*> oznaka u HTML dokumentu
2. Povezivanjem lokalne (*.js*) datoteke korištenjem <*script src*=““> oznake čija je vrijednost *src* atributa putanja do lokalne JavaScript datoteke
3. Povezivanjem vanjske (*.js*) datoteke korištenjem <*script src*=““> oznake čija je vrijednost *src* aributa internet adresa do vanjske JavaScript datoteke

Postoji mnogo biblioteka (engl. *library)* i radnih okvira baziranih na JavaScript-u koje nam daju razne mogućnosti i olakšavaju pisanje koda. Jedna od njih je JQuery s kojom se lako dohvaćaju HTML elementi, animiraju, mijenjaju CSS svojstva i slično.

Kao i kod CSS povezivanja, preporučljivo je sve datoteke imati lokalno.

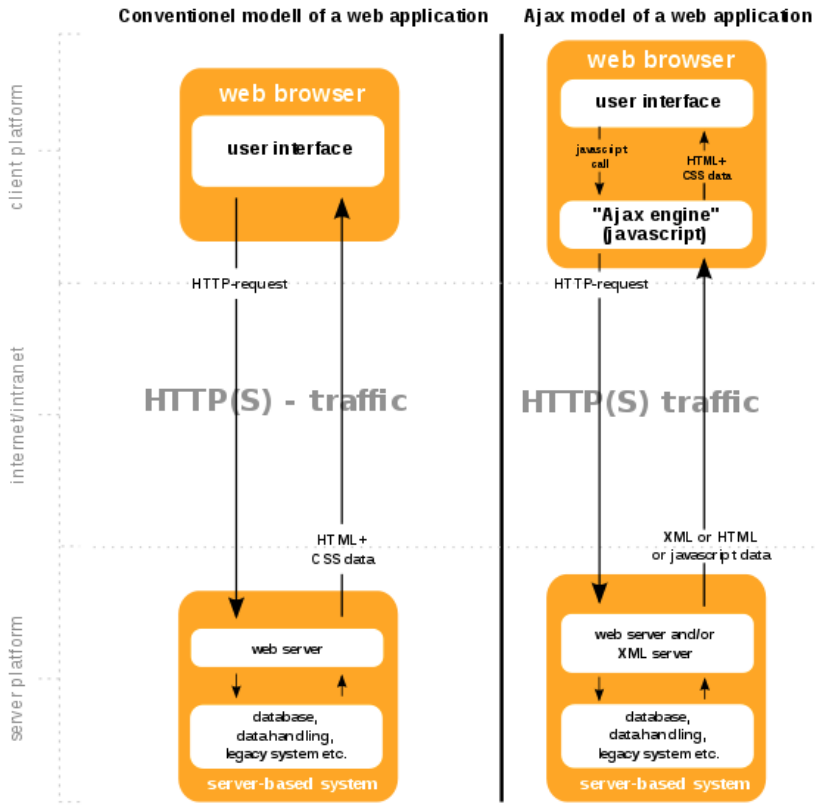
```
<!DOCTYPE html>
<html>
<body>

<p>Zbrajanje.</p>

<button onclick="zbroji()">Klikni za zbroj</button>

<p id="zbroj"></p>

<script>
function zbroji () {
    var zbroj = 100 + 50;
    document.getElementById("zbroj").innerHTML = 'Zbroj je '+zbroj;
}
</script>

</body>
</html>
```

Sl. 2.6. Primjer JavaScript kôda

Na slici 2.6. možemo vidjeti primjer HTML dokumenta s dva paragrafa, jednim dugmetom i skriptom. Dugme sadrži atribut *onclick* (hrv. *na klik*) koji se aktivira klikom miša na taj element s vrijednosti *zbroji()* što pregledniku znači da kada korisnik pritisne dugme, izvrši metodu *zbroji()*. U skripti je vidljiva implementacija te metode koja sadrži varijablu *zbroj* čija je vrijednost 150 te sadrži metodu koja kao rezultat dohvaća element čija vrijednost *id* atributa je jednaka „*zbroj*" te u njegov sadržaj upisuje vrijednost varijable *zbroj* tj. 150.

2.5. AJAX

AJAX (engl. *Asynchronous Javascript and XML*) predstavlja set tehnika za razvoj aplikacija. Koristi se za razmjenu podataka između klijentske i serverske strane bez potrebe za ponovnim učitavanjem stranice što poboljšava iskustvo rada i korisničko iskustvo korištenja aplikacije [5].



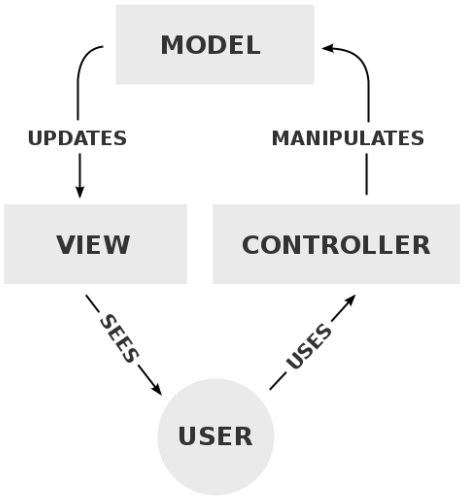
Sl. 2.7. Usporedba AJAX-a i sinkronog načina razmjene podataka u web aplikaciji [6]

Lijeva strana slike 2.7. prikazuje sinkronu metodu razmjene podataka gdje se iz korisničkog sučelja pokreće zahtjev za podacima koji dolazi do servera. Na serveru se izvršava logika koja vraća rezultat na klijentsku stranu. Za vrijeme trajanja te razmjene podataka korisnik mora čekati. Na desnoj strani je vidljiv AJAX pristup gdje se iz preglednika kreira zahtjev prema AJAX *engine-u* koji kreira *XMLHttpRequest* objekt. On se u pozadini šalje na server koji ga obrađuje i vraća rezultat AJAX metodi koja nakon toga ima mogućnost osvježavanja sučelja s novim podacima putem kôda bez potrebe za ponovnim učitavanjem stranice. Korisnik i dalje može imati mogućnost rada za to vrijeme.

2.6. Laravel (v.5.7.9)

Laravel je besplatni PHP programski okvir otvorenog kôda (engl. *open source*) koji služi za izradu web aplikacija. Temelji se na Symfony radnom okviru te prati MVC (model-pogled-kontroler) (*engl. Model-view-controller)* uzorak ili koncept za dizajn web aplikacija [7].

MVC se odnosi na arhitekturu koja razdvaja poslovnu logiku od korisničkog sučelja. *Model* se odnosi na modele podataka s kojima aplikacija rukuje, njihove podatke i stanja. *View* se odnosi na grafičko tj. korisničko sučelje koje prikazuje podatke iz modela koje je upravljač proslijedio. *Controller* je veza između modela i pogleda tj. korisnika te je zaslužan za odrađivanje logike koristeći podatke iz modela na osnovu akcija koje korisnik zahtjeva iz pogleda [8].



Sl. 2.8. Prikaz MVC arhitekture [9]

Laravel olakšava rad s bazom podataka koristeći Eloquent ORM (*engl. Object-relational mapping*) gdje svaka tablica iz baze podataka ima svoj model pomoću kojega se mogu izvršavati radnje nad tom tablicom [10]. Prije nego što uopće možemo koristiti podatke iz baze podataka potrebno ju je kreirati. Laravel također ima ugrađene metode i posebnu vrstu klasa *Seeder* (hrv. *sijač)* te migracije (*engl. migrations*) koje olakšavaju cijeli proces. Klase koje nasljeđuju klasu *Seeder* implementiraju metodu *run()* pomoću koje korisnik može na jednostavan način pomoću petlji ili ručno kreirati unose u bazu podataka bez pisanja SQL kôda dok s migracijama na jednostavan način kreira i uređuje tablice.

Kako ne bismo morali sami kreirati klase za migracije, *seeder-e*, modele i slično, s Laravel verzijom 3 se pojavio i Artisan CLI (engl. *Command Line Interface*) pomoću kojega korisnik može, koristeći naredbeni redak (engl. *Command prompt)* pozivati Artisan naredbe [11].

Laravel također dolazi s velikim brojem paketa. Korisnik može dodati nove pakete, brisati te kreirati svoje. Paketi se instaliraju uz pomoć Composer alata za upravljanje zavisnostima (engl. *dependency*). Npr. ako želimo instalirati paket kojim možemo lako manipulirati slikama, pokrenemo naredbu iz naredbenog retka „*php composer .phar require intervention/image*“ [12].

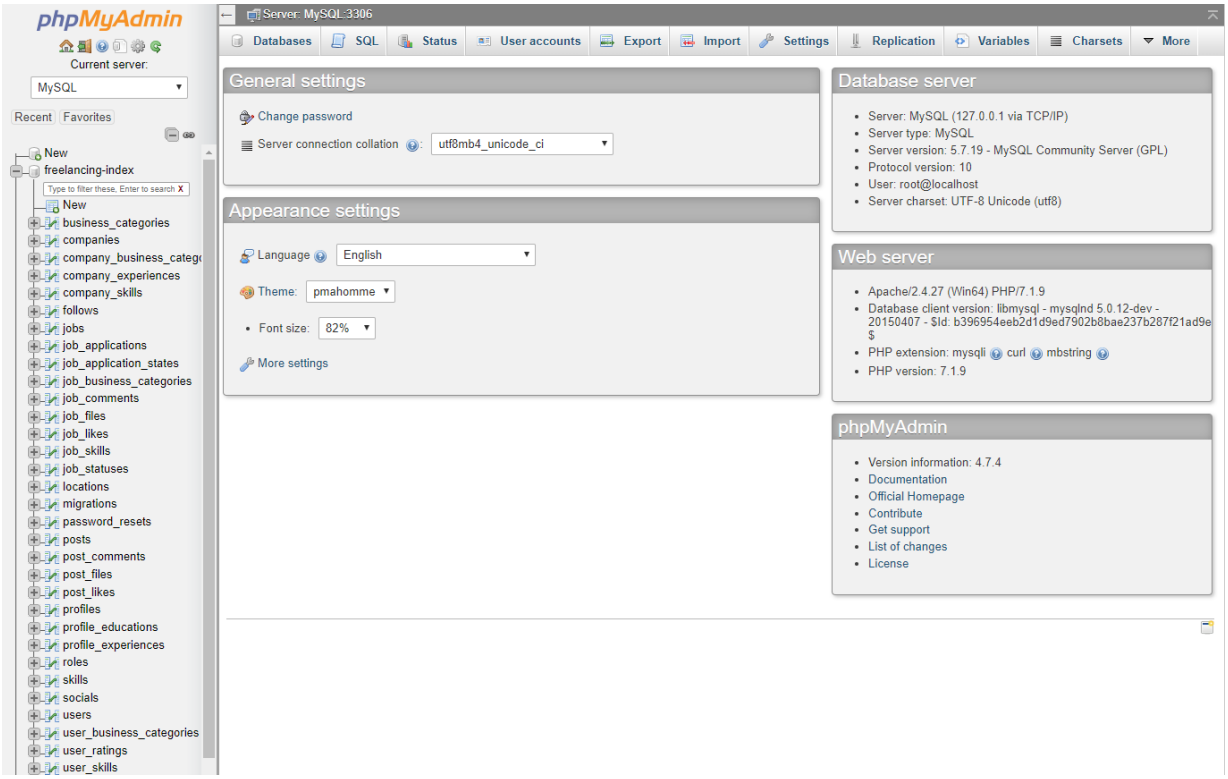
S Laravelom je isto tako vrlo lako kreirati rute i navigaciju za stranice, provjeravati podatke, kreirati poglede (engl. *view)* tj. stranice i mnoge druge radnje.

Za izradu pogleda se koristi *Blade engine* pomoću kojega možemo koristiti modele izravno u HTML dokumentu, petlje, uvjetna grananja i slično [13].

2.7. MySQL

Podaci koje koristimo na web aplikaciji kada se korisnik registrira, prijavi, objavi posao ili napravi bilo koju drugu radnju koja bi rezultirala promjenom podataka, bi trebali postojati kao takvi i nakon zatvaranja aplikacije. Iz tog razloga moramo imati bazu podataka u koju ćemo sve te podatke spremati. Za potrebu ovog projekta korišten je MySQL sustav za rad s bazom podataka. *SQL* u nazivu znači strukturirani jezik za upite (engl. *Structured Query Language*) koji služi za rad sa SQL baziranim bazama podataka.

Uzmimo na primjer naredbu „*select * from jobs“*. S ovom naredbom ćemo dobiti ispis svih podataka iz tablice *jobs* (hrv. *poslovi*).



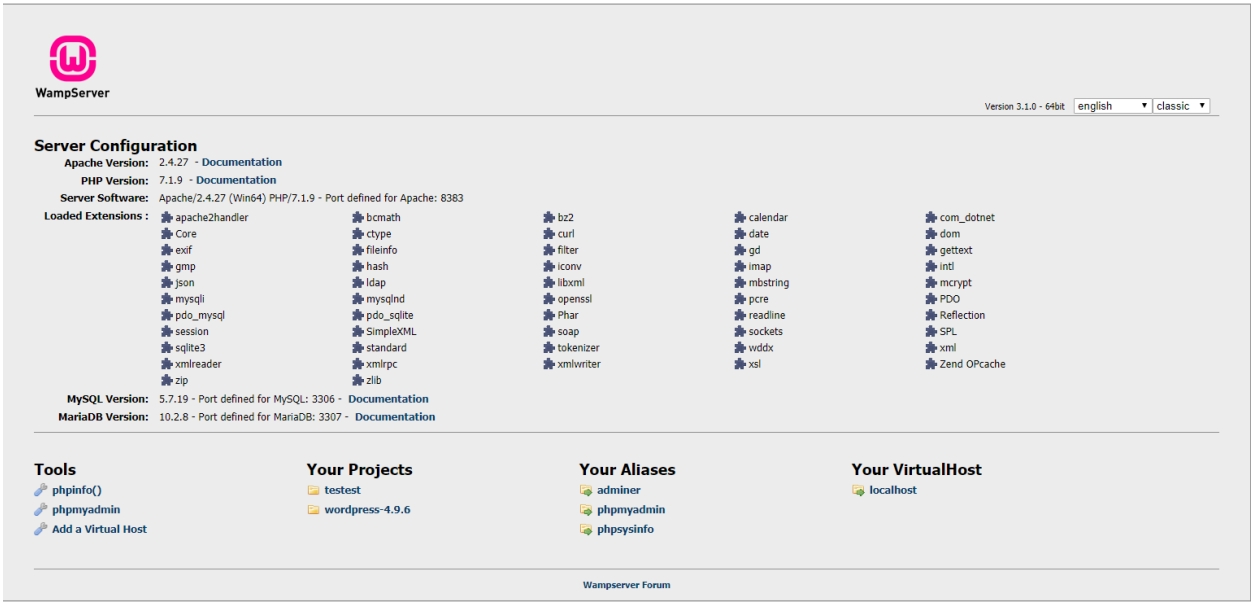
Sl. 2.9. phpMyAdmin sučelje za rad s MySQL bazom podataka

2.8. WAMP

WAMP je akronim za grupu alata koji nam omogućuju stvaranje lokalnog servera, baze podataka te rad web aplikacije. Besplatan je i otvorenog koda [14].

S obzirom na to da je WAMP akronim, on zapravo predstavlja: [14]

- *W* - predstavlja Windows u akronimu što znači da je namijenjen za korištenje na Windows operacijskom sustavu
- *A* - predstavlja Apache server koji nam služi za stvaranje lokalnog servera na kojem ćemo prikazivati našu aplikaciju
- *M* - predstavlja MySQL sustav za upravljanje bazom podataka
- *P* - predstavlja PHP, Python i PEARL programske jezike



Sl. 2.10. WAMP sučelje u pregledniku

2.9. Visual Studio Code

Visual Studio Code je uređivač teksta (engl. *text editor*) razvijen od strane Microsoft-a. Dostupan je na svim operacijskim sustavima. Glavne značajke su mu jednostavnost, skalabilnost te to da je besplatan i otvorenog koda. Korisnici mogu kreirati i preuzimati pakete te si time poboljšati iskustvo korištenja. Podržava veliki broj programskih jezika te ima ugrađene alate i pakete za prepoznavanje sintakse, ispravljanje grešaka, praćenje promjena u datotekama i mogućnost uređivanja više linija koda u isto vrijeme.

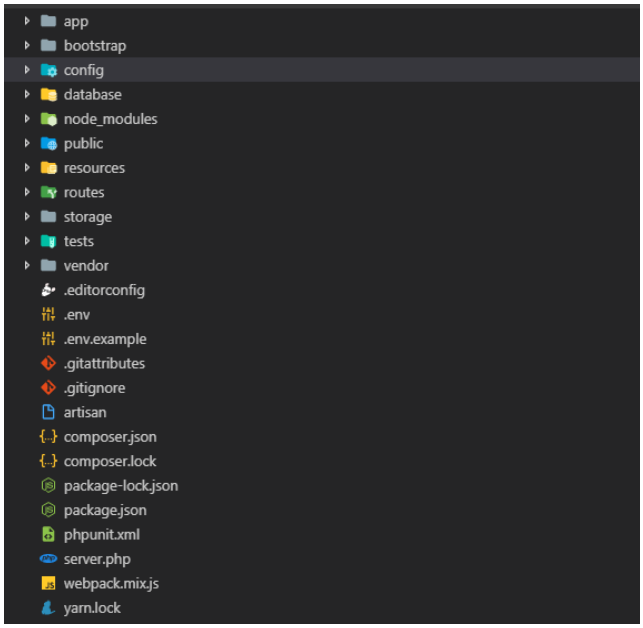
Prema anketi iz 2018. godine koju je proveo Stack Overflow, Visual Studio Code je bio najkorišteniji program sa 34.9% korisnika od ~75 tisuća korisnika [15].

3. STRUKTURA PROJEKTA

Kroz sljedeća poglavlja ćemo prikazati strukturu projekta te ju ukratko objasniti.

3.1. Laravel predložak

Kao što je već navedeno u opisu tehnologija, Laravel je PHP radni okvir koji nam olakšava i ubrzava razvoj web aplikacija nudeći razne metode, klase, naredbe, predloške i još mnogo toga. Novi projekt u Laravelu se kreira iz naredbenog retka tako što pokrenemo naredbu „*laravel new [naziv_projekta]*“. Nakon što se naredba izvede, na slici 3.1. možemo vidjeti početnu strukturu Laravel projekta.

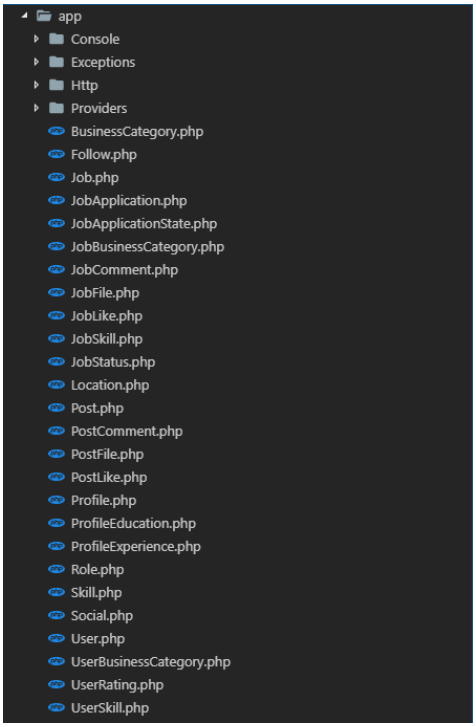


Sl. 3.1. Početna struktura Laravel projekta

Bitnije datoteke i mape su *.env* u kojoj postavljamo naziv aplikacije, vezu s bazom podataka i druge, *web.php* kojoj su navedene rute, mapa *app* u kojoj se nalaze kontroleri i modeli podataka, mapa *database* u kojoj se nalaze migracije i *seeder-i* te mapa *resources* u kojoj se nalaze pogledi, tj. *blade* datoteke. Detaljan opis možemo vidjeti pod literaturom (16).

3.2. Modeli podataka

Modele podataka možemo kreirati naredbom „*php artisan make:model [naziv_modela]“*. Modeli koje kreiramo se spremaju unutar *app* foldera. Služe nam za jednostavno pristupanje i upravljanje podacima u bazi podataka. Svaki od modela odgovara tablici u bazi podataka.



Sl. 3.2. Modeli podataka u izrađenoj aplikaciji

Iz slike 3.2. možemo vidjeti sve modele podataka koje *TheHunt* aplikacija koristi pa tako npr. *User (*hrv. *korisnik*) predstavlja tablicu *users* u bazi podataka. Konvencija naziva je takva da se modeli nazivaju u jednini, velikim početnim slovom a tablice u bazi u množini malim slovima. U slučaju relacije npr. *JobComments* model podataka bi odgovarao tablici *job_comments* u bazi podataka.

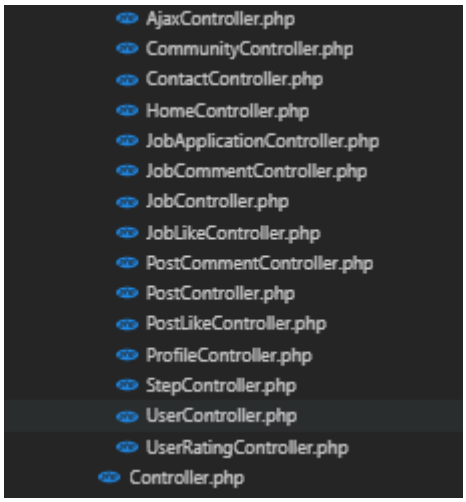
```php
<?php

You, a month ago
namespace App;

use Illuminate\Database\Eloquent\Model;

/** Job model with relation functions */
You, a few seconds ago | 2 authors (You and others)
class Job extends Model
{
    //
    protected $fillable = ['user_id','title','description','slug','offer','is_per_hour',
    'job_status_id','job_location_city','job_location_country'];


    /** Relation with user table (owner) */
    public function user(){
        return $this->belongsTo('App\User');
    }
    /** Relation with skills table (required skills) */
    public function job_skills(){
        return $this->hasMany('App\JobSkill')->join('skills', 'job_skills.skill_id',
        'skills.id')->select('job_skills.*', 'skills.name as name');
    }
    /** Relation with comments */
    public function job_comments(){
        return $this->hasMany('App\JobComment');
    }
    /** Relation with likes */
    public function job_likes(){
        return $this->hasMany('App\JobLike');
    }
    /** Relation with statuses table */
    public function job_status(){
        return $this->belongsTo('App\JobStatus');
    }
    /** Relation wtih business categories */
    public function job_business_categories() {
        return $this->hasMany('App\JobBusinessCategory')->join('business_categories',
        'job_business_categories.business_category_id', 'business_categories.id')->select
        ('job_business_categories.*', 'business_categories.name as name');
    }
    /** Relation with files table (job ads can have files attached) */
    public function job_files() {
        return $this->hasOne('App\JobFile');
    }
    /** Relation with job applications */
    public function job_applications() {
        return $this->hasMany('App\JobApplication');
    }

}
```

Sl. 3.3. Model podataka za posao (engl. *Job*)

Iz slike 3.3. možemo vidjeti model podataka za posao. *$fillable* predstavlja polje podataka u kojemu navodimo sve podatke kojima se može manipulirati u aplikaciji za taj model podataka. Također u modelu podataka postoje i funkcije koje nam predstavljaju relacije s drugim tablicama. Primjećujemo metode *hasMany()*, *belongsTo()* i slične. Pomoću njih definiramo relacije u bazi podataka koristeći Eloquent [17].

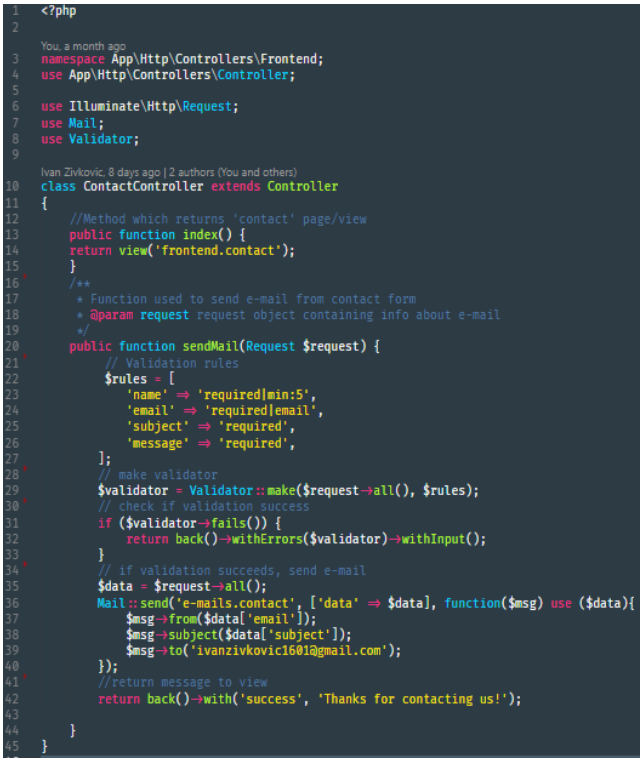
3.3. Kontroleri

S obzirom na to da se Laravel zasniva na MVC konceptu za dizajn aplikacija, sljedeća stvar koju moramo imati u aplikaciji su kontroleri. Kontrolere možemo kreirati naredbom „*php artisan make:controller [naziv_kontrolera]*“. U kontrolerima pišemo kôd koji služi za logiku aplikacije. U njemu definiramo koje podatke šaljemo korisniku na sučelje, što radimo s podacima koje korisnik pošalje serveru i slično.



Sl. 3.4. Popis kontrolera u *TheHunt* aplikaciji

Uzmimo za primjer *ContactController* koji služi za dohvaćanje stranice za kontakt te metodu za slanje e-pošte. U kontroleru postoje dvije metode, *index()* i *sendMail()*. Metoda *index()* služi za dohvaćanje pogleda za prikaz stranice za kontakt na kojoj korisnici mogu slati poruke u obliku e-pošte vlasniku aplikacije. Metoda *sendMail()* služi za slanje e-pošte. U njoj postoji *validator* koji vrši validaciju podataka unesenih u formu za kontakt te šalje e-poštu na adresu *ivanzivkovic1601@gmail*.



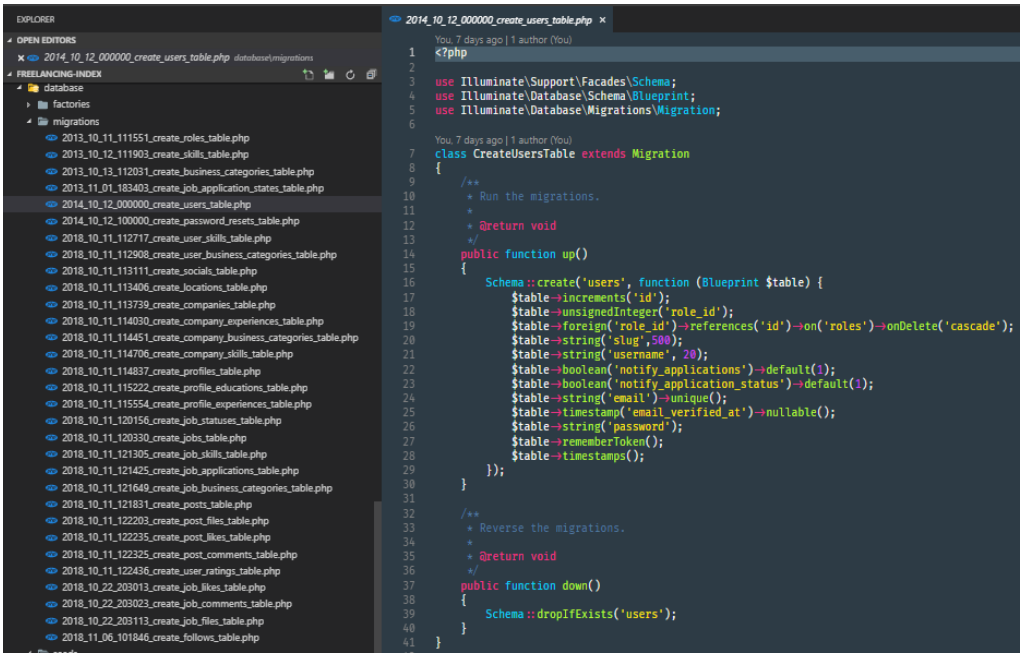
```php
<?php

You, a month ago
namespace App\Http\Controllers\Frontend;
use App\Http\Controllers\Controller;

use Illuminate\Http\Request;
use Mail;
use Validator;

Ivan Zivkovic, 8 days ago | 2 authors (You and others)
class ContactController extends Controller
{
    //Method which returns 'contact' page/view
    public function index() {
    return view('frontend.contact');
    }
    /**
     * Function used to send e-mail from contact form
     * @param request request object containing info about e-mail
     */
    public function sendMail(Request $request) {
        // Validation rules
        $rules = [
            'name' => 'required|min:5',
            'email' => 'required|email',
            'subject' => 'required',
            'message' => 'required',
        ];
        // make validator
        $validator = Validator::make($request->all(), $rules);
        // check if validation success
        if ($validator->fails()) {
            return back()->withErrors($validator)->withInput();
        }
        // if validation succeeds, send e-mail
        $data = $request->all();
        Mail::send('e-mails.contact', ['data' => $data], function($msg) use ($data){
            $msg->from($data['email']);
            $msg->subject($data['subject']);
            $msg->to('ivanzivkovic1601@gmail.com');
        });
        //return message to view
        return back()->with('success', 'Thanks for contacting us!');

    }
}
```

Sl. 3.5. ContactController

3.4. Migracije

Migracije nam služe kako bismo na jednostavan način kreirali tablice u bazi podataka, polja ili svojstva te tipove podataka za ta svojstva. Migracije kreiramo s naredbom „*php artisan make:migration [naziv_migracije]*“. Bitno je paziti kojim redom su kreirane migracije ili barem promijeniti datume u nazivu datoteke jer se tim redom i kreiraju tablice u bazi podataka. Zbog toga je bitno da prvo kreiramo osnovne tablice prije relacijskih.

Migracije pokrećemo naredbom „*php artisan migrate*“.

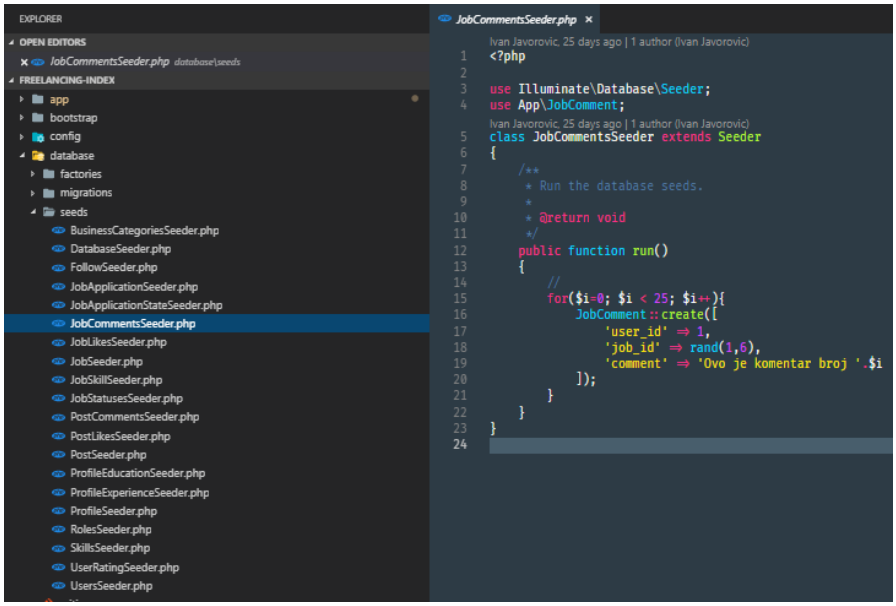


Sl. 3.6. Popis migracija i primjer migracije za kreiranje tablice za korisnike (engl. *Users)*

U metodi *up()* navodimo da želimo kreirati tablicu *users* te navodimo koja sve svojstva ili polja će sadržavati i koji tip podataka. Metoda *timestamps()* kreira dodatna dva polja koja odgovaraju datumu unosa (engl. *created_at*) i datumu izmjene retka u tablicu (engl. *updated_at*). Metoda *rememberToken()* kreira *token* za korisnike koji se može iskoristit kako bi korisnici ostali prijavljeni u aplikaciju i kada im istekne sesija. Postoje i metode *default()* s kojom postavljamo zadanu vrijednost ako nije druga navedena te *nullable()* kojom omogućujemo polju da bude prazno tj. da nema vrijednost.

3.5. *Seeder-i*

Pomoću *seeder-a* možemo na jednostavan način kreirati unose u bazu podataka što je iznimno korisno pri testiranju aplikacije. *Seeder* kreiramo naredbom „*php artisan make:seeder [naziv_seedera]“*.



Sl. 3.7. Popis *seeder*-a u aplikaciji i prikaz *seeder-a* za kreiranje komentara na oglasima za poslove

Na slici 3.7. vidimo popis *seeder-a* koje izrađena aplikacija koristi i primjer *seeder-a* za kreiranje komentara na nasumičnim oglasima za poslove, točnije njih dvadeset pet što smo izveli koristeći petlju. Ovdje je bitno da u bazi već postoje poslovi i korisnici kako se ne bi dogodila greška pri izvršavanju.

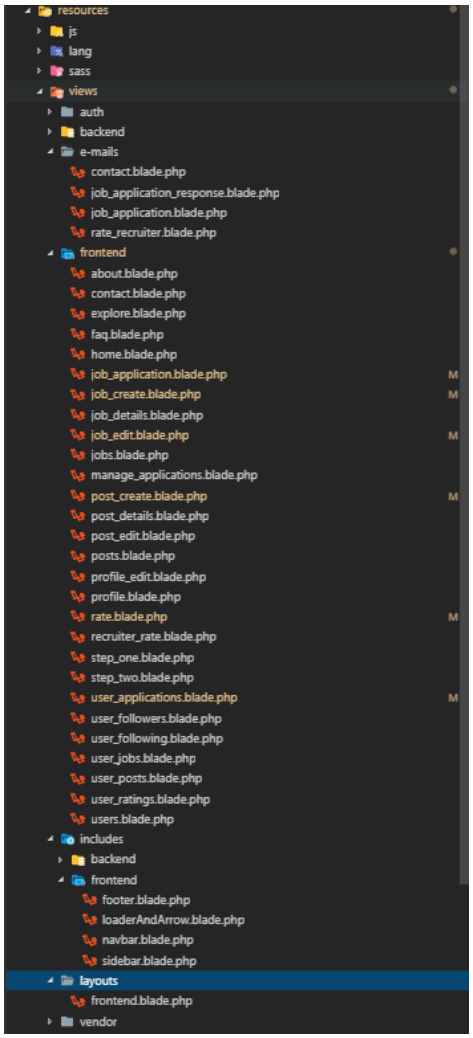
Svi *seeder-i* se moraju uključiti u glavni *DatabaseSeeder* koji se pokreće kada pokrenemo naredbu „*php artisan db:seed*“. Bitan je i redoslijed kojim su uključeni slično kao kod migracija kako se ne bi dogodila greška pri izvršavanju.

3.6. Public mapa

U mapi *public* (hrv. *javno*) se nalaze svi *asset-i* kao što su skripte, CSS datoteke, fontovi i slično. U slučaju izrađene aplikacije, u *public* mapu su spremljene Bootstrap datoteke, skripte koje se koriste za AJAX metode te druge funkcionalnosti, prenesene slike profila korisnika i drugo.

3.7. Resources mapa

U *resources* (hrv. *resursi)* mapi se nalaze svi pogledi, tj. *blade* datoteke koje sadrže HTML elemente te u većini slučajeva i podatke koje im prosljeđuje kontroler. Nalaze se i datoteke koje se koriste za jezike u slučaju da je aplikacija višejezična. U slučaju aplikacije *TheHunt* u toj mapi se nalaze i predlošci za e-poštu.



Sl. 3.8. Popis svih pogleda u izrađenoj aplikaciji i predložaka za e-poštu

3.8. Rute

Kako bi korisnici imali mogućnost navigacije kroz aplikaciju te slanja i dohvaćanja podataka, u aplikaciji moraju biti registrirane rute. Rute navodimo u *web.php* datoteci. S obzirom da ruta ima puno te bi slika zauzela cijelu stranicu, na slici 3.9. prikazan je samo dio njih.

```
// ROUTES ONLY FOR REGISTERED USERS WHICH HAVE FILLED BASE INFO
Route::group(['prefix' => '/', 'namespace' => 'Frontend', 'as' => 'frontend.', 'middleware' => ['verified','auth','steps']], function () {

    //HOME
    Route::get('/', 'HomeController@index')->name('home');
    Route::get('/faq', 'HomeController@faq')->name('faq');
    Route::get('/about-us', 'HomeController@about')->name('about');

    //PROFILE COMPLETION STEPS
    Route::get('step-1', 'StepController@getStepOne')->name('getStepOne');
    Route::get('step-2', 'StepController@getStepTwo')->name('getStepTwo');
    Route::post('post-step-2', 'StepController@postStepTwo')->name('postStepTwo');


    //COMMUNITY
    Route::get('users', 'CommunityController@index')->name('getUsers');
    Route::get('show-followers', 'CommunityController@show_followers')->name('showFollowers');
    Route::get('show-following', 'CommunityController@show_following')->name('showFollowing');
    //FOLLOW_UNFOLLOW
    Route::post('follow-unfollow/{id}', 'CommunityController@follow_unfollow')->name('followUnfollow');
    Route::any('users-filter', 'CommunityController@users_filter')->name('usersFilter');
    Route::any('followers-filter', 'CommunityController@followers_filter')->name('followersFilter');
    Route::any('following-filter', 'CommunityController@following_filter')->name('followingFilter');
```

Sl. 3.9. Dio ruta iz aplikacije *TheHunt*

Kako bi definirali rutu, moramo navesti koja metoda se izvršava na toj ruti. Metoda *get* (hrv. *dohvati*) služi za dohvat podataka ako npr. želimo prikazati početnu stranicu. Metoda *post (*hrv. *pošalji*) služi za slanje podataka na server ako želimo npr. slati podatke o registraciji i spremati ih u bazu podataka. Laravel nam nudi i opciju *resource* (hrv. *resurs*) s kojom registriramo grupu ruta sa svim metodama za manipulaciju resursa [18]. Korisnik može proizvoljno nazivati rute te im kreirati putanju. Također može navoditi koji kontroler upravlja kojim rutama te koja metoda unutar kontrolera se izvršava na toj ruti. Osim toga, rute mogu imati i neobavezne (engl. *optional*) parametre koje primaju što je korisno ako se jedna ruta koristi za više različitih podataka. To se može predočiti stranicom za pregled profila. Ovisno koji parametar pošaljemo u ruti, dobiti ćemo podatke o tom korisniku. Rute mogu biti i zaštićene koristeći *middleware*. *Middleware* možemo smatrati poveznicom između zahtjeva i odgovora. Na primjeru sa slike 3.9. vidimo da u *middleware* polju koristimo *auth* kao jedan od mnogih *middleware-a*. On nam služi kako bi provjerili je li korisnik prijavljen u našu aplikaciju, te sve rute koje su zaštićene s njim neće biti dostupne za pregled i korištenje ako korisnik nije autoriziran.

4. KORISNIČKA SUČELJA

Kako bi korisnici mogli koristiti aplikaciju i njene mogućnosti, moraju imati pristup određenim stranicama. Korisnici mogu pristupiti stranicama pod uvjetom da su registrirani i da im je registracija potvrđena (e-pošta). Taj uvjet ne uključuje stranice za prijavu i registraciju. Sučelja su opisana redom kako se nalaze i na navigacijskoj traci s iznimkama za stranice na koje se dolazi isključivo s druge stranice.

4.1. Navigacijska traka i podnožje

Jedan od glavnih dijelova sučelja je navigacijska traka koja sadrži logotip aplikacije i poveznice ili padajuće izbornike s dodatnim poveznicama na određene stranice. Navigacijska traka je vidljiva i dostupna sa svih stranica unutar aplikacije.

Sl. 4.1. Navigacijska traka

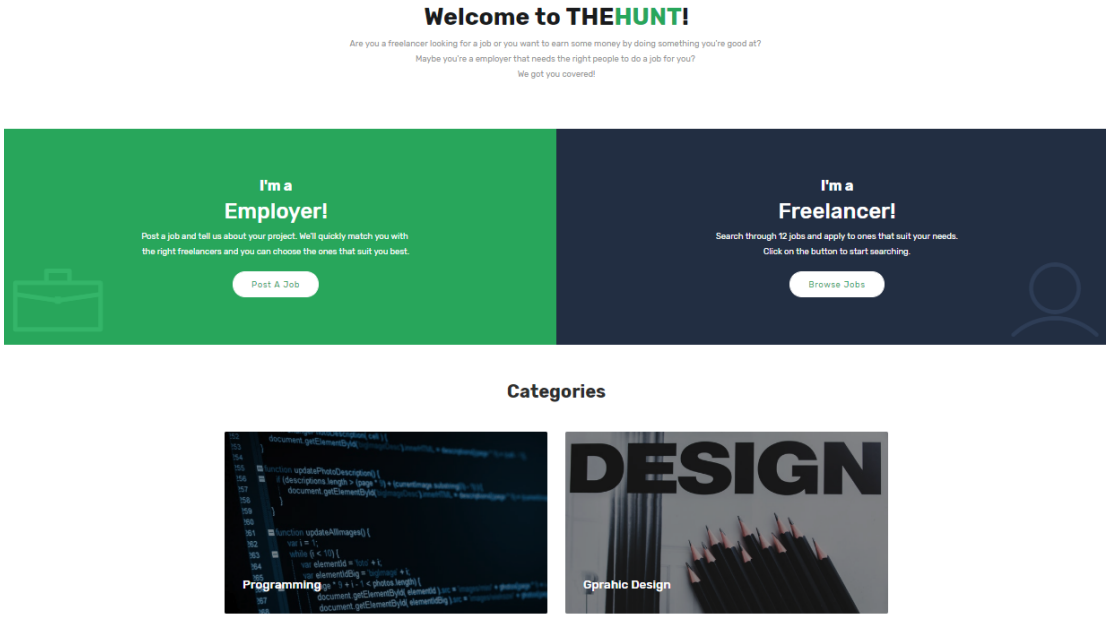
Pojedine stavke u navigacijskoj traci će biti detaljnije opisane kroz naredna poglavlja.

Drugi dio aplikacije koji je vidljiv i dostupan sa svih stranica je podnožje (*engl. footer*). Sadrži kratki opis, korisne poveznice, poveznice na društvene mreže te kontakt informacije i informaciju o studentu koji je izradio aplikaciju.

Sl. 4.2. Izgled podnožja

4.2. Home

*Home (*hrv. *dom*) ili početna stranica služi za pregled osnovnih informacija o aplikaciji kao što su zadnje objavljeni oglasi, broj registriranih korisnika, ukupan broj prijava na poslove i slično. Također sadrži poveznice na stranice za kreiranje oglasa za posao ili pretraživanje oglasa. S obzirom na to da na početnoj stranici postoji puno sadržaja, na slici 4.3. je prikazan samo dio stranice.

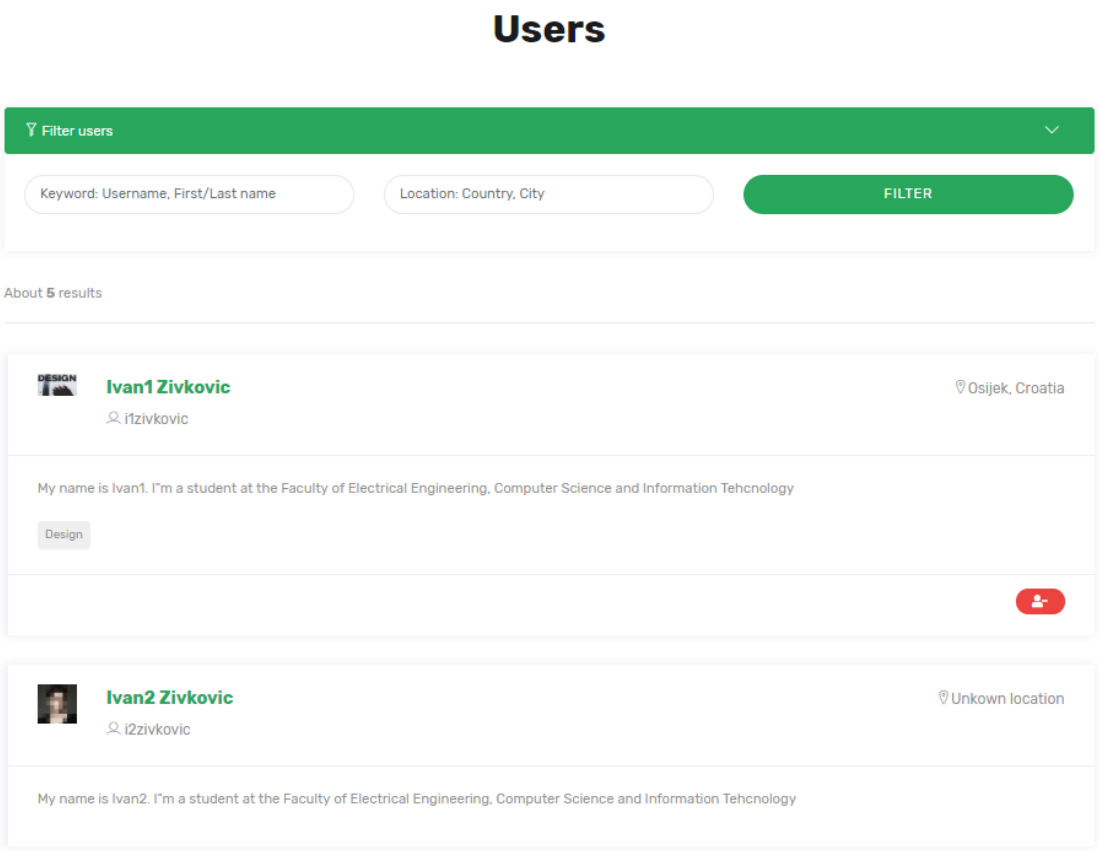


Sl. 4.3. Dio početne stranice

4.3. Community

Community (hrv. *zajednica)* predstavlja padajući izbornik u kojemu se nalaze poveznice na stranice za pregled i pretraživanje korisnika aplikacije (engl. *Users*), pregled korisnika koji prate trenutno prijavljenog korisnika (engl. *Followers*) i prikaz korisnika koje prijavljeni korisnik prati (engl. *Following*).

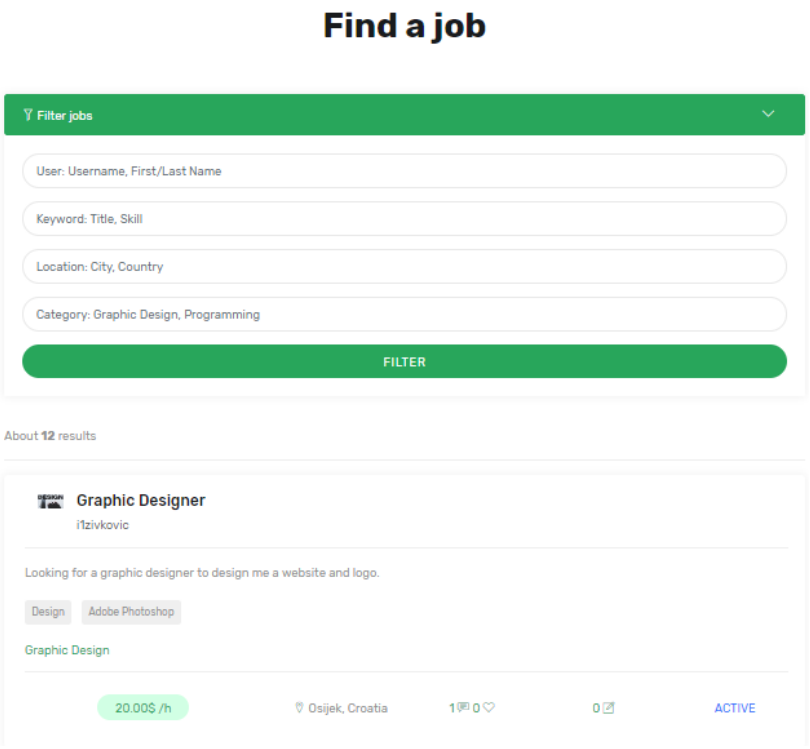
Na svakoj od tih stranica postoji filter po kojem se podaci na stranici mogu filtrirati te opcije za praćenje korisnika ili prestanak praćenja (opcije nisu dostupne nad vlastitim profilom). Izgledom su stranice identične samo prikazani podaci ovise kako je i objašnjeno.



Sl. 4.4. Prikaz *Users* (hrv. *korisnici*) stranice

4.4. Work

Work (hrv. *rad/posao)* predstavlja padajući izbornik u kojemu se nalaze poveznice na stranice za pretraživanje oglasa za posao (engl. *Find a job*), pregled prijava za korisnika koji je trenutno prijavljen u aplikaciju (engl. *My applications*) i stranica za pregled ocjena koje je korisnik dobio nakon odrađenog posla (engl. *My ratings*).

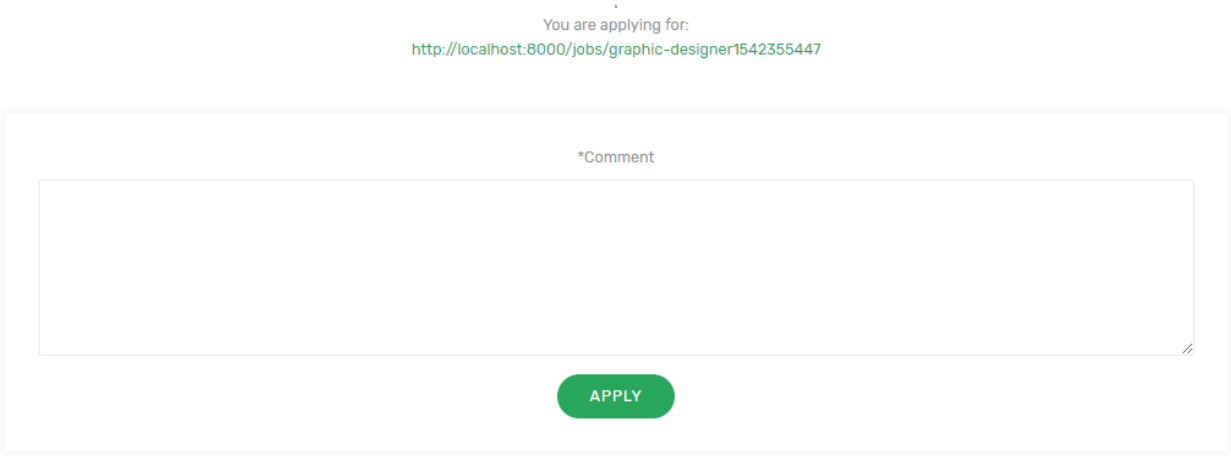


Sl. 4.5. Dio prikaza stranice *Find a job (*hrv. *pronađi posao*)

Korisnik na ovoj stranici može pretraživati oglase za poslove te klikom na naslov oglasa je preusmjeren na stranicu za prikaz detalja o tom poslu (engl. *Job details*). Na toj stranici korisnik može vidjeti detalje o poslu, komentirati posao, sviđati (engl. *like*) te se prijaviti na posao ili odjaviti ako je posao ili prijava u određenom stanju. U slučaju prijave, korisnik je preusmjeren na stranicu za prijavu na kojoj se nalazi forma za prijavu a poslodavac je obaviješten e-poštom.

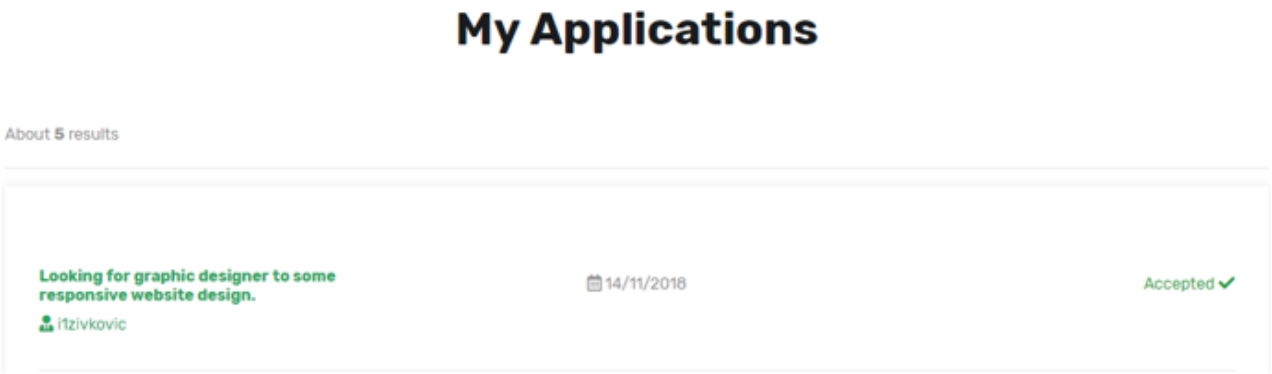
Posao može imati jedno od četiri moguća stanja:

- *Active* (hrv. *aktivan*) – posao je aktivan i otvoren za prijave
- *Unavailable* (hrv. *nedostupan*) – posao je trenutno nedostupan
- *In progress* (hrv. *u procesu*) – posao je zaključan za prijave i trenutno je u procesu odrađivanja
- *Done* (hrv. *gotov/završen/obavljen*) – posao je obavljen.



Sl. 4.6. Prikaz forme za prijavu na posao

My applications (hrv. *moje prijave*) je stranica koja služi za pregled prijava na oglase za korisnika koji je trenutno prijavljen u aplikaciju. Na stranici su prikazane informacije o naslovima poslova, datumima prijava te njihovim stanjima. Korisnik je obaviješten putem e-pošte o promjeni stanja prijave.



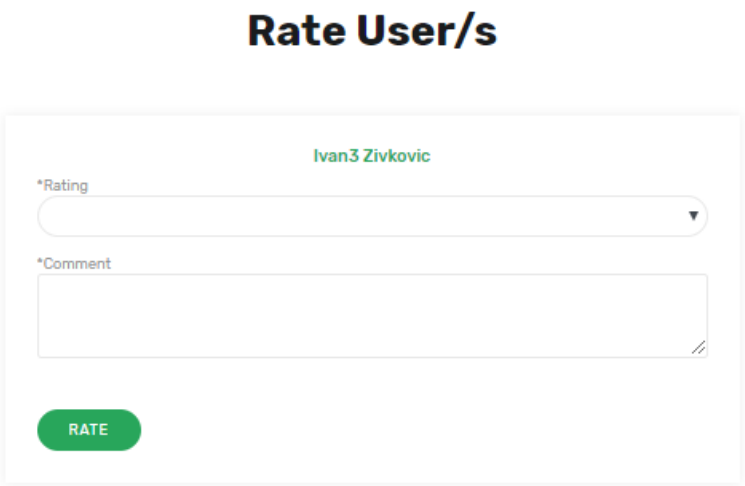
Sl. 4.7. Dio prikaza *My applications* (hrv. *moje prijave*) stranice

My ratings (hrv. *moje ocjene*) je stranica koja služi za pregled ocjena koje je korisnik dobio od drugih korisnika nakon odrađenog posla bilo kao poslodavac ili zaposlenik. Na stranici su prikazane informacije o korisniku koji je dao ocjenu, datumu, ocjeni, komentaru te poveznici na posao na koji se odnosi.

Sl. 4.8. Dio prikaza *My ratings* (hrv. *moje ocjene*) stranice

Kako bi se korisnici mogli međusobno ocijeniti, posao mora biti u stanju *Done* (hrv. *odrađen/završen/obavljen*). Promjena stanja kao i uređivanje informacija o oglasu za posao se vrši na stranici za uređivanje (engl. *Edit job)* na koju korisnik može doći ili sa stranice o detaljima oglasa ili sa stranice za pregled svojih oglasa za posao (engl. *My jobs)* (opisana u poglavlju 4.5.). Stranica za uređivanje je izgledom identična kao i stranica za objavu oglasa (engl. *Post a job ad)* (opisana u poglavlju 4.5.) no razlika je u tome što su podaci u formi već ispunjeni ovisno o oglasu koji se uređuje. Korisnik je nakon spremanja promjena obaviješten o uspješnosti ili greškama prilikom uređivanja.

Ako je korisnik postavio stanje oglasa u gotovo/završeno, tada on ima mogućnost ocjenjivanja korisnika koji su bili prihvaćeni na taj posao tako što je preusmjeren na stranicu *Rate user/s* (hrv. *ocijeni korisnika/e*) dok su ti korisnici obaviješteni putem e-pošte s porukom o mogućnosti da ocijene poslodavca te poveznicom na stranicu za ocjenjivanje. Korisnici na toj stranici za svakog korisnika kojega ocjenjuju unose ocjenu (engl. *rating*) i komentar (engl. *comment*).



Sl. 4.9. Prikaz *Rate user/s* (hrv. *ocijeni korisnika/e*) stranice

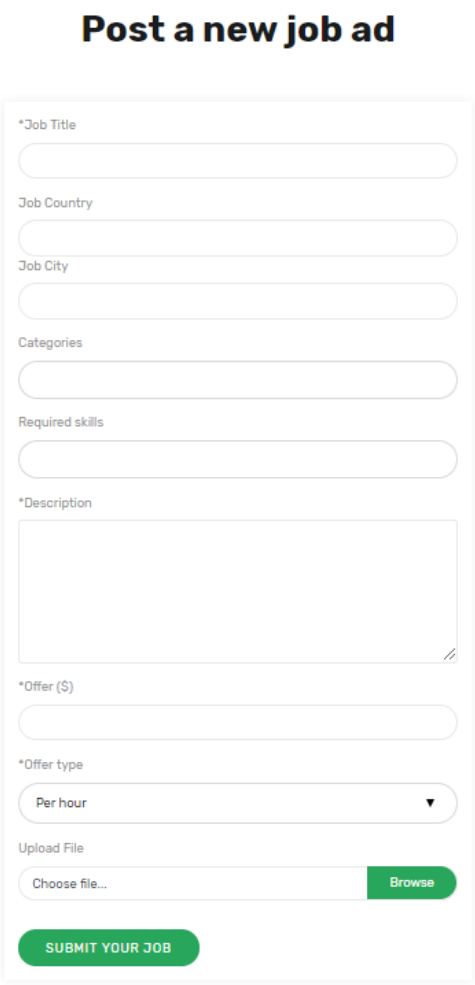
4.5. Employ

Employ (hrv. *zaposli*) predstavlja padajući izbornik s poveznicama na stranice za objavu oglasa za posao (engl. *Post a job ad)*, pregled objavljenih oglasa za poslove (engl. *My job ads)* i stranicu za pregled i upravljanje prijavama nad oglasima koje je korisnik objavio (engl. *Manage applications)*.

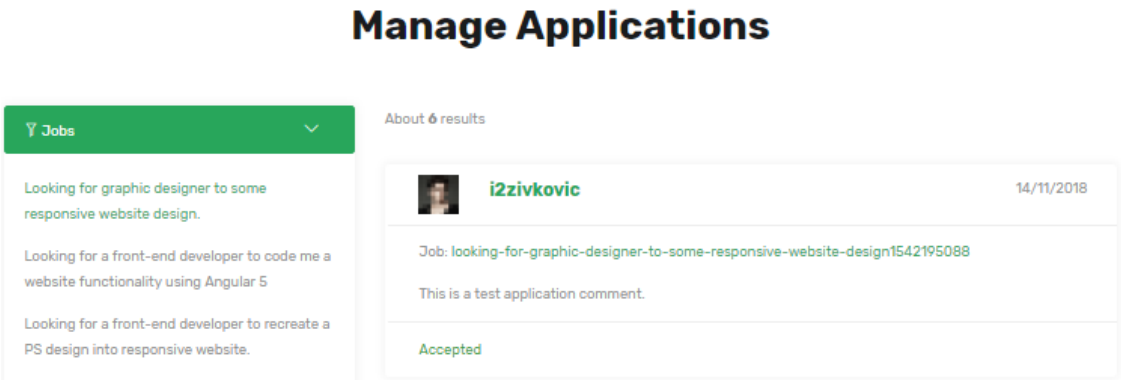
Na stranici za objavu oglasa za posao se nalazi forma gdje korisnik mora ispuniti zahtijevana polja koja su označena zvjezdicom (*). Korisnik je također nakon objave obaviješten o uspješnosti i preusmjeren na detalje o oglasu ili u suprotnom ako su nastale greške, vraćen na stranicu za objavu oglasa s obavijestima o greškama. Moguće je priložiti samo jednu datoteku (preporučljivo *.zip*).

Stranica koja služi za pregled oglasa koji je korisnik objavio (engl. *My job ads)* izgledom je identična stranici za pretragu oglasa za poslove (engl. *Find a job)* koja je opisana u prošlom poglavlju.

Manage applications (hrv. *upravljaj prijavama*) stranica služi za pregled prijava na oglase za posao koje je korisnik objavio. Svaka prijava sadrži informacije o komentaru prijave, poslu na koji se korisnik prijavio, stanju (ako je prihvaćena ili odbijena) ili mogućnostima prihvaćanja ili odbijanja prijave, datumu prijave i naravno korisničko ime korisnika i profilnu sliku.



Sl. 4.10. Prikaz stranice za objavu oglasa za posao



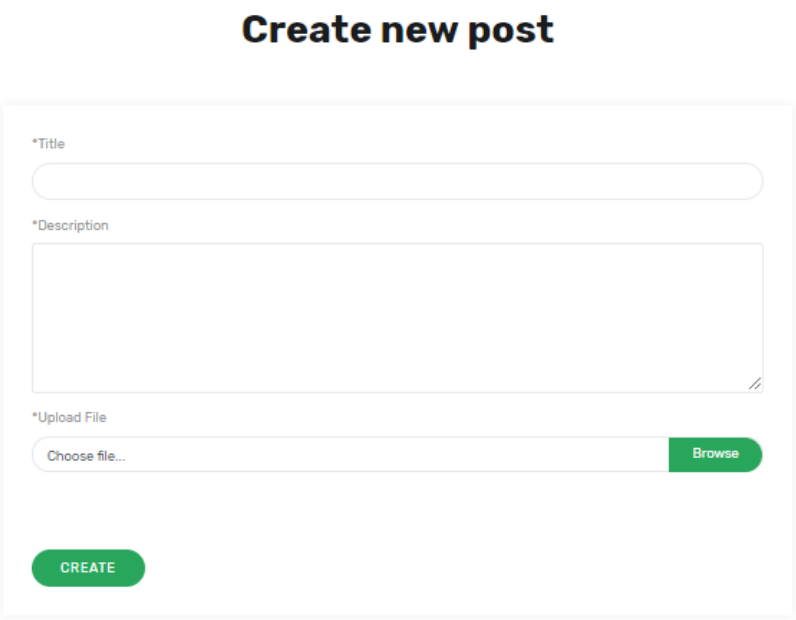
Sl. 4.11. Dio prikaza stranice za pregled prijava

4.6. Posts

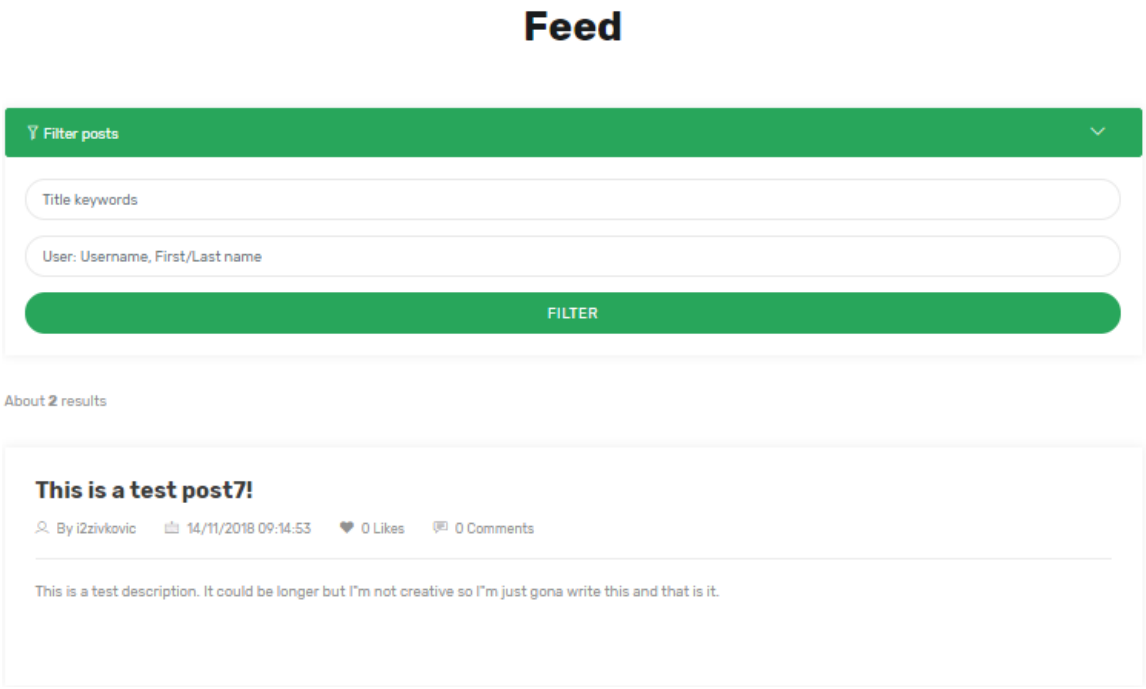
Posts (hrv. *objave*) predstavlja padajući izbornik koji sadrži poveznice na stranice za kreiranje objave (eng. *Create new post)*, pregled objava od strane korisnika koje prijavljeni korisnik prati (engl. *Feed)*, pregled svih objava na razini aplikacije i (engl. *Explore)* i stranicu za pregled osobnih objava *(*engl. *My posts*). Trenutno su objave zamišljene kao objava projekata ili portfelja s naslovom, opisom i priloženom datotekom.

Na stranici za kreiranje objave se nalazi forma s poljima za naslov, opis i dodavanje datoteke. Korisnik je obaviješten o uspješnosti objave ili o greškama nakon što pokuša kreirati objavu. Moguće je dodati jednu datoteku po objavi (preporučljivo *.zip*).

Sve tri stranice za pregled objava su izgledom identične. Također postoji filter na svakoj od stranica po kojemu se objave mogu filtrirati. Na svakoj od tih stranica korisnik može kliknuti na naslov objave pri čemu je preusmjeren na detalje o objavi gdje može vidjeti detalje, komentirati objavu ili sviđati.



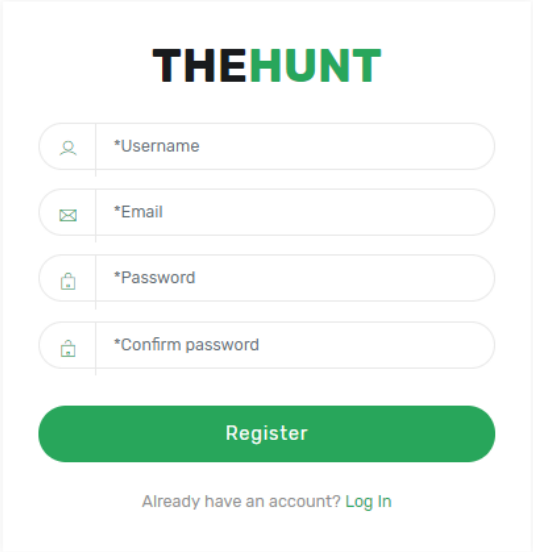
Sl. 4.12. Dio stranice za kreiranje objave



Sl. 4.13. Dio prikaza stranice za pretragu objava od strane korisnika koje korisnik prati

4.7. Log in i Register

Log in (hrv. *prijava*) i *Register* (hrv. *registracija*) predstavljaju poveznicu na stranicu na kojoj korisnik može izraditi korisnički račun koji mu služi za prijavu u aplikaciju te poveznicu na stranicu na kojoj se korisnik prijavljuje u aplikaciju. Bez potvrđenog korisničkog računa korisnik nema mogućnost pregleda ostalih stranica a samim time ni mogućnost korištenja funkcionalnosti aplikacije. Nakon što se korisnik uspješno registrira i ispuni osnovne informacije, preusmjeren je na stranicu na kojoj je prikazana obavijest o slanju e-pošte za potvrdu računa. Korisnici moraju potvrditi registraciju iz jednostavnog razloga a to je da drugi korisnici koji namjerno ili slučajno koriste tuđe adrese e-pošte ne mogu pristupiti aplikaciji jer neke od funkcionalnosti uključuju slanje obavijesti u obliku e-pošte. U slučaju da ne postoji potvrda, korisnici bi na taj način mogli drugim osobama koje možda ni ne znaju za ovu web aplikaciju stvarati neželjenu poštu (engl. *spam*) u sandučiću.



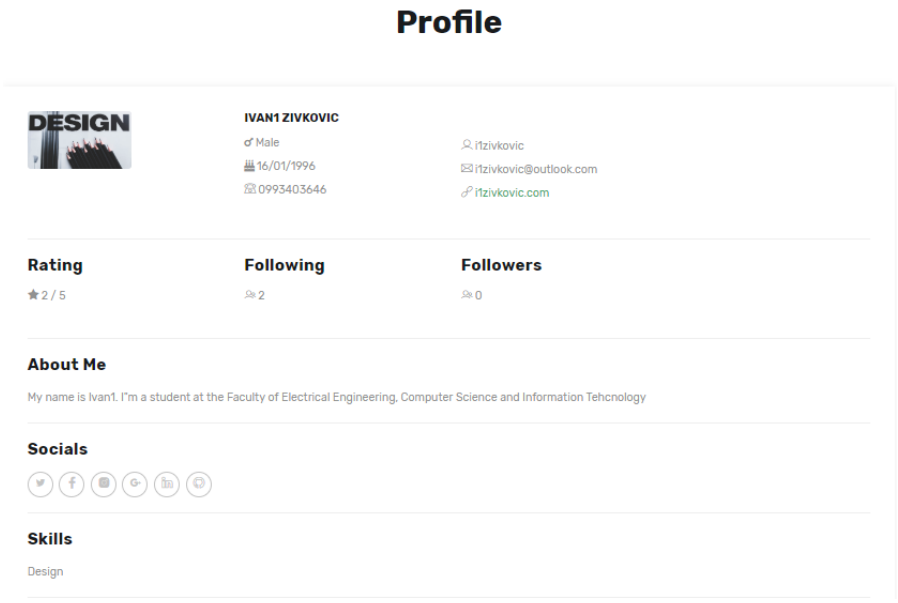
Sl. 4.14. Forma za registraciju korisnika

Kako bi se korisnik uspješno registrirao, mora ispuniti polja prikazana na slici 4.14. a to su korisničko ime, adresa e-pošte, željenu lozinku te potvrdu lozinke. Korisnici se ne mogu registrirati više puta s istim korisničkim imenima ili adresama e-pošte.

Stranica za prijavu korisnika je izgledom identična stranici za registraciju osim što sadrži samo dva polja u formi a to su polje za unos adrese e-pošte i polje za unos lozinke s kojima se korisnik registrirao. Nakon uspješne prijave, korisnik je preusmjeren na početnu stranicu.

4.8. Profil

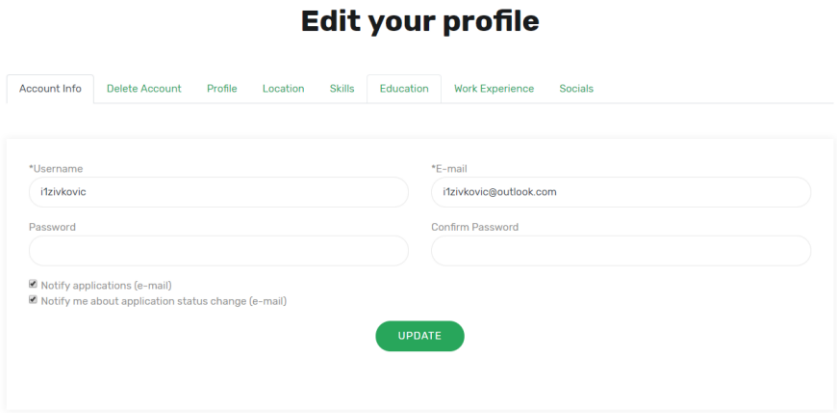
Kada se korisnik uspješno prijavi u aplikaciju, umjesto poveznica na stranice za registraciju i prijavu koje se nalaze u navigacijskoj traci, prikazana mu je padajuća lista s nazivom njegovog korisničkog imena i poveznicama na stranice za pregled osobnog profila (engl. *My profile)*, uređivanje profila (engl. *Edit profile)* i poveznicu na čiji se klik korisnik odjavljuje iz aplikacije (engl. *Logout)*.



Sl. 4.15. Dio prikaza korisničkog profila

S obzirom na to da na stranici za pregled profila može biti puno informacija na slici 4.15. je prikazan samo dio njih. Ako je korisnik dodao informacije o svojoj lokaciji, na njegovom profilu je ona prikazana u obliku Google karte (engl. *Google maps*). Također, na dnu profila se nalazi forma za kontaktiranje korisnika i to u slučaju da korisnik pregledava tuđi profil. Isto tako pri pregledavanju tuđeg profila postoje opcije za praćenje korisnika (engl. *follow*) i prestanak praćenja (engl. *unfollow*).

Korisnik može urediti profil na stranici *Edit profile* (hrv. *uredi profil*), obrisati svoj korisnički račun te uključiti ili isključiti obavijesti (e-pošta).



Sl. 4.16. Stranica za uređivanje profila

5. PROGRAMSKO RJEŠENJE

Kroz sljedećih par poglavlja ćemo ukratko proći kroz programska rješenja aplikacije te objasniti kôd. S obzirom na to da aplikacija sadrži puno funkcionalnosti, prikazati ćemo samo najbitnije od njih.

5.1. Registracija korisnika

Kako bi se korisnici mogli registrirati na aplikaciju, potrebno je kreirati kontroler i metodu u kojoj će biti implementirana logika te mora postoji tablica u bazi podataka u koju će se spremati podaci. Laravel nam nudi vrlo jednostavan način za kreiranje funkcionalnosti vezanih za registraciju i prijavu korisnika te stvaranje osnovnih tablica u bazi podataka koje su potrebne za to. Naredbom „*php artisan make:auth*“ generiraju se svi potrebi kontroleri, migracije, pogledi i slično. Uz to se kreira i grupa ruta koja nam omogućuje korištenje tih funkcionalnosti. Korisnik po želji može prilagoditi sve po njegovim potrebama.

Na slici 5.1. možemo vidjeti primjer kôda koji je zaslužan za registraciju korisnika.

```
    /**
     * Get a validator for an incoming registration request.
     *
     * @param  array  $data
     * @return \Illuminate\Contracts\Validation\Validator
     */
    protected function validator(array $data)
    {
        return Validator::make($data, [
            'username' => 'required|string|min:5|max:20|
            unique:users|alpha_dash',
            'email' => 'required|string|email|max:255|
            unique:users',
            'password' => 'required|string|min:6|confirmed',
        ]);
    }

    /**
     * Create a new user instance after a valid registration.
     *
     * @param  array  $data
     * @return \App\User
     */
    protected function create(array $data)
    {

        $slug = str_slug($data['username'], '-');

        return User::create([
            'role_id' => 2,
            'username' => $data['username'],
            'slug' => $slug.'-'.time(),
            'email' => $data['email'],
            'password' => Hash::make($data['password'])
        ]);
    }
```

Sl. 5.1. Metoda za registraciju korisnika

U primjeru vidimo metodu *validator* koja prima polje podataka te vrši validaciju. Na primjer, lozinka (engl. *password*) je zahtijevano polje, mora sadržavati najmanje 6 znakova te mora biti potvrđena (zato služi dodatno polje pri registraciji).

Metoda *create (*hrv. *stvori*) također prima polje podataka te koristi *User (*hrv. *korisnik*) model podataka i metodu *create* nad tim modelom kako bi kreirala novog korisnika aplikacije.

5.2. Kreiranje oglasa za posao

Nakon što korisnici ispune podatke o oglasu za posao te pritisnu dugme za objavu, podaci se šalju na server gdje se šalju metodi *store (*hrv. *pohrani)* unutar *JobController*-a.

```php
    /**
     * Store a newly created job in database.
     *
     * @param  \Illuminate\Http\Request  $request object containing info about new job
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {

        // Validation rules
        $rules = [
            'title' => 'required|max:200|min:10',
            'job_location_country' => 'max:200|min:2',
            'job_location_city' => 'max:100|min:2',
            'description' => 'max:1000|min:50',
            'offer' => 'required|numeric'
        ];

        // make validator
        $validator = Validator::make($request->all(), $rules);

        // check if validation success
        if ($validator->fails()) {
            return back()->withErrors($validator)->withInput();
        }

        // create job
        $job = Job::create($request->except(['business_category_id','skill_list'])+['slug' => str_slug($request->title, '-').time(), 'user_id' => Auth::id(), 'job_status_id' => 1]);


        // create relations
        $arrCategories = [];
        $arrSkills= [];
        $categories = $request->business_category_id;
        $skills = $request->skill_list;
        $now = Carbon::now();

        for ($i = 0; $i < count($categories); $i++) {
            array_push($arrCategories, ['business_category_id' => $categories[$i], 'job_id' => $job->id, 'created_at' => $now, 'updated_at' => $now]);
        }
        for ($i = 0; $i < count($skills); $i++) {
            array_push($arrSkills, ['skill_id' => $skills[$i], 'job_id' => $job->id, 'created_at' => $now, 'updated_at' => $now]);
        }

        JobBusinessCategory::insert($arrCategories);

        JobSkill::insert($arrSkills);


        // upload file if exists
        if(!empty($request->file('file'))){
            $file = $this->uploadFile($request->file('file'), Auth::user()->username, $job->id);
            $job->job_files()->create(['path' => $file]);
        }

        // redirect back to job
        return redirect()->to('jobs/'.$job->slug);
    }
```

Sl. 5.2. Metoda za objavu oglasa za posao

Nad podacima se prvo vrši validacija ovisno o pravilima koje smo postavili. Ako validacija nije prošla uspješno, korisnik je preusmjeren nazad na tu stranicu s porukama o greškama.

Nakon toga se kreira unos u tablicu *jobs (*hrv. *poslovi*) koristeći *Job* model. S obzirom na to da u oglasu možemo postaviti i zahtijevane vještine i kategoriju posla a ti podaci se spremaju u posebne tablice, nakon kreiranja unosa u tablicu *jobs* kreiramo polje podataka za kategorije i vještine s potrebnim podacima te ih unosimo u tablice metodom *insert (*hrv. *unesi*) koristeći odgovarajuće modele podataka. Nakon toga, ako je korisnik pridružio datoteku, ona se prenosi i sprema u poseban folder s putanjom *public/uploads/*[*korisničko_ime*]*/jobs/*[*id_posla]/*.

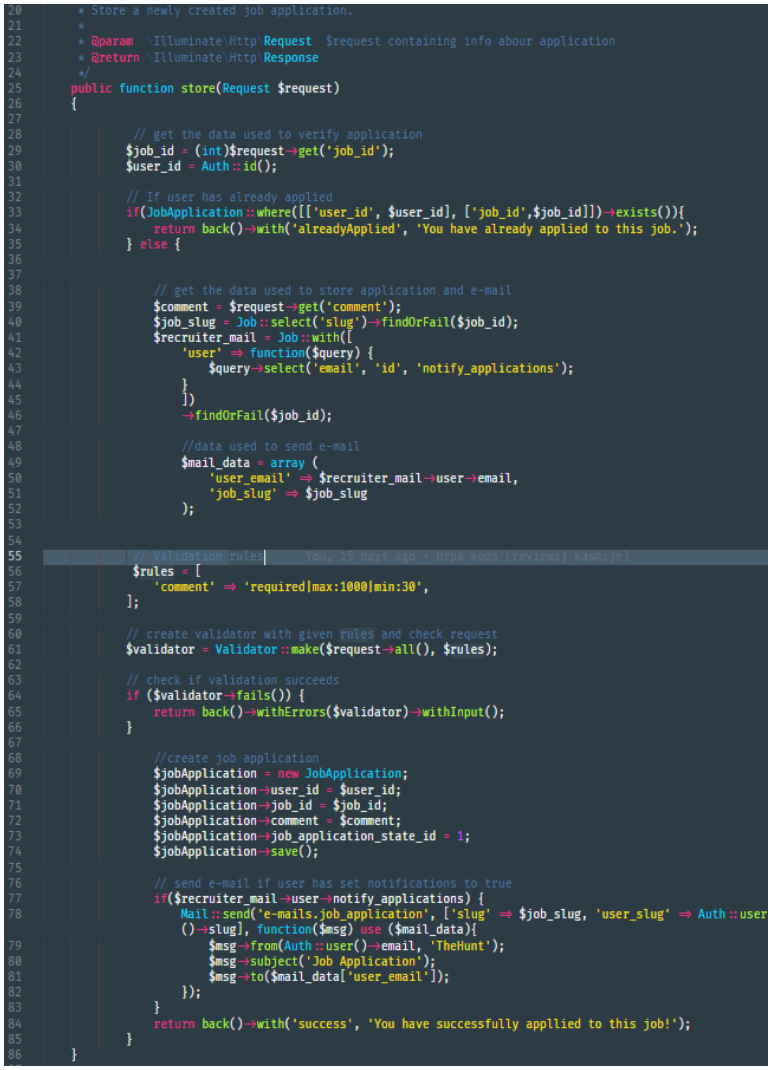
```php
/**
 * Method used to upload files
 * @param $file file
 * @param $folder name of the folder
 */
public function uploadFile($file, $username, $job_id){

    // if folder does not exists, create one with all persmissions
    if (!is_dir(public_path().'/uploads/'.$username.'/jobs/'.$job_id)) {
        mkdir(public_path().'/uploads/'.$username.'/jobs/'.$job_id, 0777, true);
    }

    // get the destination path
    $destinationPath = public_path().'/uploads/'.$username.'/jobs/'.$job_id.'/';

    // get the file name and create new one with timestamp
    $file_name = time().'-'.$file->getClientOriginalName();

    // move files / create file in $folder
    $file->move($destinationPath, $file_name);

    // return file
    return $file_name;
}
```

Sl. 5.3. Metoda za prijenos datoteke

Metoda za prijenos datoteke *uploadFile* prima podatke o datoteci, korisničkom imenu i identifikacijskom broju posla. Prvo provjerava postoji li mapa u koju će se spremiti datoteka (moguće je da je korisnik naknadno obrisao datoteku pri uređivanju oglasa). Ako ne postoji, kreira novu mapu sa svim pravima pristupa, čitanja i izvršavanja. Nakon toga u varijablu *$destinationPath* spremam putanju do te mape. U varijablu *$file_name* spremamo naziv datoteke koristeći vremensku oznaku u kombinaciji s originalnim nazivom datoteke te sljedećom naredbom datoteku prenosimo u mapu i vraćamo ju kao rezultat radnje.

5.3. Prijava na oglas

Prijava na oglas za posao se vrši tako da korisnik sa stranice o detaljima o oglasu pritisne dugme za prijavu pri čemu je preusmjeren na stranicu koja sadrži formu za prijavu. U formi postoji jedno zahtijevano polje, a to je *comment* (hrv. *komentar*). Nakon toga je korisniku prikazan poruka o uspjehu ili nastalim greškama.

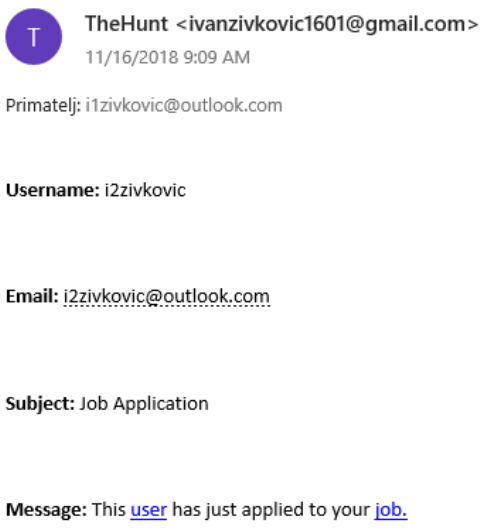


```
    * Store a newly created job application.
    *
    * @param  \Illuminate\Http\Request  $request containing info abour application
    * @return \Illuminate\Http\Response
    */
    public function store(Request $request)
    {

            // get the data used to verify application
            $job_id = (int)$request->get('job_id');
            $user_id = Auth::id();

            // If user has already applied
            if(JobApplication::where([['user_id', $user_id], ['job_id',$job_id]])->exists()){
                return back()->with('alreadyApplied', 'You have already applied to this job.');
            } else {


                // get the data used to store application and e-mail
                $comment = $request->get('comment');
                $job_slug = Job::select('slug')->findOrFail($job_id);
                $recruiter_mail = Job::with([
                    'user' => function($query) {
                        $query->select('email', 'id', 'notify_applications');
                    }
                    ])
                    ->findOrFail($job_id);

                    //data used to send e-mail
                    $mail_data = array (
                        'user_email' => $recruiter_mail->user->email,
                        'job_slug' => $job_slug
                        );


                // Validation rules:     You, 15 days ago • Hrpa koda (reviewaj kasnije)
                $rules = [
                    'comment' => 'required|max:1000|min:30',
                ];

                // create validator with given rules and check request
                $validator = Validator::make($request->all(), $rules);

                // check if validation succeeds
                if ($validator->fails()) {
                    return back()->withErrors($validator)->withInput();
                }

                    //create job application
                    $jobApplication = new JobApplication;
                    $jobApplication->user_id = $user_id;
                    $jobApplication->job_id = $job_id;
                    $jobApplication->comment = $comment;
                    $jobApplication->job_application_state_id = 1;
                    $jobApplication->save();

                    // send e-mail if user has set notifications to true
                    if($recruiter_mail->user->notify_applications) {
                        Mail::send('e-mails.job_application', ['slug' => $job_slug, 'user_slug' => Auth::user
                        ()->slug], function($msg) use ($mail_data){
                            $msg->from(Auth::user()->email, 'TheHunt');
                            $msg->subject('Job Application');
                            $msg->to($mail_data['user_email']);
                        });
                    }
                    return back()->with('success', 'You have successfully appllied to this job!');
                }
    }
```

Sl. 5.4. Metoda za prijavu na oglas

Metoda prima podatke o prijavi o obliku *Request* (hrv. *zahtjev*) objekta. Iz tog objekta čitamo podatke o identifikacijskom broju posla na koji se korisnik prijavljuje te u varijablu *$user_id* spremamo

identifikacijski broj korisnika koji se prijavljuje na posao koristeći *Auth facade*. *Facade* predstavlja statično sučelje prema ugrađenim klasama u Laravelu [19]. Nakon toga provjeravamo je li se korisnik već prijavio na taj posao zbog dodatne sigurnosti. Ako je, vraćen je na stranicu za prijavu s porukom o grešci. Ako nije, dohvaćamo komentar, *slug* za prijavljeni posao čime predstavljamo dio putanje do tog posla i adresu e-pošte poslodavca. Nakon toga se podaci za kreiranje e-pošte spremaju u polje podataka *$mail_data* i vrši se validacija komentara. Ako je validacija uspješna, kreira se nova prijava na posao koristeći *JobApplication* model podataka te se provjerava ima li poslodavac uključene obavijesti o prijavama na posao. Ako ima, na njegovu adresu e-pošte se šalje obavijest, u suprotnom se taj korak preskače. Slanje e-pošte se vrši korištenjem Mail *facade-a* i metode *send* (hrv. *pošalji)* kojoj predajemo predložak za e-poštu kao parametar, te joj možemo predati i podatke koje ćemo koristiti u predlošku. Unutar metode navodimo naslov, primatelja i pošiljatelja.



Sl. 5.5. Primjer e-pošte za obavijest o prijavi na oglas

Obavijest sadrži informacije o korisničkom imenu korisnika koji se prijavio na posao, njegovu adresu e-pošte, naslov „*Job Application*“ (hrv. *prijava na posao*) i poruku koja sadrži poveznice na profil korisnika i oglas za posao na koji se korisnik prijavljuje.

5.4. Upravljanje prijavama

Poslodavci mogu upravljati prijavama sa stranica *Manage applications (*hrv. *upravljaj prijavamai)* i *Job details (*hrv. *detalji posla)*. Prihvaćanje ili odbijanje prijave se vrši klikom na odgovarajuće ikonice korištenjem AJAX-a. AJAX koristimo iz razloga da se stranica ne mora učitavati ponovno ako se dogodi greška nego samo u slučaju uspjeha te je korisniku prikazana povratna informacija odmah nakon izvršavanja radnje. S obzirom da se koristi AJAX tehnika za razmjenu podataka, podatke šaljemo i vraćamo u *JSON* (engl. *JavaScript Object Notation)* obliku.

Sl. 5.6. Ikonice za prihvaćanje ili odbijanje prijave

Na slici 5.7. je prikazana AJAX metoda *jobApplicationHandlerAjax* koja za parametre prima identifikacijski broj prijave te identifikacijski broj novog stanja prijave. Nakon toga se u zaglavlje zahtjeva (engl. *request*) postavlja *X-CSRF-TOKEN* vrijednost na vrijednost *token-a* koju svaki prijavljeni korisnik dobije automatski prilikom prijave i on se šalje zajedno sa zahtjevom. Služi za provjeru na serverskoj strani u smislu radi li zahtjev na server korisnik koji je registriran i prijavljen u aplikaciju ili netko drugi čime se vrši autorizacija [20]. Verifikacija *token-a* se vrši korištenjem *VerifyCsrfToken middleware-a*. *Token* možemo lako dohvatiti koristeći Jquery biblioteku za JavaScript tako što u HTML dokument stranice na kojoj se nalazimo dodamo oznaku *@csrf* čime stvaramo skriveno polje za unos podataka čija je vrijednost zapravo taj *token* ili u zaglavlje dokumenta postavimo meta podatak s nazivom „*csrf-token“* i vrijednosti *{{ csrf_token() }}*. S obzirom da koristimo *blade engine* za kreiranje stranica tj. pogleda, vrijednost unutar duplih vitičastih zagrada će se promijeniti u stvarnu vrijednost *tokena*.

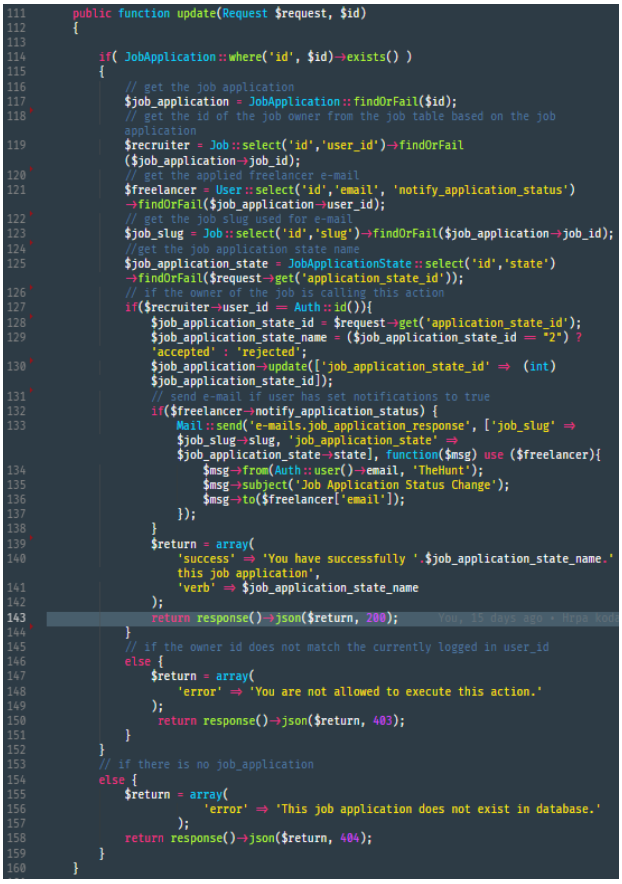
Nakon toga navodimo da želimo koristit AJAX tehniku tako što pozivamo *$.ajax* metodu kojoj predajemo putanju ili rutu na kojoj se izvršava metoda za prihvaćanje ili odbijanje prijava, metodu koju želimo izvršiti (get, post, put..), tip podataka koji šaljemo i podatke koje spremamo u JSON objekt. S obzirom na to da se AJAX izvršava asinkrono, metoda nam pruža i uzvratne mogućnosti (engl. *callback*) ili svojstva koja izvršavaju metode koje želimo u slučaju uspjeha ili greške na serveru. U ovom slučaju ako je sve prošlo u redu, pozivamo *success (*hrv. *uspjeh*) uzvratni poziv s metodom

koja će pokazati poruku u skočnom prozoru i klikom na dugme će se stranica ponovno učitati. U slučaju greške, prikazana je samo poruka u obliku skočnog prozora. Za skočni prozor se koristi metoda *swal* koja koristi SweetAlert biblioteku.

Logika za upravljanje prijavama na serveru se vrši u metodi *update (*hrv. *ažuriraj*) (Sl.5.8.) koja kao parametre prima podatke o novom stanju prijave i identifikacijski broj prijave. Podaci se šalju metodi klikom na odgovarajuću ikonicu. Nalazi se u *JobApplicationController*-u. Prvo se vrši provjera u smislu postoji li već unos u tablici *job_applications* koja je predstavljena modelom *JobApplication*. Ako ne postoji, AJAX-u su vraćeni podaci s porukom o grešci i statusom 404 što predstavlja nemogućnost servera da pronađe traženi resurs [21]. Dohvaćaju se podaci o poslu, točnije njegov identifikacijski broj i identifikacijski broj poslodavca, podaci o korisniku koji se prijavio na posao, dodatni podaci o poslu te o novom stanju posla. Nakon toga se vrši provjera u smislu vrši li promjenu stanja prijave korisnik koji je vlasnik tog posla. Ako ne vrši, AJAX-u su vraćeni podaci s porukom o grešci da nema pravo na tu radnju. U suprotnom, stanje prijave se mijenja ovisno o odabranoj opciji te se novo stanje sprema u bazu. Vrši se provjera u smislu ima li korisnik koji se prijavio na posao uključene obavijesti o promjeni stanja prijave. Ako ima, njemu se šalje obavijest u obliku e-pošte, u suprotnom se taj korak preskače. Nakon toga, ako je sve prošlo bez greške, AJAX metodi su vraćeni podaci s porukom o uspjehu i statusu 200 što se tumači kao uspjeh [21].

```
/**
 * Function used to update job application status
 * @param {*} application_id Id of the job application
 * @param {*} application_state_id State (3-Rejected / 2-Accepted)
 */
function jobApplicationHandlerAjax(application_id, application_state_id) {

    // Set x-csrf token to headers
    $.ajaxSetup({
        headers: {
            'X-CSRF-TOKEN': $("input[name=_token]").val()
        }
    });

    $.ajax({
        url: '/job-applications/' + application_id,
        type: 'PUT',
        dataType: 'JSON',
        data: {
            application_state_id: application_state_id
        },
        success: function (data) {

            console.log(data);
            // Display success message
            swalWithBootstrapButtons({
                type: 'success',
                title: 'Result:',
                text: 'Job application successfully ' + data.verb + '.',
                confirmButtonText: 'Reload',
                allowOutsideClick: false,
                allowEscapeKey: false
            }).then((result) => {
                if (result.value) {
                    location.reload();
                }
            });
        },
        error: function (error) {
            // Display error message
            swalWithBootstrapButtons({
                type: 'error',
                title: 'Oops ... ',
                text: '' + error.responseJSON.error,
            });
        }
    });
}
```

Sl. 5.7. Metoda za upravljanje prijavama na klijentskoj strani



```
    public function update(Request $request, $id)
    {

        if( JobApplication::where('id', $id)->exists() )
        {
            // get the job application
            $job_application = JobApplication::findOrFail($id);
            // get the id of the job owner from the job table based on the job
            application
            $recruiter = Job::select('id','user_id')->findOrFail
            ($job_application->job_id);
            // get the applied freelancer e-mail
            $freelancer = User::select('id','email', 'notify_application_status')
            ->findOrFail($job_application->user_id);
            // get the job slug used for e-mail
            $job_slug = Job::select('id','slug')->findOrFail($job_application->job_id);
            //get the job application state name
            $job_application_state = JobApplicationState::select('id','state')
            ->findOrFail($request->get('application_state_id'));
            // if the owner of the job is calling this action
            if($recruiter->user_id == Auth::id()){
                $job_application_state_id = $request->get('application_state_id');
                $job_application_state_name = ($job_application_state_id == "2") ?
                'accepted' : 'rejected';
                $job_application->update(['job_application_state_id' =>  (int)
                $job_application_state_id]);
                // send e-mail if user has set notifications to true
                if($freelancer->notify_application_status) {
                    Mail::send('e-mails.job_application_response', ['job_slug' =>
                    $job_slug->slug, 'job_application_state' =>
                    $job_application_state->state], function($msg) use ($freelancer){
                        $msg->from(Auth::user()->email, 'TheHunt');
                        $msg->subject('Job Application Status Change');
                        $msg->to($freelancer['email']);
                    });
                }
                $return = array(
                    'success' => 'You have successfully '.$job_application_state_name.'
                    this job application',
                    'verb' => $job_application_state_name
                );
                return response()->json($return, 200);      You, 15 days ago • Mrpa kod
            }
            // if the owner id does not match the currently logged in user_id
            else {
                $return = array(
                    'error' => 'You are not allowed to execute this action.'
                );
                 return response()->json($return, 403);
            }
        }
        // if there is no job_application
        else {
            $return = array(
                    'error' => 'This job application does not exist in database.'
                );
            return response()->json($return, 404);
        }
    }
```

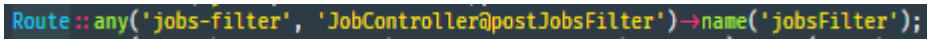
Sl. 5.8. Metoda za upravljanje prijavama na serverskoj strani

5.5. Pretraga oglasa

Pretraživanje oglasa za posao se vrši putem filtera na stranici koji ima oblik forme s odgovarajućim poljima za unos podataka. Nakon što korisnik unese odgovarajuće parametre za pretragu i pritisne dugme za pretraživanje, preusmjeren je na istu stranicu no sa podacima filtriranim prema unesenim parametrima. Na slici 5.8. je vidljiva konstrukcija forme u *blade* datoteci.

Sl. 5.9. Forma za pretragu oglasa

Forma je kreirana uz pomoć HTML elemenata za kreiranje strukture forme tj. polja za unos podataka, dugmeta, raspored istih i slično. Bootstrap klase su korištene kako bi se formi primijenio stil i prilagodljivi dizajn. Laravel Collection dodatni paket omogućuje lakše kreiranje i upravljanje formama jer možemo jednostavnije navoditi na koju rutu šaljemo podatke iz forme i slično. Za rute koristimo njihova imena, a ne stvarne adrese. *Blade engine* se koristi kako bi mogli u HTML dokumentu koristiti podatke iz modela koje šaljemo na tu stranicu. Kao što vidimo, poslovi se mogu filtrirati prema korisnicima, ključnim riječima, lokaciji i kategoriji. Nakon što korisnik ispuni te podatke, oni se šalju kroz rutu kao parametri na serversku stranu.



Sl. 5.10. Registrirana ruta u *web.php* datoteci koja služi za filtriranje oglasa

Na slici 5.10. vidimo primjer rute na koju se šalju podaci za filtriranje oglasa. Možemo primijetit kako koristimo *any* metodu. To je iz razloga što na tu rutu šaljemo podatke te metoda koja se izvršava na toj ruti nam isto tako vraća nazad podatke i preusmjerava na stranicu što je objašnjeno detaljnije niže u tekstu. Ruta koristi *JobController* kontroler i metodu *postJobsFilter* te ima naziv *jobsFilter*.

Sl. 5.11. Primjer adrese u pregledniku s parametrima za pretragu

Možemo primijetiti kako adresa u pregledniku sadrži naziv rute te parametre koji su povezani znakom & (engl. *and*). Svaka od vrijednosti parametara je pridružena nazivu parametra koji odgovara *name* (hrv. *ime*/naziv) atributu postavljenom na polje za unos podataka u formi za pretragu.

```php
    /**
     * Method which returns filtered jobs
     * @param Request $request Request object containing info about filter
     */
    public function postJobsFilter(Request $request){

        $jobs = Job::
        when($request->input('q'), function($query) use ($request) {
            return $query->where(function ($query) use ($request) {
            $query->where('title', 'like', '%'.$request->input('q').'%');
            $keywords = explode(' ', $request->input('q'));
                foreach ($keywords as $keyword) {
                    $query->orWhereHas('job_skills', function ($query) use ($keyword){
                        $query->join('skills', 'job_skills.skill_id', 'skills.id');
                        $query->select('skills.id', 'name', 'job_id');
                        $query->where('name', 'like', '%'.$keyword.'%');
                    });
                }
            });
        })
        ->when($request->input('user'), function($query) use ($request) {
            return $query->where(function ($query) use ($request) {
            $user_data = explode(' ', $request->input('user'));
                foreach ($user_data as $user_d) {
                    $query->orWhereHas('user.userProfile', function ($query) use ($user_d){
                        $query->select('*');
                        $query->where('username', 'like', '%'.$user_d.'%');
                        $query->orWhere('first_name', 'like', '%'.$user_d.'%');
                        $query->orWhere('last_name', 'like', '%'.$user_d.'%');
                    });
                }
            });
        })
        ->when($request->input('location'), function($query) use ($request) {
            return $query->where(function ($query) use ($request) {
            $locations = explode(' ', $request->input('location'));
                foreach ($locations as $location) {
                    $query->orWhere('job_location_country', 'like', '%'.$location.'%');
                    $query->orWhere('job_location_city', 'like', '%'.$location.'%');
                }
            });
        })
        ->when($request->input('category'), function($query) use ($request) {
            return $query->where(function ($query) use ($request) {
            $categories = explode(' ', $request->input('category'));
                foreach ($categories as $category) {
                    $query->orWhereHas('job_business_categories', function ($query) use ($category){
                    $query->join('business_categories', 'job_business_categories.business_category_id', 'business_categories.id');
                    $query->select('business_categories.id', 'name', 'job_id');
                    $query->where('name', 'like', '%'.$category.'%');
                });
                }
            });
        })
        ->with([
        'job_skills',
        'job_status',
        'job_business_categories',
        'user' => function($query){
            $query->select('id', 'username');
        }])
        ->withCount(['job_likes', 'job_comments'])
        ->orderBy('created_at','desc')
        ->paginate(5);

        return view('frontend.jobs', compact('jobs', 'request'));
    }
```

Sl. 5.12. Metoda za filtriranje oglasa na serverskoj strani

Metoda na serverskoj strani prima *$request* objekt koji sadrži parametre po kojima se oglasi filtriraju. Metoda vrši dohvaćanje svih poslova prema tome koji su parametri predani korištenjem *when* (hrv. *kada*) metode. U kôdu možemo primijetiti da imamo četiri *when* uvjeta što odgovara broju parametara po kojima možemo filtrirati. U slučaju da *$request* objekt sadrži taj parametar vrši se dodatno filtriranje oglasa po tom parametru. U slučaju da imamo parametar *user (*hrv. *korisnik)* kojemu smo predali vrijednosti korisničkog imena korisnika, imena i/ili prezimena, metoda će uzeti te vrijednosti i razdvojiti ih na ključne riječi umjesto da ih koristi kao rečenicu korištenjem *explode* metode. Nakon toga će za svaku riječ koju smo unijeli provjeriti postoji li u tablici *user_profile* (jer koristimo *user.userProfile* metodu koja predstavlja relaciju s *users* i *user_profiles* tablicom) korisnik s korisničkim imenom, imenom ili prezimenom koje u sebi sadrži neku od ključnih riječi tako što koristimo *like* (hrv. *kao*) ključnu riječ za pretragu. Slična stvar se događa i za druge parametre. Ako nismo naveli relacijsku metodu u modelu podataka (kao što smo naveli *userProfile* u *User* modelu), a podaci koje smo predali se nalaze u drugoj tablici koja je vezana za oglas, prilikom provjere za svaku ključnu riječ moramo tablicu povezati s odgovarajućom relacijskom.

Nakon što smo oglase filtrirali, pridružujemo im dodatne podatke korištenjem *with* (hrv. *s/sa)* metode koja prima polje relacijskih metoda koje dohvaćaju potrebne podatke iz drugih tablica za filtrirane oglase. Dodajemo i ukupan broj sviđanja i komentara korištenjem *withCount* metode kojoj predajemo isto tako relacijske metode koja nam vraća ukupan broj unosa iz odgovarajuće tablice vezane za taj oglas. Na kraju poslove posložimo prema datumu kreiranja od najnovijeg prema najstarijem. Vraćamo ih po pet koristeći metodu *paginate* (engl. *straniči*) koja nam omogućuje da na stranici imamo kontrole za stranični prikaz (engl. *pagination*) što je korisno kada imamo veliki broj podataka u bazi kako pri inicijalnom učitavanju stranice ne bi morali dohvaćati apsolutno sve podatke. Na samom kraju metode korisniku vraćamo pogled *jobs*, a u metodi *compact* (hrv.*zbij)* navodimo koje podatke i koje modele želimo predati tom pogledu tako da ih možemo i koristiti koristeći *blade engine*.

5.6. Praćenje korisnika

Na web aplikaciji postoji mogućnost međusobnog praćenja i prestanka praćenja korisnika što je moguće učiniti na stranicama za pretragu korisnika (engl. *Users*), pregled pratitelja (engl. *My followers*) i korisnika koje korisnik prati (engl. *Following*) gdje se nalaze dugmad za praćenje ili prestanak praćenja korisnika. Također se koristi AJAX kako se stranica ne bi morala učitavati pri svakom izvršavanju metode. Korisnici koji se međusobno prate mogu na stranici *Feed* pretraživati objave isključivo koje su pisali oni korisnici koje prate.

Sl. 5.13. Dugmad za praćenje korisnika (zeleno) i prestanak praćenja (crveno)

```
<a href="#!" class="btn {{$user→followers→contains('follower_id', Auth::id()) ?  'btn-danger' : 'btn-common'}} btn-xs follow-unfollow"
    data-id="{{$user→id}}" onclick="actOnFollowUnfollow(this)"><i class="fas {{$user→followers→contains('follower_id', Auth::id()) ?
    'fa-user-minus' : 'fa-user-plus'}} follow-unfollow-icon"></i></a>
```

Sl. 5.14. HTML elementi u kombinaciji s podacima iz modela koji sadrži podatke potrebne za praćenje ili prestanak praćenja korisnika

Na slici 5.14. možemo vidjeti konstrukciju dugmeta koje se sastoji od dva HTML elementa, a to su *<a>* i *<i>* elementi. Element *<a>* inače služi kao poveznica pa smo stoga u njegov *href* atribut u koji inače se navodi adresa, stavili *„#!“* vrijednost čime navodimo da ne želimo da nas preusmjerava. Na njega postavljamo određenu klasu s čime mijenjamo pozadinsku boju. Pri inicijalnom učitavanju to radimo na način da za svakog korisnika kojeg želimo pratiti ili prestati pratiti dohvatimo identifikacijske brojeve vezane za njegove pratitelje te provjerimo spada li tu i identifikacijski broj prijavljenog korisnika. Ako da, dugme će biti crveno, a u suprotnom će biti zeleno. Nakon što se metoda za praćenje ili prestanak izvrši na serveru uspješno ili neuspješno, promjenu dugmeta radimo ručno u kôdu. U atribut *data-id* spremamo vrijednost identifikacijskog broja korisnika kojega želimo pratiti ili prestati pratiti. U atributu *onclick* navodimo metodu koja će se izvršiti kada se taj element klikne (u ovom slučaju to je metoda *actOnFollowUnfollow*) s parametrom *this* (hrv. *ovo*). Parametar *this* u ovom slučaju predstavlja element koji smo kliknuli te iz tog parametra možemo u kôdu dobiti važne informacije poput trenutno postavljene klase i slično. Unutar *<a>* elementa se nalazi *<i>*

element koji zapravo predstavlja ikonicu uz pomoć klase te na isti način kako postavljamo boju za <*a*> element, postavljamo i klasu na <*i*> element koja će odgovarati ikonici. Korištenjem Font Awesome alata imamo mogućnost besplatnog korištenja ikonica pomoću klasa [22].

```
// Toast swal mixin
const toast = swal.mixin({
    toast: true,
    position: 'bottom-start',
    showConfirmButton: false,
    timer: 3000
});


// Toggle icon actins wrapper in object
var toggleIcon = {
    Follow: function (icon) {
        $(icon).find("i").removeClass('fa-user-plus').addClass('fa-user-minus');
        $(icon).removeClass('btn-common').addClass('btn-danger');
    },

    Unfollow: function (icon) {
        $(icon).find("i").removeClass('fa-user-minus').addClass('fa-user-plus');
        $(icon).removeClass('btn-danger').addClass('btn-common');
    }
};

// Follow/Unfollow handler
var actOnFollowUnfollow = function (event) {

    //get userId
    var userId = $(event).data('id');
    var icon = $(event).find("i").attr('class');

    if (icon.includes('fa-user-plus')) {
        toggleIcon.Follow(event);
        followUnfollowAjax(userId, 'Follow', event);
    } else {
        toggleIcon.Unfollow(event);
        followUnfollowAjax(userId, 'Unfollow', event);
    }

};
```

Sl. 5.15. Pomoćne metode

Na slici 5.15. možemo vidjeti pomoćne metode s kojima manipuliramo izgled dugmeta te pozivamo glavnu AJAX metodu.

```
// Ajax call to the backend
function followUnfollowAjax(user_id, action, event) {
    $.ajaxSetup({
        headers: {
            'X-CSRF-TOKEN': $("input[name=_token]").val()
        }
    });
    $.ajax({
        url: '/follow-unfollow/' + user_id,
        type: 'POST',
        dataType: 'JSON',
        data: {
            action: action    Ivan Zivkovic, 11 days ago • following done / my followers
        },
        success: function (data) {
            // Display success message
            toast({
                type: 'success',
                title: '' + data.success
            });
        },
        error: function (error) {
            // Display error message
            toast({
                type: 'error',
                title: 'Oops ... ',
                text:  error.responseJSON.error ? error.responseJSON.error : 'An error
                has occured. Please contact our administrator (ivanzivkovic1601@gmail.com)
                '
            });
            console.log(action);
            // If error happens, reverse stats and icon
            switch (action) {
                case 'Follow':
                    toggleIcon.Unfollow(event);
                    break;
                case 'Unfollow':
                    toggleIcon.Follow(event);
                    break;
            }
        }
    });
}
```

Sl. 5.16. AJAX metoda na klijentskoj strani korištena za praćenje ili prestanak praćenja korisnika

Na slici 5.16. vidimo AJAX metodu sličnu kao i za upravljanje prijava gdje navodimo sve potrebne podatke. Metoda prima identifikacijski broj korisnika kojeg prijavljeni korisnik želi pratiti, akciju koja predstavlja ili praćenje (engl. *follow*) ili obrnuto te *event* (hrv. *događaj*) parametar. Te parametre prima iz pomoćne funkcije *actOnFollowUnfollow*. Slično kao i kod upravljanja prijavama, imamo dvije uzvratne mogućnosti i metode koje se izvršavaju ovisno o uspjehu ili greškama. U ovom slučaju, ako se dogodila greška, dugmetu vraćamo stari izgled jer u pomoćnoj funkciji radimo promjenu izgleda na novo stanje prije same potvrde o uspješnosti.

```php
    /**
     * Method used to follow or unfollow user
     * @param request contains info about action
     * @param id Id of the user being followed
     */
    public function follow_unfollow(Request $request, $id){
        //get action name
        $action = $request->get('action');
        switch ($action) {
            case 'Follow':

                // if user already follows another user
                if( Follow::where([['user_id', $id], ['follower_id', Auth::id()]])->exists() ) {
                    $return = array(
                        'error' => 'You are already following this user!'
                    );
                    return response()->json($return, 400);
                }
                // if users tries to follow himself
                else if( $id == Auth::id() )
                {
                    $return = array(
                        'error' => 'You can not follow yourself!'
                    );
                    return response()->json($return, 400);
                }
                // if there is no record
                else {
                    $follow = new Follow;
                    $follow->user_id = (int)$id;
                    $follow->follower_id = Auth::id();
                    $follow->save();
                    $return = array(
                        'success' => 'You have successfully followed this user!'
                    );
                    return response()->json($return, 200);
                    break;
                }
                //  CASE UNLIKE
                case 'Unfollow':
                // if there exists this follow
                if( Follow::where([['user_id', $id], ['follower_id', Auth::id()]])->exists() ) {
                    // delete
                    Follow::where([['user_id', $id],['follower_id',Auth::id()]])->delete();
                    $return = array(
                        'success' => 'You have successfully unfollowed this user!'
                    );
                    return response()->json($return, 200);
                    }
                    // throw error
                    else
                     {
                        $return = array(
                            'error' => 'This follow does not exist in database!'
                        );
                        return response()->json($return, 400);
                    }
        }
    }
```

Sl. 5.17. Metoda za upravljanje praćenjima na serverskoj strani

Metoda sa slike 5.17. prima parametar *$id* iz rute koju poziva AJAX (Sl.5.17.) te *$request* objekt koji šaljemo kroz AJAX u *data* (hrv. *podaci)* svojstvu. Prvo provjeravamo koju radnju želimo izvršiti, praćenje (engl. *follow*) ili prestanak praćenja (engl. *unfollow*). U oba slučaja postoji provjera. U slučaju praćenja provjeravamo postoji li u bazi podataka već zapis o praćenju. Ako postoji, AJAX metodi vraćamo grešku. Ako ne postoji, provjeravamo želi li korisnik pratiti sebe (dodatna sigurnost) ili druge korisnike. U slučaju da pokušava pratiti sebe, također vraćamo grešku. U slučaju da želi pratiti drugog korisnika, kreiramo novi zapis u bazi podataka u tablici praćenja (engl. *follow*s) i vraćamo uspješan rezultat AJAX metodi uz poruku. U slučaju da želimo prestati pratiti korisnika, provjeravamo u bazi podataka postoji li zapis o postojećem praćenju između ta dva korisnika. Ako ne postoji, vraćamo grešku. Ako postoji, brišemo zapis i vraćamo poruku o uspjehu.

6. ZAKLJUČAK

Glavne mogućnosti web aplikacije *TheHunt* koja je izrađena u web tehnologijama pruža korisnicima mogućnosti za jednostavan pronalazak poslova ili osoba koje su zainteresirane za njihove poslove te pruža mogućnosti povezivanja korisnika putem praćenja. Korisnici imaju uvid u svoje oglase, njihove prijave na tuđe poslove te prijave drugih korisnika na njihove poslove. Također, korisnici u ulozi poslodavca imaju mogućnost prihvaćati ili odbijati prijave na oglase za poslove pri čemu su prijavljene osobe obaviještene o promjeni stanja njihovih prijava putem e-pošte. Korisnici u ulozi zaposlenika imaju mogućnost prijave na tuđe poslove pri čemu su poslodavci isto tako obaviješteni e-poštom. Postoji mogućnost ocjenjivanja korisnika pri čemu si povećavaju ili snižavaju reputaciju ovisno o kvaliteti njihova rada ili uloge poslodavca. Od sporednih mogućnosti, korisnici mogu kontaktirati jedni druge putem kontakt formi na profilima, komentirati i sviđati tuđe objave i poslove, pregledavati svoje ocjene te imaju uvid u to tko njih prati ili koga oni prate. Izrada praktičnog dijela zahtijevala je istraživanje i detaljno planiranje s obzirom na širinu mogućnosti, veliki broj tablica u bazi podataka te ispunjavanje korisničkih zahtjeva koje ova aplikacija treba podržavati. U aplikaciji postoji veliki broj uvjeta i provjera što zahtijeva dobru organiziranost projekta te dodaje kompleksnosti rješenja.

LITERATURA

[1] HTML (HyperText Markup Language), MDN web docs, dostupno na: https://developer.mozilla.org/hr/docs/Web/HTML [17. studeni 2018.]

[2] CSS Syntax and Selectors, W3Schools, dostupno na: https://www.w3schools.com/css/css_syntax.asp [17. studeni 2018.]

[3] Bootstrap Get Started, W3Schools, dostupno na: https://www.w3schools.com/bootstrap/bootstrap_get_started.asp [17. studeni 2018.]

[4] JavaScript Where To, W3Schools, dostupno na: https://www.w3schools.com/js/js_whereto.asp [17. studeni 2018.]

[5] What is Ajax Programming - Explained, KeyCDN, dostupno na: https://www.keycdn.com/support/ajax-programming [17. studeni 2018.]

[6] Wikipedia, dostupno na: https://en.wikipedia.org/wiki/File:Ajax-vergleich-en.svg [17. studeni 2018]

[7] Laravel, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Laravel [17. studeni 2018.]

[8] What is MVC, really?, StackExchange, dostupno na: https://softwareengineering.stackexchange.com/questions/127624/what-is-mvc-really [17. studeni 2018]

[9] Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [17. studeni 2018.]

[10] Eloquent: Getting Started, Laravel, dostupno na: https://laravel.com/docs/5.7/eloquent [17. studeni 2018.]

[11] Your One-Stop Guide to Laravel Commands, Envato Tuts+, dostupno na:

https://code.tutsplus.com/tutorials/your-one-stop-guide-to-laravel-commands--net-30349

[17. studeni 2018.]

[12] Getting Started, Composer, dostupno na: https://getcomposer.org/doc/00-intro.md [17. studeni 2018.]

[13] Blade Templates, Laravel, dostupno na: https://laravel.com/docs/5.7/blade [17. studeni 2018.]

[14] WAMP, TechTerms, dostupno na: https://techterms.com/definition/wamp [17. studeni 2018.]

[15] Visual Studio Code, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Visual_Studio_Code [17. studeni 2018.]

[16] Directory Structure, Laravel, dostupno na: https://laravel.com/docs/5.7/structure [17. studeni 2018.]

[17] Eloquent: Relationships, Laravel, dostupno na: https://laravel.com/docs/5.7/eloquent-relationships [17. studeni 2018.]

[18] HTTP Controllers: RESTful Resource Controllers, Laravel, dostupno na: https://laravel.com/docs/5.1/controllers#restful-resource-controllers [17. studeni 2018.]

[19] Facades, Laravel, dostupno na: https://laravel.com/docs/5.7/facades [17. studeni 2018.]

[20] CSRF Protection, Laravel, dostupno na: https://laravel.com/docs/5.7/csrf [17. studeni 2018.]

[21] HTTP Status Codes, REST API Tutorial, dostupno na: https://www.restapitutorial.com/httpstatuscodes.html [17. studeni 2018.]

[22] Font Awesome, dostupno na: https://fontawesome.com/?from=io [17. studeni 2018.]

SAŽETAK

U ovom završnom radu izrađena je web aplikacija za objavu i prijavu na honorarne poslove. Korisnici se mogu registrirati i izraditi korisničke profile te se isti mogu uređivati i brisati. Korisnici imaju mogućnosti objava oglasa za posao te uređivanje i brisanje istih a i sami mogu pretraživati i filtrirati poslove te se mogu prijaviti na tuđe poslove pri čemu su poslodavci obaviješteni e-poštom. Kao poslodavci, korisnici imaju mogućnosti pregleda svih prijava na njihove poslove te opcije prihvaćanja ili odbijanja prijava pri čemu su potencijali kandidati obaviješteni e-poštom. Kao korisnici koji su se prijavili na poslove, imaju uvid u svoje prijave te njihov status. Iste prijave mogu i otkazati. Nakon odrađenog posla, korisnici se mogu međusobno ocijeniti u smislu da poslodavci ocjenjuju korisnike koji su radili na poslu i obrnuto. Korisnici se međusobno mogu pratiti i prestati pratiti te mogu pregledavati tuđe profile i objave. Isto tako imaju uvid koga prate i tko njih prati. Korisnici se mogu kontaktirati putem forme na njihovim profilnim stranicama. Također postoji mogućnost objave komentara na objavama i poslovima te mogućnosti sviđanja istih. U teorijskom dijelu rada opisane su korištene, struktura projekta, korisnička sučelja i programsko rješenje aplikacije. U praktičnom dijelu rada je izrađena funkcionalna web aplikacija korištenjem web tehnologija koje su i opisane.

Ključne riječi: aplikacija, e-pošta, honorar, oglas, posao

ABSTRACT

Web application for publishing and applying to freelance jobs

In this thesis, web application for publishing and applying to freelance jobs was created. Users can register and create their user profiles which can be edited and deleted. Users have the possibilities of publishing job ads which can be edited and deleted while they can also search and filter other user's jobs and apply to them in which case employers are notified by e-mail. As employers, users can check all the job applications for their job ads and have options of accepting or rejecting them in which case the potential candidates are notified by e-mail. As users which have applied to job ads, users have access to their job applications and their status. They can also cancel those as well. After the job is done, users can rate each other in means of recruiters rating the users which have worked on the job and vice versa. Users can follow and unfollow each other and look at other user profiles. They also have an insight into who are they following and who is following them. Users can be contacted by contact forms on their profile pages. There is also a possibility of posting comments on posts and jobs and liking them as well. Theoretical part describes the used techologies, project structure, user interfaces and program solution. In the practical part, the functional web application was made by using the described technologies.

Keywords: ad, application, e-mail, freelance, job

ŽIVOTOPIS

Ivan Živković rođen je 16. siječnja 1996. godine u Vinkovcima. Od 2002. do 2010. godine pohađa Osnovnu školu Bartola Kašića u Vinkovcima. Godine 2010. upisuje Tehničku školu Ruđera Boškovića u Vinkovcima koju završava 2014. godine. Iste godine polaže državnu maturu s vrlo dobrim uspjehom i upisuje Elektrotehnički fakultet (sadašnji Fakultet elektrotehnike, računarstva i informacijskih tehnologija) Osijek na Sveučilištu Josipa Jurja Strossmayera u Osijeku na stručnom studiju Informatike.

Ivan Živković