Izrada web-aplikacija pomoću Laravela

Zbiljski, Stjepan

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:597464

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Stjepan Zbiljski

IZRADA WEB-APLIKACIJA POMOĆU LARAVELA

Diplomski rad

Voditelj rada:
doc. dr. sc. Zvonimir Bujanović

Zagreb, studeni, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **2**

1 Razvojni okvir Laravel **3**
1.1 Instalacija Laravela 4
1.2 Prva aplikacija 5
1.3 Artisan CLI 6
1.4 Migrations i Seeds 7
1.5 Eloquent 11
1.6 Controller 14
1.7 View i Blade 17
1.8 Usmjeravanje 20
1.9 Autentifikacija 23
1.10 Korisnička strana web-aplikacije (frontend) 24

2 Složena web-aplikacija **26**
2.1 Aplikacija za podršku kolegijima 26
2.2 Opis modela 30
2.3 Controlleri i rute 36
2.4 Middleware 41
2.5 Pohrana datoteka 42
2.6 Korisničko sučelje 43

Bibliografija **45**

Uvod

PHP je jedan od najraširenijih programskih jezika za razvoj web-aplikacija. Za razliku od većine popularnih programskih jezika za razvoj web-aplikacija koji prikaz podataka stvaraju na klijentskoj strani, PHP se izvršava u potpunosti na poslužiteljskoj strani. Kao pomoć pri izradi sve složenijih aplikacija, pojavili su se brojni razvojni okviri koji ubrzavaju njihovu implementaciju i pomažu pri organizaciji koda. Jedan od najpopularnijih razvojnih okvira za PHP je Laravel. Laravel je besplatni okvir otvorenog koda (eng. *open-source*) koji slijedi arhitekturalni obrazac *model-view-controller*, donosi moćan alat za web-predloške, implementaciju objektno-relacijskog preslikavanja (ORM) za komunikaciju s bazom podataka, te brojne druge mogućnosti koje se zahtjevaju od moderne platforme za razvoj web-aplikacija.

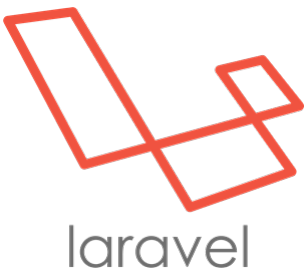
U prvom poglavlju opisujemo općeniti razvojni okvir Laravel. S obzirom da je Laravel opširan razvojni okvir, te korisniku pruža mnogo mogućnosti i načina rada, ovdje se usredotočujemo samo na njegov osnovni okvir koji je potreban za početak razvoja web-aplikacije. Prvo se bavimo postavljanjem razvojnog okruženja i kreiranjem nove web-aplikacije. Zatim, opisujemo osnovne dijelove arhitekturalnog obrasca *model-view-controller* i novosti te korisne mehanizme koje Laravel donosi programeru pri implementaciji njegovih dijelova. Posebno su zanimljivi Eloquent za modele i Blade za view-ove. Također, spomenut ćemo i usmjeravanje koje je važan dio obrasca i u koje Laravel uvodi razne metode za lakše definiranje i imenovanje. Novost u Laravelu su migracijske skripte, a važne su za rad i izmjene na bazi podataka, te olakšavaju rad u timu. Spomenut ćemo i mogućnost jednostavnog kreiranja autentifikacije. Na kraju ovoga poglavlja govorimo o mogućnostima rada na korisničkoj strani web-aplikacije.

U drugom poglavlju opisujemo složenije web-aplikacije u Laravelu. Za potrebe ovoga rada kreirana je web-aplikacija za podršku kolegijima Matematičkog odsjeka Prirodoslovno-matematičkog fakulteta u Zagrebu. Na početku prikazujemo samu web-aplikaciju, kako se ona upotrebljava i koje su njezine mogućnosti, a zatim ulazimo više u strukturu same web-aplikacije razlažući njene modele, rute te njihov rad. Navodimo i složenije mogućnosti Laravela koje nismo spominjali u prvom poglavlju, a izrazito su korisne, te se pojavljuju u našoj web-aplikaciji. Na kraju ovoga poglavlja ponovno govorimo o korisničkoj strani web-aplikacije, te navodimo i ukratko opisujemo sve pakete koje smo ko-

ristili u našoj web-aplikaciji.

Poglavlje 1

Razvojni okvir Laravel



Laravel je razvojni okvir za programski jezik PHP koji služi za izradu web-aplikacija. On je jedan od najpopularnijih razvojnih okvira, besplatan je i otvorenog je koda (eng. *open-source*). Razvojni okvir je baziran na arhitekturalnom obrascu MVC (eng. *model-view-controller*). Osnovna ideja MVC-a je razdvajanje aplikacije na tri osnovne komponente: model, view i controller. Model predstavlja logiku aplikacije i pripadne strukture podataka. On je smislen i bez weba, te ne vrši nikakvu komunikaciju s korisnikom aplikacije. View služi za vizualni prikaz modela korisniku u nekom pogodnom formatu, što je u slučaju web-aplikacija najčešće HTML. Controller je spojnica između modela i view-a. On obrađuje podatke poslane od strane korisnika te na temelju njih ažurira model. Također, radi upite na modele i na temelju odgovora priprema podatke koje će prikazati view.

Laravel nam donosi izrazito moćne web-predloške, implementaciju objektno-relacijskog preslikavanja (ORM) za komunikaciju s bazom podataka, te brojne druge mogućnosti koje se zahtijevaju od moderne platforme za razvoj web-aplikacija. U ovom radu opisujemo verziju 5.6.

1.1 Instalacija Laravela

Serverski zahtjevi

Laravel je PHP razvojni okvir te zahtjeva HTTP server na kojem će se pokretati. Na serveru treba postojati instalacija PHP-a u verziji $\geq$ 7.1.3, te sljedeće PHP ekstenzije:

- OpenSSL
- PDO
- Mbstring
- Tokenizer
- XML
- Ctype
- JSON

Da bi priprema programskog okruženje za razvoj web-aplikacija bilo što jednostavnija, Laravel nudi alternativu lokalnom serveru u obliku virtualnog stroja Laravel Homestead. To je službeni, pripremljeni Vagrant box koji služi lakšem razvoju aplikacija bez potrebe za postavljanjem web servera, instalacijom PHP-a, i bilo kojeg drugog serverskog softvera na lokalno računalo. Vagrant je *open-source* softverski produkt koji služi za kreiranje i upravljanje prijenosnim virtualnim okolinama za razvoj softvera. Vagrant boxevi su manje verzije virtualne mašine kojima se lakše i brže upravlja. Velika je prednost korištenja Homesteada što se korisnik ne mora brinuti o mogućim problemima koje može stvoriti na lokalnom računalu. Vagrant boxovi se lako uništavaju i stvaraju novi u slučaju pojave ikakve greške.

Composer i instalacija

Composer je alat koji služi za upravljanje zavisnostima u PHP-u. Pomoću njega se deklariraju biblioteke koje su nam potrebne u nekom projektu. Jednostavnim pozivom iz komandne linije instaliramo nove, ažuriramo postojeće te brišemo suvišne biblioteke. On nam je potreban za instalaciju samog Laravela.

Instalacija Laravela se zatim provodi sljedećim pozivima u komandnoj liniji:

```
1 composer global require "laravel/installer"
```

i zatim

```
laravel new ime_direktorija
```

Prva naredba globalno dohvaća instalacijski paket za Laravel te nam omogućava pozivanje naredbe **laravel** za kreiranje novih projekata. Pomoću druge naredbe kreiramo željeni direktorij u trenutnom, te dohvaćamo i instaliramo sve zavisnosti potrebne za novi projekt.

Alternativno, instalaciju možemo vršiti i isključivo pomoću Composera, tako da pozovemo naredbu:

```
composer create-project --prefer-dist laravel/laravel ime_direktrorija
```

U ovom slučaju izravno kreiramo novi projekt i instaliramo sve potrebne zavisnosti.

1.2 Prva aplikacija

Laravel dolazi s ugrađenim razvojnim HTTP serverom za PHP koji možemo koristiti za posluživanje naše aplikacije, ako imamo lokalno instaliran PHP.

Na adresi `http://localhost:8000` pokrećemo razvojni server sljedećom naredbom:

```
php artisan serve
```

Nakon uspješne instalacije, otvaranjem adrese `http://localhost:8000`, ili neke druge pripadajuće adrese ako koristimo npr. Laravel Homestead, u web pregledniku otvara se naša nova web-aplikacija, koja je oblika prikazanog na slici 1.1.

Slika 1.1: Inicijalna stranica

Laravel ima izrazito razgranatu strukturu direktorija. Pošto se koristi MVC obrazac ključno je na početku znati gdje se nalaze modeli, controlleri i view-ovi.

Modeli se nalaze unutar direktorija ***app/***. Pošto se modeli brzo gomilaju dobro im je kreirati vlastiti poddirektorij. Uz modele se vežu i vrlo su važne migracijske skripte koje se nalaze unutar ***database/migrations/***. O njima ćemo detaljnije pričati kasnije.

Controlleri se nalaze unutar direktorija ***app/Http/Controllers/***. Također ih je moguće grupirati u razne poddirektorije ovisno o funkcionalnosti koju izvršavaju.

View-ovi se nalaze unutar direktorija ***resources/views/***. Njih se također može grupirati i vrlo je dobra praksa razdvajati jedan view na više dijelova ovisno o funkcionalnosti koju implementiraju.

Važno je još spomenuti da se rute nalaze u direktoriju ***routes/***. O njima i važnim elementima toga direktorija ćemo detaljnije kasnije.

1.3 Artisan CLI

Artisan je sučelje komandne linije uključeno u Laravel. Pomoću njega imamo pristup brojnim naredbama koje su izrazito korisne u kreiranju aplikacije, na primjer, pomoću jedne naredbe kreiramo novi model s pripadajućim controllerom i migracijskom skriptom. Same naredbe i primjere pozivanja pokazat ćemo u narednim sekcijama.

Kako bismo dobili listu svih naredbi pozivamo sljedeću komandnu liniju:

```
1 php artisan list
```

Kada odaberemo željenu naredbu njene argumente i opcije možemo saznati pozivom linije:

```
1 php artisan help ime_naredbe
```

Svaka Laravel aplikacija sadrži i Tinker. Tinker je interaktivna okolina za testiranje unosa i komandi, koji ga zatim izvršava i vraća korisniku rezultat, tzv. REPL (eng. *Read–eval–print loop*). Pomoću njega možemo vršiti interakciju s cijelom aplikacijom iz komandne linije, kao na primjer s Eloquent ORM-om, događajima itd. Time se značajno olakšava testiranje raznih linija koda za koje nismo sigurni da li nam vraćaju željeni rezultat i imaju li očekivano ponašanje.

Tinker pokrećemo pomoću Artisana pozivom sljedeće linije:

```
1 php artisan tinker
```

Artisan omogućuje i kreiranje vlastitih naredbi unutar aplikacije. Vidimo da je to izrazito moćan alat pri kreiranju Laravel aplikacija i uvelike ubrzava izradu raznih dijelova.

1.4 Migrations i Seeds

Većina modernih web-aplikacija koristi baze podataka za pohranu svojih podataka. Laravel omogućuje jednostavno kreiranje tablica u bazi podataka i njihovo inicijalno punjenje podacima. Migracijske skripte služe kao svojevrsna kontrola verzija baze podataka i za lakše izmjene i dijeljenje promjena na njenoj shemi unutar tima. Seederi su s druge strane skripte pomoću kojih jednostavno punimo bazu testnim podacima.

Migrations

Migracijske skripte služe za kontrolu verzija baze podataka. Pomoću njih je moguće jednostavno kreiranje novih tablica ili izmjene na postojećim te samim time jednostavnije dijeljenje izmjena s ostalim članovima tima. Kreiramo ih pomoću sljedeće naredbe:

```
1 php artisan make:migration ime_skripte
```

Uobičajna je konvencija da se skripta imenuje s prefiksima koji naglašavaju tip izmjene, npr. *create*, *add_ime_stupca_to*, zatim ime tablice i na kraju *table*. Direktorij u koji su spremaju je ***database/migrations***. Svaka kreirana skripta ima na početku dodani vremenski žig kojim Laravel utvrđuje redoslijed migracijskih skripti.

Dodavanjem zastavica **--table** i **--create** možemo naglasiti samom Laravelu da se radi o izmjeni na postojećoj tablici ili kreiranju nove. Pozivi, uz navedenu konvenciju imenovanja, su sljedeći:

```
php artisan make:migration create_tablica_table --create=tablica
php artisan make:migration add_stupac_to_tablica_table --table=tablica
```

Nakon poziva navedenih naredbi, Laravel stvara datoteku u navedenom direktoriju te automatski generira kod za klasu **ImeSkripte** s dvjema praznim metodama **up** i **down**, koje programer zatim implementira. Metoda **up** služi za kreiranje novih tablica, dodavanje stupaca ili indeksa u bazu podataka, dok metoda **down** uklanja sve izmjene, tj. u njoj radimo sve suprotne radnje od onih u **up** metodi. Kod 1.1 prikazuje primjer jedne migracijske skripte za kreiranje tablice *users* u koju će biti spremljeni svi korisnici aplikacije i njihove osnovne informacije.

```
<?php

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('surname');
            $table->string('email')->unique();
            $table->string('password');
            $table->smallInteger('role');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
```

```
31          * @return void
32          */
33         public function down()
34         {
35             Schema::dropIfExists('users');
36         }
37 }
```

Kod 1.1: Primjer migracijske skripte

Kao što vidimo, kreiranje tablice i deklaracija stupaca je jednostavno implementirana pomoću Laravelove klase **Schema**. Zanimljivo je spomenuti da Laravel ima jednostavnu deklaraciju vremenskog žiga (linija 24.) koji će kreirati dva stupca u tablici za vrijeme kreiranja i zadnje izmjene, kojima će kasnije lako sam pozadinski manipulirati prilikom unošenja izmjena retka.

Pokretanje migracija se vrši sljedećom naredbom:

```
1 php artisan migrate [--seed]
```

Opcionalnom zastavicom **--seed** možemo istovremeno napuniti tablice testnim podacima koje će generirati pripadne klase tipa Seeder.

Migracije stvaraju automatski i svoju posebnu tablicu pomoću koje se pamte izmjene i pozivanja migracija. Time nam se omogućava povratak na prijašnja stanja ako nismo zadovoljni promjenama i slične radnje.

Ako želimo izbrisati sve tablice i pokrenuti ponovno migraciju, pozivamo:

```
1 php artisan migrate:fresh [--seed]
```

Seeds

Klase tipa Seeder su jednostavne metode kojima nam je omogućeno punjenje baze testnim podacima. Sve takve klase se nalaze unutar direktorija ***database/seeds***. Seeder klase mogu imati proizvoljna imena, ali konvencija je da budu oblika **ImeTabliceTableSeeder**. Sljedećim pozivom kreiramo klasu:

```
1 php artisan make:seeder ImeTabliceTableSeeder
```

Seederi implementiraju metodu **run** koja je odgovorna za punjenje tablice u bazi podataka. Početno je definirana klasa **DatabaseSeeder**. Kao što vidimo u primjeru 1.2, ona služi za pozivanje ostalih seedera koji rade punjenje određenih tablica.

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UsersTableSeeder::class);
        $this->call(ProgrammesTableSeeder::class);
        $this->call(CoursesTableSeeder::class);
        $this->call(ExamTypesTableSeeder::class);
    }
}
```

Kod 1.2: Primjer klase **DatabaseSeeder**

U primjeru 1.3 vidimo jednostavnu implementaciju gdje u tablicu *users* dodajemo dva korisnika, te u tablicu *students_infos* unosimo dodatne informacije za drugog korisnika.

```php
<?php

use Illuminate\Database\Seeder;

class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //
        DB::table('users')->insert(
            array(
                'name' => 'Marko',
                'surname'=> 'Horvat',
                'email' => 'marko.horvat@mail.com',
                'role' => 0,
                'password' => Hash::make('markovasifra'),
            )
        );
```

```
24
25          $id = DB::table('users')->insertGetId(
26              array(
27                  'name' => 'Ivan',
28                  'surname'=> 'Novak',
29                  'email' => 'ivan.novak@mail.com',
30                  'role' => 1,
31                  'password' => Hash::make('ivanovasifra'),
32              )
33          );
34
35          DB::table('students_infos')->insert(
36              array(
37                  'user_id' => $id,
38                  'JMBAG' => '0123456789'
39              )
40          );
41      }
42  }
```

Kod 1.3: Primjer jednostavnije seeder klase

Laravel omogućava i generiranje velikih količina podataka pomoću modela factory.

Osim već navedenog pokretanja zajedno s migracijskim skriptama, seederi se mogu pokretati i samostalno. Pozivom:

```
1  php artisan db:seed
```

izvršava se, kao i prilikom poziva zajedno s migracijama, metoda **run** klase **DatabaseSeeder**. Zato je uobičajna praksa da ona poziva sve ostale klase, a da se u njoj samoj ne izvodi populacija baze.

Ako želimo pozvati samo određeni seeder tada dodavanjem zastavice **--class** definiramo koju klasu želimo pozvati:

```
1  php artisan db:seed --class=ImeTabliceTableSeeder
```

1.5 Eloquent

Modeli služe za logiku aplikacije i rad s pripadnom bazom podataka u arhitekturalnom obrascu MVC. Laravel implementira modele pomoću Eloquent ORM-a (objektno-relacijskog preslikavanja). Eloquent omogućuje jednostavan rad s bazom podataka, oblikovan

je arhitekturalnim obrascem ActiveRecord. Svaki model je vezan za jednu tablicu u bazi podataka i omogućuje jednostavno pretraživanje, dodavanje i brisanje podataka.

Model imenujemo u skladu s tablicom koju će predstavljati i kreiramo sljedećim pozivom

```
php artisan make:model ImeModela [--migrations --controller --resource
  ]
```

Postavljanjem zastavica istovremeno za model možemo stvoriti pripadajuću migracijsku skriptu i controller. Štoviše, za controller možemo postaviti zastavicu **--resource** kako bi se stvorio CRUD controller, ali o tome više kasnije.

Prvo pogledajmo neke opće postavke na modelu i njihove zadane vrijednosti. Neka se, na primjer, naš model zove **User**. Uzimamo engleski naziv jer Eloquent koristi englesku množinu u automatskom imenovanju tablice u odnosu na model. Osnovna svojstva klase **User** su sljedeća:

```
protected $table = 'users';

protected $primaryKey = 'id';
public $incrementing = true;

public $timestamps = true;
const CREATED_AT = 'created_at';
const UPDATED_AT = 'updated_at';
```

Pomoću **$table** postavljamo željeno ime tablice na koju se model **User** spaja. Vidimo da je moguće i postaviti primarni ključ, te povećava li se on automatski. Automatsko povećanje je važno primjerice kod dodavanja novih podataka. Također, potrebno je naglasiti postoje li polja za vremenski žig, jer njega Eloquent samostalno izmjenjuje prilikom dodavanja i uređivanja. Ako je potrebno, mogu se i promijeniti imena tih polja. Postoje još razne opcije koje je moguće namještati. Spomenimo još **$fillable** koja prilikom kreiranja pomaže pri masovnom postavljanju vrijednosti stupaca. Samo se stupci navedeni u tom polju mogu masovno postaviti, vidi npr. kod 1.4.

Eloquent metode kao što su **all** i **get**, koje vraćaju više redaka, stvaraju instancu klase **Collection**. Ona nam omogućava jednostavan rad s podacima i nudi razne dodatne metode. Po takvim kolekcijama se može prelaziti petljom kao po polju. Metodama kao što su **toArray** i **toJson** možemo kolekcije jednostavno pretvarati u polje, odnosno JSON. Moguće je i dodavati različite uvjete na upite. Eloquent implementira metode kao što su **where**, **orderBy**, **groupBy** i slične metode čijim ulančavanjem stvaramo standardne upite na bazu podataka. Jedan takav primjer vidimo u kodu 1.4, u metodi **getAllProfessors**.

Eloquent omogućuje da jednom dohvaćenom retku lako pročitamo ili promijenimo vrijednost stupca preko svojstva koje nosi isti naziv kao ime stupca.

Jedna od najkorisnijih metoda koje nudi Eloquent je lako stvaranje relacija između tablica, odnosno modela. Tako se nude metode za veze "jedan naprema jedan", "jedan naprema mnogo", i "mnogo naprema mnogo". Posljednjom se najviše iščitava moć Eloquenta gdje jednostavno preko pivotne tablice stvara relacije. Samo korištenje veza se odvija preko metoda koje se dodaju u model, a zatim pozivaju kao svojstva koja vraćaju kolekcije. Primjer raznih veza vidimo u kodu 1.4.

```
1  <?php
2
3  namespace App\Models;
4
5  use ...
6
7  class User extends Authenticatable
8  {
9    use Notifiable;
10
11   protected $fillable = [
12     'name', 'surname', 'email', 'password', 'role'
13   ];
14
15   protected $hidden = [
16     'password', 'remember_token',
17   ];
18
19   ...
20
21   public static function getAllProfessors()
22   {
23     return self::where('role', 0)->orderBy('surname')->get();
24   }
25
26   ...
27
28   public function coursesAdmin()
29   {
30     return $this->belongsToMany(Courses::class, 'admin_course', 'user_id
       ', 'course_id');
31   }
32
33   public function posts()
34   {
35     return $this->hasMany(Post::class);
36   }
```

```
37
38    public function publish(Post $post)
39    {
40        return $this->posts()->save($post);
41    }
42
43    ...
44
45 }
```

Kod 1.4: Primjer modela za korisnike

Objasnimo sada metode **coursesAdmin** i **posts**. Model **User** i model **Courses**, tj. njihove pripadajuće tablice, su u vezi "mnogo naprema mnogo". Njihova pivotna tablica je *admin_course*. Metodom **coursesAdmin** definiramo relaciju među modelima pomoću Eloquent metode **belongsToMany**. Ona zahtjeva kao argument model (u našem primjeru **Courses**) s kojom je dani model (**User**) povezan. Drugi argument je naziv pivotne tablice (*admin_course*), te zatim ključevi na koje se spaja primarni ključ danog modela (*user_id*), i na koji se spaja primarni ključ modela u relaciji (*course_id*). Zadnja tri parametra nije potrebno navoditi ako stupce i tablice imenujemo po nekim pravilima. Recimo, strane ključeve bi trebalo imenovati kao *ime_tablice_id*, pri čemu je *ime_tablice* tablica na koju se odnosi strani ključ. Za pivotne tablice bi se trebalo imenovati spajanjem imena tablica na koje se odnosi, i to abecednim redom.

Metoda **posts** stvara relaciju za vezu "jedan naprema mnogo" između modela **User** i **Post**. Vidimo da se koristi Eloquent metoda **hasMany** i da se kao argument šalje model **Post**. U ovom primjeru vidimo da se ne navode strani ključevi, premda ih je moguće naglasiti, zato što Eloquent sada može raditi ispravnu predikciju po prije navedenim pravilima.

Za kraj spomenimo metodu **publish** koja koristi jednu vrlo korisnu mogućnost Eloquenta. Eloquent omogućuje jednostavno dodavanje novih modela u relaciju. U konkretnom primjeru želimo dodati novi **Post** za neki **User** model. Umjesto da sami postavljamo člana **user_id**, moguće je to napraviti preko postojeće relacije, kao što vidimo u primjeru, i jednostavnim pozivom metode za spremanje **save**, ili za neki drugi slučaj metodom za stvaranje novog modela **create**.

1.6 Controller

Unutar arhitekturalnog obrasca MVC controlleri služe kao veza između modela i viewa. Oni odrađuju pripremu podataka za prikaz, te obradu podataka i ažuriranje modela. U Laravelu se controlleri nalaze unutar direktorija ***app/Http/Controllers/***. Konvencija je da ih imenujemo najčešće po modelu na kojem rade ili nekom prigodnom tipu obrade koji rade,

i dodavanjem sufiksa *Controller*, tj. **ImeModelaController**. Controllere možemo kreirati pozivom Artisan naredbe:

```
1  php artisan make:controller ImeModelaController
```

Zastavicom **--invokable** kreiramo controller koji ima samo jednu moguću akciju. Takvi controlleri implementiraju samo metodu **__invoke**.

Dodavanjem zastavice **--resource** ili kraće **-r** kreiramo CRUD (eng. *create, read, update, and delete*) controllere s definicijama svih pripadajućih metoda, koje je zatim potrebno samo popuniti. Primjer jednog takvog controllera je prikazan u kodu 1.5. Za takav tip controllera Laravel nudi jednostavnu definiciju ruta o čemu će biti više riječi u sljedećim sekcijama.

Laravel omogućuje izravno povezivanje parametra iz rute s odgovarajućim modelom u kojem je taj parametar identifikator (stupac id u pripadnoj tablici baze podataka). Zastavicom **--model** možemo povezati CRUD controller s pripadajućim modelom. To je vrlo korisna opcija koja olakšava dohvaćanje modela jer automatski stvara instancu koja je spremna za daljnji rad.

U kodu 1.5 možemo vidjeti primjere metoda kada controlleru nije proslijeđen parametar (metoda **index**) i kada mu je proslijeđen (metoda **edit**). Metoda **index** vrši preusmjeravanje na controller **HomeController** i njegovu pripadajuću metodu **index**. U metodi **edit** vidimo izravno povezivanje parametra rute s odgovarajućim modelom. Metoda poziva view i šalje mu odgovarajuće podatke za prikaz. Također, vidimo da se unutar samog poziva vrši dohvaćanje i priprema podataka za prikaz pozivima raznih metoda modela.

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use ...
6
7  class CourseController extends Controller
8  {
9    public function __construct()
10   {
11     $this->middleware('prof', ['except' => ['index', 'show']]);
12     $this->middleware('cadmin', ['only' => ['edit', 'update', 'destroy'
       ]]);
13   }
14
15   public function index()
16   {
17     return Redirect::action('HomeController@index');
18   }
```

```
19
20    public function create()
21    {
22        ...
23    }
24
25    public function store(Request $request)
26    {
27        $this->validate($request, [
28            'name' => 'required|string',
29            'programms' => 'required'
30        ]);
31
32        ...
33    }
34
35    public function show( Courses $course)
36    {
37        ...
38    }
39
40    public function edit( Courses $course )
41    {
42        return view('course.edit', [
43            'course' => $course,
44            'programmes' => Programm::programmes(),
45            'exams' => ExamType::examTypes(),
46            'professors' => User::getAllProfessors(),
47            'admins' => $course->courseAdmin()->pluck('id')->toArray(),
48            'selected_programms' => $course->programmes()->pluck('id')->
      toArray(),
49            'selected_exams' => $course->examsTypes()->pluck('id')->toArray()
50        ]);
51    }
52
53    public function update(Request $request, Courses $course)
54    {
55
56        $this->validate($request, [
57            'name' => 'required|string',
58            'programms' => 'required',
59            'users' => 'required'
60        ]);
61
62        ...
63    }
64
```

```
65    public function destroy(Courses $course)
66    {
67        ...
68    }
69 }
```

Kod 1.5: Primjer CRUD controllera za upravljanje kolegijima

U Laravelu postoji bazna klasa *Controller* koju bi svaki controller trebao nasljeđivati. Iz nje nasljeđujemo metode **middleware**, **validate** i **dispatch**.

Kako bismo zaštitili određene rute od neautoriziranog pristupa moguće je definirati middleware. Poziv middlewarea moguće je dodati i pri definiranju same rute, ali u slučaju korištenja controllera prikladnije ga je dodati u njegov konstruktor **__construct**. Pri tome moguće je definirati middleware na samo određenim metodama. Primjer toga vidimo u kodu 1.5. Više o samom middleweru i njegovoj implemetaciji se može naći u [10], a i u idućem poglavlju.

Važan dio rada controllera je validacija podataka. Najčešće se radi o podacima koje prima iz HTML formi. Pomoću metode **validate** imamo široki spektar opcija koje možemo provjeravati na poslanim podacima. Od opcija možemo zahtijevati da polje postoji, kojeg je tipa, kolike je veličine, datotekama možemo provjeravati i tip. U slučaju da svi uvjeti nisu zadovoljeni automatski se vraća na prethodnu stranicu i zaustavlja daljnju obradu. U metodama **store** i **update** iz koda 1.5 možemo vidjeti jednostavne primjere validacije.

1.7 View i Blade

U arhitekturalnom obrascu MVC HTML kod se odvaja u view-ove i služi isključivo za prikaz podataka uz minimalnu logiku. Laravel nam nudi jednostavan i izrazito moćan alat za predloške zvan Blade. Sve Blade datoteke se nalaze u direktoriju ***resources/views*** i imaju ekstenziju *blade.php*. Za razliku od ostalih do sada nabrojanih elemenata, Blade nema pripadajuću Artisan naredbu s kojom bi se kreirao. Njegovo dodavanje se vrši kreiranjem prazne datoteke s odgovarajućim imenom, koje ne smije sadržavati točke, i navedenom ekstenzijom. Dobra je praksa odvajati Blade datoteke u direktorije ovisno o funkcionalnosti koju implementiraju.

Glavne prednosti Blade-a su nasljeđivanje predložaka i sekcije. Pošto većina stranica ima određeni dio koji se uvijek koristi možemo kreirati glavni predložak koji će ostale datoteke proširivati, npr. 1.6.

```
1 <!DOCTYPE html>
2 <html lang="{{ app()->getLocale() }}">
3 <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
6       <meta name="viewport" content="width=device-width, initial-scale=1">
7
8       <!-- CSRF Token -->
9       <meta name="csrf-token" content="{{ csrf_token() }}">
10
11      <title>{{ config('app.name') }}</title>
12
13      <!-- Scripts -->
14      <script src="{{ asset('js/app.js') }}" defer></script>
15      <script src="https://cloud.tinymce.com/stable/tinymce.min.js"></
        script>
16
17      <!-- Fonts -->
18      <link rel="dns-prefetch" href="https://fonts.gstatic.com">
19      <link href="https://fonts.googleapis.com/css?family=Raleway
        :300,400,600" rel="stylesheet" type="text/css">
20
21      <!-- Styles -->
22      <link href="{{ asset('css/app.css') }}" rel="stylesheet">
23      <link href="{{ asset('css/blog.css') }}" rel="stylesheet">
24      <link href="{{ asset('css/dashboard.css') }}" rel="stylesheet">
25      <link href="{{ asset('css/sticky-footer.css') }}" rel="stylesheet">
26
27      @yield('header-scripts')
28  </head>
29  <body>
30      @include ('layouts.navbar')
31      <div id="app">
32
33          @yield('content')
34
35          @include ('layouts.footer')
36      </div>
37
38      @yield('footer-scripts')
39  </body>
40  </html>
```

Kod 1.6: Glavni Blade predložak

Primjećujemo dvije direktive **@yield** i **@include**. Prva služi za prikaz sadržaja određene sekcije. Druga služi za umetanje predloška na određeno mjesto i pomaže pri čestoj upotrebi nekog segmenta i kada želimo kod držati što čišćim. Prilikom umetanja šalje se ime Blade datoteke, a ako se ona nalazi unutar nekog direktorija onda prvo pišemo ime direktorija, pa točku, i na kraju ime Blade datoteke. Zato je važno da imena ne sadrže točke.

Direktiva **@section**, čiju primjenu vidimo u 1.7, dolazi u paru s direktivom **@yield**.

U njoj definiramo sadržaj sekcije, koji se onda uvrštava na mjesto gdje pozivamo **@yield**. Direktiva **@extends** služi za nasljeđivanje nekog roditeljskog view-a, najčešće je to već spomenuti glavni predložak.

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

@section('content')
<div class="container-fluid">
    <div class="row">
        @include('layouts.sidebar')
        <main class="col-sm-9 offset-sm-3 col-md-10 offset-md-2 pt-3">
            <h1>Obavijesti</h1>

            <div class="col-sm-12 blog-main">
                @includeWhen(( Auth::check() && Auth::user()->isAdmin(
    $course->id ) ), 'posts.create')
                @foreach ($posts as $post)
                @include('posts.show')
                @endforeach
                {{ $posts->links('vendor.pagination.bootstrap-4')}}
            </div>
            @includeWhen(( Auth::check() && Auth::user()->isAdmin(
    $course->id ) ), 'posts.editModal')
            @includeWhen(( Auth::check() && Auth::user()->isAdmin(
    $course->id ) ), 'layouts.deleteModal')
        </main>
    </div>
</div>
@endsection

@section('footer-scripts')
<script src="{{ asset('js/modalPosts.js') }}" defer></script>
<script src="{{ asset('js/modalDelete.js') }}" defer></script>
@endsection
```

Kod 1.7: Blade predložak za stranicu s objavama

Kao što primjećujemo u primjeru 1.7, Blade omogućava i jednostavno dodavanje petlji (direktiva **@foreach**) i grananja (direktiva **@if**). Također, postoje direktive kao što su **@includeWhen** koje su ustvari grananje te se view uključuje samo kada je uvjet zadovoljen.

U sljedećem primjeru vidimo jednu metodu nekog controllera kako poziva view i šalje mu podatke:

```
public function index( Courses $course)
{
  return view('posts.index', ['posts' => $course->getOrderedPosts(), '
    course' => $course ]);
}
```

Daljnja manipulacija s podacima se odvija uobičajno pri čemu Blade dobiva varijable koje imaju imena vrijednosti pod navodnicima. U konkretnom primjeru, view-u spremljanom u ***posts/index.blade.php*** bi bile dostupne varijable **$posts** i **$course**.

Kada želimo ispisati vrijednost neke varijable to radimo na sljedeći način.

```
{{ $variable }}
```

Blade tada automatski poziva PHP-ovu funkciju **htmlspecialchars** na danoj varijabli kako bi spriječio XSS (eng. *Cross-site scripting*) napade. Moguće je izbjeći pozivanje funkcije **htmlspecialchars** na sljedeći način, ali tada smo izloženi mogućim napadima:

```
{!! $variable !!}
```

Blade ima i direktive za odabir prikaza ovisno o postojanju autoriziranog posjetitelja ili ako je posjetitelj gost. Direktiva **@guest** služi za dijelove prikaza koji se prikazuju neautoriziranom posjetitelju. S druge strane, postoji i direktiva **@auth** unutar čijeg se bloka nalazi prikaz za autoriziranog posjetitelja.

1.8 Usmjeravanje

Usmjeravanje (eng. *routing*) je dio polužiteljske strane web-aplikacije koji preslikava HTTP zahtjeve na pripadajuće metode i funkcije. Stoga su rute važan dio naše aplikacije jer one definiraju što će naša aplikacija raditi kada korisnik pristupi određenom URI-ju (eng. *Uniform Resource Identifier*). Unutar Laravela rute se definiraju unutar direktorija ***routes/***, te tamo nalazimo četiri datoteke: *api.php*, *channels*, *console.php* i *web.php*. Nama je najvažnija posljednja. Ona nam pruža stanje sesije, CSRF (eng. *Cross-site request forgery*) zaštitu i enkripciju kolačića. U slučaju da imamo aplikaciju bez stanja sesije (eng. *session*) koristimo *api.php*. Tada se autentifikacija mora odvijati tokenima. Njena najčešća upotreba je kada imamo RESTful API [9].

Usmjerivač (eng. *router*) pruža registraciju ruta za sve vrste HTTP poziva. Oni su sljedećeg oblika:

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
```

```
6 Route::options($uri, $callback);
```

Vidimo da metode primaju odgovarajući URI, te povratni poziv, što može biti ili neka funkcija ili, najčešće, odgovarajuća metoda na odgovarajućem controlleru. Ako ruta treba odgovarati na više vrsta HTTP poziva to možemo implementirati pozivom metode **match**. U slučaju da ruta treba odgovarati na sve vrste HTTP poziva onda koristimo metodu **any**.

Laravel dolazi s ugrađenim mehanizmom zaštite od CSRF napada. Zato je za pozive tipa **POST**, **PUT** i **DELETE** važno u svaku pripadajuću HTML formu staviti Blade direktivu **@csrf** za CSRF token kako bi zahtjev bio validan, a ne okarakteriziran kao napad. Također, HTML forme ne mogu raditi **PUT**, **PATCH** i **DELETE** zahtjeve. Kako bismo njih mogli ostvariti, unutar forme dodajemo Blade direktivu *@method* i odgovarajuću vrstu zahtjeva kao argument. Primjer:

```
1 <form action="/foo/bar" method="POST">
2   @method('PUT')
3   @csrf
4 </form>
```

Promotrimo sada složeniji primjer 1.8. Na njemu vidimo deklaraciju svih web ruta za neku aplikaciju. Uz standardne navedene metode koje imaju svoje pripadajuće controllere prvo zamjećujemo da je rute moguće imenovati pomoću metode **name**. To nam omogućava lakše pozivanje ruta unutar aplikacije tako da ne moramo pamtiti koje smo URI-je definirali nego je dovoljno pamtiti imena. To također uvelike olakšava promjene samog URI-ja, jer ako ga izmijenimo, a ime ostane isto, ne moramo u ostatku koda raditi izmjene svugdje gdje se spominje.

```
1  <?php
2
3  /*
4  |--------------------------------------------------------------------------
5  | Web Routes
6  |--------------------------------------------------------------------------
7  |
8  | Here is where you can register web routes for your application. These
9  | routes are loaded by the RouteServiceProvider within a group which
10 | contains the "web" middleware group. Now create something great!
11 |
12 */
13
14 Auth::routes();
15
16 Route::get('/', 'HomeController@index')->name('home');
```

```
17 Route::get('home', 'HomeController@index');
18 Route::get('results/all', 'HomeController@results')->name('results.all')
       ;
19
20 Route::resource('course', 'CourseController');
21
22 Route::group(['prefix' => 'course/{course}'], function() {
23     Route::resource('posts', 'PostController')->except('create', 'show',
       'edit');
24
25     Route::get('about/{type}', 'CourseViewController@show')->name('
       courseview.show');
26     Route::match(['put', 'patch' ], 'about/{type}', '
       CourseViewController@update')->name('courseview.update');
27
28     Route::resource('exams', 'ExamController')->only('index', 'store', '
       destroy');
29     Route::resource('assignments', 'AssignmentController')->only('store'
       , 'destroy');
30
31     Route::resource('results', 'ResultsController')->except('create', '
       edit');
32 });
```

Kod 1.8: Primjer rutera *web.php*

Druga važna stvar koju nam omogućuje Laravel je slanje parametara preko URI-ja. Dovoljno je staviti ime parametra unutar vitičastih zagrada i pripadajuća funkcija ga može dohvatiti. Kao što smo vidjeli u prijašnjim sekcijama, ta mogućnost je izrazito važna i korisna za rad controllera. Također, parametar možemo postaviti da bude opcionalan dodavanjem upitnika iza imena.

Kada imamo CRUD controllere, izrazito je korisna metoda **resource**. Ona nam omogućava da jednostavno kreiramo pozive za sve metode jednom linijom koda. Ukoliko nemamo implementirane sve metode, pozivom metode **only** možemo navesti sve metode koje smo implementirali, odnosno, pozivom metode **except** sve one koje nismo.

Zadnja veća opcija koju Laravel pruža pri kreiranju ruta je grupiranje. Pozivom metode **group** je moguće je postavljanje prefiksa za URL, postavljanje raznih middlewarea i sličnih opcija na skupu ruta.

Listu svih definiranih ruta možemo saznati pozivom:

```
1 php artisan route:list
```

U popisu možemo vidjeti vidjeti razne korisne informacije kao što su URL rute, vrste HTTP poziva na ruti, ime rute, te postoje li kakve zaštite na ruti.

1.9 Autentifikacija

Autentifikacija je izrazito važna mogućnost većine modernih web-aplikacija. Kod nekih dio sadržaja nije dostupan neautentificiranim korisnicima, dok je kod drugih sadržaj u potpunosti nedostupan bez autentifikacije. Laravel omogućava jednostavno implementiranje autentifikacije. Štoviše, Laravel dolazi s cijelom autentifikacijom pripremljenom, te je samo potrebno ju dodati u projekt. To radimo sljedećim pozivom:

```
php artisan make:auth
```

Laravel pri samom kreiranju novog projekta uključuje model **User** i njegovu pripadnu migracijsku skriptu. Svi controlleri i view-ovi kreirani prošlom naredbom se spajaju na taj postojeći model.

Naredba kreira skup controllera unutar direktorija ***App/Http/Controllers/Auth***. **RegisterController** obrađuje zahtjeve za registracijom novog korisnika, **LoginController** obrađuje prijavu, dok **ForgotPasswordController** i **ResetPasswordController** obrađuju izgubljene lozinke, njihovo ponovno postavljanje, te slanje novih korisniku na njegov e-mail.

Naredba kreira i pripadajuće Blade view-ove unutar direktorija ***resources/views/auth***. Također, kreira direktorij ***resources/views/layouts*** u kojem stvara bazni predložak. Svi view-ovi su bazirani na Bootstrap-u [3].

Svi svi navedeni dijelovi su podložni prilagodbi kako bi ostvarile dodatne funkcionalnosti koje su nam potrebne, ili kako bi prikaz bio u skladu s našim željama.

Klasa **Auth** nam pomaže pri radu s autentificiranim korisnikom. Jednostavno možemo dohvatiti sve njegove podatke

```
$user = Auth::user();
```

ili samo njegov identifikator

```
$id = Auth::id();
```

ili odrediti da li postoji autentificirani korisnik

```
if (Auth::check()) {
  // Korisnik je autentificiran
}
```

Također, imamo i middleware *auth* kojima možemo ograničiti pristup određenoj ruti neautoriziranim korisnicima.

1.10 Korisnička strana web-aplikacije (frontend)

Slika 1.2: Laravel dolazi s predinstaliranim Bootstrapom i Vue-om

Laravel omogućava korisniku slobodni odabir i upotrebu JavaScripta i CSS-a za izradu korisničke strane web-aplikacije, (eng. *frontend*). Međutim, dolazi s pripremljenim bibliotekama Bootstrap [3] i Vue (1.2) za lakši početak rada. Laravel je početno podešen da za instalaciju frontend paketa koristi NPM, ali moguće je namjestiti i neki drugi.

Node package manager ili skraćeno NPM je upravitelj paketima za JavaScript pomoću kojega se instaliraju paketi, tj. moduli, u bilo koji projekt. Time nam je omogućeno jednostavno dodavanje raznih novih paketa, a svi instalirani na našem projektu se nalaze u datoteci *package.json* (kod 1.9).

```
{
    "private": true,
    "scripts": {
        "dev": "npm run development",
        "development": "cross-env NODE_ENV=development node_modules/webpack/bin/webpack.js --progress --hide-modules --config=node_modules/laravel-mix/setup/webpack.config.js",
        "watch": "npm run development -- --watch",
        "watch-poll": "npm run watch -- --watch-poll",
        "hot": "cross-env NODE_ENV=development node_modules/webpack-dev-server/bin/webpack-dev-server.js --inline --hot --config=node_modules/laravel-mix/setup/webpack.config.js",
        "prod": "npm run production",
        "production": "cross-env NODE_ENV=production node_modules/webpack/bin/webpack.js --no-progress --hide-modules --config=node_modules/laravel-mix/setup/webpack.config.js"
    },
    "devDependencies": {
        "axios": "^0.18",
```

```
14          "bootstrap": "^4.0.0",
15          "popper.js": "^1.12",
16          "cross-env": "^5.1",
17          "jquery": "^3.2",
18          "laravel-mix": "^2.0",
19          "lodash": "^4.17.4",
20          "vue": "^2.5.7"
21      }
22  }
```

Kod 1.9: Inicijalne postavke *package.json*

Unutar *devDependencies* možemo vidjeti sve inicijalno pripremljene module kao što su već spomenuti Bootstrap i Vue, te jQuery i lodash od ostalih poznatijih. Svaki novi modul koji instaliramo putem NPM-a će se pojaviti unutar njega.

U idućem će poglavlju biti riječi o još nekim zanimljivim paketima.

Ovime smo opisali osnovne dijelove razvojnog okvira Laravel koji su potrebni za izradu jednostavnijih web-aplikacija. Postoje još razne naprednije komponente koje se nude i koje su izrazito korisne. Spomenuli smo već middleware, o kojem će više riječi biti kasnije. Tu je i klasa za pohranu datoteka, o kojoj će isto biti riječi kasnije. Od ostalih spomenimo redom: *Notification* za slanje raznih obavijesti putem e-maila, SMS-a i sličnih servisa, *Queue* koji nam dopušta upravljanje redoslijedom i odgađanje izvršavanja vremenski zahtjevnih operacija, *Events* za upravljanje događajima, te cijelo okruženje za kreiranje testova za dijelove aplikacije. Više o njima je moguće saznati u [1], [10] i [11].

Poglavlje 2

Složena web-aplikacija

Laravel pomoću arhitekturalnog obrasca MVC nudi lako stvaranje nove aplikacije i njenog osnovnog okvira. Međutim, nudi i razne dodatne, naprednije mogućnosti. Stoga smo izradili složeniju web-aplikaciju koja koristi još neke dodatne opcije.

2.1 Aplikacija za podršku kolegijima

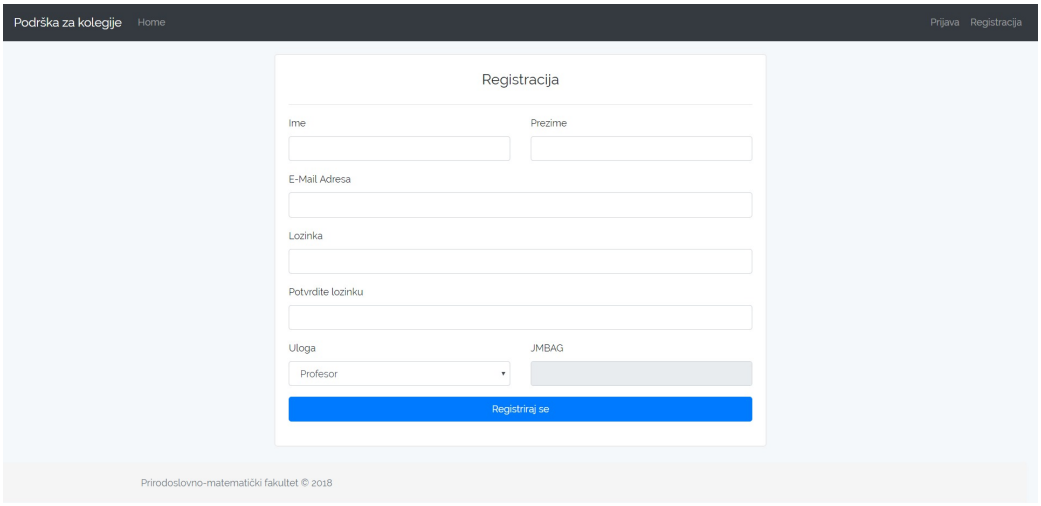
Odsjek za matematiku Prirodoslovno-matematičkog fakulteta ima mnogo kolegija i neki od njih imaju svoje web stranice za objavljivanje novosti. Izrada tih web stranica ovisi isključivo o voditeljima tih kolegija što dovodi do neujednačenosti u dizajnu i organizaciji podataka na njima, te ih nije moguće pronaći sve s jedne lokacije.

Za potrebe ovog rada, pomoću Laravela je izrađena web-aplikacija koja bi trebala olakšati dijeljenje obavijesti sa studentima, izradu samih web stranica te objavu rezultata. Svaki voditelj će moći jednostavno kreirati novu stranicu za svoj kolegij i stavljati obavijesti, dok će studenti s jednog mjesta imati pristup svim dostupnim stranicama kolegija.

Korisnički načini rada

Aplikacija prepoznaje tri vrste korisnika: profesore, studente i goste. Profesori i studenti su prijavljeni korisnici, dok gost nije.

Gost može pregledati sve kolegije, njihove obavijesti, kolokvije, materijale i slične opcije, ali ne može pristupiti rezultatima ili uređivanju kolegija. Gost se može prijaviti, ako ima postojeći korisnički račun, ili se registrirati (slika 2.1). Prilikom registracije se unose osobni podaci, te se izabire uloga korisnika. Korisnik može biti ili profesor ili student. O njihovim specifičnim mogućnostima će biti više govora u daljnjem tekstu.



Slika 2.1: Forma za registraciju novog korisnika

Administracija kolegija

Prvo recimo nešto o samoj administraciji kolegija. Svaki prijavljeni profesor može stvoriti vlastiti predmet odabirom *Dodaj novi kolegij* u gornjoj navigacijskoj traci. Prilikom stvaranja postavlja se ime kolegija, biraju se ostali profesori koji imaju mogućnost uređivanja, odabire se na kojim je smjerovima dostupan kolegij, te koji koji se tipovi kolokvija pišu na kolegiju. Posljednje je izrazito važno za kasniju objavu rezultata i starih kolokvija.

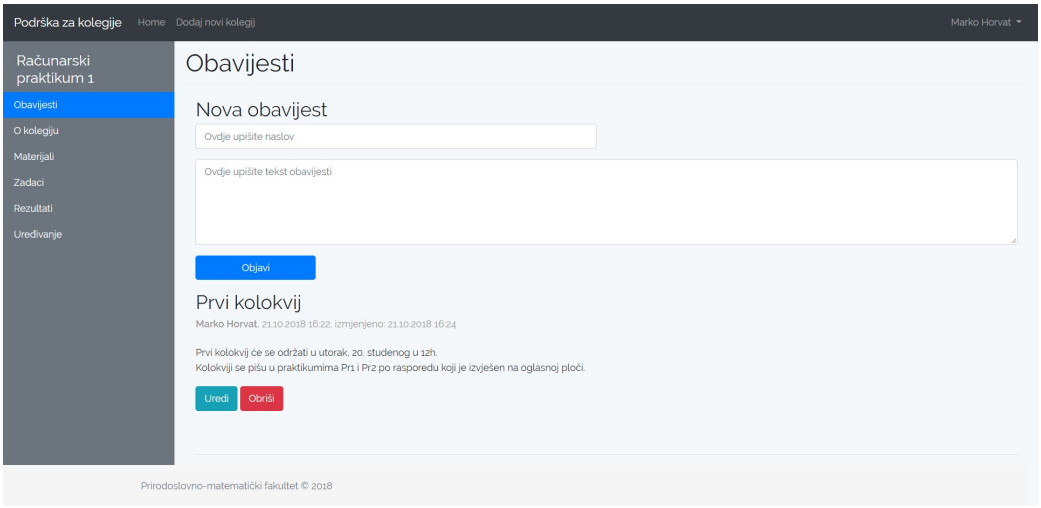
Na svakom kolegiju samo njegovi administratori imaju mogućnost objava, izmjena i brisanja podataka. Sve opcije koje se postavljaju prilikom kreiranja moguće je naknadno izmijeniti. Time je omogućeno davanje pristupa novim profesorima, oduzimanje starima, te ostale izmjene na navedenim opcijama.

Administratori jedini imaju mogućnost i brisanja kolegija. U slučaju toga odabira brišu se svi podaci vezani za kolegij, sve objave, rezultati, primjeri kolokvija i ostali vezani podaci.

Kolegiji

Početna stranica nudi posjetitelju izbor između svih postojećih kolegija. Također, nudi se i mogućnost filtriranja po smjeru. Nakon izbora, posjetitelju se otvara stranica kolegija s obavijestima.

Na stranici kolegija s obavijestima se nalaze sve obavijesti sortirane po vremenu objavljivanja, od novijih prema starijim. Svi posjetitelji mogu pregledavati obavijesti, ali samo administratori kolegija mogu obavijesti dodavati, uređivati i brisati (slika 2.2).



Slika 2.2: Administratorov pogled na stranicu za obavijesti

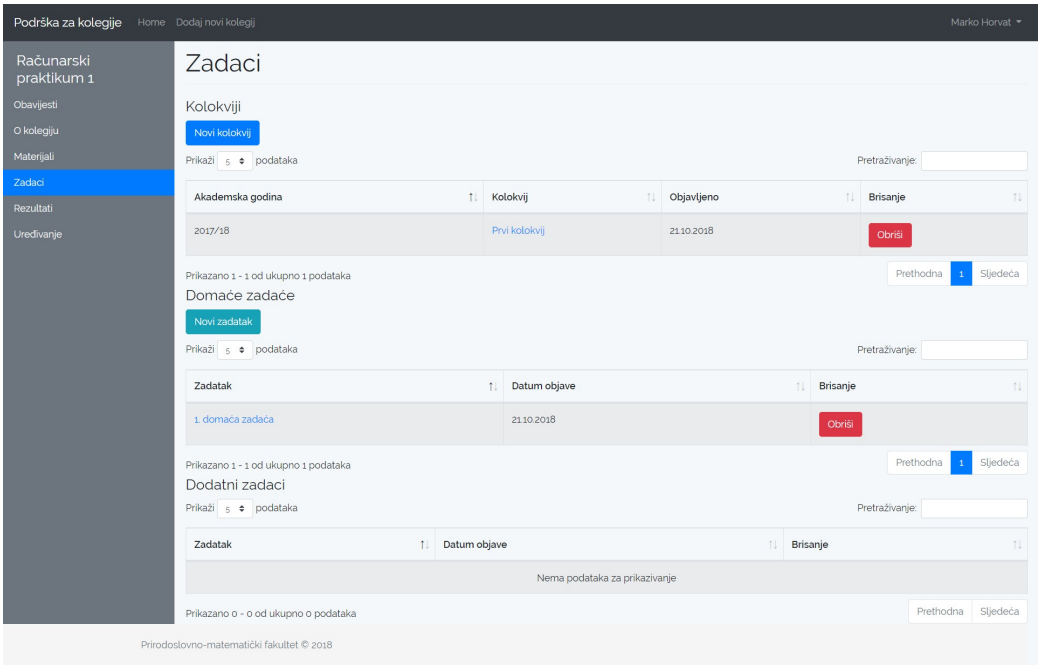
Na slici 2.2 s lijeve strane vidimo izbornik koji nudi sve podstranice za odabrani kolegij. Opcija *Uređivanje* je dostupna i vidljiva samo administratorima kolegija, dok je opcija *Rezultati* dostupna samo prijavljenim korisnicima.

Podstranice *O kolegiju* i *Materijali* nude osnovne informacije za kolegij. Njih administrator uređuje pomoću dodatka TinyMCE. Njime je omogućeno jednostavno dodavanje i uređivanje teksta bez korištenja HTML koda. Više o samom TinyMCE ćemo govoriti kasnije.

Zadaci

Podstranica *Zadaci* nudi posjetiteljima prikaz svih dosadašnjih kolokvija, objavljene domaće zadaće, te dodatne zadatke. Zadaci su radi lakšeg pretraživanja grupirani i prikazani pomoću tablica. Tablice su napravljene pomoću dodatka DataTables, o kojem ćemo više u sljedećim sekcijama. On nam omogućava jednostavno sortiranje, filtriranje i pretraživanje. Time je korisniku omogućeno lakše pronalaženje željenih zadataka. Prikaz vidimo na slici 2.3

Zadatke mogu dodavati jedino administratori. Dodavanje se izvršava pomoću jednostavnih modala kojima unosimo željenu datoteku u PDF-u (eng. *Portable Document Format*) i ostale potrebne informacije. Zadatke je moguće brisati pomoću gumba koji se nalazi uz pripadajući zadatak.



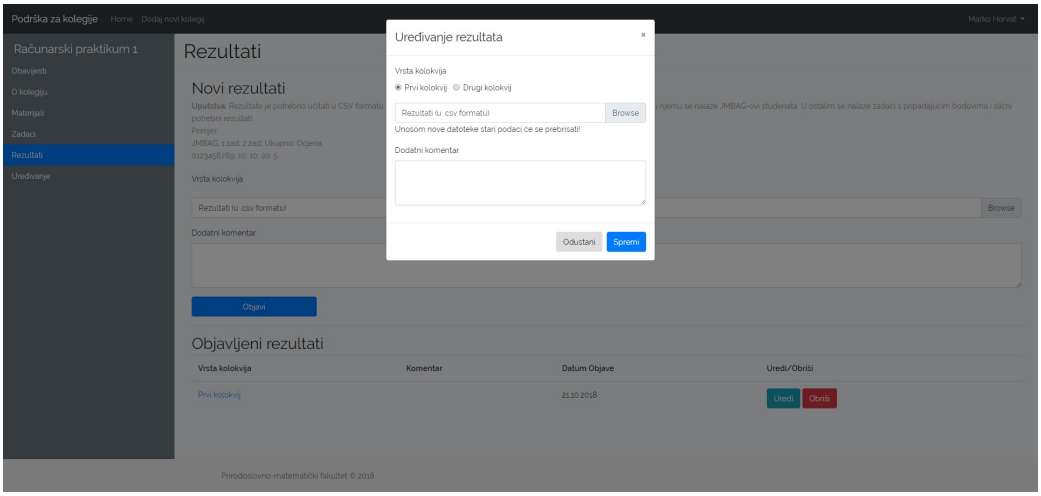
Slika 2.3: Prikaz stranice za zadatke

Rezultati

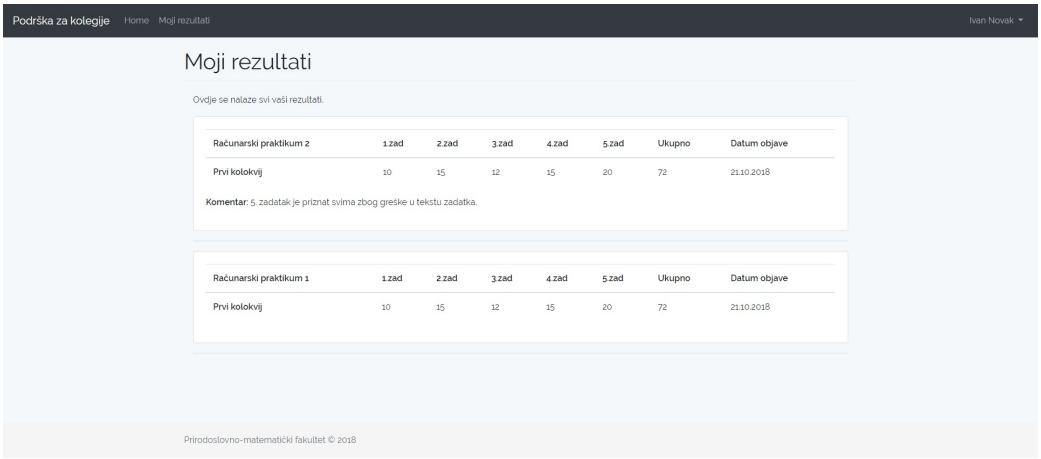
Podstranici *Rezultati* imaju pristup samo prijavljeni korisnici. Ako je korisnik student, tu će se pojaviti svi njegovi dosadašnji rezultati na kolokvijima iz danog kolegija.

Administratoru kolegija se tu nudi mogućnost objavljivanja rezultata. Na vrhu stranice se nalazi mala forma kojom unosi rezultate uz pomoć CSV (eng. *Comma-separated values*) datoteke, u obliku opisanom u uputstvu na vrhu stranice. Također, označava se i vrsta kolokvija, te je još moguće napisati dodatni komentar ako je potrebno. Svi objavljeni rezultati za dani kolegij se nalaze na dnu stranice u tablici. Rezultate je moguće uređivati unutar modalnog dijaloškog okvira, pri čemu svaki novi unos datoteke s rezultatima briše stare. Također, moguće je obrisati sve rezultate. Primjer stranice s otvorenim dijaloškim okvirom za uređivanje vidimo na slici 2.4

Student, uz to što može pogledati svoje rezultate za dani kolegij, može pogledati i svoje rezultate iz svih kolegija. U gornjoj navigacijskoj traci odabirom *Moji rezultati* otvara se stranica na kojoj su oni prikazani (slika 2.5).



Slika 2.4: Prikaz stranice za rezultate s dijaloškim okvirom za uređivanje



Slika 2.5: Prikaz svih studentovih rezultata

2.2 Opis modela

Web-aplikacija za podršku kolegijima koristi mnogo modela za svoj rad. Modeli imaju svoja svojstva koja su povezana sa strukturom tablice na koju se spajaju u bazi podataka, te su u raznim relacijama. Opisujemo sve modele kako bi dobili bolji uvid u organizaciju aplikacije. Modele navodimo abecednim redom.

Assignment

Model **Assignment** služi nam za rad sa zadacima, pamti njihovu lokaciju te sve dodatne informacije potrebne za njih, koja bolje vidimo kroz svojstva. Njegova su svojstva:

- *id* je primarni ključ i služi identifikaciji modela.
- *course_id* je strani ključ koji označava u sklopu kojeg se kolegija zadatak nalazi.
- *path* služi za spremanje lokacije na kojoj je spremljen zadatak. On nam služi kako bi korisniku mogli prikazati željenu datoteku.
- *link_text* nam govori što će pisati na poveznici za dani zadatak.
- *additional* označava o kakvom je zadatku riječ, običnom ili dodatnom. To je logička varijabla čija istinitost označava dodatni zadatak.
- *created_at* i *updated_at* su vremenski žigovi koji označavaju kada je u tablici u bazi podataka kreiran dani model i kada je zadnji put izmijenjen.

Svaka instanca modela je povezana s jednim modelom **Courses** preko svojstva *course_id*.

Courses

Model **Courses** je glavni model u web-aplikaciji. On služi predstavlja kolegije i u relaciji je s većinom drugih modela. Njegova su svojstva:

- *id* je primarni ključ i služi identifikaciji modela.
- *name* je ime kolegija.

Iako sam model nema puno svojstva njegove relacije iskazuju koliko je razgranat i važan. S modelom **Assignment** ima relaciju ”jedan naprema mnogo”. Istu relaciju tvori s modelima **CourseView**, **Exam**, **Post** i **ResultsInfo**. S modelima **ExamType**, **Programm** i **User** ima relacije ”mnogo naprema mnogo”.

Osim metoda koje opisuju pripadajuće relacije, model se odlikuje s još par zanimljivih metoda. Statičke metode **courses** i **courseFilter** nam nude zanimljive mogućnosti prikaza svih kolegija, odnosno svih kolegija koji se predaju na određenom smjeru, s Eloquentovom paginacijom koja olakšava prikaz velikog broja podataka. Metode **getOrderedPosts** i **getResultsInfos** nam vraćaju sortirane kolekcije modela **Post**, odnosno **ResultsInfo**, koristeći ustanovljene relacije na modelu. Liste su također prilagođene za paginaciju. Metoda **getAssignmentsByType** nam vraća kolekciju svih zadataka ili dodatnih zadataka, ovisno

o proslijeđenom argumentu, koristeći relaciju s modelom **Assignment**. Spomenimo još metodu **addView** koja prihvaća i sprema model **CourseView**, pri čemu mu dodaje relaciju s danom instancom modela.

CourseView

Model **CourseView** služi za stranice gdje je korisniku omgućena maksimalna kontrola uređivanja sadržaja, kao što je dano s TinyMCE-om. Primjer toga su podstranice *O kolegiju* i *Materijali*, gdje svaka predstavlja tip stranice. Svojstva modela su:

- *course_id* koji predstavlja vezu s pripadajućim kolegijem.
- *type* služi za razlikovanje o kojoj se podstranici radi, pošto ih za jedan kolegij ima više. Ova prva dva svojstva zajedno tvore primarni ključ.
- *view* predstavlja sadržaj koji će se posjetitelju prikazati i koji administrator može uređivati. Najčešće se radi o HTML kodu.
- *created_at* i *updated_at* su vremenski žigovi koji označavaju kada je u tablici u bazi podataka kreiran dani model i kada je zadnji put izmijenjen.

Model je u relaciji s modelom **Courses** preko svojstva *course_id*. Za model je važna statička metoda **getByCourseAndType** kojom dobivamo instancu modela pomoću para koji tvori primarni ključ.

Exam

Model **Exam** služi za rad s kolokvijima. Primarna svrha mu je pamćenje lokacije gdje su datoteke s kolokvijskim zadacima spremljeni kako bi im korisnici mogli pristupiti. Svojstva su mu:

- *id* je primarni ključ koji služi za identifikaciju.
- *course_id* je strani ključ koji označava u sklopu kojeg se kolegija kolokvij nalazi.
- *type_id* je strani ključ koji označava o kojoj je vrsti kolokvija riječ.
- *path* služi za spremanje lokacije na kojoj se datoteka nalazi.
- *year* je akademska godina u kojoj je dani kolokvij pisan.
- *created_at* i *updated_at* su vremenski žigovi koji označavaju kada je u tablici u bazi podataka kreiran dani model i kada je zadnji put izmijenjen.

Model je u relaciji s modelom **Courses** pomoću svojstva *course_id*, te modelom **ExamType** pomoću svojstva *type_id*.

ExamType

Model **ExamType** služi za definiranje vrste kolokvija. U njemu se nalaze glavne informacije o tome kakvi kolokviji postoje. Sve vrste možemo dohvatiti statičkom metodom **examTypes**. Model ima sljedeća svojstva:

- *id* je primarni ključ koji služi za identifikaciju.
- *name* je ime određenog tipa kolokvija, npr. ”1. kolokvij” ili ”Završni kolokvij”.

Model je u relaciji ”mnogo naprema mnogo” s modelom **Courses** i u relaciji ”jedan naprema mnogo” s modelom **Exam**.

Post

Model **Post** sadrži sve podatke vezane za obavijesti. Ima sljedeća svojstva:

- *id* je primarni ključ koji služi za identifikaciju.
- *user_id* je strani ključ koji označava korisnika koji je objavio obavijest.
- *course_id* je strani ključ koji označava kolegij u sklopu kojeg se obavijest nalazi.
- *title* je naslov obavijesti.
- *note* je sadržaj obavijesti.
- *created_at* i *updated_at* su vremenski žigovi koji označavaju kada je u tablici u bazi podataka kreiran dani model i kada je zadnji put izmijenjen.

Model je u realciji s modelima **Courses** i **User** preko svojstava *course_id*, odnosno *user_id*.

Programm

Model **Programm** predstavlja smjerove na Matematičkom odjelu Prirodoslovno-matematičkog fakulteta. Sve smjerove možemo dohvatiti statičkom metodom **programmes**. Svojstva su mu sljedeća:

- *id* je primarni ključ koji služi za identifikaciju.

- *name* označava ime smjera.
- *year_no* označava koliko godina traje studij.
- *type* označava razinu studija, tj. je li riječ o preddiplomskom ili diplomskom studiju.

Model je u relaciji ”mnogo naprema mnogo” s modelom **Courses**.

Results

Model **Results** predstavlja rezultat studenata na kolokviju. Sastoji se od sljedećih svojstava:

- *id* je primarni ključ koji služi za identifikaciju.
- *info_id* je strani ključ koji označava dodatne informacije za rezultate.
- *user_id* je studentov JMBAG i s kojim ga povezujemo s korisnikom aplikacije.
- *data* sadrži rezultate kolokvija. Rezultati su u JSON formatu i sljedećeg su oblika: {*”ime_stupca”: ”broj_bodova”, ...*} .

Model je u relaciji s modelom **StudentsInfo** pomoću svojstva *user_id* i modelom **ResultsInfo** pomoću svojstva *info_id*.

Model sadrži i tri važne statičke metode. Metoda **deleteByInfo** briše sve modele kojima je *info_id* jednak onom iz argumenta metode. Metoda **getResultsForUserByCourse** dohvaća sve rezultate studenta za određeni kolegij, dok metoda **getResultsForUser** dohvaća sve rezultate neovisno o kolegiju.

ResultsInfo

Model **ResultsInfo** sadrži sve općenite informacije za rezultate nekog kolokvija. Svojstva su sljedeća:

- *id* je primarni ključ koji služi za identifikaciju.
- *course_id* je strani ključ koji označava kolegij u sklopu kojeg se rezultati nalaze.
- *type_id* je strani ključ koji označava vrstu kolokvija koji je objavljen.
- *header* sadrži imena stupaca iz CSV datoteke koja je sadržala dane rezultate.
- *comment* sadrži dodatni komentar administratora kolegija na dane rezulate.

- *created_at* i *updated_at* su vremenski žigovi koji označavaju kada je u tablici u bazi podataka kreiran dani model i kada je zadnji put izmijenjen.

Model je u relaciji s modelom **Courses** pomoću svojstva *course_id* i modelom **ExamType** pomoću svojstva *type_id*. Također, model je u relaciji "jedan naprema mnogo" s modelom **Results**.

Model sadrži metodu **saveResults** koja prima podatke iz CSV datoteke, te ih prilagođava i sprema u model **Results** pomoću relacije među modelima.

StudentsInfo

Model **StudentsInfo** služi za proširivanje informacija o korisniku koji je student. Njegova su svojstva:

- *user_id* koji je strani ključ i označava korisnika.
- *JMBAG* koji predstavjlja jedinstveni matični broj akademskog građanina ili skraćeno JMBAG, od njega zahtijevamo jedinstvenost.

Model je u relaciji s modelom **User** preko svojstva *user_id*, te modelom **Results** pomoću svojstva *JMBAG*.

User

Model **User** opisuje korisnika aplikacije i usko je povezan sa svim opcijama vezanim za autentifikaciju. Njegova svojstva su:

- *id* koji je primarni ključ i služi za identifikaciju.
- *name* koji predstavlja ime korisnika.
- *surname* koji predstavlja prezime korisnika.
- *email* predstavlja jedinstvenu e-mail adresu pomoću koje se korisnik prijavljuje.
- *password* sadrži hashiranu lozinku korisnika.
- *role* predstavlja tip korisnika, tj. je li on profesor ili student.
- *remember_token* sadrži token koji se brine o pamćenju prijave korisnika, kada on izabere opciju "Zapamti me".
- *created_at* i *updated_at* su vremenski žigovi koji označavaju kada je u tablici u bazi podataka kreiran dani model i kada je zadnji put izmijenjen.

Model je u relaciji ”jedan naprema jedan” s modelom **StudentsInfo**. Ta relacija postoji samo kada je korisnik student. S modelom **Post** je u relaciji ”jedan naprema mnogo”, a s modelom **Courses** ”mnogo naprema mnogo”.

Metode **isStudent**, **isProfessor** i **isAdmin** redom određuju je li korisnik student, profesor ili administrator nekog kolegija. Statička metoda **getAllProfessors** služi za dohvaćanje svih profesora, poredanih po prezimenu.

Metoda **publish** za svoj rad koriste Eloquentove mogućnosti s relacijama. Ona sprema novu objavu koristeći relaciju između modela **Post** i **User**, te dodaje korisnika kao autora objave.

2.3 Controlleri i rute

Web-aplikacija za podršku kolegijima ima rute kojima korisnik dolazi do željenog sadržaja. Opisujemo sve web rute i njihove pripadajuće controllere, te za što su one zadužene. Svakoj ruti navodimo i tip ili tipove HTTP zahtjeva, pošto se naoko iste rute razlikuju po njima. Rute dijelimo u skupine s komplementarnim zadaćama i tako ih opisujemo.

Početna stranica

Rute / i *home* odgovaraju na GET i HEAD HTTP zahtjeve. One pozivaju controller **HomeController** i njegovu metodu **index**, te prikazuju početnu stranicu sa svim kolegijima.

Ruta *course*, također, odgovara na GET i HEAD HTTP zahtjeve, ali poziva controller **CourseController** i njegovu metodu **index**. Međutim, ta metoda radi preusmjeravanje na controller **HomeController** i njegovu metodu **index**.

Prijava u aplikaciju

Rutu *login* razdvajamo na dva dijela ovisno o tipu HTTP zahtjeva na koji odgovara. Oba tipa su zadužena za prijavu korisnika i posjetitelj tih ruta mora biti gost, tj. ispunjavati uvjete middlewarea *guest* da korisnik nije trenutno prijavljen.

Ruta s GET i HEAD HTTP zahtjevima prikazuje formu za prijavu pozivajući metodu **showLoginForm** controllera **LoginController**, dok ruta s POST zahtjevom vrši autentifikaciju korisnika i njegovih podataka pozivajući metodu **login** controllera **LoginController**. Ako su podaci ispravni postavlja sesiju za autentificiranog korisnika i preusmjerava na početnu stranicu, dok se u suprotnom vraća na formu za prijavu s pripadajućom obavijesti greške.

Ruta *logout* radi suprotno od rute *login*. Ona odgovara na POST HTTP zahtjeve, te poziva metodu **logout** controllera **LoginController**. Metoda vrši odjavu korisnika brisanjem korisničkih podataka iz sesije, te vraća odjavljenoga korisnika na početnu stranicu.

Izgubljena lozinka

U ovoj skupini se nalaze rute koje su zadužene za promjenu lozinke kada ju korisnik zaboravi. Sve ovdje navedene rute dolaze s middlewarom koji zahtijeva da korisnik nije prijavljen.

Ruta *password/reset*, koja dolazi s GET ili HEAD HTTP zahtjevom, poziva metodu **showLinkRequestForm** controllera **ForgotPasswordController**. Metoda prikazuje formu koja služi za stvaranje zahtjeva za promjenom lozinke.

Ruta *password/email*, koja je pozvana s POST HTTP zahtjevom, poziva metodu **sendResetLinkEmail** controllera **ForgotPasswordController**. Metoda provjerava e-mail adresu iz prethodne forme i šalje na nju link poništavanje lozinke. Ovisno o uspješnosti slanja ispisuje se pripadajuća poruka.

Ruta *password/reset/{token}* poziva se s GET ili HEAD HTTP zahtjevom i nju korisnik dobiva na svoju e-mail adresu prilikom prethodnog koraka. Ona poziva metodu **showResetForm** controllera **ResetPasswordController** koja prikazuje formu za promjenu lozinke, ako je parametar **token** proslijeđen. U suprotnom, prikazuje formu za stvaranje zahtjeva za promjenom lozinke.

Ruta *password/reset*, pozvana s POST HTTP zahtjevom, pokreće metodu **reset** controllera **ResetPasswordController**. Metoda vrši izmjenu lozinke i sprema ju u bazu podataka. U slučaju uspješne promjene vraća korisnika na početnu stranicu, a inače ga vraća na prethodnu stranicu.

Registracija

Ruta *register*, koja je pozvana s GET ili HEAD HTTP zahtjevom, pokreće metodu **showRegistrationForm** controllera **RegisterController**. Metoda prikazuje formu za registraciju novog korisnika.

Ruta *register*, koja je pozvana s POST HTTP zahtjevom, pokreće metodu **register** controllera **RegisterController**. Metoda vrši validaciju podataka. U slučaju da su svi podaci ispravni stvara novog korisnika i automatski ga prijavljuje u aplikaciju te preusmjerava na početnu stranicu. U suprotnom vraća korisnika na prethodnu formu.

Obje rute imaju middleware da korisnik mora biti gost prilikom njihovog pozivanja.

Rad s kolegijima

Opišimo sada rute zadužene za kreiranje, uređivanje i brisanje stranica kolegija.

Ruta *course/create*, koja je pozvana s HTTP zahtjevima GET ili HEAD, pokreće metodu **create** controllera **CourseController**. Metoda prikazuje formu za kreiranje novog kolegija. Također, na rutu je postavljen middleware koji pristupanje ovoj ruti dopušta samo prijavljenim korisnicima koji su profesori.

Ruta *course*, koja je pozvana s POST HTTP zahtjevom, pokreće controller metodu **store CourseController**. Metoda vrši validaciju podataka iz prethodne forme. U slučaju da neki od podataka ne zadovoljava uvjete vraća korisnika na prethodnu formu. Ako je validacija uspješna, kreira novi kolegij u bazi podataka pomoću modela **Courses**, te dodaje sve potrebne relacije za novi kolegij. Na kraju, metoda preusmjerava na stranicu za prikaz obavijesti novokreiranog kolegija. Ova ruta, također, ima middleware koji zahtjeva da je korisnik profesor.

Ruta *course/{course}/edit*, koja ja pozvana s GET ili HEAD HTTP zahtjevom, pokreće metodu **edit** controllera **CourseController**. Metoda prikazuje ispunjenu formu s podacima kolegija čiji identifikator ima vrijednost parametra *course*. Forma nudi mogućnost uređivanja podataka za taj kolegij. Ova ruta ima middleware koji omogućuje pristupanje samo administratorima kolegija. Ovakav middleware imaju i sljedeće dvije rute koje navodimo.

Ruta *course/{course}*, koja je pozvana s PUT ili PATCH HTTP zahtjevom, pokreće metodu **update** controllera **CourseController**. Metoda vrši validaciju kao i metoda **store**. U slučaju neuspjeha vraća se na formu za uređivanje. U slučaju uspjeha ažurira podatke za kolegij čiji je identifikator jednak parametru *course* i sprema ih, te se također preusmjerava na stranicu za prikaz obavijesti.

Ruta *course/{course}*, koja je pozvana s DELETE HTTP zahtjevom, pokreće metodu **destroy** controllera **CourseController**. Metoda briše sve podatke za kolegij čiji identifikator odgovara parametru *course*, i sve podatke vezane nekom relacijom za taj kolegij. Također, briše se direktorij s datotekama tog kolegija.

Obavijesti

Sada opisujemo rute za podstranicu s obavijestima, koja je glavna podstranica za kolegije.

Ruta *course/{course}/posts*, koju pozivamo s GET ili HEAD HTTP zahtjevom, pokreće metodu **index** controllera **PostController**. Metoda prikazuje sve obavijesti za kolegij koji odgovara parametru *course*, s mogućnošću uređivanja i dodavanja novih za administratora kolegija. Ruta *course/course* s identičnim HTTP zahtjevima poziva controllerom **CourseController** i metodu **show**, te se preusmjerava na prethodnu rutu.

Sljedeće rute su zaštićene middlewareom tako da im može pristupiti samo administrator kolegija.

Ruta *course/{course}/posts*, pozvana s POST HTTP zahtjevom, pokreće metodu **store** controllera **PostController**. Metoda vrši validaciju podataka unesenih u formu za novu obavijest. Ako su svi podaci ispravni obavijest se dodaje te se prikaže ponovno stranica s obavijestima gdje sada vidimo i novu obavijest. Ako podaci nisu ispravni vraćamo se na prethodnu stranicu bez unosa promjena.

Ruta *course/{course}/posts/{post}*, pozvana s PUT ili PATCH HTTP zahtjevom, pokreće metodu **update** controllera **PostController**. Metoda prihvaća uređene podatke za obavijest koja odgovara parametru *post*, te nad njima vrši validaciju. U slučaju neuspjeha se vraća na prethodnu stranicu, dok u slučaju uspjeha sprema izmjene te se preusmjerava na početnu stranicu s obavijestima za kolegij koji odgovara parametru *course*.

Ruta *course/{course}/posts/{post}*, pozvana s DELETE HTTP zahtjevom, pokreće metodu **destroy** controllera **PostController**. Metoda briše obavijest koja odgovara parametru *post* i preusmjerava se na stranicu s obavijestima kolegija koji odgovara parametru *course*.

Rezultati

Rute vezane za rezultate izvršavaju složenije radnje od ostalih jer se kod njih javlja i parsiranje.

Prva ruta je koju spominjemo je *results/all*. Ona se poziva s GET ili HEAD HTTP zahtjevom, te ima middleware koji dopušta pristup samo korisnicima koji su studenti. Ona pokreće metodu **results** constrollera **HomeController**, koja prikazuje sve rezultate prijavljenog studenta. Sve ostale rute koje opisujemo su vezane za podstranicu *Rezultati*.

Ruta *course/{course}/results*, pozvana s GET ili HEAD HTTP zahtjevom, pokreće metodu **index** controllera **ResultsController**. Također, ona ima middleware i mogu joj pristupiti samo korisnici koji su ili administratori kolegija ili studenti. Metoda **index** priprema rezultate kolegija, koji odgovara parametru *course*, za prikaz. Prikazi i podaci se razlikuju ovisno o tome je li korisnik student ili administrator kolegija.

Ostale rute su zaštićene middlewareom tako da im mogu pristupiti samo administratori kolegija koji odgovara parametru *course*.

Ruta *course/{course}/results*, pozvana s POST HTTP zahtjevom, pokreće metodu **store** controllera **ResultsController**. Metoda prvo vrši validaciju podataka. Posebno je važna provjera tipa datoteke kako bismo datoteku mogli uspješno parsirati. U slučaju neuspješne validacije vraća se na prethodnu formu za unos rezultata. Zatim se vrši parsiranje CSV datoteke kojim dobivamo rezultate za sve studente koji su pristupili danom kolokviju. Rezultati se zatim spremaju pomoću metode modela zaduženog za rezultate. Nakon uspješnog parsiranja i spremanja, metoda nas preusmjerava na početnu rutu za rezultate.

Ruta *course/{course}/results/{result}*, pozvana s GET ili HEAD HTTP zahtjevom, pokreće metodu **show** controllera **ResultsController**. Metoda dohvaća sve rezultate studenata za kolegij koji odgovara parametru *course* i čije informacije o rezultatima (model **ResultsInfo**) odgovaraju parametru *result*. Svi dohvaćeni rezultati se prikazuju administratoru kolegija.

Ruta *course/{course}/results/{result}*, pozvana s PUT ili PATCH HTTP zahtjevom, pokreće metodu **update** controllera **ResultsController**. Metoda radi analogno metodi **store**, uz par izmjena. Datoteka ne mora biti nužno učitana. Kada je datoteka učitana onda

se prvo brišu svi rezultati čije informacije odgovaraju parametru *results*, te se tek onda provodi parsiranje. Također, normalno se vrši ažuriranje informacija o rezultatima koji odgovaraju parametru *results*. U slučaju kada je ažuriranje uspješno, metoda preusmjerava na početnu rutu rezultata kolegija koji odgovara parametru *course*.

Ruta *course/{course}/results/{result}*, pozvana s DELETE HTTP zahtjevom, pokreće metodu **destroy** controllera **ResultsController**. Metoda pokreće brisanje informacija o rezultatu koja odgovara parametru *result*, i svih rezultata povezanih s tim informacijama o rezultatu. Nakon toga vrši preusmjeravanje na rutu za prikaz rezultata kolegija koji odgovara parametru *course*.

Kolokviji i zadaci

Rute za kolokvije i zadatke vezane su za podstranicu *Zadaci* i vidjet ćemo da je njihov najzanimljiviji dio rad s datotekama.

Ruta *course/{course}/exams*, pozvana s GET ili HEAD HTTP zahtjevom, pokreće metodu **index** controllera **ExamController**. Metoda je zadužena za prikazivanje podstranice *Zadaci* kolegija koji odgovara parametru *course*, pri čemu dohvaća sve potrebne kolokvije i zadatke koji su potrebni za prikaz. Administratoru kolegija se nudi i mogućnost dodavanja i brisanja zadataka.

Daljnje rute su zaštićene middlewareom koji dopušta pristup samo administratoru kolegija.

Ruta *course/{course}/exams*, pozvana s POST HTTP zahtjevom, pokreće metodu **store** controllera **ExamController**. Metoda vrši validaciju podataka iz forme za dodavanje novog kolokvija. Ako je validacija neuspješna metoda se vraća na prethodnu formu. Posebno važan dio ove metode je učitavanje i spremanje PDF datoteke u odgovarajući direktorij. Spremanje se vrši u direktorij vezan s kolegijem koji odgovara parametru *course* iz rute. Zatim se spremaju i podaci modela za dani kolegij. Metoda se nakon uspješnih navedenih radnji preusmjerava rutu za prikaz podstranice *Zadaci*.

Ruta *course/{course}/exams/{exam}*, pozvana s DELETE HTTP zahtjevom, pokreće metodu **destroy** controllera **ExamController**. Metoda briše datoteku i podatke kolokvija koji odgovara parametru *exam*. Metoda se nakon uspješnog izvršavanja preusmjerava na rutu za prikaz podstranice *Zadaci* kolegija koji odgovara parametru *course*.

Ruta *course/{course}/assignments*, pozvana s POST HTTP zahtjevom, pokreće metodu **store** controllera **AssignmentController**. Metoda je analogna metodi **store** controllera **ExamController**, pri čemu umjesto kolokvija radi s zadacima.

Ruta *course/{course}/assignments/{assignment}*, pozvana s DELETE HTTP zahtjevom, pokreće metodu **destroy** controllera **AssignmentController**. I ova metoda je analogna istoimenoj metodi iz controllera **ExamController**, pri čemu radi s zadacima i parametrom *assignment*.

Ostale rute

Ostale su nam još rute koje se javljaju za podstranice kao što su *O kolegiju* i *Materijali*, gdje je korisniku omogućeno maksimalna sloboda uređivanja sadržaja.

Ruta *course/{course}/about/{type}*, koja je pozvana s GET ili HEAD HTTP zahtjevom, poziva metodu **show** controllera **CourseViewController**. Metoda pomoću para parametara *course* i *type* dohvaća model, koji se onda prikazuje. U prikazu ako je korisnik administrator kolegija ima mogućnost uređivanja podataka.

Ako administrator kolegija odluči spremiti izmjene poziva se ruta *course/{course}/about/{type}*, ali ovoga puta s PUT ili PATCH HTTP zahtjevom. Ruta je zaštićena middlewarom koji pristupanje dopušta samo administratoru kolegija. Ruta pokreće metodu **update** controllera **CourseViewController**. Metoda vrši validaciju podataka, sprema izmjene, te preusmjerava korisnika na prethodnu rutu.

Ovime smo opisali sve web rute naše aplikacije.

2.4 Middleware

Middleware nam nudi jednostavan i koristan mehanizam za filtriranje HTTP zahtjeva koji dolaze našoj aplikaciji. Laravel ima unaprijed definirane middleware za autentifikaciju i zaštitu od CSRF napada. Svi middlewarei se nalaze u direktoriju ***app/Http/Middleware***. Kreiramo ih pomoću sljedeće naredbe:

```
1 php artisan make:middleware ImeZaMiddleware
```

Middleware klase implementiraju metodu **handle**. Ona provjerava određene uvjete te ovisno jesu li zadovoljeni dopušta traženi zahtjev, odnosno, preusmjerava na željenu stranicu u slučaju da nisu. U primjeru 2.1 vidimo jedan jednostavni middleware za provjeru da li je korisnik student. Ako je korisnik prijavljen i ako je student dopuštamo izvršavanje zahtjeva, dok u suprotnom preusmjeravamo korisnika na početnu stranicu.

```
1  <?php
2
3  namespace App\Http\Middleware;
4
5  use Closure;
6  use Illuminate\Support\Facades\Auth;
7
8  class Student
9  {
10   public function handle($request, Closure $next)
11   {
12     if( Auth::check() && Auth::user()->isStudent() ) {
```

```
13            return $next($request);
14        }
15
16        return redirect('home');
17    }
18 }
```

Kod 2.1: Middleware za provjeru je li korisnik student

Ponekad nam je potrebno da middleware odradi neke dodatne radnje nakon odgovora HTTP zahtjeva, npr. spremanje nekih dodatnih podataka u sesiju. U takvim slučajevima možemo definirati metodu **terminate**.

Middleware možemo registrirati unutar klase **Kernel** koja se nalazi unutar datoteke ***app/Http/Kernel.php***. Postoje tri tipa middlewarea. Prvi je tip globalni. Takvi middlewarei se pozivaju prilikom svakog HTTP zahtjeva i za njih navodimo našu klasu u svojstvu **$middleware**. Drugi je tip kada želimo specificirati neki middleware samo za određenu rutu ili controller (kao što smo vidjeli u kodu 1.5). Takve navodimo u svojstvu **$routeMiddleware**. Treći su tip grupe. Ponekad želimo pozvati više middlewarea zajedno jednim pozivom kako bismo ih lakše pridružili ruti. Takve navodimo u svojstvu **$middlewareGroups**. Laravel dolazi s pripremljenim grupama **web** i **api**. Ovdje je middleware grupa **web** automatski postavljena na sve rute unutar datoteke ***routes/web.php***

2.5 Pohrana datoteka

Laravel nam nudi moćni alat za rad s datotečnim sustavom pomoću PHP paketa Flysystem. Laravelova integracija Flysystem-a nudi jednostavne upravljačke programe za rad s lokalnim datotečnim sustavom, Amazonom S3 [2] i Rackspace Cloud Storage-om [6]. Štoviše, jednostavno je mijenjati između navedenih opcija za spremanje, dok aplikacija ostaje ista za svaki sistem. Datoteka za podešavanje se nalazi u ***config/filesystems.php***. Unutar njega je moguće podešavati diskove koji predstavljaju određeno spremište podacima i njegovu lokaciju.

Javni disk služi za datoteke koje će biti javno dostupne. Inicijalne su postavke da je lokalni datotečni sustav javni disk i da se datoteke spremaju unutar direktorija ***storage/app/public***. Kako bi datoteke bile dostupne s weba potrebno je stvoriti simboličku vezu između ***public/storage*** i ***storage/app/public***. Nju stvaramo sljedećim pozivom:

```
1 php artisan storage:link
```

Za rad sa spremištem podataka se koristi klasa **Storage**. Ona nam nudi razne metode kao što su: **put** za spremanje datoteka, **get** za dohvaćanje, **download** za kreiranje odgovora

koji prisiljava korisnikov preglednik na preuzimanje podataka, **url** za dobivanje URL-a željene datoteke, pa do raznih metoda za dohvaćanje metapodataka, itd. U sljedećem dijelu koda možemo vidjeti upotrebu metode **storeAs** koja služi za spremanje datoteka koje nam korisnik šalje preko HTTP zahtjeva:

```
$file_name = time() . '_' . $request->examFile->getClientOriginalName();
$directory_name = 'public/exams';
$path = $request->examFile->storeAs($directory_name, $file_name);
```

Također, imamo metode za brisanje. U sljedećem primjeru vidimo primjer metode **delete** koja služi za brisanje datoteka:

```
Storage::delete($path);
```

Isto tako, moguće je raditi s cijelim direktorijima i za to postoje razne metode. Važno je napomenuti da se za svaku od navedenih metoda jednostavno može postaviti i željeni disk. Sustav za pohranu datoteka nudi još razne mogućnosti o kojima se više može pronaći u [10].

2.6 Korisničko sučelje

U web-aplikaciji za podršku kolegijima korištene su i neke vanjske biblioteke. Ukratko opisujemo koje su sve korištene i koja je njihova namjena.

Bootstrap [3] je najpopularnija biblioteka za responzivni prikaz web-stranica. On omogućava korisniku da postavljanjem klasa HTML objektima jednostavno uređuje sadržaj, bez da mora brinuti o prikazu na različitim zaslonima. U našoj aplikaciji dizajn je u potpunosti baziran na njemu.

DataTables [4] je dodatak Javascript biblioteci jQuery. Njime nam je omogućeno dodavanje različitih korisnih svojstava HTML tablicama, npr. paginaciju, pretraživanje, filtriranje po stupcima, sortiranje po jednom ili više stupaca itd. U našoj web-aplikaciji služi za prikaz zadataka s ciljem da se lakše mogu naći zadaci od interesa korisniku.

jQuery [5] je mala i brza Javascript biblioteka puna mogućnosti. Ona omogućava jednostavniju manipulaciju HTML objektima, rad s događajima, i sličnim zahtjevima. U našoj web-aplikaciji korišten je u kombinaciji s Bootstrapom za rad s modalnim dijaloškim okvirima, te je neophodan za rad ostalih biblioteka kao što su DataTables, Select2 i TinyMCE.

Select2 [7] omogućuje veću funkcionalnost padajućih izbornika u web-aplikaciji. Njime se pojednostavljuje rad s većim količinama podataka, omogućuje pretraživanje među izborima, te nudi višestruki odabir. U našoj web-aplikaciji se koristi prilikom dodavanja i izbacivanja administratora na kolegiju.

TinyMCE [8] je moćan i jednostavan uređivač teksta koji omogućuje ljudima s raznim razinama znanja stvaranje različitog sadržaja na web-stranicama. Njime se jednostavno kreira složeni sadržaj bez ikakvog znanja HTML-a ili CSS-a. U našoj web-aplikaciji korišten je za stranice kao što su *O kolegiju* i *Materijali* gdje korisniku želimo dati maksimalnu slobodu kreiranju i organizaciji sadržaja.

Ovime smo opisali sve važne dijelove naše složene web-aplikacije za podršku kolegijima. Premda je sama web-aplikacija složena, zahvaljujući Laravelu se ona jednostavno kreira. Metode su većinom jednostavne zahvaljujući mnogim Laravelovim mogućnostima i njegovoj strukturalnoj raspodijeli, kod se jednostavno dijeli, te je lak za pratiti prilikom rada u timu. Zaključujemo da je Laravel odličan razvojni okvir koji olakšava i ubrzava proces izrade raznih web-aplikacija, pogotovo kada ga uspoređujemo s PHP-om bez razvojnog okvira. Također, važno je napomenuti da Laravel nema strmu krivulju učenja, što je programerima izrazito važno kada se susretnu s njime.

Bibliografija

[1] *Laravel 5.4 From Scratch*, https://laracasts.com/series/laravel-from-scratch-2017, 2017, posjećeno u kolovozu 2018.

[2] *Amazon S3*, https://aws.amazon.com/s3/, 2018, posjećeno u listopadu 2018.

[3] *Bootstrap*, https://getbootstrap.com/, 2018, posjećeno u listopadu 2018.

[4] *DataTables*, https://datatables.net/, 2018, posjećeno u listopadu 2018.

[5] *jQuery*, https://jquery.com/, 2018, posjećeno u listopadu 2018.

[6] *Rackspace - Cloud Files Online Object Storage*, https://www.rackspace.com/cloud/files, 2018, posjećeno u listopadu 2018.

[7] *Select2 - Getting started*, https://select2.org/getting-started, 2018, posjećeno u listopadu 2018.

[8] *Tiny - Quick Start*, https://www.tiny.cloud/docs/quick-start/, 2018, posjećeno u listopadu 2018.

[9] *Tutorialspoint - RESTful Web Services*, https://www.tutorialspoint.com/restful/restful_introduction.htm, 2018, posjećeno u listopadu 2018.

[10] *Web stranice razvojnog okvira Laravel*, https://laravel.com/docs/5.6, 2018, posjećeno u kolovozu 2018.

[11] M. Stauffer, *Laravel: Up and Running*, O'Reilly Media, 2016.

Sažetak

U ovom smo radu prezentirali razvojni okvir Laravel za programski jezik PHP. Baziran je na arhitekturalnom obrascu *model-view-controller*, besplatan je i otvorenog je koda.

U prvom smo poglavlju opisivali osnovne dijelove Laravela. Za početak smo opisali kako ga instalirati i koje nam zanimljivosti nudi pri kreiranju web-aplikacija. Naglasak smo stavili na opisivanju kako izgleda arhitekturalni obrazac MVC u Laravelu. Vidjeli smo da donosi dvije veoma korisne novosti: Eloquent i Blade. Opisali smo i kako izgleda usmjeravanje, kao važan dio za rad web-aplikacija. Također, spomenuli smo i kako Laravel proširuje arhitekturalni obrazac MVC pomoću migracijskih skripti i klasa za punjenje baze podataka. Dodatno smo govorili o Laravelovom pripremljenoj autentifikaciji. Za kraj poglavlja smo pričali i o mogućnostima s korisničke strane web-aplikacije.

U drugom smo poglavlju opisivali složeniju web-aplikaciju za podršku kolegijima na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta. Prvo smo prikazali upotrebu web-aplikacije i sve njezine mogućnosti. Zatim smo krenuli u opisivanje modela pomoću kojih je web-aplikacija napravljena. Naveli smo i sve rute u aplikaciji, te za što služe i kako rade. Opcije Laravel smo proširili s opisivanjem naprednijih mogućnosti, kao što su middleware i pohrana datoteka. Za kraj smo opet govorili o korisničkom sučelju. Proširili smo ga bibliotekama korištenim u web-aplikaciji.

Summary

In this thesis we have presented PHP framework Laravel. Laravel is free and open-source framework based on model-view-controller architectural pattern.

In the first chapter, we described the main parts of Laravel. We began with the installation and the description of the features it offers for creation of web-applications. We focused on how the MVC architectural pattern is implemented in Laravel. We saw two new tools which Laravel has: Eloquent and Blade. We described how to implement routing, as it is a vital part of a web-application. Moreover, we mentioned Laravel's migration scripts and the seeder class which extend the MVC architectural pattern. Additionally, we talked about Laravel's out-of-the-box solution for authentication. In the end of the chapter, we talked about frontend development for Laravel web-applications.

In second chapter, we described a more complex web-application whose purpose is to provide web-support for the courses taught at the Department of Mathematics, Faculty of Science. Firstly, we shown use cases of the web-application and all the possibilities of using it. Afterwards, we described all of the models in the web-application. We mentioned all of the routes, what are they used for and how they function. We extended our description of the Laravel framework with the more advanced options: middleware and file storage. In the end, we discussed the frontend side of our application, and the various libraries that were used for its implementation.

Životopis

Rođen sam 21. prosinca 1994. godine u Zagrebu. Nakon završene Osnovne škole Borovje 2009. godine upisujem V. gimnaziju u Zagrebu. Nakon završene srednje škole 2013. godine upisujem preddiplomski studij matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. Stjecanjem titule sveučilišnog prvostupnika matematike 2016. godine završavam preddiplomski studij. Iste godine upisujem diplomski studij Računarstva i matematike, također na Prirodoslovno-matematičkom fakultetu u Zagrebu.

Sudionik sam Ericsson Nikola Tesla Summer Camp-a 2017. na kojem stječem certifikat "Challenges for networked society". Nakon njega nastavljam studentski raditi u Ericsson Nikola Tesla Servisi kao programer u programskom jeziku PHP u razdoblju od rujna 2017. godine do veljače 2018. godine. U travnju 2018. godine počinjem studentski raditi u AVL-AST kao frontend developer na aplikacijskoj platformi Angular.