



UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

---

**Elektronske lekcije  
o radnom okviru Laravel**

---

MASTER RAD

*Student*  
Katarina Andrejević

*Mentor*  
dr Miroslav Marić

Beograd,  
2018

## Sadržaj

Uvod	2
1 Istorija radnog okvira Laravel	3
2 Arhitektura MVC	3
3 Instalacija radnog okvira Laravel	4
4 Struktura fajlova radnog okvira Laravel	7
5 Rutiranje	8
6 Kontroleri	10
7 Izgled stranice i nasleđivanje interfejs šablona	12
8 Redirekcija	15
9 Konekcija sa bazom i migracije	16
10 Rad sa tabelama i kolonama	18
11 Objektno-relaciono preslikavanje	21
11.1 Upravljanje podacima iz komandnog prozora . . . . .	22
12 Upisivanje podataka u bazu, izmena podataka i brisanje po- dataka	23
12.1 Ispis podataka iz baze . . . . .	27
12.2 Izmjena podataka u bazi . . . . .	29
12.3 Brisanje podataka iz baze . . . . .	31
13 Autentifikacija	32
14 Ubacivanje fajla na sajt	35
15 Kreiranje aplikacije u radnom okviru Laravel	40
16 Elektronske lekcije	44
Zaključak	47
Literatura	48

## Uvod

Internet je u današnje vreme sredstvo komunikacije, informisanja i deo posla miliona ljudi širom sveta. Zbog toga svakodnevno raste broj veb aplikacija. Dostupno je mnoštvo tutorijala za učenje veb tehnologija. Među njima je platforma eŠkola veba koja je dostupna na linku [http://edusoft.matf.bg.ac.rs/eskola\\_veba/#/home](http://edusoft.matf.bg.ac.rs/eskola_veba/#/home). Na ovoj platformi se nalaze besplatni kursevi različitih veb tehnologija. Svaki kurs se sastoji od elektronskih lekcija. Više o eŠkoli veba može se videti u [2].

Mnogo je programskih jezika koji se koriste za izradu veb aplikacija, pa se razvijaju i radni okviri koji olakšavaju njihovu upotrebu. Laravel je radni okvir otvorenog koda za programski jezik PHP, koji je jedan od najpopularnijih programskih jezika za izradu veb aplikacija. Međutim, kod velikih aplikacija kôd je obiman i ima mnogo ponavljanja, a za rešavanje takvih problema koristi se radni okvir. Među najpopularnijim radnim okvirima za programski jezik PHP su: Laravel, Symfony, CodeIgniter, CakePHP i Zend.

U ovom radu biće prikazani osnovni koncepti radnog okvira Laravel. Na početku će biti opisana arhitektura MVC na kojoj je izgrađen Laravel, biće prikazano kako se instalira ovaj radni okvir i biće opisana njegova struktura fajlova. Nakon toga biće prikazano rutiranje i kako se koristi za kreiranje URL adresa. Jedan od najosnovnijih koncepata radnog okvira Laravel su kontroleri, jer se sve funkcije za rad sa podacima iz baze podataka nalaze u njima. Jedna od mogućnosti radnog okvira Laravel je šablon za nasleđivanje stranica pomoću kojeg se jednostavno kreiraju veb stranice, bez nepotrebnog ponavljanja koda, što će detaljnije biti objašnjeno u narednim poglavljima. Nakon toga biće prikazana redirekcija i kako se ona koristi za vraćanje na željenu veb stranicu ili na istu stranicu ukoliko je došlo do greške. Biće objašnjena konekcija sa bazom podataka i rad sa podacima iz baze. Zatim će biti kreirana aplikacija o studentima, kao primer pomoću kojeg će biti objašnjeno kako se ispisuju podaci iz baze, kako se upisuju u bazu, kako se menjaju podaci i kako se brišu podaci iz baze podataka. U poglavljima nakon toga biće prikazana autentifikacija i kako se pravi sistem za logovanje, a zatim kako ubaciti fajl na sajt. Za razumevanje radnog okvira Laravel neophodno je poznavanje programskog jezika PHP.

Na kraju, sve obrađene teme biće iskorišćene za kreiranje aplikacije koja je zamišljena kao blog stranica za turizam gde će registrovani korisnici moći da podele svoja iskustva pišući o mestima koja su posetili. Korisnici će moći da unose slike i opis putovanja i moći će da menjaju svoje objave i da ih brišu. Stranice aplikacije koja predstavlja blog o putovanjima biće stilizovane u radnom okruženju Bootstrap.

## 1 Istorija radnog okvira Laravel

Tvorac radnog okvira Laravel je Tejlor Otvel (Taylor Otwell). Prva verzija Laravela se pojavljuje u junu 2011. godine. Do pojave radnog okvira Laravel, najpopularniji radni okvir za programski jezik PHP bio je CodeIgniter, a prva verzija Laravela imala je za cilj da popuni nedostatke prethodnih radnih okvira (bilo ih je više).

Laravel je još u svojoj prvoj verziji imao ugrađenu autentifikaciju, Eloquent ORM za operacije sa podacima iz baze, imao je keševe, sesije, opciju view za izgled veb stranica, kao i jednostavan mehanizam za rutiranje. Tejlor je proširivao ovaj radni okvir i za šest meseci se pojavila njegova druga verzija.

Nova verzija objavljena je u novembru 2011. godine. Jedna od značajnijih nadogradnji radnog okvira Laravel bilo je dodavanje kontrolera, koji su preko zahteva koji korisnik šalje regulisali pristup veb stranici. O kontrolerima će više reći biti u narednim poglavljima. Značajan deo nadogradnje su šabloni za nasleđivanje strukture izgleda stranica, što će kasnije u radu biti detaljno objašnjeno.

Treća verzija radnog okvira Laravel objavljena je u februaru 2012. godine. Ova verzija bila je fokusirana na komande koje su pisane u liniji komandnog prozora, na migracije baze podataka i više na sesije.

Četvrta verzija radnog okvira Laravel objavljena je u maju 2013. godine. Iako je dobro što se okvir razvija, pojavljivanje novih verzija smanjilo je kredibilitet ovog radnog okvira. Problem je bio velike aplikacije prevesti na novu verziju. Laravel 4 je napisan kao kolekcija paketa koji se integrišu u kôd. Upravljanje komponentama vrši se preko menadžera programskog jezika PHP koji se zove Composer. Najnovija, peta verzija radnog okvira Laravel izašla je u januaru 2015. godine. Za razliku od nekih prethodnih verzija čiji su kodovi pisani od početka, sada izlaze kodovi sa unapređenom prethodnom verzijom i izlaze na svakih šest meseci. Više o istoriji radnog okvira Laravel se može pročitati u literaturi [8].

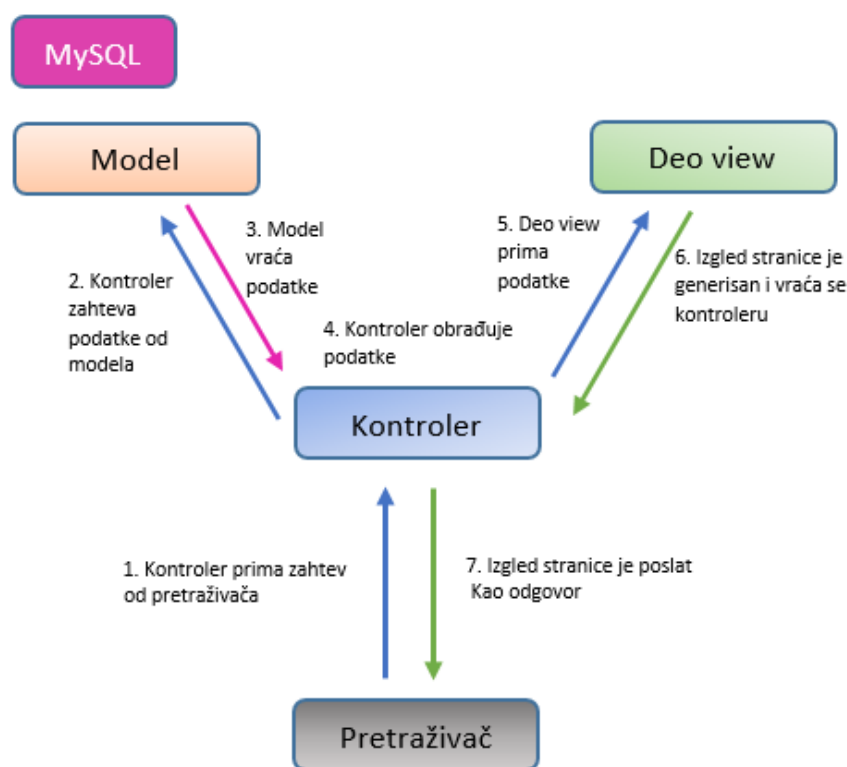
## 2 Arhitektura MVC

Na arhitekturi MVC, čiji je pun naziv Model View Controller, zasnivaju se neki od najpopularnijih radnih okvira za izradu veb sajtova i aplikacija, a među njima i Laravel. Koristi se u velikim aplikacijama koje rade sa ogromnom količinom podataka. Programeri ih koriste da bi razdvojili kôd koji se bavi podacima od koda koji se bavi interfejsom, jer je razvoj tako odvojenih delova aplikacije jednostavniji.

Logika aplikacije koja je izgrađena na arhitekturi MVC je takva da se zahtevi šalju kontroleru koji pomoću modela generiše potrebne podatke. Podaci pomoću dela view stižu do korisnika. Deo view, odnosno pogled

ili izgled stranice je interfejs aplikacije i sve ono što korisnik vidi, model predstavlja logiku aplikacije, a kontroler ima ulogu koordinatora između njih. Deo view i kontroler zavise od modela, dok model od njih ne zavisi.

Model sadrži glavne podatke kao što su informacije o objektima iz baze podataka. Svi podaci se dobijaju od modela, ali se on ne može direktno pozvati, već sve ide preko kontrolera. Kad obradi zahtev, kontroler traži od modela podatke. Kontroler podatke prosleđuje pogledu, koji ih prikazuje krajnjem korisniku. Deo view ili pogled je poslednji sloj arhitekture MVC. Korisnik vidi samo ono što pogled prikazuje, dok su model i kontroler skriveni za korisnika. Pogled nikad ne obrađuje podatke i njegova uloga u aplikaciji se završava kada su podaci prikazani. Kontroler definiše ponašanje same aplikacije. On je prvi sloj koji se poziva nakon što se u pretraživač unese URL adresa. Kontroler obično poziva model za izvršenje zahteva i bira odgovarajući pogled. Lep prikaz arhitekture MVC dat je na slici 1.



Slika 1: Arhitektura MVC

### 3 Instalacija radnog okvira Laravel

Laravel je besplatan radni okvir otvorenog koda. U narednom delu biće prikazani koraci kako se instalira. Laravel je radni okvir programskog jezika

PHP koji se interpretira na serveru, pa je neophodno instalirati program koji u sebi sadrži Apache server koji izvršava programski jezik PHP. U radu će biti korišćen program EasyPHP. Potrebna je 7.0.0 verzija jezika PHP, pa ju je neophodno preuzeti. EasyPHP se može preuzeti sa <http://www.easyphp.org/>. Potrebno je preuzeti verziju 17.0 programa EasyPHP. Kada se instalira, na adresi <http://127.0.0.1:1111/> se otvara stranica kao na slici 2.



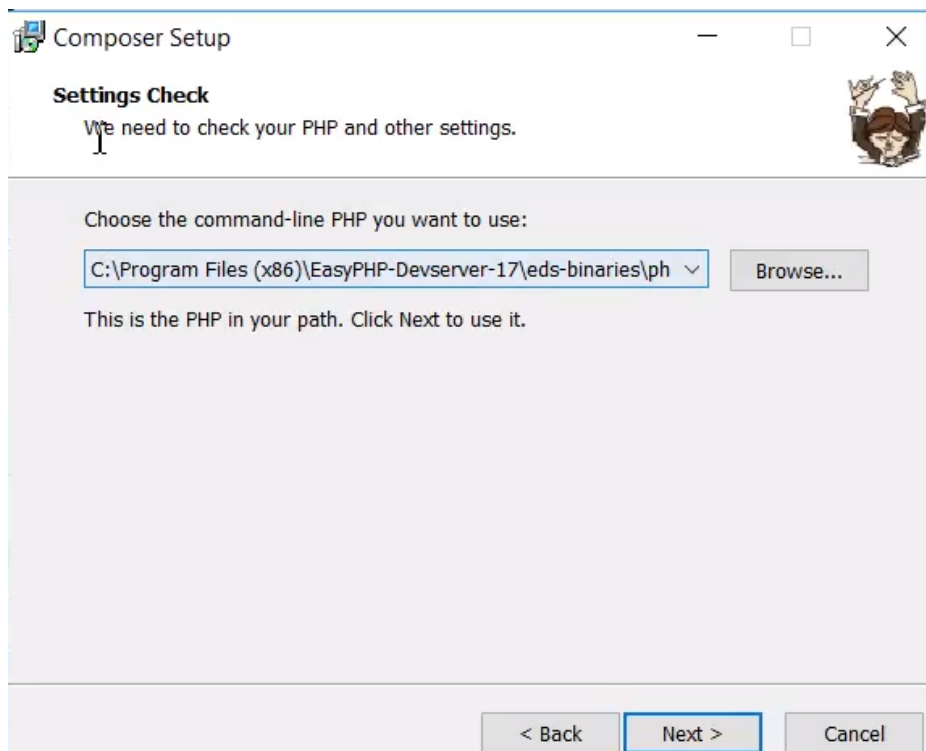
Slika 2: Provera verzije programskog jezika PHP

Ako je u selektovanom delu 5.6.3 verzija jezika PHP, onda je neophodno u podešavanjima servera, u padajućem meniju, odabrati 7.1.3 verziju jezika PHP i kliknuti dugme start kao što je prikazano na slici 3.



Slika 3: Podešavanje odgovarajuće verzije jezika PHP

Da bi se Laravel koristio na korisničkom računaru, mora se instalirati Composer. Composer je aplikacija koja omogućava korisnicima da upravljaju komponentama od kojih se sastoji radni okvir Laravel. Koristi se za instalaciju, ažuriranje i uklanjanje paketa programskog jezika PHP. Instalacija aplikacije Composer se može preuzeti sa adrese <https://getcomposer.org/>. Klikom na Composer-Setup.exe pokreće se instalacija. Kada se otvori prozor kao na slici 4, treba kliknuti na browse.



Slika 4: Odgovarajuća putanja

Zatim treba naći EasyPHP-Devserver-17, pa odabrati folder `eds-binaries`. Sledeće što treba odabrati je folder `php`, a zatim folder čiji je naziv `php713vc14x86x171101093849`. Kada se otvori poslednji folder treba kliknuti na `php.exe` i `open`. Nakon što se ovo uradi, treba pokrenuti instalaciju klikom na `next`, pa ponovo na `next` i zatim kliknuti `install`. Instalacija se nastavlja iz komandnog prozora. Neophodno je prekopirati

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

u komandni prozor i instalacija će se pokrenuti. Nakon toga se može nastaviti sa radom. Iz komandnog prozora se zadaje ime aplikacije. To se radi tako što se u komandnom prozoru napiše sledeća komanda,

```
laravel new nazivProjekta
```

ali se pre ove komande treba pozicionirati u folder `eds-www` komandom `cd`. Putanja tog foldera je podrazumevano zadata sa `C:\Program Files (x86)\EasyPHP-Devserver-17\eds-www`, pa je u sledećem redu prikazano kako se treba pozicionirati u folder čija je putanja zadata u prethodnom redu.

```
cd C:\Program Files(x86)\EasyPHP-Devserver-17\eds-www
```

Folder `nazivProjekta` se nalazi u folderu `eds-www`. Na kraju, da bi se pokrenula aplikacija u projektu u radnom okviru Laravel, neophodno je po-

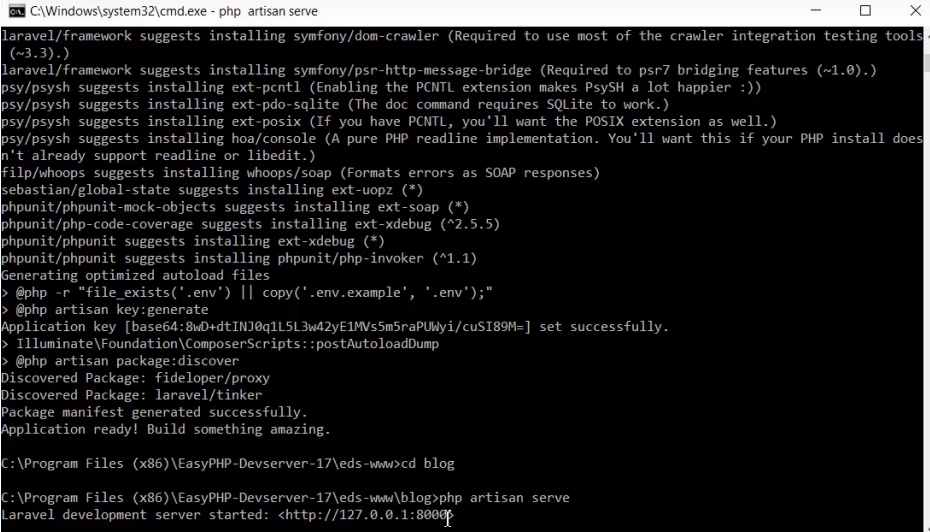
zicionirati se u kreirani folder nazivProjekta komandom koja je prikazana u sledećem redu.

```
cd nazivProjekta
```

Nakon toga sa:

```
php artisan serve
```

iz komandnog prozora, server će biti pokrenut.



```
C:\Windows\system32\cmd.exe - php artisan serve
laravel/framework suggests installing symfony/dom-crawler (Required to use most of the crawler integration testing tools (~3.3).)
laravel/framework suggests installing symfony/psr-http-message-bridge (Required to psr7 bridging features (~1.0).)
psy/psysh suggests installing ext-pcntl (Enabling the PCNTL extension makes PsySH a lot happier :)
psy/psysh suggests installing ext-pdo-sqlite (The doc command requires SQLite to work.)
psy/psysh suggests installing ext-posix (If you have PCNTL, you'll want the POSIX extension as well.)
psy/psysh suggests installing hoa/console (A pure PHP readline implementation. You'll want this if your PHP install does n't already support readline or libedit.)
filp/whoops suggests installing whoops/soap (Formats errors as SOAP responses)
sebastian/global-state suggests installing ext-uopz (*)
phpunit/phpunit-mock-objects suggests installing ext-soap (*)
phpunit/php-code-coverage suggests installing ext-xdebug (^2.5.5)
phpunit/phpunit suggests installing ext-xdebug (*)
phpunit/phpunit suggests installing phpunit/php-invoker (^1.1)
Generating optimized autoload files
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
> @php artisan key:generate
Application key [base64:8w0+dtIND0q1L5L3w42ye1MN5m5naPUWyi/cuSI89M=] set successfully.
> illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover
Discovered Package: fideloper/proxy
Discovered Package: laravel/tinker
Package manifest generated successfully.
Application ready! Build something amazing.

C:\Program Files (x86)\EasyPHP-Devserver-17\eds-www>cd blog
C:\Program Files (x86)\EasyPHP-Devserver-17\eds-www\blog>php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
```

Slika 5: Pravljenje projekta i pokretanje servera

Kada se `http://127.0.0.1:8000` kopira u pretraživač, prva aplikacija u radnom okviru Laravel će biti pokrenuta.

## 4 Struktura fajlova radnog okvira Laravel

Glavni folder koji je kreiran u komandnom prozoru podrazumevano sadrži mnoštvo foldera. Najbitniji i često korišćeni folderi pri izradi aplikacija su:

- **app** - u njemu se nalazi logika aplikacije i tu se nalazi osnovni kôd razvijene aplikacije koji se nalazi u kontrolerima i modelima,
- **config** - sadrži konfiguracijske datoteke aplikacije,
- **database** - sadrži u sebi sve migracijske fajlove,
- **public** - sadrži sve medijske fajlove i fajlove pisane u jeziku CSS i JavaScript,
- **resources** - u ovom folderu se nalaze svi fajlovi u kojima su definisani izgledi veb stranica koji će nakon svih potrebnih povezivanja korisnik moći da vidi i svi fajlovi se imenuju sa `nazivFajla.blade.php`,



- `routes` - u ovom folderu, u fajlu `web.php` su sve definisane rute aplikacije,
- i `.env` fajl - zadužen za konekciju sa bazom podataka.

Više o strukturi fajlova u radnom okviru Laravel se može pročitati u literaturi [3] i [5]. U radu će više puta biti naglašeno gde se čuvaju fajlovi koji će biti korišćeni za izradu aplikacija.

## 5 Rutiranje

Rutiranje služi za podešavanje putanja unutar aplikacije. Putanje su URL adrese ka stranicama aplikacije. Može se reći da rute određuju navigaciju unutar aplikacije, jer vode na određite koje predstavljaju veb strane aplikacije. Na primer sa:

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

otvara se stranica `welcome.blade.php`, a URL koji vodi do nje bi bio `http://127.0.0.1:8000`. Prvi argument metode `get` određuje šta je potrebno dodati u pretraživač nakon porta 8000. U konkretnom primeru, ako korisnik ode na adresu `http://127.0.0.1:8000`, što je adresa kojom je pokrenuta aplikacija u radnom okviru Laravel, tada se otvara stranica čiji je izgled definisan u fajlu `welcome.blade.php`. Bez rutiranja ne postoji mogućnost povezivanja sa krajnjim korisnikom. U fajlu `web.php` koji se nalazi u folderu `routes` se nalaze sve definisane rute, tj. putanje, kojima su povezane stranice aplikacije. Stranicama na koje vode zadate rute se pristupa navođenjem rute u pretraživaču. Na primer sa:

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

je definisana putanja `http://127.0.0.1:8000/about`. Ako se u pretraživaču posle porta 8000 doda `about`, otvara se stranica čiji je izgled zadat u fajlu `about.blade.php` koji se čuva u folderu `views`. Taj folder se nalazi u folderu `resources`. Kôd fajla `about.blade.php` je dat u nastavku.

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

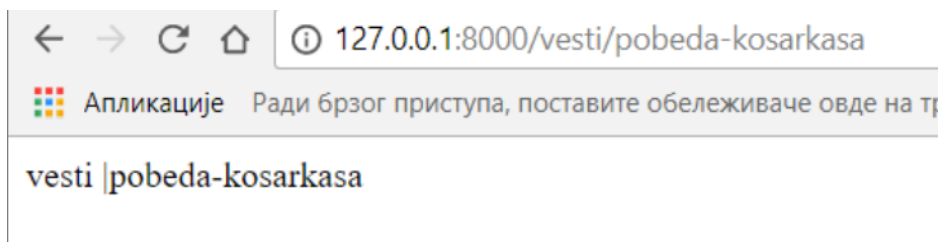
```
About Us
</body>
</html>
```

Na veb stranici se samo ispisuje naslov About Us.

Kao argument mogu se preneti parametri koji ne moraju biti fiksni. To se zove parametarsko rutiranje. Tada se oni navode kao argumenti anonimne funkcije koja je prosleđena kao drugi argument definisane putanje. Tako se na primer sa:

```
Route::get('/vesti/{$naslov}', function ($naslov) {
return echo "vesti |".$naslov
});
```

definiše putanja `http://127.0.0.1:8000/vesti/pobeda-kosarkasa`. Ovaj link vodi do stranice koja je prikazana na slici 6.



Slika 6: Parametarsko rutiranje

Svaki HTTP zahtev ima glagol i akciju koja ga prati. Glagolom GET podaci se šalju preko URL zahteva, dok se sa POST ne mogu slati preko URL zahteva. Osim ovih, mogu se koristiti PUT, DELETE i PATCH glagoli.

Moguće je praviti imenovane rute, a to doslovno znači da se cela ruta imenuje i da se njeno ime koristi kasnije, na primer u redirekciji. O redirekciji će biti više reči u nekom od narednih poglavlja. Jedan primer imenovane rute dat je u sledećem redu.

```
Route::get('user/profile', 'UserController@showProfile')->name
    -> ('profile');
```

Na primer ako se ova ruta iskoristi kao link, trebalo bi njeno ime navesti kao vrednost atributa href. Taj deo koda je prikazan u sledećem redu.

```
<a href="{{route('profile')}}">Idi na profil</a>
```

Tada se otvara stranica `http://127.0.0.1:8000/user/profile` koja poziva funkciju `showProfile` iz kontrolera. Više o imenovanim rutama se može pročitati u literaturi [3]. O kontrolerima će više biti reči u sledećem poglavlju.

## 6 Kontroleri

Kontroleri su spona između izgleda, tj. dela view i modela aplikacije, a to znači da se na osnovu unosa podataka i zahteva koji korisnik šalje kontroliše pristup veb stranici. Kontrolisanje pristupa veb stranici znači da kontroler preuzima korisničke zahteve od dela view, prosleđuje ih modelu koji na osnovu zahteva uzima podatke iz baze i prosleđuje ih u deo aplikacije koji je zadužen za izgled, tj. u deo view. Kontroler se može kreirati ručno, tako što se u folderu `app\Http\Controllers` kreira fajl naziv `Controller.php`. Međutim, kontroler se može kreirati korišćenjem komande `php artisan make:controller nazivController`. Na primer sa:

```
php artisan make:controller PagesController
```

kreiran je kontroler. Kôd u kontroleru se prethodnom komandom automatski generiše i prikazan je u nastavku.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PagesController extends Controller
{
}

```

Ako se doda sledeća funkcija

```
public function home()
{
    return view('welcome');
}

```

u telo klase u kontroleru, onda ona vraća početnu stranicu `welcome`. Potrebno je definisati putanju do te stranice. Deo koda koji to definiše dat je u sledećem redu.

```
Route::get('/', 'PagesController@home');
```

Dakle, ovim je definisano da ako se posle porta 8000 ne navede ništa, onda se poziva funkcija `home` iz kontrolera koja vodi na stranicu `welcome`. Na primer, posle porta se može navesti putanja, a drugi argument je funkcija koja se poziva iz kontrolera i ona mora biti globalna. Tako da ako se u fajlu `web.php` doda

```
Route::get('about', 'PagesController@about');
```

i ako se u kontroleru doda globalna funkcija

```
public function about()
{
    return view('aboutUs');
}
```

onda se sa `http://127.0.0.1:8000/about` ide na stranicu čiji je kôd u fajlu `aboutUs.blade.php`.

Laravel ima mogućnost upravljanja podacima pomoću takozvanog resursnog kontrolera. Ovaj kontroler sadrži metode:

- index,
- create,
- store,
- show,
- edit,
- update,
- i destroy.

U sledećem redu prikazana je komanda kojom se automatski kreira kontroler sa nabrojanim metodama za obradu podataka.

```
php artisan make:controller nazivKontrolera --resource
```

Komandom koja je prikazana u sledećem redu mogu se izlistati sve metode sa svojim putanjama i akcijama.

```
php artisan route:list
```

Na primer, na slici 7 izlistane su sve metode sa putanjama za napravljeni kontroler koji se zove `StudentController`. Ovaj kontroler će kasnije u radu biti korišćen za izradu aplikacije kroz koju će biti prikazan rad sa podacima iz baze.

GET HEAD	student	student.index	App\Http\Controllers\StudentController@index
POST	student	student.store	App\Http\Controllers\StudentController@store
GET HEAD	student/create	student.create	App\Http\Controllers\StudentController@create
GET HEAD	student/{student}	student.show	App\Http\Controllers\StudentController@show
PUT PATCH	student/{student}	student.update	App\Http\Controllers\StudentController@update
DELETE	student/{student}	student.destroy	App\Http\Controllers\StudentController@destroy
GET HEAD	student/{student}/edit	student.edit	App\Http\Controllers\StudentController@edit

Slika 7: Putanje svih funkcija iz resursnog kontrolera

Sada se postavlja pitanje kako sve putanje napisati u fajlu `web.php`. Vrlo jednostavno, tako što se u fajlu `web.php` doda kôd koji je prikazan u sledećem redu.

```
Route::resource('student', 'StudentController');
```

Prethodnom linijom koda dobijaju se rute za sve metode koje su definisane u resursnom kontroleru, a koje su prikazane na slici 7. Više o kontrolerima se može pročitati u literaturi [1], [6] i [10].

## 7 Izgled stranice i nasleđivanje interfejs šablona

Izgled ili deo view arhitekture MVC na kojoj je izgrađen Laravel ima mogućnost kreiranja veb stranica pomoću šablona Blade. Blade je jednostavan, ali veoma jak alat Laravela i služi za pravljenje interfejs šablona, što će biti prikazano pravljenjem jednostavne stranice. Šablonom se može nazvati kreirana stranica koju će naslediti sve druge stranice. Ona se naziva šablonom jer ima statički kreiran kostur koji se ne menja, a sve ostale stranice je nasleđuju sa dodatkom dinamičkog dela koji je drugačiji za svaku od stranica. Primer je preuzet iz literature [6].

Folder `views` se nalazi u folderu `resources`. Tu se čuva kôd kojim je definisan izgled svih stranica. Svi potrebni fajlovi pisani u jeziku CSS i JavaScript se čuvaju u folderu `public`. U ovoj lekciji biće kreiran prvo statički deo, a zatim i dinamički deo stranice. Biće napravljen kontroler sa nazivom `MyController`. Za potrebe bolje organizacije fajlova mogu se kreirati dodatni folderi. Ovde će biti kreiran folder `pages`. Zatim, u folderu `pages` treba kreirati fajl `master.blade.php` (može se nazvati i drugačije). To će biti osnova koju će naslediti sve druge stranice. Kôd koji sledi je kôd fajla `master.blade.php`.

```
<!DOCTYPE html>
<html>

  <head>
    @yield('head')
    <link rel='stylesheet' href='/css/style.css'>

    <title>
      @yield('title')
    </title>

  </head>

  <body>
```

```

<div class="container">
<div class="heading">
@yield('heading')
</div>
<div class="content">
@yield('content')
</div>
<div class="footer">@yield('footer')</div>
</div>

</body>

</html>

```

Putanja fajla `master.blade.php` je `resources/views/pages/master.blade.php`.

Direktiva `yield` koja se koristi se poziva sa `@`. Ona uključuje dinamički promenljiv sadržaj. Kôd stranice koju će ostale stranice naslediti je podeljen na celine direktivom `yield`. Kôd stilizovanja stranice za ovaj primer se može pogledati u literaturi [6] u sekciji View and Blade. Naredni kôd predstavlja fajl `umetnost.blade.php`. Stranica `umetnost.blade.php` će biti jedna od stranica koja nasleđuje šablon.

```

@extends('pages.master')

@section('head')
<!--<link rel='stylesheet' href='/css/style.css'-->
@stop
@section('title')
Ovo je neka stranica
@stop
@section('heading')
{{ $ime }} {{ $profesija }}
@stop
@section('content')
<div class='h1'>
Sloboda pisca je dar.
</div>
<p>
Dovoljno je nekoliko sekundi inspiracije. Dovoljno je nekoliko
    ↪ sekundi inspiracije.
Dovoljno je nekoliko sekundi inspiracije. Dovoljno je nekoliko
    ↪ sekundi inspiracije.
Dovoljno je nekoliko sekundi inspiracije.
</p>

```

```
<div class='h2'>
Inspiracija
</div>
<p>
Jave i san
</p>
@stop
@section('footer')
Home of Katrin
@stop
```

Korišćena je direktiva `@extend`, što znači da stranica nasleđuje fajl `master.blade.php`. Korišćene su još dve direktive, `section` i `stop`, pri čemu svaka sekcija ima svoj sadržaj. `@stop` služi za kraj direktive `@section`, a za kraj ove direktive koristi se i `@endsection`. Unutar sekcije `head` ubačen je link kojim se uključuje CSS. Međutim, kako je stil ubačen u fajl `master.blade.php`, nema potrebe da se ubacuje u fajl `umetnost.blade.php`, jer će i stizovanje biti nasleđeno. U sekciji `heading`, na primer, može se pisati sledeće:

```
@section('heading')
{{ $ime }} {{ $profesija }}
@stop
```

i ono što je `echo $ime;` u jeziku PHP, to je `{{ $ime }}` u radnom okviru Laravel. Ime i profesija su promenljive koje se koriste u kontroleru ovog primera. U njemu je napisana samo globalna funkcija `umetnost` koja je prikazana u nastavku.

```
public function umetnost() {
    return view('pages.umetnost')->with([
        'ime'=>'Pera',
        'profesija'=>'Pisac'
    ]);
}
```

U fajlu `web.php` treba dodati putanju ka fajlu. Taj deo koda prikazan je u sledećem redu.

```
Route::get('umetnost', 'MyController@umetnost');
```

To znači da je putanja ka stranici `http://127.0.0.1:8000/umetnost`, odnosno nakon porta 8000 treba dodati `umetnost`. Aplikacija će biti pokrenuta sa `http://127.0.0.1:8000/umetnost` i izgleda kao na slici 8.

# Pera Pisac

## Sloboda pisca je dar.

Dovoljno je nekoliko sekundi inspiracije. Dovoljno je nekoliko sekundi inspiracije. Dovoljno je nekoliko sekundi inspiracije. Dovoljno je nekoliko sekundi inspiracije. Dovoljno je nekoliko sekundi inspiracije.

### Inspiracija

Java i san

Home of Katrin

Slika 8: Stranica kreirana šablonom nasleđivanja stranica

Važno je napomenuti da se koristi i direktiva `@include` kojom se uključuju drugi fajlovi čiji je naziv `naziv.blade.php`. Primer fajla koji se ubacuje na stranicu direktivom `@include` je fajl za navigaciju stranice. Ako bi naziv tog fajla bio `navbar.blade.php`, fajl bi bio uključen kodom koji je prikazan u sledećem redu.

```
@include('navbar')
```

Koriste se još i direktive `@for`, `@if`, `@while`, `@foreach`, pri čemu se svaka mora završiti sa `@endfor`, `@endif`, `@endwhile` i `@endforeach` redom.

## 8 Redirekcija

Redirekcije su zahtevi za skok na neku drugu stranicu i instance su klase `Illuminate\Http\RedirectResponse`. Najjednostavnije korišćenje redirekcije dato je sa:

```
Route::get('redirekcija',function(){
    return redirect('umetnost');
});
```

gde je `redirect` funkcija koja kao argument prima rutu koja se piše posle porta 8000 i ovaj deo koda se kuca u fajlu `web.php`. Nakon pokretanja aplikacije sa `http://127.0.0.1:8000/redirekcija` u pregledaču, otvara se stranica koja je prikazana na slici 8 čiji je link `http://127.0.0.1:8000/umetnost`. Ako treba preusmeriti korisnika nazad na prethodnu stranicu, što se često koristi u situacijama kada je validacija podataka neuspela, može se koristiti kôd koji je prikazan u sledećem redu.

```
return redirect()->back();
```

Dakle, na metodu `redirect` može se dodati još metoda ili čak više njih. Može se u slučaju neuspele validacije, nakon korišćenja metode `back` poslati odgovarajuća poruka. Kôd koji ilustruje takav primer prikazan je u sledećem redu.



```
return redirect()->back()->with('error', 'Probajte ponovo');
```

U slučaju imenovanih ruta redirekcija bi bila

```
return redirect()->route('naziv_rute');
```

i ovaj deo koda se piše u kontroleru. U slučaju parametarskog rutiranja, ako bi ruta bila zadata sa:

```
Route::get('book/{id}', 'BooksController@show')->name('↪ book_view');
```

redirekcija bi bila zadata kodom u sledećem redu.

```
return redirect()->route('book_view', 1);
```

Imenovanom rutom se korisnik vraća na rutu koja posle porta 8000 ima zadato `book/id_vrednost`, ali zbog toga što je id deo parametarskog rutiranja, mora se kao drugi argument metode route u redirekciji poslati željeni id.

## 9 Konekcija sa bazom i migracije

Mnoge aplikacije rade sa bazama podataka. Laravel podržava mnoge sisteme za upravljanje bazama podataka uključujući: MySQL, SQLite, PostgreSQL i SQL. Za aplikaciju se može izabrati bilo koja, ali u ovom radu i za potrebe aplikacije biće korišćena baza MySQL. Među folderima u aplikaciji nalazi se i fajl `.env`. U folderu `config` i fajlu `database.php` linija koda

```
'default' => env('DB_CONNECTION', 'mysql')
```

znači da Laravel automatski ima konekciju sa bazom podataka MySQL. U fajlu `.env` treba promeniti vrednosti `DB_DATABASE`, `DB_USERNAME` i `DB_PASSWORD`, da bi se uspostavila konekcija sa bazom. Podešeno će biti na `test`, `root` i prazno polje redom. Sada se može pristupiti bazi podataka. Pre nego što se ovo uradi, potrebno je u MySQL phpMyAdmin napraviti bazu.

Migracije su vid kontrole baze podataka, omogućavajući timu da lako modifikuje bazu podataka. Laravel koristi migracijske fajlove kako bi programeri koji koriste istu bazu znali šta je menjano u bazi podataka. To se može znati tako što svaki kreirani migracijski fajl automatski dobija naziv po datumu kad je kreiran. Migracijski fajlovi se čuvaju u folderu `app\database\migrations`. Iz komandne linije, migracijski fajl se kreira komandom koja je prikazana u sledećem redu.

```
php artisan make:migration create_books_table --create=books
```

Automatski se generiše migracijski fajl u kojem je definisana tabela books. Kôd kreiranog migracijskog fajla prikazan je u nastavku.

```
<?php

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

class CreateBooksTable extends Migration
{
    /**
     * Run the migrations.
     * @return void
     */
    public function up()
    {
        Schema::create('books', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }

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

Svaki migracijski fajl sastoji se od metoda up i down. Metoda up služi za dodavanje tabela i kolona, dok metoda down služi da poništi sve što je kreirano metodom up. Nakon što se tabeli dodaju željene kolone, treba pokrenuti migraciju. Komanda kojom se pokreće migracija prikazana je u sledećem redu.

```
php artisan migrate
```

Ponovno definisanje migracije zadaje se komandom

```
php artisan migrate:rollback --step=2
```

gde se delom `--step=2` određuje koliko migracija će se ponovo definisati. U konkretnom primeru, ponovo će biti definisane poslednje dve migracije. Naredbom

```
php artisan migrate:reset
```

će se ponovo definisati sve migracije, a naredbom

```
php artisan migrate:refresh
```

će se ponovo definisati sve migracije i sprovedeće se naredba migrate.

## 10 Rad sa tabelama i kolonama

U prethodnom poglavlju je opisano kako se kreiraju migracijski fajlovi. Komandom iz komandnog prozora

```
php artisan make:migration create_nazivTabele_table --create=  
↪ nazivTabele
```

se kreira migracijski fajl, čiji je kôd automatski generisan i nalazi se u podfolderu `migrations` foldera `app`. U kodu tog fajla generisana je metoda `create`. Tabela se kreira koristeći metodu `create`. Metoda `create` kao prvi argument prima naziv tabele, a kao drugi argument objekat `Blueprint` koji se koristi za definisanje promenljive kojom će se unositi kolone koje će tabela sadržati. Svaki migracijski fajl podrazumevano ima kolonu `id` i podatak `timestamp` koji se sastoji od dve kolone. To su kolone `created_at` i `updated_at`. Podaci iz kolona `created_at` i `updated_at` su značajni jer se svakom izmenom beleži kada je izmena nastala. Osim navedenih podrazumevanih kolona, može se dodati koliko god kolona da je potrebno i taj deo koda se piše u metodi `create`. Deo koda koji treba dodati u metodi `create` prikazan je u sledećem redu.

```
$table->tip_podatka('naziv_kolone');
```

U konkretnom primeru, ako je naziv kolone `name`, a tip podatka `string`, kôd za dodavanje kolone `name` dat je u sledećem redu.

```
$table->string('name');
```

Nakon dodavanja svih željenih kolona može se pokrenuti migracija. Ako se zaboravi dodavanje kolone, ona se može naknadno dodati. Na primer, kolona opis, se može dodati na sledeći način. Najpre se komandom iz komandnog prozora pravi nova migracija sa:

```
php artisan make:migration add_opis_to_place
```

i ovom komandom je kreiran novi migracijski fajl. Njegov kôd je prikazan u nastavku.

```

<?php

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

class AddZanrToUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('place', function (Blueprint $table) {
            //
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('place', function (Blueprint $table) {
            //
        });
    }
}

```

U metodi up treba dodati,

```
$table->string('opis');
```

a u metodi down treba dodati

```
$table->dropColumn('opis');
```

i nakon unetih izmena se pokreće migracija sa `php artisan migrate`. Dešava se da se nakon osvežavanja baze ne vidi nikakva promena. Ako do toga dođe, potrebno je zaustaviti server ili ga restartovati pre dodavanja naredbe u komandnom prozoru. Zatim, pre pokretanja migracije treba ponovo pokrenuti server. Kako se kolona može dodati, tako se može i izbrisati. Prvo treba

zadati naredbu iz komandnog prozora koja je prikazana u sledećem redu.

```
php artisan make:migration remove_name_from_place
```

Zatim u metodi up kreiranog migracijskog fajla treba dodati kôd koji je prikazan u nastavku.

```
Schema::table('place', function (Blueprint $table) {  
    $table->dropColumn('name');  
});
```

U metodi down treba dodati sledeći kôd:

```
Schema::table('place', function (Blueprint $table) {  
    $table->string('name');  
});
```

i sada se metoda dropColumn piše u metodi up, jer je brisanje kolona ono što se očekuje od nove migracije.

Podaci u tabeli mogu biti tipa: integer, boolean, string, enum, date, time... Sintaksa za definisanje kolone tipa enum je prikazana u sledećem redu.

```
Schema::create('primer', function($table) {  
    $table->enum('who', ['Pera', 'Mika', 'Zika']);  
});
```

Može se definisati dužina stringa sa:

```
$table->string('name',60);
```

gde je name naziv kolone. Ako treba naglasiti da ne sme doći do ponavljanja iste vrednosti, onda se dodaje metoda unique kao što je prikazano u sledećem redu.

```
$table->string('name')->unique();
```

Ako treba neku drugu vrednost postaviti za primarni ključ, onda se to piše kao što je prikazano u nastavku.

```
$table->string('name');  
$table->primary('name');
```

Primarnih ključeva može biti i više, a deo koda koji definiše više primarnih ključeva prikazan je u narednom primeru.

```
Schema::create('example', function($table) {  
    $table->integer('id');  
    $table->string('username');  
    $table->string('email');
```

```
$table->primary(['id', 'username', 'email']);
});
```

Tabela i kolona se mogu preimenovati na sledeći način:

```
Schema::rename($nazivPrije, $nazivPoslije);//preimenovanje
↳ tabelle
Schema::table('primer', function($table) {
    $table->renameColumn('name', 'ime');//preimenovanje kolone
```

i sve navedeno se osim iz koda može uraditi direktno u bazi podataka. Više o tabelama se može pročitati u literaturi [1].

## 11 Objektno-relaciono preslikavanje

Objektno-relaciono preslikavanje (engl. Object-relational mapping - ORM) predstavlja tehniku prevođenja elemenata objektnog modela u elemente relacionog modela i obrnuto. Alat Eloquent ORM je moćan alat u radnom okviru Laravel za upravljanje bazom podataka. Tabele iz baze podataka se predstavljaju kao objekti, odnosno modeli, što omogućava lakše upravljanje i manipulaciju podacima. Svaki model predstavlja jednu tabelu u bazi podataka. Pomoću modela se komunicira sa tom tabelom. Model se stvara komandom koja je prikazana u sledećem redu.

```
php artisan make:model nazivModela
```

Ako sa modelom treba kreirati migracijski fajl za taj model, onda se model pravi komandom koja je prikazana u sledećem redu.

```
php artisan make:model nazivModela -m
```

Model se čuva u folderu `app`. Naziv modela i tabele su usklađeni. Model se piše u jednini, a naziv tabele se piše u množini. Može se zadati drugi naziv tabele ako se u modelu doda linija koda koja je prikazana u sledećem redu.

```
protected $table = 'neki_drugi_naziv';
```

Podrazumevano, primarni ključ tabele je id koji je tipa integer. Kao što se definisanjem u modelu, naziv tabele može postaviti na željeni, koji je drugačiji od naziva modela, tako se i primarni ključ može promeniti. Primarni ključ može biti kolona koja je tipa string, ali se onda svojstvo `incrementing` postavlja na `false`. U tom slučaju, u kôd modela treba dodati kôd koji sledi.

```
protected $primaryKey = 'naziv';
public $incrementing = false;
```

Može se desiti da korisnik unese neočekivani parametar putem zahteva i pritom se pojavljuje greška. Zbog toga se svojstvom `fillable` to rešava tako što se navedu kolone koje se mogu menjati od strane korisnika. To se radi tako što se u modelu doda kôd koji je prikazan u sledećem redu.

```
protected $fillable = ['naziv'];
```

Svojstvom `guarded` može se definisati koje atribute korisnik ne sme menjati. To se na isti način postiže kao i kod svojstva `fillable`, tj. u modelu se dodaje kôd koji je prikazan u sledećem redu.

```
protected $guarded = ['naziv'];
```

Dovoljno je podesiti jedno od ova dva svojstva. Ako su svi atributi izmenjivi od strane korisnika, onda bi se pisalo:

```
protected $guarded = [];
```

što znači da korisnik nema zabranjen pristup nijednom atributu. U narednom poglavlju će biti prikazano kako iz komandnog prozora upisati podatke u bazu i kako ih izbrisati iz baze.

## 11.1 Upravljanje podacima iz komandnog prozora

U radnom okviru Laravel postoji mogućnost za upisivanje u bazu i brisanje podataka iz baze iz komandnog prozora. Za upisivanje i brisanje podataka iz komandnog prozora se koristi interaktivna konzola Tinker za upravljanje bazom podataka. U komandnoj liniji, Tinker se pokreće komandom koja je prikazana u sledećem redu.

```
php artisan tinker
```

Za upis podataka u bazu piše se sledeći kôd:

```
$s=new App\Student;  
$s->first_name='Ana';  
$s->last_name='Jovanovic';  
$s->save();
```

ako je u pitanju tabela sa unosom studenata. Nakon svakog reda, treba pritisnuti enter. Ako treba izbrisati podatak iz baze, on se mora prvo dohvatiti po primarnom ključu, a onda se iskoristi metoda `delete`, kao što je prikazano u primeru koji sledi.

```
$s = App\Student::find(2);  
$s->delete();
```

Naravno, najjednostavnije je upisivati podatke direktno iz baze, brisati ih, menjati ih i ostalo. U daljem tekstu biće kreirana jednostavna aplikacija

koja će imati polja za unos imena i prezimena studenata i dugmiće za menjanje i brisanje podataka sa stranice u bazu. U kreiranju ove aplikacije biće iskorišćen kontroler resursa.

## 12 Upisivanje podataka u bazu, izmena podataka i brisanje podataka

Rad sa podacima iz baze biće prikazan u aplikaciji koja sledi. Treba napraviti bazu koja se zove studenti. Neophodno je promeniti vrednosti `DB_DATABASE`, `DB_USERNAME` i `DB_PASSWORD` u fajlu `.env` kao što je prikazano u nastavku.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=studenti
DB_USERNAME=root
DB_PASSWORD=
```

Zatim treba napraviti model `Student` komandom iz komandnog prozora koja je prikazana u sledećem redu.

```
php artisan make:model Student -m
```

Migracijske fajlove koji su generisani pravljenjem novog projekta treba izbrišati. Nakon pravljenja modela, automatski će biti kreiran migracijski fajl koji predstavlja tabelu `students`. Osim kolona `id` i `timestamps`, koje su automatski definisane, može se dodati još kolona. Aplikacija je zamišljena tako da se za svakog studenta unosi ime i prezime. Stoga, treba dodati kolone za ime i prezime u metodi `up` kao što je prikazano u sledećem redu.

```
$table->string('first_name');
$table->string('last_name');
```

Nakon što su dodate sve kolone unutar migracijskih fajlova, treba pokrenuti migraciju komandom koja je prikazana u sledećem redu.

```
php artisan migrate
```

Za izradu ove aplikacije biće napravljen resursni kontroler. Taj kontroler se pravi iz komandnog prozora sa:

```
php artisan make:controller StudentController --resource
```

gde je `StudentController` naziv kontrolera. Kreiran je kontroler koji ima funkcije: `index`, `create`, `store`, `show`, `edit`, `update` i `destroy`. Ove metode čine takozvani `CRUD` (`CreateReadUpdateDestroy`) sistem. Da bi se napravile rute za sve `CRUD` funkcije, treba dodati



```
Route::resource('student', 'StudentController');
```

u fajlu `web.php` koji se nalazi u folderu `routes`. Rute svih funkcija i imena ruta mogu se izlistati komandom iz komandnog prozora koja je prikazana u sledećem redu.

```
php artisan route:list
```

GET HEAD	student	student.index	App\Http\Controllers\StudentController@index
POST	student	student.store	App\Http\Controllers\StudentController@store
GET HEAD	student/create	student.create	App\Http\Controllers\StudentController@create
GET HEAD	student/{student}	student.show	App\Http\Controllers\StudentController@show
PUT PATCH	student/{student}	student.update	App\Http\Controllers\StudentController@update
DELETE	student/{student}	student.destroy	App\Http\Controllers\StudentController@destroy
GET HEAD	student/{student}/edit	student.edit	App\Http\Controllers\StudentController@edit

Slika 9: Rute funkcija resursnog kontrolera

Na slici 9 prikazane su sve rute aplikacije.

Aplikacija treba da se sastoji od veb stranica sa kojih je moguće upisati studente u bazu, unositi izmene ili brisati studente iz baze. Zbog toga će biti iskorišćen šablon Blade u radnom okviru Laravel. Treba napraviti stranicu koju će sve druge stranice naslediti. Stranica koju druge stranice nasleđuju može se nazvati `master.blade.php`. Njen kôd je dat sa:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
    ↪ scale=1">
  <title>Laravel CRUD</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery
    ↪ /3.2.1/jquery.min.js"></script>
  <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-
    ↪ bootstrap/4.0.0-alpha/css/bootstrap.css" rel="
    ↪ stylesheet">
</head>
<body>

<div class="container">
  @yield('content')
</div>
```

```
</body>
</html>
```

gde je sa:

```
<link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-
  ↳ bootstrap/4.0.0-alpha/css/bootstrap.css" rel="stylesheet
  ↳ ">
```

integrisana biblioteka za stilizovanje veb stranica u radnom okruženju Bootstrap 4. Elektronske lekcije iz radnog okruženja Bootstrap 4 dostupne su na eŠkoli veba na linku [http://edusoft.matf.bg.ac.rs/eskola\\_veba/#/course-details/bs4](http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/bs4). U svaku stranicu `blade.php` biće uključen fajl `master.blade.php`. Sledeće što treba napraviti je stranica `create.blade.php` čiji je kôd prikazan u nastavku.

```
@extends('master')

@section('content')
<div class="row">
  <div class="col-md-12">
    <br />
    <h3 align="center">Add Data</h3>
    <br />

    <form method="post" action="{{url('student')}}">
      {{csrf_field()}}
      <div class="form-group">
        <input type="text" name="first_name" class="form-control"
          ↳ placeholder="Enter First Name" />
      </div>
      <div class="form-group">
        <input type="text" name="last_name" class="form-control"
          ↳ placeholder="Enter Last Name" />
      </div>
      <div class="form-group">
        <input type="submit" class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
@endsection
```

Akcija koja je zadata u formi znači da klik na dugme vodi na stranicu koja se nalazi na linku <http://127.0.0.1:8000/student>.

Kada se šalje zahtev metodom POST, može doći do falsifikovanja korisnikovog zahteva. Laravel štiti aplikaciju od falsifikovanja zahteva tako što svaki put kada se koristi forma i metoda POST treba dodati polje `csrf_field()`. Ovo polje automatski generiše token `csrf` za svaku aktivnu sesiju korisnika koji upravlja aplikacijom. Kada se pošalje zahtev metodom POST, međusoftver za zaštitu proverava da li se token iz zahteva poklapa sa tokenom iz sesije koji je kreirao Laravel. Objašnjenje je preuzeto iz literature [10]. U kontroleru u metodi `create` treba dodati liniju koda koja je prikazana u sledećem redu.

```
return view('student.create');
```

Podaci koji su uneti se metodom `store` šalju u bazu klikom na dugme `submit`. Prvo se klikom na dugme `submit` šalje zahtev kontroleru koji obrađuje zahtev u metodi `store`. U metodi `store` treba napisati kôd koji je prikazan u nastavku.

```
$this->validate($request, [
    'first_name' => 'required',
    'last_name' => 'required'
]);
$student = new Student([
    'first_name' => $request->get('first_name'),
    'last_name' => $request->get('last_name')
]);
$student->save();
return return redirect()->route('student.create')->with('
    ↪ success', 'Data Added');
```

Metoda `store` kao argument prima zahtev, a metodom `validate` se zahteva unos podataka. Bitno je u modelu naglasiti da su podaci izmenjivi od strane korisnika. To se podešava svojstvom `fillable`. Dakle, u modelu `Student` treba dodati kôd koji je prikazan u sledećem redu.

```
protected $fillable = ['first_name', 'last_name'];
```

U slučaju da su uneti ili nisu uneti podaci, mogu se ispisati odgovarajuće poruke kodom koji je prikazan u nastavku. Taj deo koda se dodaje u fajlu `create.blade.php`.

```
@if(count($errors) > 0)
<div class="alert alert-danger">
  <ul>
    @foreach($errors->all() as $error)
      <li>{{$error}}</li>
    @endforeach
  </ul>
```

```
</div>
@endif
@if(\Session::has('success'))
<div class="alert alert-success">
  <p>{{ \Session::get('success') }}</p>
</div>
@endif
```

U metodi store je metodom with poslata poruka u slučaju uspešno unetih podataka. U kodu za proveru sa:

```
@if(\Session::has('success'))
```

se proverava da li u sesiji postoji ta poruka i ako postoji vraća se vrednost true, pa se sa:

```
<p>{{ \Session::get('success') }}</p>
```

uzima poruka iz sesije. Nakon što je kreirana prva stranica, ona se može otvoriti na linku <http://127.0.0.1:8000/student/create>. U slučaju da se klikne dugme pre upisa podataka, ispisuju se odgovarajuće poruke kao na slici 10.



The screenshot shows a web form with a light red background for error messages. The messages are:

- The first name field is required.
- The last name field is required.

Below the messages are two input fields:

Slika 10: Poruke u slučaju da podaci nisu uneti

Klikom na dugme submit, podaci se upisuju u bazu.

## 12.1 Ispis podataka iz baze

Nakon upisivanja podataka u bazu, treba prikazati ispisvanje podataka iz baze. Najpre, treba dodati sledeći kôd u metodi index u kontroleru.

```
$students = Student::all()->toArray();
return view('student.index', compact('students'));
```

Funkcija `compact` je funkcija jezika PHP, a ne radnog okvira Laravel. Ova funkcija kreira niz koji sadrži promenljive i njihove vrednosti. Nazivi promenljivih su ključevi, a vrednosti su konkretne vrednosti koje odgovaraju zadatom ključu. U radnom okviru Laravel se uglavnom koristi da bi se vrednosti poslale pogledu. `Compact` je pomoćna funkcija koja kao argument prima ključeve promenljivih, a odgovarajuće vrednosti biće dostupne pogledu. Potrebno je napraviti fajl `index.blade.php`, koji, koristeći šablon Blade, nasleđuje fajl `master.blade.php`. Kôd fajla `index.blade.php` je prikazan u nastavku.

```
@extends('master')

@section('content')
<div class="row">
  <div class="col-md-12">
    <br />
    <h3 align="center">Student Data</h3>
    <br />
    <div align="right">
      <a href="{{route('student.create')}}" class="btn btn-
        ↳ primary">Add</a>
    <br />
    <br />
  </div>
  <table class="table table-bordered table-striped">
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Edit</th>
      <th>Delete</th>
    </tr>
    @foreach($students as $row)
    <tr>
      <td>{{ $row['first_name'] }}</td>
      <td>{{ $row['last_name'] }}</td>
      <td><a href="{{action('StudentController@edit', $row['id'])
        ↳ }}" class="btn btn-warning">Edit</a></td>
    </tr>
    @endforeach
  </table>
</div>
```

```
</div>
@endsection
```

Kada se pokrene aplikacija sa `http://127.0.0.1:8000/student`, podaci su prikazani kao na slici 11.



First Name	Last Name	Edit	Delete
Katarina	Andrejević		
Ana	Jovanovic		
Petar	Petrović		

Slika 11: Prikaz podataka iz baze

U kolonama edit i delete treba napraviti dugmiće za menjanje i brisanje podataka, jer je to sledeće što treba omogućiti u aplikaciji. Dodato je dugme add, da bi se nakon ispisivanja podataka ponovo moglo otići na stranicu za unos podataka. Ideja je da se nakon klika na dugme submit ode na stranicu na kojoj su podaci iz baze predstavljeni u tabeli, nakon što su uneti u bazu podataka. Putanja tog dugmeta se nalazi u funkciji store kontrolera i treba je zameniti novom putanjom koja je prikazana u sledećem redu.

```
return redirect()->route('student.index')->with('success', '
    ↪ Data Added');
```

Nako što je omogućen upis i ispis podataka, treba omogućiti izmenu i brisanje podataka.

## 12.2 Izmena podataka u bazi

Za izmenu podataka, odgovorne su funkcije edit i update. To su funkcije kontrolera resursa. Funkcija edit je zadužena za promenu podataka na stranici koju korisnik vidi, a funkcija update je zadužena za izmenu podataka u bazi. Treba napraviti dugmiće za izmenu podataka. Taj deo koda treba dodati u fajl `index.blade.php`. Dugmići će biti linkovi koji vode na stranicu za izmene preko metode edit u kontroleru, pa treba dodati kôd koji je prikazan u sledećem redu.

```
<td><a href="{action('StudentController@edit', $row['id'])}"
    ↪ class="btn btn-warning">Edit</a></td>
```

U metodi edit u kontroleru treba dodati kôd koji je prikazan u nastavku.

```
$student = Student::find($id);
return view('student.edit', compact('student', 'id'));
```

Izmena u bazi izvršava se po primarnom ključu podatka. Zbog toga je neophodno proslediti id kao argument funkcije. Potrebno je napraviti fajl `edit.blade.php`, koji koristeći šablon Blade, nasleđuje osnovni fajl koji se može nazvati `master.blade.php`. Kôd fajla `edit.blade.php` je prikazan u nastavku.

```
@extends('master')

@section('content')

<div class="row">
  <div class="col-md-12">
    <br />
    <h3>Edit Record</h3>
    <br />
    <form method="post" action="{{action('
      ↳ StudentController@update', $id)}}">
      {{csrf_field()}}
      <input type="hidden" name="_method" value="PATCH" />
      <div class="form-group">
        <input type="text" name="first_name" class="form-control"
          ↳ value="{{ $student->first_name }}" placeholder="Enter
          ↳ First Name" />
      </div>
      <div class="form-group">
        <input type="text" name="last_name" class="form-control"
          ↳ value="{{ $student->last_name }}" placeholder="Enter
          ↳ Last Name" />
      </div>
      <div class="form-group">
        <input type="submit" class="btn btn-primary" value="Edit"
          ↳ />
      </div>
    </form>
  </div>
</div>

@endsection
```

Sa `php artisan route:list` može se videti da je metoda za pozivanje funkcije `update` `PATCH`. Međutim, forme ne podržavaju metode `PUT`, `PATCH` i `DELETE` koje su pozvane iz forme. Ovo se može izbeći dodavanjem

```
<input type="hidden" name="_method" value="PATCH" />
```

u formu. Vrednost poslata poljem `_method` će se koristiti kao metoda zahteva HTTP. Funkcija `update` kao argumente prima zahtev i `id`, iz zahteva uzima prosleđene podatke, pronalazi podatak u bazi po argumentu `id` koji je prosleđen i menja podatke u bazi. Kôd funkcije `update` je prikazan u nastavku.

```
$this->validate($request, [
    'first_name' => 'required',
    'last_name' => 'required'
]);
$student = Student::find($id);
$student->first_name = $request->get('first_name');
$student->last_name = $request->get('last_name');
$student->save();
return redirect()->route('student.index')->with('success', '
    ↪ Data Updated');
```

Svojevremeno `required` naglašava se da je unos podataka obavezan. Ukoliko nije uneta izmena podataka, a klikne se na dugme `edit`, treba obezbediti odgovarajuće poruke upozorenja čiji je kôd prikazan u nastavku. Taj deo koda se dodaje u fajlu `edit.blade.php` pre forme.

```
@if(count($errors) > 0)

<div class="alert alert-danger">
    <ul>
        @foreach($errors->all() as $error)
            <li>{{$error}}</li>
        @endforeach
    </ul>
@endif
```

### 12.3 Brisanje podataka iz baze

U fajlu `index.blade.php` gde su u tabeli prikazani podaci iz baze treba dodati dugmiće za brisanje. Kôd za brisanje dugmića prikazan je u nastavku.

```
<td><form method="post" class="delete_form" action="{action('
    ↪ StudentController@destroy', $row['id'])}">
    {{csrf_field()}}
    <input type="hidden" name="_method" value="DELETE" />
    <button type="submit" class="btn btn-danger">Delete</
        ↪ button>
</form></td>
```



Ovo dugme poziva funkciju `destroy` koja kao argument prima `id`, zatim pronalazi podatak u bazi po `id` i briše ga. Kao i funkcija `update`, tako se i funkcija `destroy` poziva metodom koja nije `POST` ili `GET`. Funkcija `destroy` se poziva metodom `DELETE`. Kako forma ne vidi nijednu drugu metodu osim `POST` i `GET`, ovde je opet neophodno skrivenim poljem proslediti metodu `DELETE` kao vrednost polja `_method`. Da bi se izbrisao podatak iz baze, najpre ga treba pronaći po primarnom ključu, a zatim se može primeniti metoda `destroy`. U funkciji `destroy` u kontroleru treba napisati kôd koji je prikazan u nastavku.

```
$student = Student::find($id);
    $student->delete();
return redirect()->route('student.index')->with('success', '
    ↪ Data Deleted');
```

Funkcija `destroy`, kao i funkcija `update` metodom `with` šalju poruke o izvršenoj izmeni u bazi, bilo da je to izmena podataka ili brisanje podataka. Poruka se nalazi u sesiji. Da bi se poruka ispisala, treba u fajlu `index.blade.php` fajlu dodati deo koda koji proverava da li ima poruke u sesiji. Kôd je prikazan u nastavku.

```
@if($message = Session::get('success'))
    <div class="alert alert-success">
        <p>{{ $message }}</p>
    </div>
@endif
```

Zaključno sa ovom lekcijom je kompletiran rad sa podacima u bazi. Primer za aplikaciju je preuzet iz literature [11].

## 13 Autentifikacija

U ovom poglavlju biće prikazano kako se može napraviti stranica za logovanje. U komandnom prozoru biće napravljen nov projekat koji se može nazvati `login`. Prvo se komandom `cd` treba pozicionirati u folder `eds-www`. Komanda za pravljenje novog projekta prikazana je u sledećem redu.

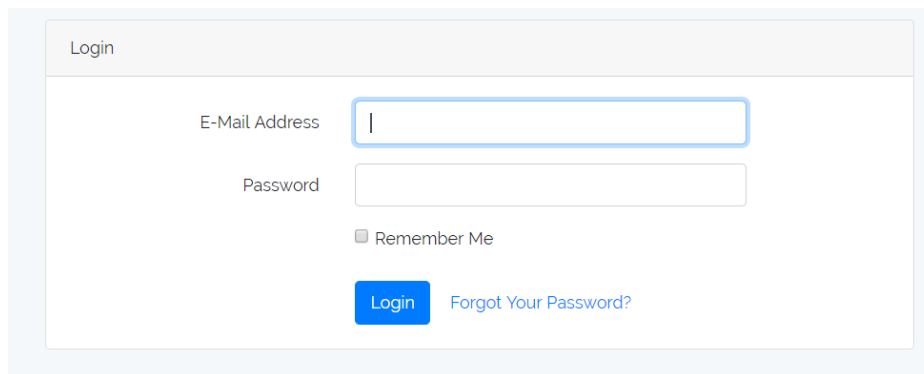
```
laravel new login
```

Nakon kreiranja novog projekta se komandom `cd login` treba pozicionirati u folder `login`. Zatim treba napraviti bazu i u fajlu `.env` se treba konektovati na bazu. Komandom iz komandnog prozora se sa:

```
php artisan make:auth
```

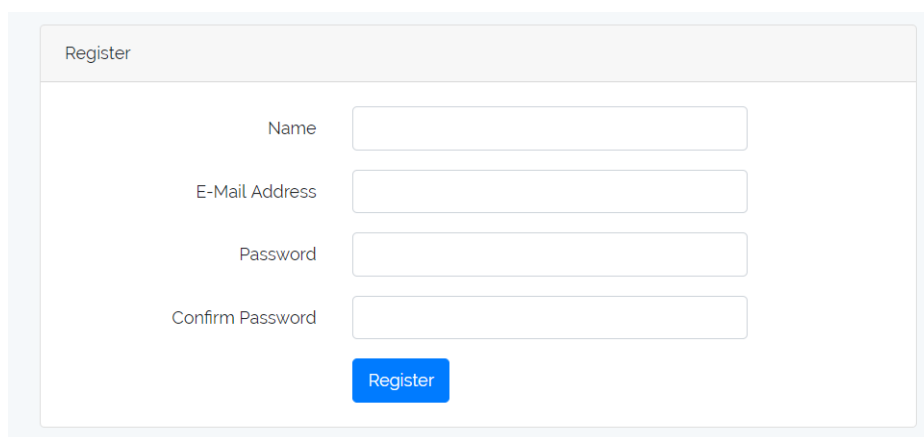
pravi sistem `login`. Dobija se poruka da je autentifikacija uspešno generisana. Kada se pokrene aplikacija i ode na stranicu `http://127.0.0.1:8000/` u

gornjem desnom uglu pojavili su se linkovi login i register. Klikom na login otvara se stranica kao na slici 12.



Slika 12: Stranica za logovanje

Klikom na register otvara se stranica kao na slici 13.



Slika 13: Stranica za registraciju

Stranice za logovanje i registraciju samo vizuelno funkcionišu, u smislu da se klikom na dugme login ili na dugme register ništa ne dešava, jer im nije dodeljena nikakva uloga. U folderu `database\migrations` se nalaze dva migracijska fajla koja imaju svoju ulogu u autentifikaciji. U migracijskom fajlu `users`, tabela `users` ima kolonu `name` kojoj treba zadati dužinu na sledeći način. Treba otići u folder `app` i u okviru njega se nalazi podfolder `providers`. U fajlu `AppServiceProvider.php`, u metodi `boot` treba dodati klasu `Schema`, ali prvo treba navesti gde se ona nalazi. Pre koda koji počinje sa `class` treba dodati liniju koda koja je prikazana u sledećem redu.

```
use Illuminate\Support\Facades\Schema;
```

Nakon toga u globalnoj funkciji `boot` treba dodati kôd koji je prikazan u sledećem redu.

```
schema::defaultStringLength(191);
```

Na kraju sa:

```
php artisan migrate
```

treba pokrenuti migracije.

Sve je spremno za registraciju prvog korisnika. Laravel već ima zaštitu za sva polja u sistemu login, u smislu da ima poruke upozorenja ako se unese nešto što nije mejl, ako se šifre ne poklapaju ili ako se unese šifra koja ima manje od 6 karaktera. Primer validacije prikazan je na slici 14.

Slika 14: Validacija

Kada se korisnik registruje, klikom na register, korisnik je upisan u bazi u tabeli users, a otvara se stranica kao na slici 15 sa odgovarajućom porukom.

Slika 15: Uspešna registracija korisnika

Sa sistemom za autentifikaciju generisan je i fajl `home.blade.php` koji se nalazi u podfolderu `views` foldera `resources`. Ideja je da se na stranici ispiše odgovarajuća poruka u slučaju da je korisnik logovan. Ako korisnik nije logovan treba da se prikaže stranica za registraciju. Prvo što treba uraditi je da se u fajlu `web.php` promeni ruta i da se u kodu:

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

`welcome` zameni sa `home`. Nakon promene, kada se otvori stranica na linku `http://127.0.0.1:8000`, pojavljuje se poruka da je korisnik logovan, iako

nije. Ako korisnik nije registrovan, treba ga odvesti na stranicu za registraciju. Deo koda koji to reguliše dodaje se u fajlu `home.blade.php`. Ispod naredbe `if` treba dodati kôd koji je prikazan u nastavku.

```
@guest
    @include('auth.login')
@else
    You are logged in!
@endguest
```

Sa `@guest` je definisana kontrola pristupa i može da se tumači kao `if`. Koristeći `@guest` proverava se da li je neko gost ili je već logovan korisnik. Ako je već logovan ispisuje se poruka “You are logged in!”, a ako nije onda se sa `@include`, što znači uključivanje stranice, otvara stranica `login`. Fajl `login.blade.php` se nalazi u folderu `auth` i generisan je kada je napravljen sistem za logovanje. Zbog toga se prvo piše naziv foldera, pa naziv fajla, tj. `auth.login`.

## 14 Ubacivanje fajla na sajt

U ovoj sekciji će biti prikazano kako se ubacuje slika na stranicu. Za potrebe te jednostavne aplikacije biće kreiran nov projekat komandom iz komandnog prozora koja je prikazana u sledećem redu.

```
laravel new upload
```

Sve funkcije aplikacije biće u kontroleru, pa ga treba kreirati. Kontroler se kreira komandom iz komandnog prozora sa:

```
php artisan make:controller UploadController
```

gde je `UploadController` izabran naziv kontrolera. Za pogled će biti iskorišćen šablon `Blade`, a to znači da se pravi fajl, koji će ostali fajlovi naslediti. Neka se fajl zove `layout.blade.php` i nalazi se u podfolderu `views` foldera `resources`. Kôd tog fajla je prikazan u nastavku.

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-
    ↪ scale=1, shrink-to-fit=no">
<meta name="description" content="">
<meta name="author" content="">
<title>Upload image</title>
<!-- Bootstrap core CSS -->
```

```

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
  ↳ bootstrap/4.0.0-beta.2/css/bootstrap.min.css">
</head>
<body>
  <main role="main" class="container">
    <div class="starter-template">
      <div class="row">
        <div class="col-sm-8">
          @yield('content')
        </div>
      </div>
    </div>
  </main>
</body>
</html>

```

Kôd fajla `uploadfile.blade.php` koji treba napisati, a koji nasleđuje fajl `layout.blade.php` je prikazan u nastavku.

```

@extends('layout')

@section('content')
  <div class="container">
    <div class="row justify-content-center">
      <div class="card">
        <div class="card-header">Upload File Example
        </div>
        <div class="card-body">
          <form method="post" action="{{url('/uploadfile')}}"
            ↳ enctype="multipart/form-data">
            {{csrf_field() }}
            <div class="form-group">
              <table class="table">
                <tr>
                  <td width="40%" align="right"><label>Select File for
                    ↳ Upload</label></td>
                  <td width="30"><input type="file" name="select_file"
                    ↳ /></td>
                  <td width="30%" align="left"><input type="submit"
                    ↳ name="upload" class="btn btn-primary" value="
                    ↳ Upload"></td>
                </tr>
                <tr>
                  <td width="40%" align="right"></td>
                  <td width="30"><span class="text-muted">jpg, png, gif

```



Datoteka se učitava klikom na dugme upload. Kako je datoteka poslata metodom POST, putanja koju treba napisati u fajlu `web.php` prikazana je u sledećem redu.

```
Route::post('uploadfile', 'UploadController@uploadFilePost');
```

Nakon zahteva `http://127.0.0.1:8000/uploadfile`, poziva se funkcija `uploadFilePost` iz kontrolera čiji je kôd prikazan u nastavku.

```
public function uploadFilePost(Request $request) {
    $this->validate($request, [
        'select_file' => 'required|image|mimes:jpeg,png,gif|max
        ↪ :2048'
    ]);

    $image = $request->file('select_file');

    $new_name = rand() . '.' . $image->
        ↪ getClientOriginalExtension();

    $image->move(public_path('images'), $new_name);
    return back()->with('success', 'Image Uploaded Successfully
    ↪ ')->with('path', $new_name);
}
```

Objašnjenje prethodnog koda koje sledi je preuzeto iz literature [4]. Prvo se metodom `validate` zahteva da fajl mora biti izabran, a zatim da je fajl tipa `image`. Sa `mimes` se određuju dozvoljene ekstenzije slike, a sledeća provera je veličina fajla, koji ne može biti veći od 2 MB. Deo koda

```
$image = $request->file('select_file');
```

znači da se iz zahteva uzima selektovani fajl, a zatim se sa:

```
$new_name = rand() . '.' . $image->getClientOriginalExtension
    ↪ ();
```

pravi novo ime slike, uz uzimanje ekstenzije iz zahteva. Na kraju se sa:

```
$image->move(public_path('images'), $new_name);
return back()->with('success', 'Image Uploaded Successfully')
    ↪ ->with('path', $new_name);
```

slika skladišti u podfolderu `public` foldera `images`. Vraća se poruka o uspešnom slanju slike u folder. Poruka je sačuvana u sesiji. Treba dodati poruke o eventualnim greškama koje nastaju ukoliko se ne izabere slika, a klikne se na dugme `upload` ili ako se ne izabere odgovarajuća slika, bilo po ekstenziji ili veličini. Taj deo koda treba dodati u fajlu `upload.blade.php` pre forme i prikazan je u nastavku. Etiketom `img` obezbeđen je prikaz slike.

```

@if ($message = Session::get('success'))
  <div class="alert alert-success alert-block">
    <button type="button" class="close" data-dismiss="alert"></
      ↪ button>
    <strong>{{ $message }}</strong>
  </div>
  
@endif
@if (count($errors) > 0)
  <div class="alert alert-danger">
    <strong>Whoops!</strong> There were some problems with your
      ↪ input.<br><br>
    <ul>
      @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
      @endforeach
    </ul>
  </div>
@endif

```

Ukoliko se, nakon što se odabere slika i klikne na dugme upload, pojavi greška *Unable to guess the mime type as no guessers are available (Did you enable the php\_fileinfo extension?)*, potrebno je da se u fajlu `php.ini` skine komentar sa `enable=php_fileinfo.dll`. Ukoliko se koristi EasyPHP taj fajl se nalazi u podfolderu `php` foldera `eds-binaries`. Nakon toga, može se pojaviti ista greška. U tom slučaju je potrebno ponovo pokrenuti server. Kad god se nešto promeni u podešavanju za PHP, potrebno je restartovati veb server. Kada se ubaci slika na stranicu, stranica izgleda kao na slici 17.



Image Uploaded Successfully



Select File for Upload

Одабери датотеку

Није одабрано

Upload

jpg, png, gif

Slika 17: Stranica sa ubačenom slikom

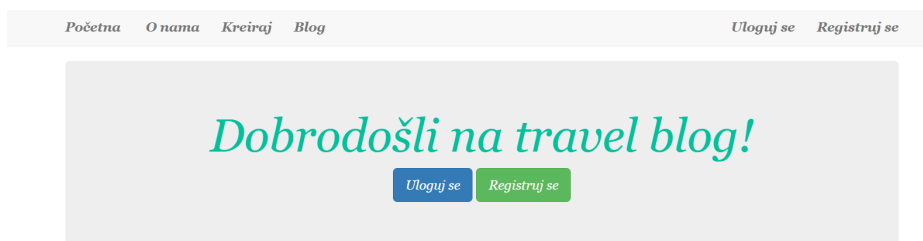
## 15 Kreiranje aplikacije u radnom okviru Laravel

U ovom poglavlju biće prikazan opis aplikacije koja je kreirana u radnom okviru Laravel. Aplikacija predstavlja blog o razmeni iskustava sa putovanja. Aplikacija je predstavljena kao sajt koji ima navigaciju i sastoji se od nekoliko stranica, stranice za kreiranje, za izmenu, za brisanje podataka, kao i stranice za pregled svih unetih putovanja. Blog o putovanjima mogu čitati svi, tj. svi imaju pristup objavama, a samo registrovani korisnici mogu da kače postove sa opisima putovanja. U opisu se može navesti bilo šta što bi koristilo nekome kao informacija o mestu koje želi da poseti, a pre toga želi da se informiše o tom mestu. Korisnik može okačiti i slike sa putovanja. Registrovan korisnik osim što može da kači objave, može i da unosi izmene u svojim objavama kao i da ih briše. Registrovanih korisnika može biti više, pa je potrebno pristup izmeni ili brisanju objava ograničiti kontrolom pristupa. To znači da svaki korisnik može unositi izmene i brisati samo svoje objave, tj. registrovani korisnici ne mogu da menjaju ili brišu tuđe objave. Deo koda koji to reguliše je if naredba koja je prikazana u sledećem redu.

```
@if(Auth::user()->id==$place->user_id)
```

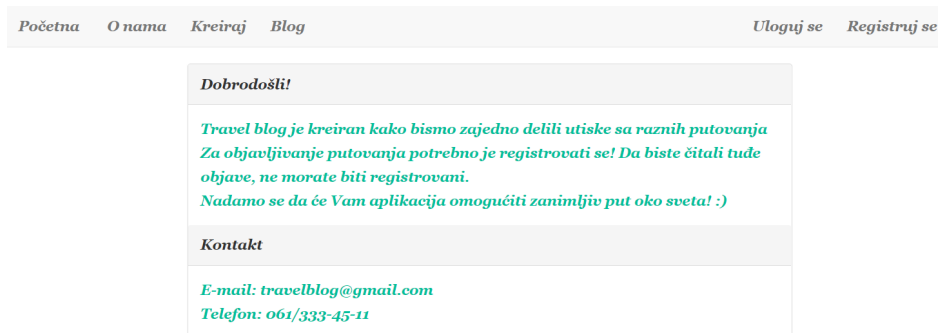
Da bi korisnik imao pregled svojih putovanja, omogućen je pregled putovanja za svakog korisnika i zadata su u tabeli. Uz svako putovanje prikazani su dugmići izmeni i obriši, ukoliko korisnik želi odmah da pristupi izmeni putovanja ili da ga obriše.

Naslovna stranica data je na slici 18.



Slika 18: Naslovna strana

Klikom na početnu stranicu otvara se naslovna strana kao na slici 18. Klikom na link o nama otvara se strana kao na slici 19.



Slika 19: Strana o nama

Klikom na link kreiraj otvara se stranica za kreiranje novog putovanja, ali ukoliko korisnik nije registrovan ili nije ulogovan, link vodi na stranicu za logovanje. Kako je potrebno uneti opis putovanja, polje za unos ne može biti input polje tipa text, već bi najviše odgovarao editor. Uputstvo za instalaciju editora se nalazi na linku <https://github.com/UniSharp/laravel-ckeditor>. Editor se instalira tako što se u liniji komandnog prozora napiše naredba koja je prikazana u sledećem redu.

```
composer require unisharp/laravel-ckeditor
```

Kada se instalacija završi, potrebno je u fajlu `app.php`, koji se nalazi u folderu `config`, dodati liniju koda koja je prikazana u sledećem redu. Ta linija koda se dodaje u sekciji `Application Service Providers` fajla `app.php`.

```
Unisharp\Ckeditor\ServiceProviders::class,
```

Zatim je potrebno u liniji komandnog prozora napisati liniju koda koja je prikazana u sledećem redu.

```
php artisan vendor:publish --tag=ckeditor
```

Poslednje što treba uraditi je dodavanje koda u fajl iz foldera `view` koji će ostali fajlovi nasleđivati. Kôd koji treba dodati je:

```
<script src="/vendor/unisharp/laravel-ckeditor/ckeditor.js"></
  ↪ script>
<script>
  CKEDITOR.replace( 'product-body' );
</script>
```

gde je `product-body` vrednost atributa `id` polja `textarea` kojim se uključuje editor. Izgled fajla `create.blade.php` koji se otvara klikom na link kreiraj prikazan je na slici 20.

Početna O nama Kreiraj Blog admin -

Novo putovanje

Mesto

Mesto koje ste posetili

Opis

Одабери датотеку | Није одабрано

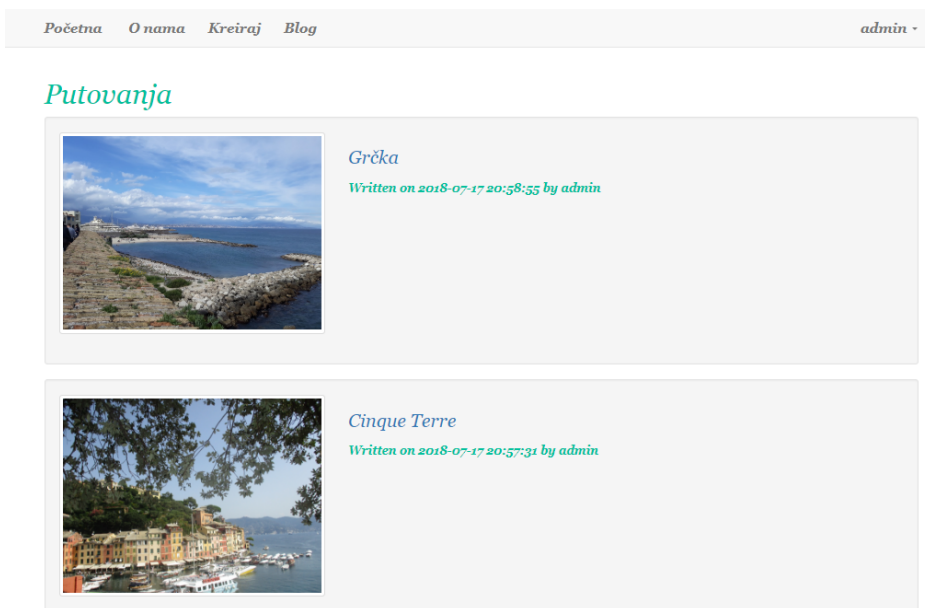
Избор датотека | Није одабрано

Kreiraj

Ac

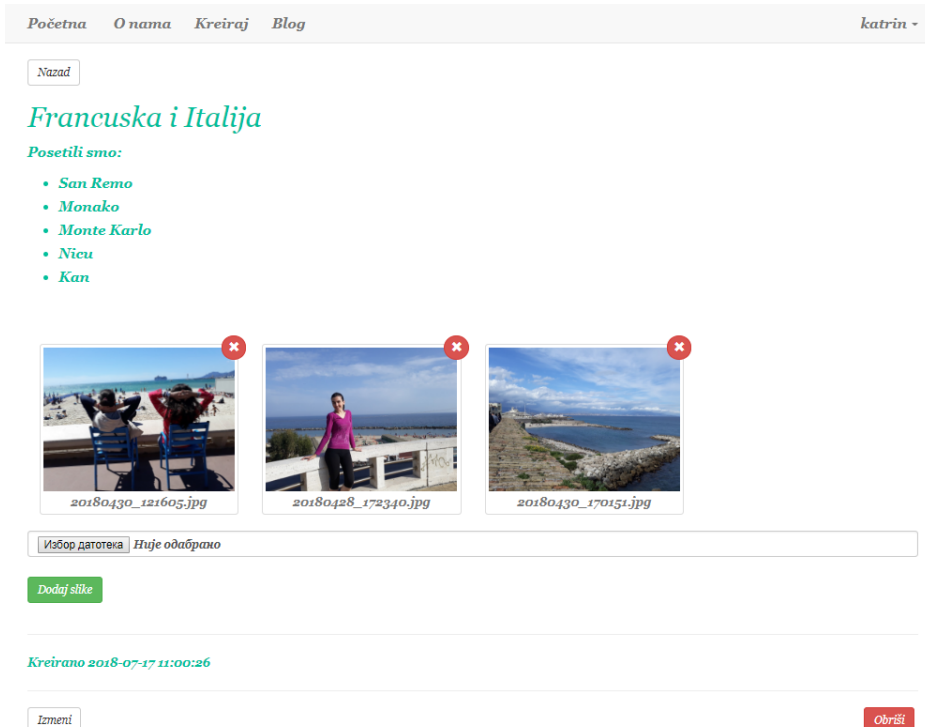
Slika 20: Izgled strane kreiraj

Objavljenim putovanjima mogu da pristupe svi, bez obzira na to da li su registrovani. Klikom na link `blog` otvara se stranica koja prikazuje sva do sada uneta putovanja. Svaki naslov mesta je link i vodi na stranicu koja otvara određeno putovanje. Ukoliko je trenutno registrovani korisnik kreirao putovanje na koje je kliknuto, onda on ima pristup izmenama, može izbrisati putovanje ili direktno na stranici brisati ili dodavati slike. Izgled stranice koja se otvara klikom na link `blog` dat je na slici 21.



Slika 21: Prikaz unetih putovanja

Izgled stranice sa tačno određenim putovanjem ukoliko ga je trenutno registrovani korisnik kreirao dat je na slici 22.

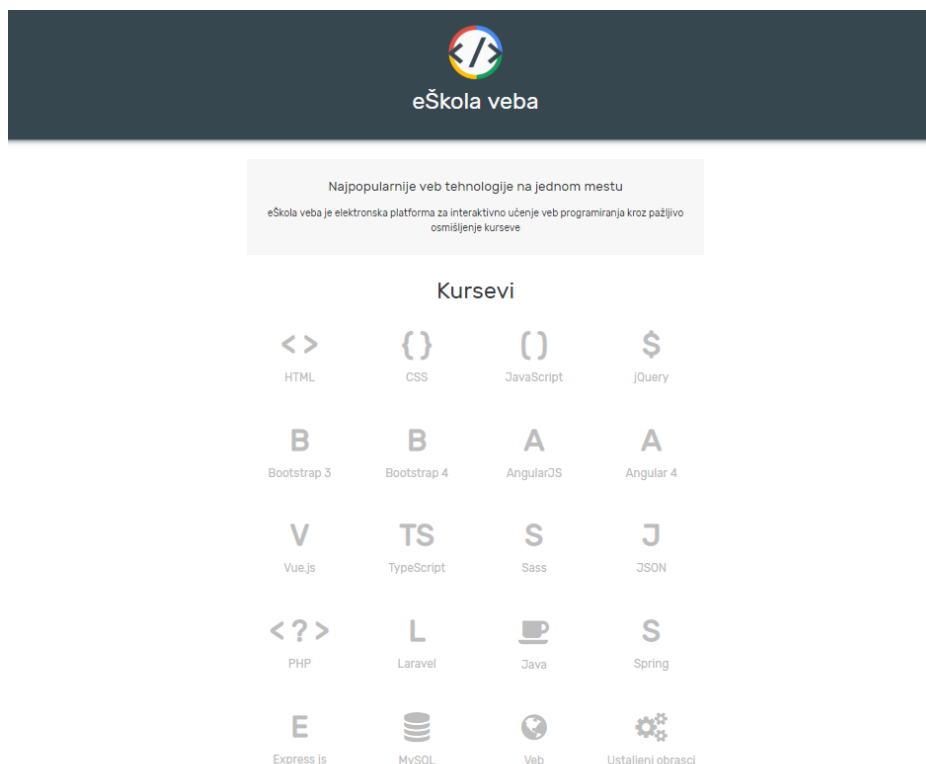


Slika 22: Putovanje

Kôd za pravljenje galerije kao na slici 22 preuzet je iz literature [7]. Za navigaciju aplikacije korišćeno je radno okruženje Bootstrap iz literature [9].

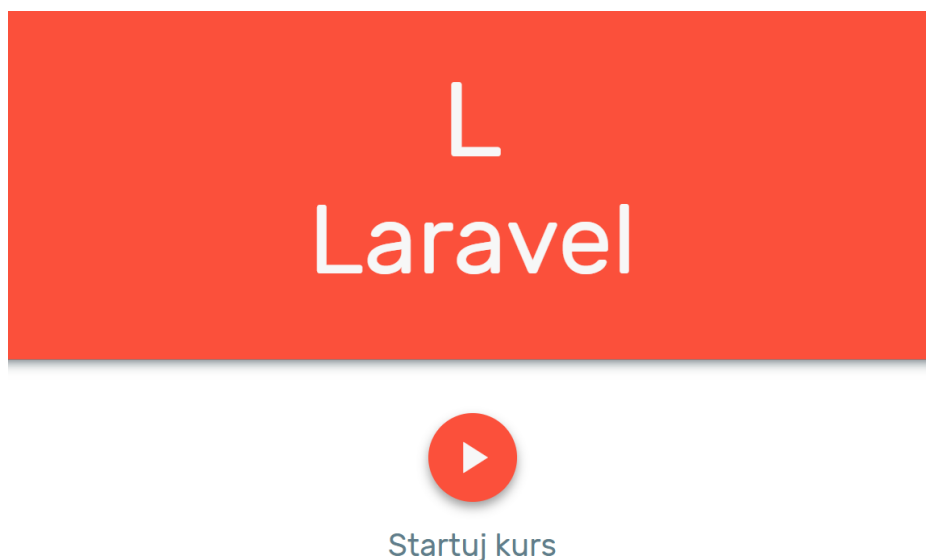
## 16 Elektronske lekcije

Elektronske lekcije o radnom okviru Laravel su dostupne na platformi eŠkola veba. Lekcije se nalaze na linku [http://edusoft.matf.bg.ac.rs/eskola\\_veba/#/course-details/lara](http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/lara) i namenjene su svima koji žele da unaprede svoje znanje u oblasti veb tehnologija. Sve lekcije su besplatne i dostupne na srpskom jeziku. Na slici 23 prikazan je izgled platforme.



Slika 23: Dostupni kursevi na platformi eŠkola veba

Na platformi se nalaze kursevi različitih veb tehnologija koje prate sadržajne elektronske lekcije sa mnoštvom primera. Jedan od kurseva na platformi je kurs o radnom okviru Laravel koji je kreiran za potrebe ovog rada i izborom ovog kursa otvara se početna strana kao na slici 24.



Slika 24: Naslovna strana izabranog kursa

Elektronske lekcije o radnom okviru Laravel bogate su primerima koje imaju za cilj da praktično približe sve koncepte koji su u njima prikazani. Sve lekcije su bogate slikama, naročito lekcija koja opisuje instalaciju radnog okvira Laravel. Ovakvim pristupom detaljnije su opisani koraci i izbegnute su nejasnoće koje mogu nastati prilikom instalacije. Komande u radnom okviru Laravel se zadaju u komandnom prozoru. Kontroleri, modeli, migracijski fajlovi i tabele u bazi podataka se kreiraju naredbama iz komandnog prozora, pa je u svakoj lekciji istaknuta naredba koja se za njihovo kreiranje koristi.

Radni okvir Laravel sadrži mnogo fajlova, pa je zbog toga na bitnim mestima u lekcijama svaki put naglašeno gde se čuvaju fajlovi koji su neophodni za dalji rad. Elektronske lekcije o rutiranju prate jednostavni primeri koji imaju za cilj da prikažu čemu služi rutiranje i kako se koristi. Ideja je nove pojmove i koncepte prikazati na lakim primerima, da bi se postepeno usvajali.

Osim kratkih primera, elektronske lekcije sadrže i aplikacije kreirane u radnom okviru Laravel. Među njima je aplikacija koja ilustruje korišćenje nasleđivanja stranica. Primer ima za cilj da opisane koncepte i korišćene direktive prikaže u praksi, pa je stranica jednostavna, bez mnogo detalja i sa minimalnim stilizovanjem u jeziku CSS. Rad sa bazom podataka može se smatrati najbitnijim za ozbiljniji rad i veće projekte, stoga je kreirana aplikacija o studentima koja detaljno opisuje funkcije za rad sa podacima iz baze. U elektronskim lekcijama je prikazan kôd kreiranih stranica aplikacije. Svi prethodno opisani koncepti se koriste u kreiranju ove aplikacije. Aplikacija se ne može otvoriti sa platforme, pa je prikazano kako se pokreće

i prikazane su slike izgleda aplikacije. Nijedan primer se ne može pokrenuti direktno sa platforme. Zbog toga je u elektronskim lekcijama opisano kako se pokreće svaki od primera ili su dati linkovi koji otvaraju kreirane stranice.

Sajtovi se često sastoje od stranice za logovanje, pa elektronska lekcija o autentifikaciji ima za cilj da prikaže kako se jednom komandom u radnom okviru Laravel automatski generišu stranice za logovanje i registraciju uz korišćenje direktiva za kontrolu pristupa. Da bi se zaokružila celina svih potrebnih komponenti za kreiranje aplikacija, prikazano je kako dodati fajl na sajt.

Elektronske lekcije su kreirane sistematično, redosledom koji prati osnovno razumevanje, a sa svakim novim pojmom korišćeni su i prethodni pojmovi. Cilj ovakvog pristupa je obanvaljanje naučenog gradiva ili otklanjanje nejasnoća iz prethodnog gradiva.

## Zaključak

Elektronske lekcije o radnom okviru Laravel imaju za cilj da olakšaju proces usvajanja njegovih koncepata. Aplikacija koja je opisana u radu je kreirana da bi se praktično prikazalo kako napraviti veb stranice i kako ih povezati u funkcionalnu celinu.

Svaka elektornska lekcija se oslanja na neku od tri komponente arhitekture MVC, na modele, kontrolere ili deo zadužen za izgled. Neki primeri u radu su stilizovani u jeziku CSS, korišćene su etikete jezika HTML i korišćen je JavaScript, što znači da korisnici koji su ranije koristili nabrojane tehnologije mogu koristiti te tehnologije i u radnom okviru Laravel. Nasleđivanje i kreiranje stranica preko šablona pokazali su se kao jedna od značajnijih pogodnosti u radnom okviru Laravel, jer je moguće napraviti mnogo novih stranica koristeći jednu osnovnu stranicu. Konekcija sa bazom podataka se svodi na izmenu nekoliko vrednosti u fajlu `.env` zbog čega je postupak konekcije jednostavniji nego isti postupak u programskom jeziku PHP. Sve metode koje su zadužene za ispis, upis, menjanje ili brisanje podataka smeštene su u kontrolerima koji su najbitniji za kreiranje aplikacija. Na kraju, svi koncepti koji su prikazani u radu, iskorišćeni su za kreiranje aplikacije koja je opisana u radu i koja predstavlja blog o putovanjima. Kôd kreirane aplikacije je dostupan na linku <https://github.com/detlic/blog>.

Literatura za učenje veb tehnologija je obimna i dostupna na internetu, često na stranim jezicima, najčešće na engleskom, pa učenje može da bude sporije, jer zahteva dobro razumevanje strane literature. Zbog toga elektronske lekcije o radnom okviru Laravel, koje su dostupne na platformi eŠkola veba, imaju posebnu pogodnost što su dostupne na srpskom jeziku. Elektronske lekcije o radnom okviru Laravel imaju za cilj da osposobe čitaoca za kreiranje veb stranica i jednostavnih aplikacija.



## Literatura

- [1] Dayle Rees, *Code smart, the Laravel framework version 5 for beginners*, 2016.
- [2] Jurić Nemanja, Marić Miroslav, *eŠkola veća, VII Simpozijum Matematika i primene*, Matematički fakultet, Beograd, 5. novembar 2016.
- [3] Matt Stauffer, *Laravel Up and Running, a framework for building modern PHP apps*, 2017.
- [4] Nathan Wu., *Laravel 5 cookbook, enhance your amazing applications*, 2016.
- [5] Nathan Wu, *Learning Laravel 5, building practical applications*, 2016.
- [6] Sanjib Sinha, *Beginning Laravel, a beginner's guide to application development with Laravel 5.3*, 2017.
- [7] Sajt <http://itsolutionstuff.com>
- [8] Sajt <https://maxoffsky.com>
- [9] Zvaničan sajt radnog okruženja Bootstrap: <https://getbootstrap.com/>
- [10] Zvaničan sajt radnog okvira Laravel: <https://laravel.com/>
- [11] Webslesson tutorijal: <https://www.webslesson.info>