

UNIVERZITET U BIHAĆU

TEHNIČKI FAKULTET

Odsjek: *Elektrotehnički*

Smjer: *Računarstvo i informatika*

OBJEKTNO-ORIJENTIRANE BAZE PODATAKA
VJEŽBE

Asistent: Zinaid Kapić, MA ing.el.

Akadska godina: 2021/2022.

Tutorijal OODB – Laravel

Cilj:

Upoznati se sa MVC arhitekturom i ORM modelom koristeći razvojno okruženje Laravel. Kreirati jednostavnu CRUD aplikaciju koja će biti osnova za izradu projektnih zadataka. Kreirati složene upite nad većim brojem podataka i tabela.

Vježbe 1 – Instalacija Laravel-a

1. Instalirati zadnju verziju XAMPP-a (paziti da se prilikom instalacije Laravela php doda u path. U suprotnom ručno dodati php u path. <https://www.apachefriends.org/index.html>
2. Instalirati composer <https://getcomposer.org/>
3. Provjeriti verziju php-a. Otvoriti cmd i ukucati **php -v**. Ako je uspješno izvršena komanda i ako je ispisana verzija PHP-a 8.0 i iznad nastaviti dalje korake.
4. Nakon instalacije composera otvoriti cmd i pokrenuti naredbu:

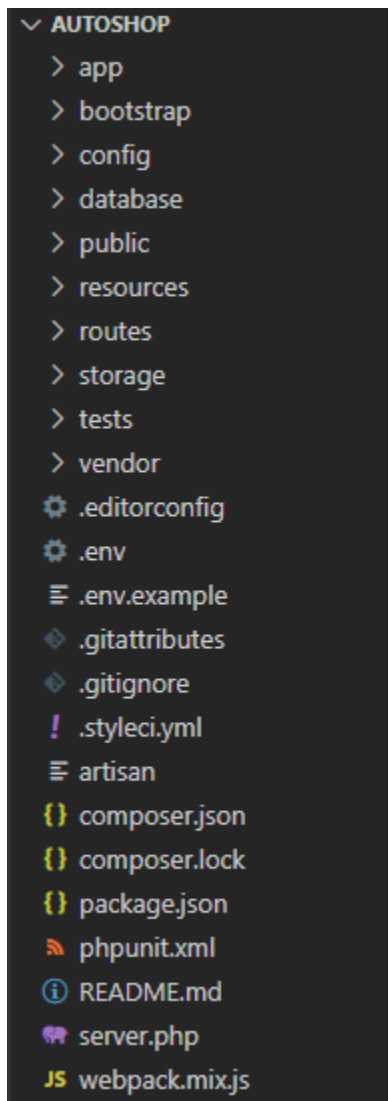
composer global require laravel/installer

5. Nakon uspješne instalacije laravela trebali bi biti u mogućnosti kreirati novi projekat. Prije svega kroz cmd odaberite folder unutar kojeg ćete instalirati novu aplikaciju npr. **cd Desktop** pa nakon toga pokrenuti: **laravel new NAZIVPROJEKTA** npr. **laravel new AutoShop**
6. Nakon izvršene komande uspješno je kreirana nova aplikacija naziva AutoShop na našem desktopu. Sljedeći korak je otvoriti aplikaciju u nekom od Editora (prijedlog je Visual Studio Code).
7. Da bi provjerili ispravnost rada kreirane aplikacije pokrenuti naredbu **php artisan serve**. Najjednostavniji način je nakon otvaranja aplikacije u Visual Studio Code-u otvoriti terminal koji se nalazi u meniju ovog editora i pokrenuti naredbu **php artisan serve**. Nakon toga ispisat će se link na kojem se nalazi aplikacija a to je najčešće **localhost:8000**

Otvoriti taj link u browseru i otvorit će se početna stranica novokreirane Laravel aplikacije koja sadrži reference na dokumentaciju i servise Laravela. Dokumentacija Laravela je najbolja referenca za učenje i istraživanje o Laravelu. Link dokumentacije: <https://laravel.com/docs/8.x>

Vježbe 2 – Objašnjavanje strukture foldera kreirane aplikacije

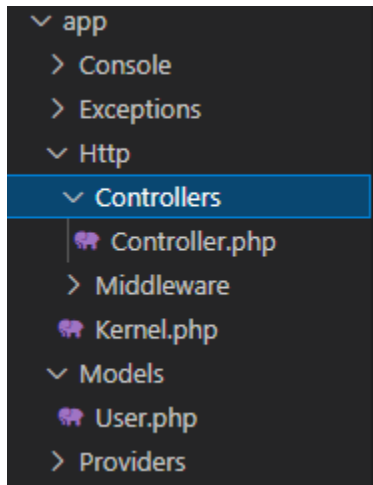
Laravel je kreirao veliki broj foldera i napravio je cjelokupnu strukturu za funkcionisanje aplikacije. Struktura novokreirane aplikacije je sljedeća:



Folderi i fajlovi koji će se koristiti u ovom tutorijalu su:

.env (glavni konfiguracijski fajl koji sadrži naziv, url aplikacije, konekciju sa bazom podataka i druge mnogobrojne postavke aplikacije)

app – folder koji sadrži najvažnije funkcionalne elemente aplikacije, a to su kontroleri, modeli, middleware, ...



database – folder koji sadrži stvari vezane za bazu podataka, a od kojih su nam bitne migracije i seederi.

public – jedini folder koji se prilikom postavljanja aplikacije na server postavlja kao vidljivi dio i sadrži index.php.

resources – sadrži bitne prezentacijske stvari kao što su Views, jezik, Css, JavaScript. Od ključnog značaja su pogledi Views koji će sadržavati html kod naših stranica

routes – unutar ovog foldera koristit će se fajl web.php koji sadrži instrukcije za rutiranja. Prilikom pokušaja odlaska na određeni link aplikacija prvo pregleda web.php fajl i ustanovi lokaciju na koju želimo pristupiti.

Kroz vježbe će se većina fajlova i foldera upoznati. Za početak ćemo izdvojiti:

ROUTES>web.php

APP>HTTP>CONTROLLERS

APP>MODELS

DATABASE>MIGRATIONS

RESOURCES>VIEWS

.env

Vježbe 3 – Kreiranje login i register funkcionalnosti aplikacije

Laravel kao i svaki drugi framework ima mogućnost eksploatacije gotove login i register logike. Laravel nudi nekoliko opcija za kreiranje ovih funkcionalnosti, a za vježbe je odabran Laravel Jetstream, servis Laravela koji u sebi sadrži cjelokupnu autentikacijsku logiku i funkcionalnost, kao i već predefinisane Views, Modele i Controllere kako bi to sve uspješno funkcioniralo. Ovim ubrzavamo razvoj aplikacije. Naravno novokreirane funkcije možemo izmjeniti i prilagoditi svojoj upotrebi.

Laravel Jetstream se podešava na sljedeći način:

```
composer require laravel/jetstream
```

```
php artisan jetstream:install livewire
```

Za sljedeće komande potrebno je imati instaliran Node.js. (<https://nodejs.org/en/>). Restartovati računar i ponovno otvoriti aplikaciju u editoru i izvršiti komande.

```
npm install
```

```
npm run dev
```

Nakon uspješno izvršenih naredbi Laravel je instalirao i dodao nove fajlove. Ti fajlovi su nove rute, resursi, controlleri, modeli i slično.

Da bi testirali rad logina i registra. Potrebno je prije svega unutar **localhost/phpmyadmin** dodati novu praznu tabelu naziva **autoshop**.

Zatim otvoriti **.env** fajl i prilagoditi konekciju sa bazom na sljedeći način:

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:yxNdzj3MGtU3T+mP8TXd8NPFOqcePhC/E1IHkNK24TI=
4 APP_DEBUG=true
5 APP_URL=http://AutoShop.test
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=autoshop
15 DB_USERNAME=root
16 DB_PASSWORD=
```

Nakon dodane baze na phpmyadmin i nakon promjene .env fajla (ako ste sve po tutorijalu instalirali i imenovali neće biti potrebe ni za mijenjanjem env fajla) radi se migracija tabela.

Sad je potrebno izvršiti migraciju odnosno slanje novokreiranih tabela (tabele je Laravel sam kreirao) na bazu. To se vrši sljedećom komandom:

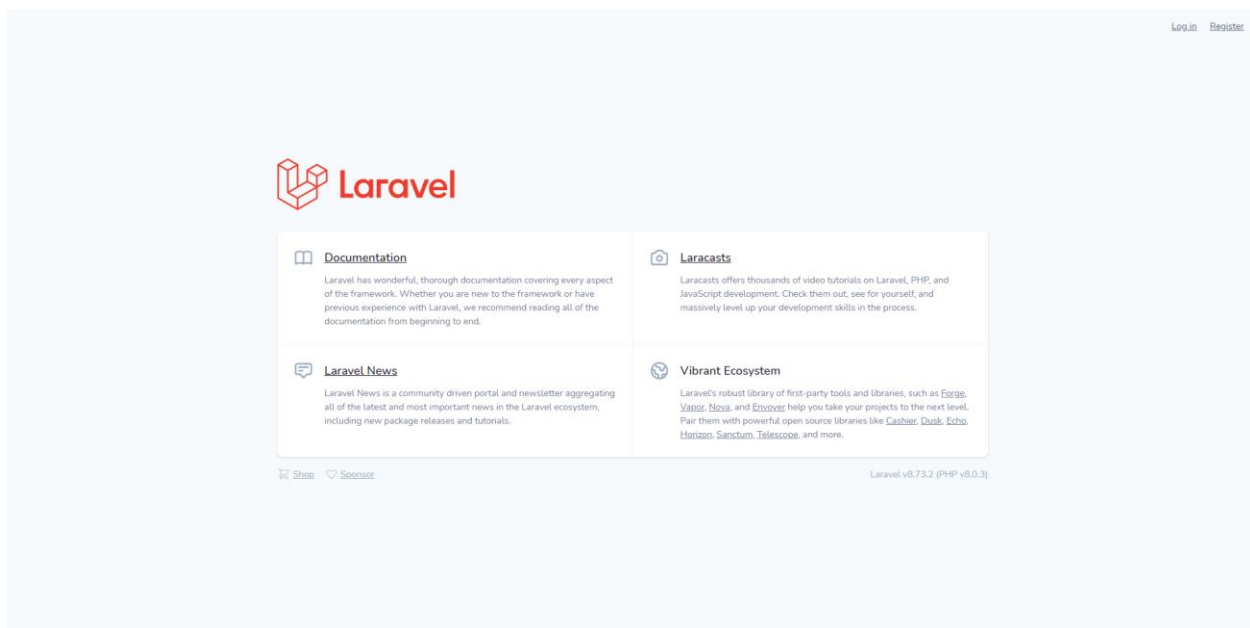
php artisan migrate

Ovom naredbom je poslana cijela migracija na bazu i baza sad izgleda ovako:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> failed_jobs	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
<input type="checkbox"/> migrations	★ Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> password_resets	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
<input type="checkbox"/> personal_access_tokens	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
<input type="checkbox"/> sessions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
6 tables	Sum	6	InnoDB	latin1_swedish_ci	208.0 KiB	0 B

Sljedeće što je potrebno je upaliti Xampp Apache i MySQL i ponovno pokrenuti server naredbom:

php artisan serve



Vježbe 4 – Modifikacija ruta i login/register

Otvoriti folder routes i fajl web.php

```
<?php

use Illuminate\Support\Facades\Route;

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

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

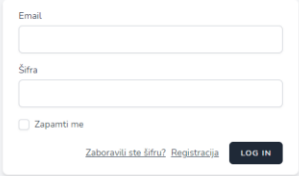
Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {
    return view('dashboard');
})->name('dashboard');
```

Obrisati prvu rutu cjelokupnu ili je komentirati, a dashboard zamjeniti sa praznim zahtjevom tako da će defaultno naša aplikacija pri prvom pokretanju zahtijevati da korisnik bude logiran. Ako nije logiran odvest će ga na login View, a suprotno otvorit će dashboard View.

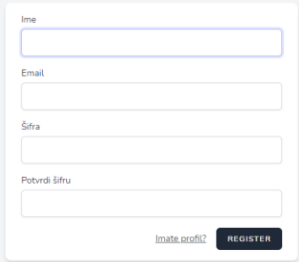
Sad ćemo izmjeniti izgled login i register View-a. Otvoriti folder *resources > views > auth*. U ovom folderu se nalaze svi Views za autentikaciju a među kojima je i login i register.

ZADATAK:

Izmjeniti login i register da izgledaju kao na slikama:

A login form with a light gray background. It features two input fields: 'Email' and 'Šifra' (Password). Below the password field is a checkbox labeled 'Zapamti me' (Remember me). At the bottom, there are two links: 'Zaboravili ste šifru?' (Forgot your password?) and 'Registracija' (Registration), followed by a dark gray button labeled 'LOG IN'.

Login

A registration form with a light gray background. It features four input fields: 'Ime' (Name), 'Email', 'Šifra' (Password), and 'Potvrdi šifru' (Confirm password). At the bottom, there are two links: 'Imate profil?' (Do you have a profile?) and a dark gray button labeled 'REGISTER'.

Register

Zatim registrirati se i prijaviti na aplikaciju.

Vježbe 5 – Kreiranje nove rute i pogleda Cars

Unutar foldera *resources>views* se nalazi fajl **navigation.blade.php** unutar kojeg treba dodati još jedan **nav-link** koji će voditi na stranicu Auta. Taj link se dodaje na sljedeći način:

```
1 <nav x-data="{ open: false }" class="bg-white border-b border-gray-100">
2     <!-- Primary Navigation Menu -->
3     <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
4         <div class="flex justify-between h-16">
5             <div class="flex">
6                 <!-- Logo -->
7                 <div class="flex-shrink-0 flex items-center">
8                     <a href="{{ route('dashboard') }}">
9                         <x-jet-application-mark class="block h-9 w-auto" />
10                    </a>
11                </div>
12
13                <!-- Navigation Links -->
14                <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
15                    <x-jet-nav-link href="{{ route('dashboard') }}" :active="request()->routeIs('dashboard')">
16                        {{ __('Početna') }}
17                    </x-jet-nav-link>
18                </div>
19
20                <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
21                    <x-jet-nav-link href="{{ route('cars') }}" :active="request()->routeIs('cars')">
22                        {{ __('Auta') }}
23                    </x-jet-nav-link>
24                </div>
25            </div>
26        </div>
27    </div>
28 </nav>
```

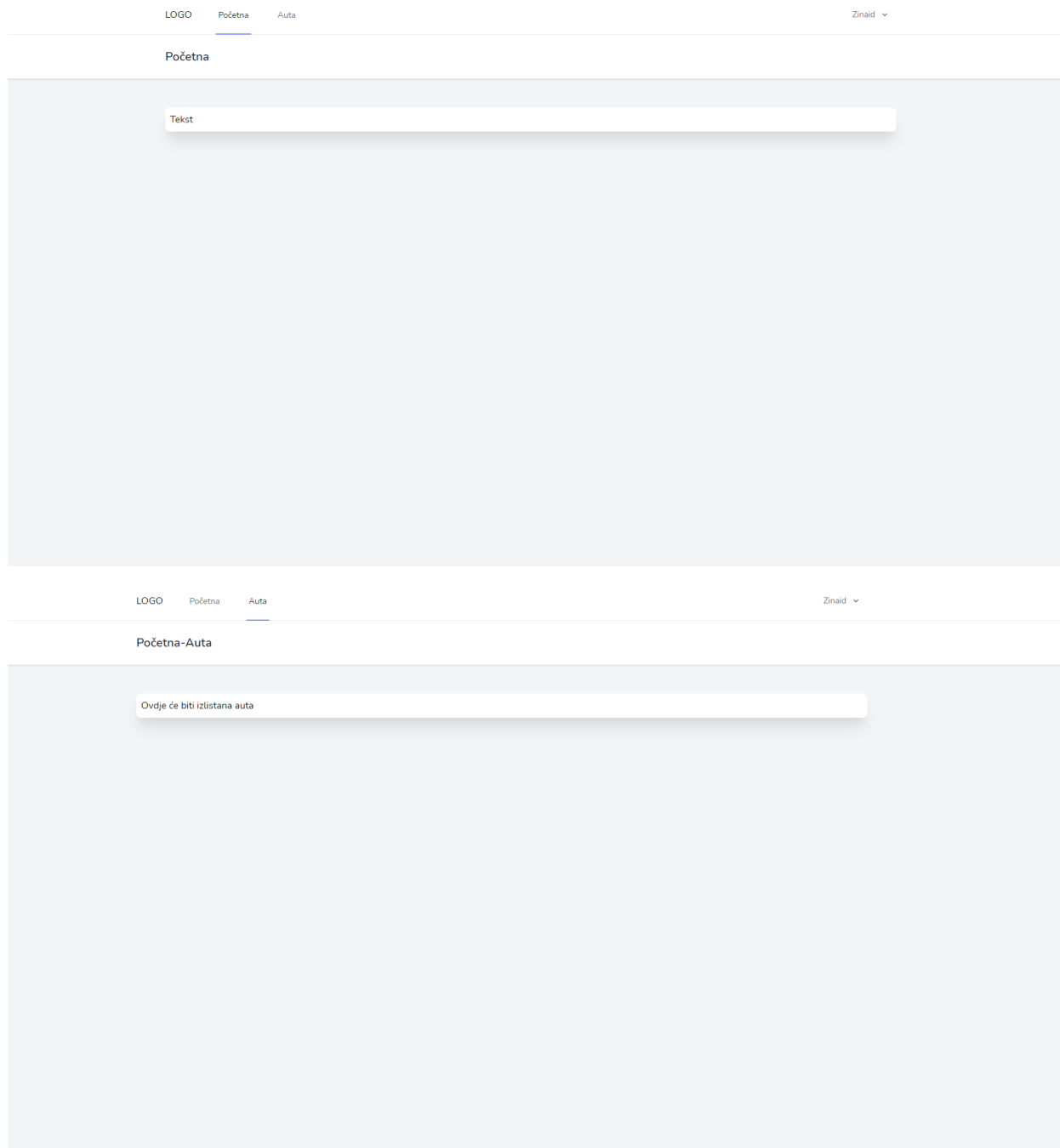
Kao što vidimo promjenili smo link zahtjeva i tekst koji će pisati na navigacionom meniju.

Dalje je potrebno dodati novi pogled unutar resursa. Obzirom da je lijepa praksa da svaki model ili svaku imenicu odvojimo u zaseban folder to ćemo i uraditi. Unutar *resources>views* ćemo kreirati novi folder naziva **cars**. Unutar tog foldera kreirat ćemo fajl naziva **index.blade.php** i u taj fajl ćemo kopirati cjelokupan sadržaj **dashboard.blade.php** pogleda. Sada prilikom pokretanja stranice javit će se greška koja kaže da ruta *cars* nije definisana. Istu je potrebno definisati u rutama na sljedeći način:

```
16 Route::middleware(['auth:sanctum', 'verified'])->get('/', function () {
17     return view('dashboard');
18 }->name('dashboard');
19
20 Route::middleware(['auth:sanctum', 'verified'])->get('/cars', function () {
21     return view('cars.index');
22 }->name('cars');
23
```

ZADATAK:

Izmjeniti poglede *dashboard.blade.php* i *cars>index.blade.php* tako da odgovaraju sljedećim slikama:



Sljedeći korak će biti kreiranje modela, kontrolera i migracije za objekat Cars.

Vježbe 6 – Kreiranje modela, kontrolera i migracije

Svi kontroleri se nalaze u folderu *app>http>controllers* i standardno se imenuju po objektu čije metode sadrže npr. ***CarsController.php***.

Sve migracije se nalaze u folderu *database>migrations*.

Svi modeli se nalaze u folderu *app>models* i standardno se imenuju po objektu kojeg opisuju npr. ***Cars.php***.

Svi ovi fajlovi se mogu kreirati i ručno, ali Laravelov artisan daje mogućnost automatskog kreiranja sva tri fajla i to sa unaprijed kreiranom strukturom.

Naredba za kreiranje modela, migracije i kontrolera

php artisan make:model -mcr Car

Sada je potrebno izmjeniti migraciju i model tako da odgovaraju tabeli koja nam treba. Cilj nam je napraviti sljedeću tabelu:

CARS

#id

*** name (varchar)**

*** year (date)**

*** engine (double)**

*** code (integer) unique**

o air_condition (integer)

*** brand (integer) FK** (Ovo ćemo staviti jer će kasnije biti potreban ovaj Foreign Key za povezivanje sa tabelom brands (tj. marke auta). Laravel nudi i mogućnost kreiranja FK Constraints ovdje na sljedeći način: `$table->foreign('user_id')->references('id')->on('users');` kao i neke naprednije stvari rada sa stranim ključevima. Zasad to nećemo koristiti i ovaj FK ćemo kreirati kao obični integer koji je ustvari id iz tabele koju ćete kasnije kreirati i zove se **brands**.

Dakle otvaramo posljednju kreiranu migraciju za cars i mjenjamo je na sljedeći način:

```
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateCarsTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('cars', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->date('year');
20             $table->double('engine');
21             $table->integer('code')->unique();
22             $table->integer('air_condition')->nullable();
23             $table->integer('brand');
24         });
25     }
26 }
```

Model mora odgovarati migraciji.

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Car extends Model
9  {
10     public $timestamps = false;
11
12     use HasFactory;
13     /**
14      * The attributes that are mass assignable.
15      *
16      * @var string[]
17      */
18     protected $fillable = [
19         'name',
20         'year',
21         'engine',
22         'code',
23         'air_condition',
24         'brand',
25     ];
26
27 }
```

Sada migriramo migraciju da prenese ovu tabelu u našu bazu podataka autoshop. Migracija se radi na dosad pokazani način: *php artisan migrate*.

Sada želimo izmjeniti rutu '\cars' na način da nakon upita aplikacija prvo posjeti controller (gdje bi naprimjer mogli uraditi neke upite prema bazi, obradu podataka i tek onda vratiti pogled cars.blade.php).

Dakle mijenjamo **web.php** na sljedeći način:

```
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\CarController;
5
6  /*
7  |-----
8  | Web Routes
9  |-----
10 |
11 | Here is where you can register web routes for your application. These
12 | routes are loaded by the RouteServiceProvider within a group which
13 | contains the "web" middleware group. Now create something great!
14 |
15 */
16
17 Route::middleware(['auth:sanctum', 'verified'])->get('/', function () {
18     return view('dashboard');
19 })->name('dashboard');
20
21 /*
22 Route::middleware(['auth:sanctum', 'verified'])->get('/cars', function () {
23     return view('cars.index');
24 })->name('cars');
25 */
26
27 Route::middleware(['auth:sanctum', 'verified'])->get('cars', [CarController::class, 'index'])->name('cars');
```

Dakle prije svega treba uključiti kontroler odmah na vrhu fajla.

use App\Http\Controllers\CarController;

da bi mogli koristiti sve metode unutar ovog kontrolera. Dakle sad prilikom upita '\cars' prvo se pogleda **CarController** i nađe metoda '**index**' koja je zasad prazna.

Ako otvorimo **CarController** i u metodu index upišemo npr. **dd('ZINAID');** dobit ćemo ispis tog teksta u browseru jer naredba **dd** ustvari mijenja **php-ov var_dump + exit;**

Sav proces obrade podataka, upisa, ispisa, modificiranja baze podataka ili samih podataka se izvršava unutar controllera. Nakon uspješno izvršenih funkcija vraćamo se na View na isti način kao što je i prije rađeno na web.php. Ovdje npr. možemo vratiti i povučene podatke iz baze podataka zajedno sa View-om.

Radi testiranja ručno unijeti dva auta u tabelu cars preko phpmyadmina.

Sada ćemo unutar CarController index metode izvršiti upit prema bazi podataka i ispisati povučene podatke.

To se radi na sljedeći način

```
app > Http > Controllers > CarController.php
1  <?php
2
3  namespace App\Http\Controllers;
4  use DB;
5
6  use App\Models\Car;
7  use Illuminate\Http\Request;
8
9  class CarController extends Controller
10 {
11     /**
12      * Display a listing of the resource.
13      *
14      * @return \Illuminate\Http\Response
15      */
16     public function index()
17     {
18         $cars = DB::table('cars')
19             ->get();
20
21         dd($cars);
22     }
23 }
```

Paziti da se uključi **use DB**; kako bi se moglo pristupiti funkcijama Laravel Query Buildera.

Dakle, povukli smo sve iz tabele cars i spremili to u php varijablu **\$cars**. Ovdje je ilustriran **ORM** princip gdje su se podaci iz relacione baze podataka spremili u vidu objekata tj. kolekcije objekata u ovom slučaju. Podaci kad se ispišu dobije se sljedeće:

```
Illuminate\Support\Collection {#1390 ▼
  #items: array:2 [▼
    0 => {#1394 ▼
      +"id": 1
      +"name": "Audi A3"
      +"year": "2011-11-09"
      +"engine": 2.0
      +"code": 1234
      +"air_condition": 1
      +"brand": 1
    }
    1 => {#1395 ▼
      +"id": 2
      +"name": "Renault Clio"
      +"year": "2021-11-03"
      +"engine": 1.2
      +"code": 2233
      +"air_condition": 0
      +"brand": 2
    }
  ]
  #escapeWhenCastingToString: false
}
```

Sada želimo povučene podatke poslati na View **cars.index.blade.php**. To radimo na sljedeći način:

```
public function index()
{
    $cars = DB::table('cars')
        ->get();

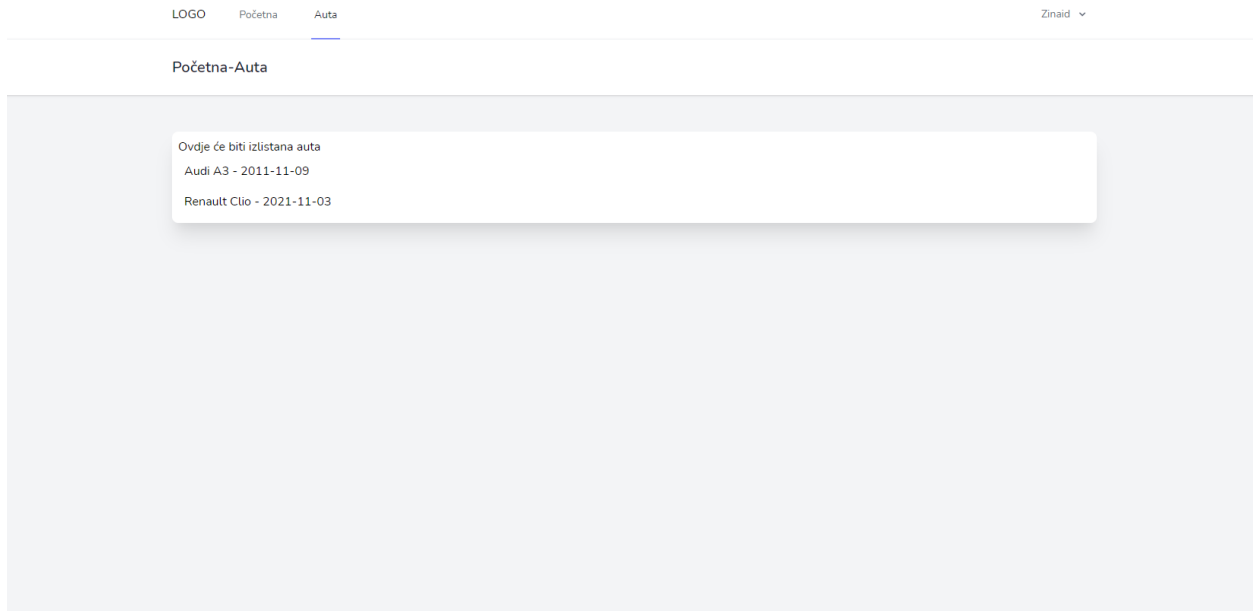
    return view('cars.index', ['cars' => $cars]);
}
```

Sada, osim što je učitani pogled **cars.index.blade.php** poslana je i kolekcija svih auta iz baze podataka. Te podatke sada koristimo kao globalne varijable u tom pogledu ili koristimo Laravelov Blade template rendering kako bi olakšali rad sa objektima.

Npr. podatke možemo ispisati na View-u **cars.index.blade.php** na sljedeći način:

```
resources > views > cars > index.blade.php
1  <x-app-layout>
2      <x-slot name="header">
3          <h2 class="font-semibold text-xl text-gray-800 leading-tight">
4              {{ __('Početna-Auta') }}
5          </h2>
6      </x-slot>
7
8      <div class="py-12">
9          <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10             <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11                 <div class="p-2">
12                     <h1 class="font-xl">Ovdje će biti izlistana auta</h1>
13                     @foreach ($cars as $car)
14                         <p class="p-2"> {{ $car->name }} - {{ $car->year }}</p>
15                     @endforeach
16                 </div>
17             </div>
18         </div>
19     </div>
20 </x-app-layout>
```

Što daje rezultat:



ZADATAK:

Isto ponoviti za brendove auta. Dakle, kreirati model, migraciju i controller. Dodati brendove u navigacioni meni i ispisati sve brendove iz baze.

RJEŠENJE:

Prvo ćemo kreirati novi folder unutar *views* naziva **brands** i u njemu fajl **index.blade.php** i kopirat ćemo sadržaj iz *cars>index.blade.php* fajla u ovaj novi fajl. Zatim ćemo kreirati migraciju, model i controller za brandove naredbom:

php artisan make:model -mcr Brand

Dalje ćemo izmjeniti fajl *navigation.blade.php* koji se nalazi u resources i dodati novi nav-link:

```
12
13
14 <!-- Navigation Links -->
15 <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
16   <x-jet-nav-link href="{{ route('dashboard') }}" :active="request()->routeIs('dashboard')">
17     {{ __('Početna') }}
18   </x-jet-nav-link>
19 </div>
20
21 <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
22   <x-jet-nav-link href="{{ route('cars') }}" :active="request()->routeIs('cars')">
23     {{ __('Auta') }}
24   </x-jet-nav-link>
25 </div>
26
27 <div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
28   <x-jet-nav-link href="{{ route('brands') }}" :active="request()->routeIs('brands')">
29     {{ __('Brendovi') }}
30   </x-jet-nav-link>
```


Samim tim mora se dodati nova ruta u **web.php** naziva brands, s tim da se mora i dodati ruta kontrolera na vrhu fajla kako bi mogli koristiti funkcije iz ovog kontrolera:

```
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\CarController;
5  use App\Http\Controllers\BrandController;
6
7  /*
8  |-----
9  | Web Routes
10 |-----
11 |
12 | Here is where you can register web routes for your application. These
13 | routes are loaded by the RouteServiceProvider within a group which
14 | contains the "web" middleware group. Now create something great!
15 |
16 */
17
18 Route::middleware(['auth:sanctum', 'verified'])->get('/', function () {
19     return view('dashboard');
20 }->name('dashboard'));
21
22 /*
23 Route::middleware(['auth:sanctum', 'verified'])->get('/cars', function () {
24     return view('cars.index');
25 }->name('cars'));
26 */
27
28 Route::middleware(['auth:sanctum', 'verified'])->get('cars', [CarController::class, 'index'])->name('cars');
29
30 Route::middleware(['auth:sanctum', 'verified'])->get('brands', [BrandController::class, 'index'])->name('brands');
31
```

Sad ćemo modificirati migraciju i model vezan za brendove na način da reprezentiraju sljedeću tabelu:

BRANDS

#id

* name (varchar)

* country (varchar)

Dakle fajl za migraciju izgleda kao:

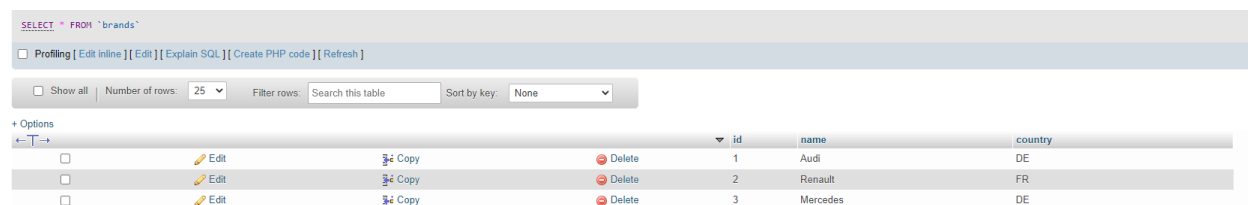
```
14     public function up()
15     {
16         Schema::create('brands', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->string('country');
20         });
21     }
```

A model ovako:

```
app > Models > Brand.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Brand extends Model
9  {
10     public $timestamps = false;
11
12     use HasFactory;
13     /**
14      * The attributes that are mass assignable.
15      *
16      * @var string[]
17      */
18     protected $fillable = [
19         'name',
20         'country',
21     ];
22 }
```

Sada je potrebno migrirati tabelu u bazu autoshop naredbom *php artisan migrate*, a zatim ponovno upaliti server sa *php artisan serve*.

Ručno ćemo u tabelu brands dodati tri brenda auta.



	id	name	country
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	1	Audi	DE
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2	Renault	FR
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	3	Mercedes	DE

Sada ćemo isto ispisati na view *brands>index.blade.php* tako što ćemo u kontroleru povući sve brendove iz baze i grupirati ih po državi kojoj pripadaju i to prikazati na *brands>index.blade.php*.

Paziti da se na početku fajla doda **use DB** kako bi se mogle vršiti operacije nad bazom podataka. Upit odgovara upitu za povlačenje svih auta sa dodatkom naredbe **orderBy('country')**. O mogućnostima rada sa Laravel Database Query Builder posavjetovati se na sljedećem linku:

<https://laravel.com/docs/8.x/queries>

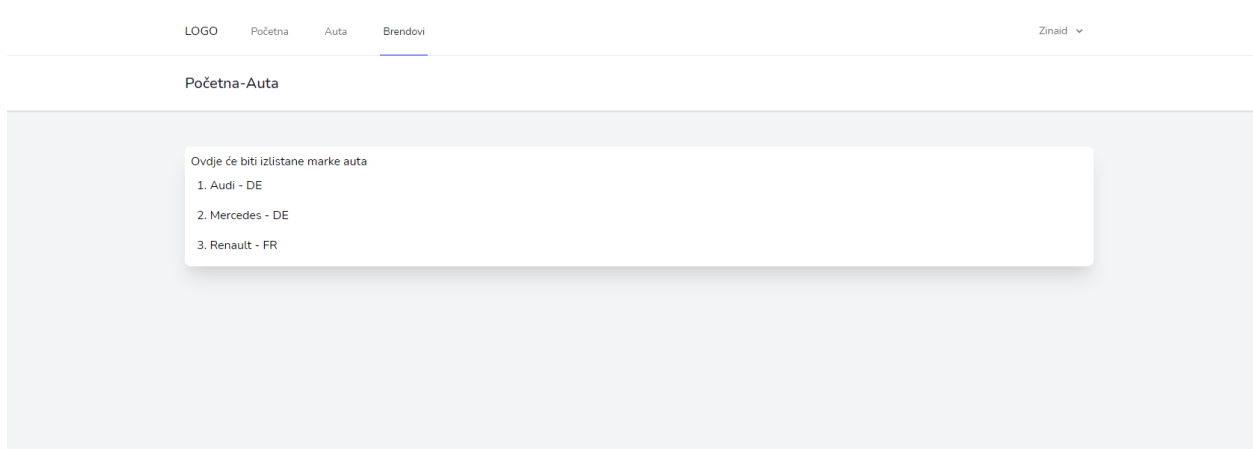
Sljedeća slika pokazuje izgled **BrandControllera**:

```
16     public function index()
17     {
18         $brands = DB::table('brands')
19             ->orderBy('country')
20             ->get();
21
22         return view('brands.index', ['brands' => $brands]);
23     }
```

Potrebno je izmjeniti i *brands>index.blade.php*:

```
resources > views > brands > index.blade.php
1  <x-app-layout>
2      <x-slot name="header">
3          <h2 class="font-semibold text-xl text-gray-800 leading-tight">
4              {{ __('Početna-Auta') }}
5          </h2>
6      </x-slot>
7
8      <div class="py-12">
9          <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10             <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11                 <div class="p-2">
12                     <h1 class="font-xl">Ovdje će biti izlistane marke auta</h1>
13                     @foreach ($brands as $brand)
14                         <p class="p-2">{{ $loop->iteration }}. {{ $brand->name }} - {{ $brand->country }}</p>
15                     @endforeach
16                 </div>
17             </div>
18         </div>
19     </div>
20 </x-app-layout>
```

Ispis toga svega je sljedeći:



Vježbe 7 – Dodavanje auta u bazu

U ovim vježbama pokazat ćemo dodavanje novog reda u tabelu koristeći formu. Dodat ćemo novo auto u bazu. Prije svega na stranici `cars>index.blade.php` dodat ćemo anchor koji vodi na stranicu za dodavanje auta.

```
7
8     <div class="py-12">
9         <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10             <a href="\add_car" class="m-2 p-2 text-xl">Dodaj auto</a>
11             <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
12                 <div class="p-2">
13                     <h1 class="font-xl">Ovdje će biti izlistana auta</h1>
14                     @foreach ($cars as $car)
15                         <p class="p-2"> {{ $car->name }} - {{ $car->year }}</p>
16                     @endforeach
17                 </div>
18             </div>
19         </div>
20     </div>
21 </x-app-layout>
```

Nakon toga je potrebno definisati tu rutu **add_car** kao i kreirati view za tu rutu. Sada u folder `resources>views>cars` dodati view **add.blade.php** i izmjeniti fajl **web.php**:

```
27
28 Route::middleware(['auth:sanctum', 'verified'])->get('cars', [CarController::class, 'index'])->name('cars');
29 Route::middleware(['auth:sanctum', 'verified'])->get('add_car', [CarController::class, 'create'])->name('add_car');
30
31 Route::middleware(['auth:sanctum', 'verified'])->get('brands', [BrandController::class, 'index'])->name('brands');
32
```

Dakle kreirali smo novi *get* zahtjev na poziv url-a **add_car** koji ide na **CarController** i metodu **create**.

U metodi `create` vraćamo taj novokreirani view **add.blade.php** na sljedeći način:

```
30     public function create()
31     {
32         return view('cars.add');
33     }
```

View **add.blade.php** ćemo izmjeniti kao:

```
8 <div class="py-12">
9 <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10 <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11 <div class="p-2">
12 <form method="POST" action="{{ route('store_car') }}">
13 @csrf
14 <div>
15 <x-jet-label for="name" value="{{ __('Naziv') }}" />
16 <x-jet-input id="name" class="block mt-1 w-full" type="text" name="name" :value="old('name')" required autofocus />
17 </div>
18 <div class="mt-4">
19 <x-jet-label for="year" value="{{ __('Godina') }}" />
20 <x-jet-input id="year" class="block mt-1 w-full" type="date" name="year" required autofocus />
21 </div>
22 <div class="mt-4">
23 <x-jet-label for="engine" value="{{ __('Motor') }}" />
24 <x-jet-input id="engine" class="block mt-1 w-full" type="number" name="engine" required autofocus />
25 </div>
26 <div class="mt-4">
27 <x-jet-label for="code" value="{{ __('Sifra') }}" />
28 <x-jet-input id="code" class="block mt-1 w-full" type="number" name="code" required autofocus />
29 </div>
30 <div class="mt-4">
31 <x-jet-label for="brand" value="{{ __('Brend') }}" />
32 <x-jet-input id="brand" class="block mt-1 w-full" type="number" name="brand" required autofocus />
33 </div>
34 <div class="flex items-center justify-end mt-4">
35 <x-jet-button class="ml-4">
36 {{ __('Spremi') }}
37 </x-jet-button>
38 </div>
39 </form>
40 </div>
41 </div>
42 </div>
```

Dakle napravljena je forma metode *POST* i akcije **store_car** i određenih inputa koje trebamo popuniti i poslati radi spremanja na bazu. Prije svega treba napraviti rutu **store_car** u **web.php**.

```
28 Route::middleware(['auth:sanctum', 'verified'])->get('cars', [CarController::class, 'index'])->name('cars');
29 Route::middleware(['auth:sanctum', 'verified'])->get('add_car', [CarController::class, 'create'])->name('add_car');
30 Route::middleware(['auth:sanctum', 'verified'])->post('store_car', [CarController::class, 'store'])->name('store_car');
31
32 Route::middleware(['auth:sanctum', 'verified'])->get('brands', [BrandController::class, 'index'])->name('brands');
```

Popunjavanjem ovih inputa i klikom na dugme *Spremi* poslat će se zahtjev na metodu **CarControllera** koja se naziva **store** i koja ima parametar *Request \$request* a koji ustvari sadrži sve poslano tim zahtjevom.

```
41 public function store(Request $request)
42 {
43     $request->validate([
44         'name' => 'required|string|max:255',
45     ]);
46
47     DB::table('cars')->insert([
48         'name' => $request->name,
49         'year' => $request->year,
50         'engine' => $request->engine,
51         'code' => $request->code,
52         'brand' => $request->brand,
53     ]);
54
55     return redirect()->route('cars');
56
57 }
```

ZADATAK:

U formi umjesto unosa broja za upisivanje *Marke/Brenda* auta dodati **select** koji ispisuje sve brendove unutar tabele *brands* gdje će biti ispisan njihov **naziv** a u **value** opcije selecta će biti dodan njihov **id** kao parametar koji se ustvari i sprema u tabeli *cars*.

RJEŠENJE:

Dakle, cilj je imati izlistane brendove iz tabele u selectu unutar forme. Ono što nama treba na pogledu **add.blade.php** je informacija o svim brendovima iz tabele *brands*. To ćemo dobiti tako što ćemo u metodi koja nam vraća view **add.blade.php** proslijediti i kolekciju brendova. Ta metoda se naziva **create** i nalazi se u **CarControlleru**. Pokupit ćemo sve brendove i poslati ih na view u sklopu return-a.

```
30     public function create()
31     {
32         $brands = DB::table('brands')
33             ->get();
34
35         return view('cars.add', ['brands' => $brands]);
36     }
```

Brendove koji su poslani na view *add.blade.php* treba ispisati u selectu, a to ćemo odraditi na sljedeći način:

```
30     <div class="mt-4">
31         <x-jet-label for="brand" value="{{ __('Brend') }}" />
32         <select id="brand" name="brand" class="form-select block w-full mt-1 border-gray-300 focus:border-indigo-300
33             focus:ring focus:ring-indigo-200 focus:ring-opacity-50 rounded-md shadow-sm">
34             <option selected="true" disabled="disabled">Odaberi</option>
35             @foreach($brands as $brand)
36                 <option value="{{ $brand->id }}">{{ $brand->name }}</option>
37             @endforeach
38         </select>
39     </div>
```

Spremanje će raditi isto kao što je i radilo jer select ima isti name kao i input koji je stojao prije njega, tako da se pozadina ne mijenja. U opciju selecta je ispisan naziv marke, a value je jednak id-u marke.

Vježbe 8 – Brisanje i editovanje auta u bazi

Prije svega treba napraviti *button* koji je okidač za brisanje auta. Pored svakog ispisanog auta na view-u dodati dva buttona-a. Jedan za edit, a jedan za delete. To se radi na view-u *cars>index.blade.php*:

```
resources > views > cars > index.blade.php
7
8 <div class="py-12">
9 <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10 <a href="#" add_car" class="m-2 p-2 text-xl flex items-center justify-end">Dodaj auto</a>
11 <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
12 <div class="p-2">
13 <h1 class="font-xl mb-4 text-center">Ovdje će biti izlistana auta</h1>
14 <hr/>
15 @foreach ($cars as $car)
16 <div class="flex space-x-4">
17 <div class="flex-1"><p class="p-2"> {{ $car->name }} - {{ $car->year }} </div>
18 <div class="flex-1">
19 <form method="POST" action="{{ route('delete_car') }}">
20 @csrf
21 <input type="hidden" name="id" value="{{ $car->id }}">
22 <div class="p-2">
23 <button class="ml-4 inline-flex items-center px-4 py-2 bg-red-700 border border-transparent rounded-md font-semibold text-xs text-white uppercase ml-4">
24 {{ __('Obrisi') }}
25 </button>
26 </div>
27 </form>
28 </div>
29 <div class="flex-1">
30 <form method="POST" action="{{ route('edit_car') }}">
31 @csrf
32 <input type="hidden" name="id" value="{{ $car->id }}">
33 <div class="p-2">
34 <button class="ml-4 inline-flex items-center px-4 py-2 bg-green-700 border border-transparent rounded-md font-semibold text-xs text-white uppercase">
35 {{ __('Uredi') }}
36 </button>
37 </div>
38 </form>
39 </div>
40 </div>
41 @endforeach
42 </div>
43
```

Nakon toga potrebno je kreirati rute za brisanje i editovanje. Prvo ćemo riješiti brisanje, tako što ćemo dodati rutu **delete_car**, a koja vodi na kontroler **CarController** i metodu **delete** koju isto tako kreiramo, a koja prima request. Ovdje se na osnovu poslanog id-a auta vrši brisanje istog iz baze podataka, te se nakon uspješnog brisanja vraća na pogled sa listom auta.

```
31 Route::middleware(['auth:sanctum', 'verified'])->post('delete_car', [CarController::class, 'delete'])->name('delete_car');
32 Route::middleware(['auth:sanctum', 'verified'])->post('edit_car', [CarController::class, 'edit'])->name('edit_car');
```

Metoda delete unutar kontrolera CarController izgleda ovako:

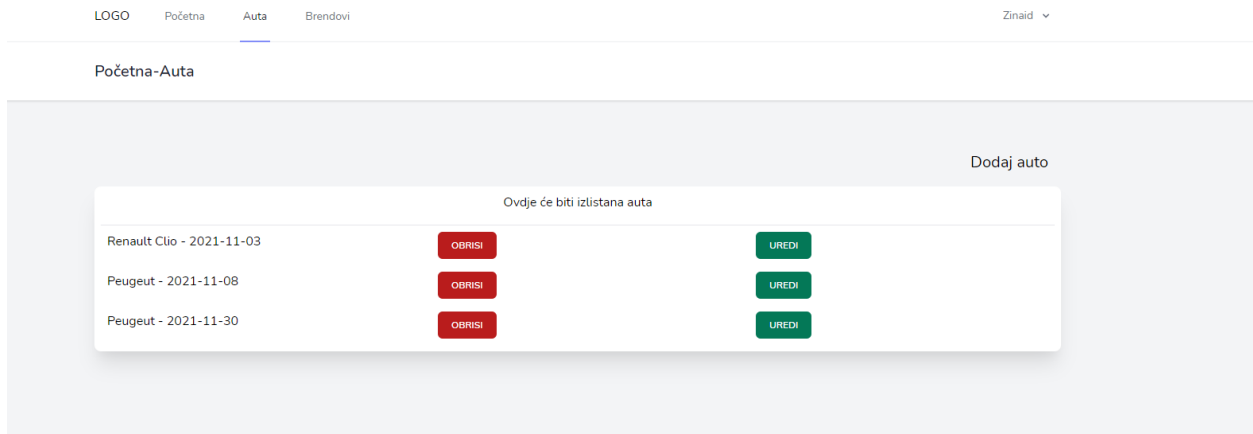
```
59 public function delete(Request $request){
60     $id=$request->id;
61
62     Car::destroy($id);
63
64     return redirect()->route('cars');
65
66 }
```

Na prethodni način završena je funkcionalnost brisanja auta. Preko POST forme pošalje se id određenog auta koji je spremljen u skrivenom inputu. Koristeći metodu **destroy(\$id)** nad modelom Car se izvrši brisanje auta koji ima određen id. Ovo se moglo riješiti i koristeći Database Query Builder i standardnim zahtjevom za brisanje iz baze podataka koji izgleda ovako:

```
DB::table('cars')->where('id', $id)->delete();
```

Poslije toga radi se redirekcija na rutu *cars*.

View *cars>index.blade.php* sad izgleda ovako:



Sada je potrebno da se klikom na dugme Uredi otvori forma za uređivanje određenog auta. Dakle već imamo dugme koje proslijeđuje id i dalje je potrebno napraviti novi view *edit.blade.php* u folderu *cars*. Taj view ćemo otvarati klikom na dugme Uredi i aktiviranjem rute *edit_car* koja pristupa **CarControlleru** i metodi **edit**.

Metoda **edit** izgleda kao na slici:

```
89 public function edit(Request $request)
90 {
91     $id = $request->id;
92
93     $cars = DB::table('cars')
94         ->where('id', $id)
95         ->get();
96
97     $brands = DB::table('brands')
98         ->get();
99
100     return view('cars.edit', ['cars' => $cars , 'brands' => $brands]);
101 }
102
```

Obzirom da je dodan select koji sadrži brendove, pored određenog auta kojeg uređujemo moramo vratiti i brendove kako bi mogli isti promjeniti.

Sada treba izmjeniti view *edit.blade.php*.

View *edit.blade.php* izgleda kao:

```
8      <div class="py-12">
9          <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10             <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11                 <div class="p-2">
12                     @foreach($cars as $car)
13                         <form method="POST" action="{{ route('update_car') }}">
14                             @csrf
15                             <input type="hidden" name="id" value="{{ $car->id }}" />
16                             <div>
17                                 <x-jet-label for="name" value="{{ __( 'Naziv' ) }}" />
18                                 <x-jet-input id="name" class="block mt-1 w-full" type="text" name="name" value="{{ $car->name }}" required autofocus />
19                             </div>
20                             <div class="mt-4">
21                                 <x-jet-label for="year" value="{{ __( 'Godina' ) }}" />
22                                 <x-jet-input id="year" class="block mt-1 w-full" type="date" name="year" value="{{ $car->year }}" required autofocus />
23                             </div>
24                             <div class="mt-4">
25                                 <x-jet-label for="engine" value="{{ __( 'Motor' ) }}" />
26                                 <x-jet-input id="engine" class="block mt-1 w-full" type="text" name="engine" value="{{ $car->engine }}" required autofocus />
27                             </div>
28                             <div class="mt-4">
29                                 <x-jet-label for="code" value="{{ __( 'Sifra' ) }}" />
30                                 <x-jet-input id="code" class="block mt-1 w-full" type="number" name="code" value="{{ $car->code }}" required autofocus />
31                             </div>
32                             <div class="mt-4">
33                                 <x-jet-label for="brand" value="{{ __( 'Brend' ) }}" />
34                                 <select id="brand" name="brand" class="form-select block w-full mt-1 border-gray-300 focus:border-indigo-300
35                                     focus:ring focus:ring-indigo-200 focus:ring-opacity-50 rounded-md shadow-sm">
36                                     <option>Odaberi</option>
37                                     @foreach($brands as $brand)
38                                         <option value="{{ $brand->id }}"
39                                             @if($car->brand == $brand->id) selected
40                                         @endif>{{ $brand->name }}</option>
41                                     @endforeach
42                                 </select>
43                             </div>
44                             <div class="flex items-center justify-end mt-4">
45                                 <x-jet-button class="ml-4">
46                                     {{ __( 'Spremi' ) }}
47                                 </x-jet-button>
48                             </div>
49                         </form>
50                     @endforeach
51                 </div>
52             </div>
53         </div>
54     </div>
```

Radi se foreach auta koji je vraćen i inputi se popunjavaju sa vrijednostima vezanih za taj objekat **\$cars**. U selectu se radi foreach svih brendova i provjerava se da li vrijednost **\$car->brand** odgovara id-u od nekog selecta, odnosno **\$brand->id**. Ako je to zadovoljeno opciji selecta se dodaje atribut **selected** što znači da će taj **option** biti odabran. Pored toga doda se i **input** koji je hidden i nosi naziv **id** kako bi prosljedili id auta kojeg uređujemo.

Obzirom da u ovoj formi imamo novu rutu **update_car** istu moramo dodati u **web.php** na sljedeći način:

```
28 Route::middleware(['auth:sanctum', 'verified'])->get('cars', [CarController::class, 'index'])->name('cars');
29 Route::middleware(['auth:sanctum', 'verified'])->get('add_car', [CarController::class, 'create'])->name('add_car');
30 Route::middleware(['auth:sanctum', 'verified'])->post('store_car', [CarController::class, 'store'])->name('store_car');
31 Route::middleware(['auth:sanctum', 'verified'])->post('delete_car', [CarController::class, 'delete'])->name('delete_car');
32 Route::middleware(['auth:sanctum', 'verified'])->post('edit_car', [CarController::class, 'edit'])->name('edit_car');
33 Route::middleware(['auth:sanctum', 'verified'])->post('update_car', [CarController::class, 'update'])->name('update_car');
```

Metoda **update** se nalazi u **CarControlleru**.

Metoda **update** prima request poslan sa POST zahtjevom putem forme i ima oblik:

```
110 public function update(Request $request)
111 {
112     $id = $request->id;
113
114     $request->validate([
115         'name' => 'required|string|max:255',
116         'engine' => 'required|integer',
117     ]);
118
119     $update_query = DB::table('cars')
120     ->where('id', $id)
121     ->update([
122         'name' => $request->name,
123         'year' => $request->year,
124         'engine' => $request->engine,
125         'code' => $request->code,
126         'brand' => $request->brand,
127     ]);
128
129     return redirect()->route('cars');
130 }
```

Dakle, prvo se pokupi id od auta, uradi se validate nekih parametara. Nakon toga se izvrši query update nad tabelom **cars** gdje je id jednak poslanom id-u. Poslije toga redirecta se na rutu **cars**.

Edit forma izgleda ovako:

[LOGO](#) [Početna](#) [Auta](#) [Brendovi](#) Zinaid ▾

Početna-Auta-Uredi

Naziv

Godina

📅

Motor

Sifra

Brend

▾

SPREMI

Vježbe 9 – Dodavanje tabele Vozači i kreiranje REST API-ja

Laravel predstavlja odlično okruženje za brz razvoj api-ja. Danas mnoge web stranice ili aplikacije koriste front-end JavaScript okruženja poput React.js ili Vue.js koji zahtjevaju da pozadina ima neki Restful API. Restful API je potreban i za mobilne aplikacije koje koriste pozadinske API-je za obavljanje radnji u ime korisnika. Restful API-ji se koriste kada treba da obezbjedite neke metode integracije za sisteme neke treće strane.

REST API (poznat kao i RESTful API) je interfejs za programiranje aplikacija (API ili web API) koji u skladu sa ograničenjima REST arhitektonskog stila omogućava interakciju sa RESTful web servisima. REST je skraćenica za prenos stanja reprezentacije. REST je skup arhitektonskih ograničenja, a ne protokol ili standard. API programeri mogu implementirati REST na razne načine.

Osnovne radnje REST api-ja omogućuje HTTP protokol koji ima sljedeće radnje:

GET – preuzimanje resursa

POST – kreiranje resursa

PUT – ažuriranje resursa

DELETE – brisanje resursa

U ovom dijelu pokazat ćemo kako se kreira api za Vozače koristeći Laravel. Cjelokupan CRUD sistem će biti reprezentiran kroz API. Odnosno omogućit će se čitanje svih vozača, pojedinačnog vozača, uređivanje vozača, brisanje vozača i kreiranje novog vozača, a sve to kroz api endpoints. Ujedno ćemo predstaviti jedan alat za testiranje rada kreiranih api-ja, a to je Insomnia (neki od poznatijih alata ovog tipa je popularni Postman).

Prvo ćemo kreirati novi model i migraciju za vozače na sljedeći način:

php artisan make:model Driver –migration

Zatim modificirajmo migraciju i model na sljedeći način:

```
14     public function up()
15     {
16         Schema::create('drivers', function (Blueprint $table) {
17             $table->id();
18             $table->string('name');
19             $table->string('lastname');
20             $table->integer('level')->default(0);
21             $table->string('description')->nullable();
22             $table->timestamps();
23         });
24     }
```

```

app > Models > 🐘 Driver.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Driver extends Model
9  {
10     use HasFactory;
11
12     protected $fillable = [
13         'name',
14         'lastname',
15         'level',
16         'description',
17     ];
18 }
19

```

Sada je potrebno migrirati tabelu u bazu podataka sa poznatom naredbom:

php artisan migrate

Kada kreiramo api naše rute ćemo definirati u fajlu **api.php** koji se isto kao i web.php nalazi u routes. Prije svega na vrh ovog fajla treba definirati putanju modela **Driver**.

```

5  use App\Models\Driver;
6  use App\Http\Controllers\DriverController;

```

Sada ćemo kreirati kontroler **DriverController** ali ćemo na naredbu dodati ekstenziju - - api tako da kreira sve metode koje sadrži neki api (odnosno CRUD radnje). To radimo sljedećom naredbom:

php artisan make:controller DriverController --api

Rute u api.php možemo ručno na sljedeći način:

```
18
19 Route::get('/drivers', 'DriverController@index');
20 Route::post('/drivers', 'DriverController@store');
21 Route::put('/drivers', 'DriverController@update');
22 Route::delete('/drivers', 'DriverController@destroy');
```

Ukoliko pratimo konvenciju imenovanja unutar Laravela sve pobrojane rute možemo zamjeniti sa sljedećim:

```
21 Route::resource('drivers', DriverController::class);
```

Da bi provjerili koje su to sve rute dostupne u našoj aplikaciji potrebno je ukucati naredbu:

php artisan route:list

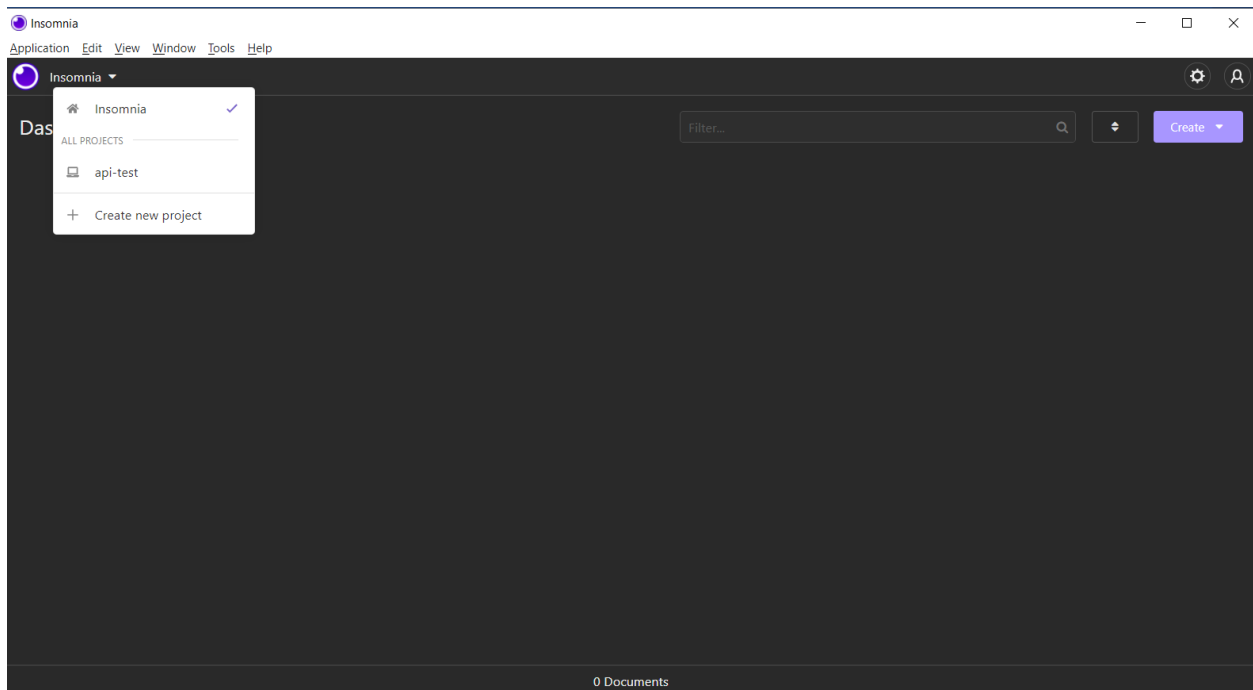
Vidimo da su definisane sve CRUD rute za vozače:

GET HEAD	api/drivers	drivers.index	App\Http\Controllers\DriverController@index
api			
POST	api/drivers	drivers.store	App\Http\Controllers\DriverController@store
api			
GET HEAD	api/drivers/create	drivers.create	App\Http\Controllers\DriverController@create
api			
GET HEAD	api/drivers/{driver}	drivers.show	App\Http\Controllers\DriverController@show
api			
PUT PATCH	api/drivers/{driver}	drivers.update	App\Http\Controllers\DriverController@update
api			
DELETE	api/drivers/{driver}	drivers.destroy	App\Http\Controllers\DriverController@destroy
api			
GET HEAD	api/drivers/{driver}/edit	drivers.edit	App\Http\Controllers\DriverController@edit
api			
GET HEAD	api/user		Closure
api			

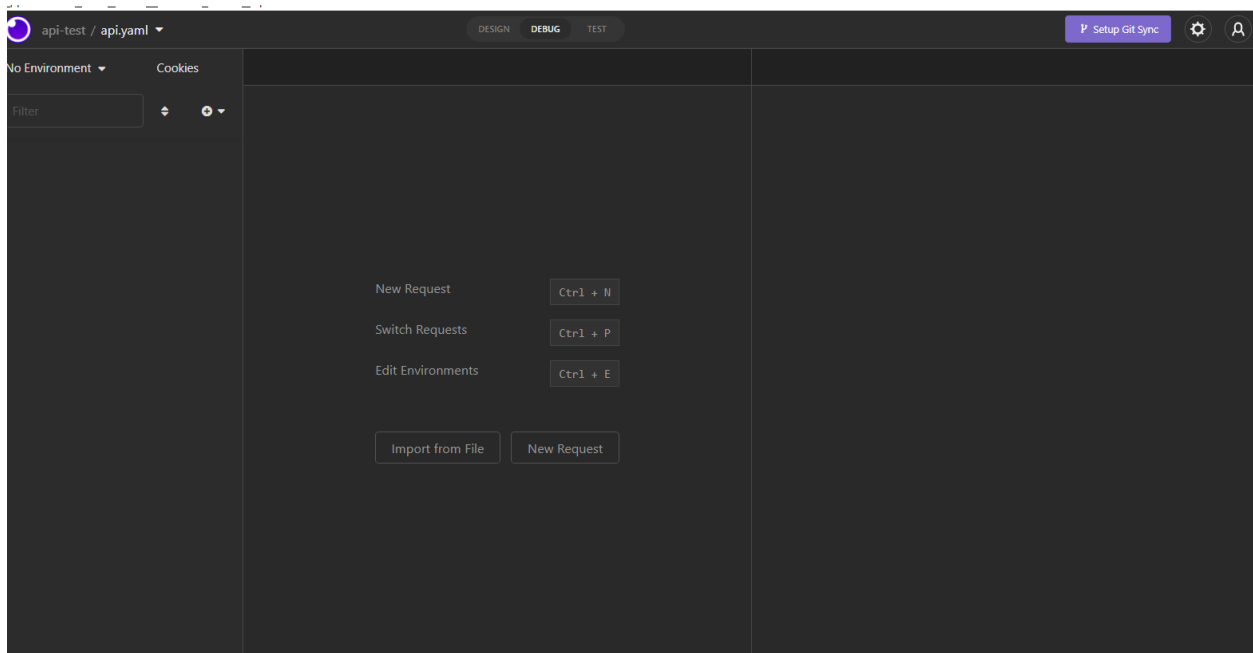
Dakle, slanjem **GET** zahtjeva **api/drivers** Laravel zna da treba pristupiti **DriverControlleru** i metodi **index**. Naprimjer slanjem **DELETE** zahtjeva na **api/drivers/{driver}** gdje **driver** u vitičastim zagradama označava **id** vozača kojeg želimo obrisati Laravel zna da treba pristupiti **DriverControlleru** i metodi **destroy**.

Da bi dalje nastavili razvijati API, moramo instalirati i alat za njegovo testiranje. Neke od opcija su Postman ili Insomnia. U ovom tutorialu koristit ćemo Insomniu koja se može pronaći na sljedećem linku: <https://insomnia.rest/download>

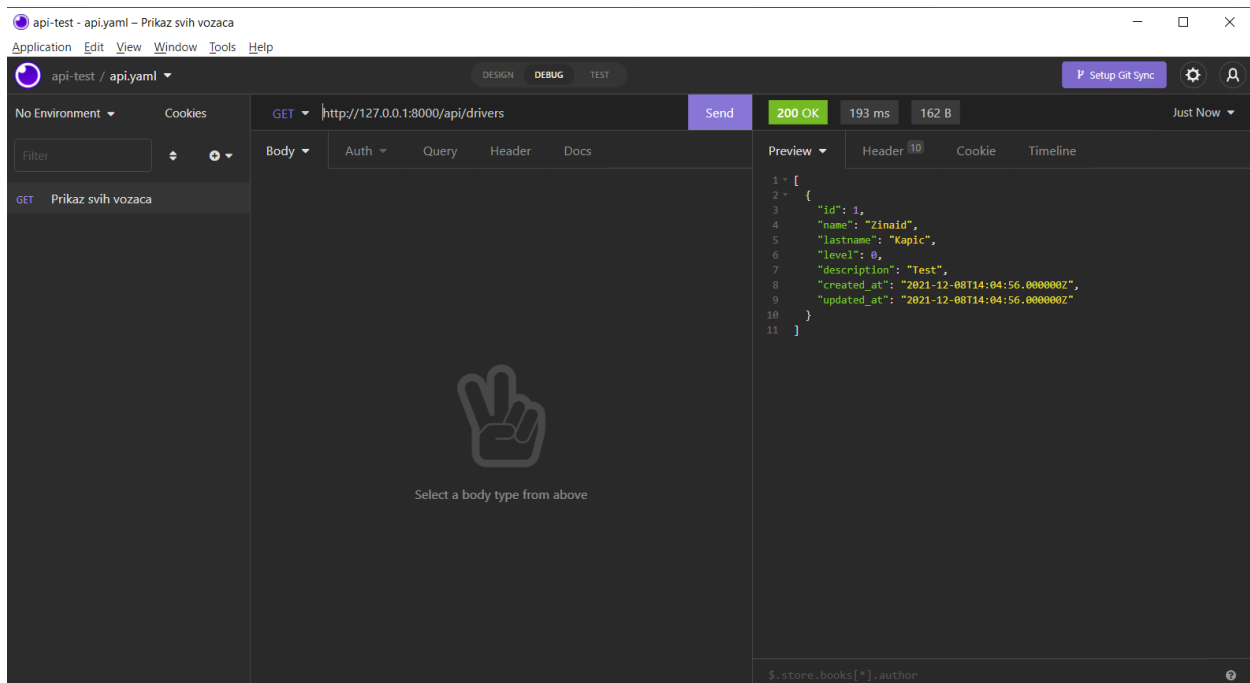
Nakon instalacije Insomnie i njenog pokretanja potrebno je uraditi sljedeće stvari:



Create new project i nazvati ga recimo **api-test**. Zatim na desnoj strani kliknuti na **Create** i kreirati novi document **api.yaml**. Otići na debug koji se nalazi na sredini ekrana.



Sada smo spremni da vršimo zahtjeve prema našem apiju. Prije nastavka ručno u tabelu vozači dodati jednog vozača. Dalje, kliknuti na **New Request** i unijeti ime tog requesta npr. Prikaz svih vozaca. Odabrati da to bude **GET** zahtjev. U polje za unos URL-a unijeti adresu našeg apija a to je: **http://127.0.0.1:8000/api/drivers**



Klikom na **Send** uputit će se **GET** zahtjev na rutu `drivers`, a to znači da će ući u **DriverController** i na metodu **index**. Dakle, treba na metodi `index` dodati kod za ispis svih vozača. To se radi na sljedeći način (Paziti da se u vrhu fajla **DriverController** doda pristup **Driver** model na način `use App\Models\Driver`).

Vidi sljedeću sliku:

```

app > Http > Controllers > DriverController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use App\Models\Driver;
7
8  class DriverController extends Controller
9  {
10     /**
11      * Display a listing of the resource.
12      *
13      * @return \Illuminate\Http\Response
14      */
15     public function index()
16     {
17         return Driver::all();
18     }

```

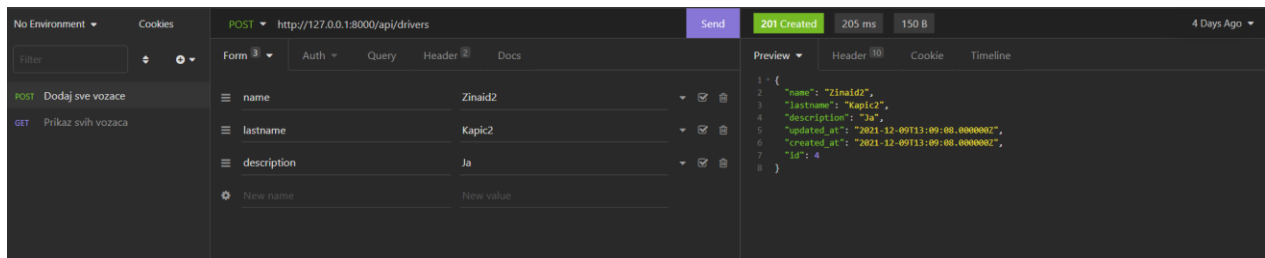
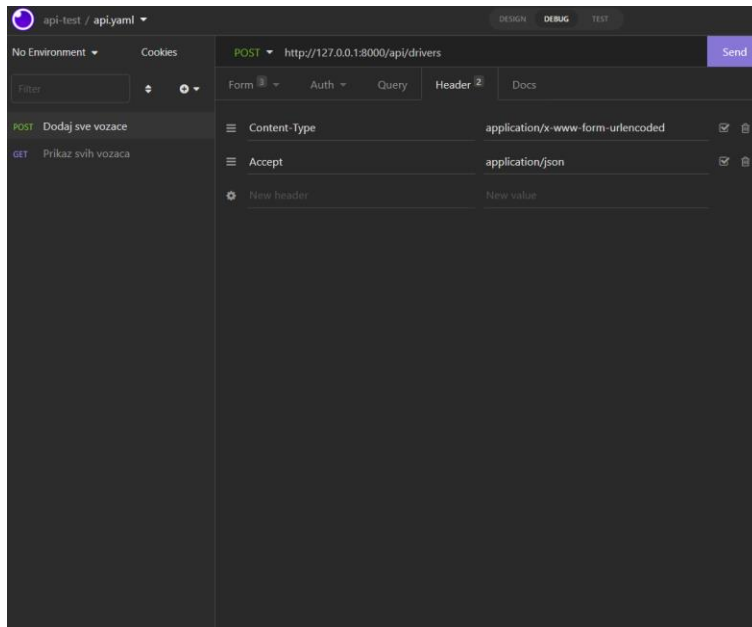
I sad klikom na *Send* u Insomniji ispisat će se vozači koje ste unijeli.

Idemo sad dodati sve ostale CRUD operacije u **DriverController**.

```
20  /**
21   * Store a newly created resource in storage.
22   *
23   * @param \Illuminate\Http\Request $request
24   * @return \Illuminate\Http\Response
25   */
26  public function store(Request $request)
27  {
28      return Driver::create($request->all());
29  }
30
31  /**
32   * Display the specified resource.
33   *
34   * @param int $id
35   * @return \Illuminate\Http\Response
36   */
37  public function show($id)
38  {
39      return Driver::find($id);
40  }
41
42  /**
43   * Update the specified resource in storage.
44   *
45   * @param \Illuminate\Http\Request $request
46   * @param int $id
47   * @return \Illuminate\Http\Response
48   */
49  public function update(Request $request, $id)
50  {
51      $driver = Driver::find($id);
52      $driver->update($request->all());
53      return $driver;
54  }
```

```
56
57  /**
58   * Remove the specified resource from storage.
59   *
60   * @param int $id
61   * @return \Illuminate\Http\Response
62   */
63  public function destroy($id)
64  {
65      return Driver::destroy($id);
66  }
67  }
68
```

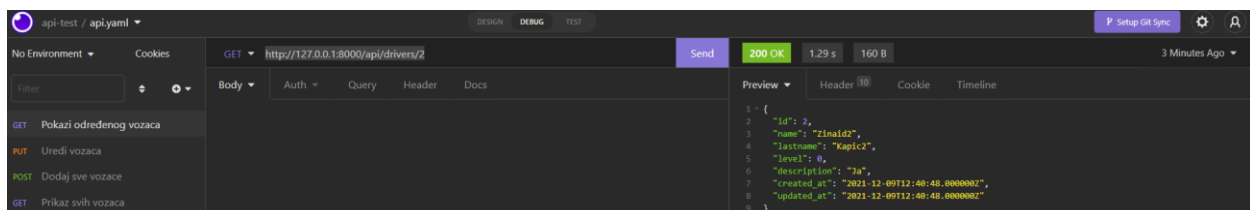

Sada kreirajmo novi zahtjev ***Dodaj novog vozača*** koji radi ***POST*** na isti link na koji smo radili i prethodni zahtjev. Ovdje želimo dodati novog vozača tako da moramo izmijeniti **Header** i **Body** zahtjeva jer ustvari mi šaljem neke podatke koji će se spremi u tabelu **drivers**.



Testirajmo sada prikaz samo određenog vozača. To radimo tako što upućujemo **GET** zahtjev na rutu koju smo i dosad koristili samo što dodamo na kraju i id od vozača.

http://127.0.0.1:8000/api/drivers/2

Dakle, iz ovog zahtjeva pristupit će se DriverControlleru i metodi **show** koja vraća tačno traženog vozača na osnovu upita: ***Driver::find(\$id)***.



Sada ćemo testirati rad uređivanja vozača.

Dodajmo novi request koji će biti **PUT** i koji će ići na istu rutu

http://127.0.0.1:8000/api/drivers

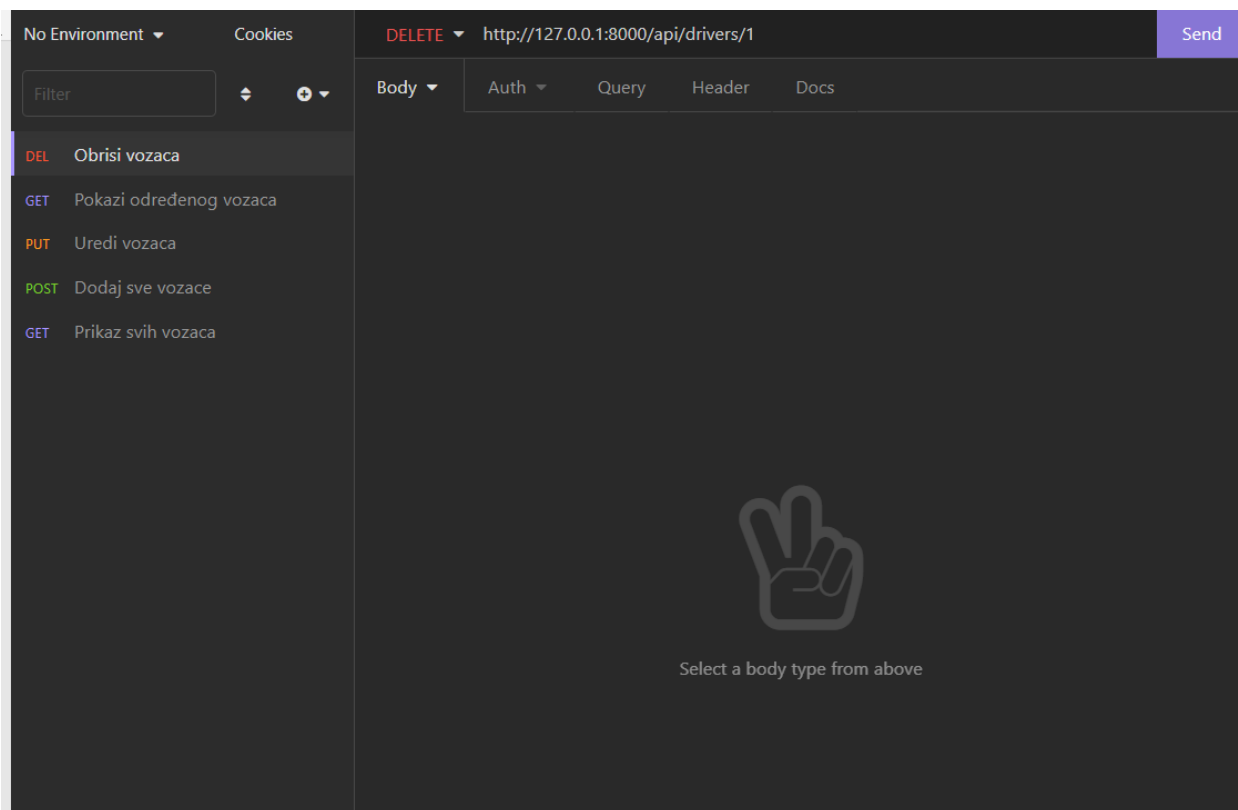
The screenshot shows the 'api-test / api.yaml' interface. The 'PUT' method is selected for the endpoint 'http://127.0.0.1:8000/api/drivers/2'. The 'Header' tab is active, showing two headers: 'Content-Type' with value 'application/x-www-form-urlencoded' and 'Accept' with value 'application/json'. A 'Send' button is visible in the top right corner. On the left, a list of requests is shown, with 'Uredi vozaca' (PUT) selected.

The screenshot shows the 'api-test / api.yaml' interface after the request has been sent. The status is '200 OK' with a response time of '208 ms' and a body size of '167 B'. The 'Preview' tab is active, displaying the JSON response:

```
1 {
2   "id": 2,
3   "name": "Zinaid new",
4   "lastname": "Kapic2",
5   "level": 0,
6   "description": "Tester",
7   "created_at": "2021-12-09T12:40:48.000000Z",
8   "updated_at": "2021-12-13T15:36:45.000000Z"
9 }
```

 The 'Header' tab is also visible, showing the 'Content-Type' header. On the left, the 'Uredi vozaca' (PUT) request is still selected.

Posljednja opcija koja je ostala je brisanje Vozača. Dakle kreiramo **DELETE** request na Insomniji i proslijedimo id vozača kojeg želimo obrisati.



Vježbe 10 – Popunjavanje tabele testnim (dummy) podacima koristeći Laravel Factories i Seeders

Prilikom razvoja, a naročito testiranja određenih funkcionalnosti, često budemo u potrebi za brzim popunjavanjem tabela sa testnim podacima (tzv. Dummy data). Unutar Laravela to nam omogućuje Laravel Factory i Laravel Seeders.

Prije svega kreira se Laravel Factory naredbom:

php artisan make:factory CarFactory --model=Car

Unutar database>factories je kreiran novi fajl **CarFactory.php**. Taj fajl ćemo izmjeniti na sljedeći način:

```
database > factories > 🐞 CarFactory.php
1  <?php
2
3  namespace Database\Factories;
4  use App\Models\Car;
5
6  use Illuminate\Database\Eloquent\Factories\Factory;
7
8  class CarFactory extends Factory
9  {
10     /**
11      * The name of the factory's corresponding model.
12      *
13      * @var string
14      */
15
16     protected $model = Car::class;
17
18     /**
19      * Define the model's default state.
20      *
21      * @return array
22      */
23
24     public function definition()
25     {
26         return [
27             'name' => $this->faker->name(),
28             'year' => now(),
29             'engine' => $this->faker->randomDigit(),
30             'code' => $this->faker->unique()->numberBetween(1,1000),
31             'air_condition' => $this->faker->numberBetween(0,1),
32             'brand' => $this->faker->numberBetween(1,3),
33         ];
34     }
35 }
```

Dalje trebamo napraviti Seeder fajl naredbom:

php artisan make:seed CarTableSeeder

koja kreira fajl u database>seeders.

```
database > seeders > CarTableSeeder.php
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use App\Models\Car;
7
8  class CarTableSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         Car::factory()->count(10)->create();
18     }
19 }
```

Dummy data se šalje naredbom:

php artisan db:seed --class=CarTableSeeder

Sada je tabela Cars popunjena sa random dummy podacima definiranim u CarFactory.php.

Sage Abbott - 2021-12-13	OBRISI	UREDJI
Bertha Satterfield DVM - 2021-12-13	OBRISI	UREDJI
Francesco Lang - 2021-12-13	OBRISI	UREDJI
Andrew Huel - 2021-12-13	OBRISI	UREDJI
Lydia Sipes - 2021-12-13	OBRISI	UREDJI
Lucius Huel IV - 2021-12-13	OBRISI	UREDJI
Mr. Bernardo Bergnaum MD - 2021-12-13	OBRISI	UREDJI
Tyreek Runolfsdottir V - 2021-12-13	OBRISI	UREDJI
Gudrun Emard - 2021-12-13	OBRISI	UREDJI
Mr. Leif Hane - 2021-12-13	OBRISI	UREDJI
Prof. Mohammad Breitenberg - 2021-12-13	OBRISI	UREDJI
Anabel Swaniawski - 2021-12-13	OBRISI	UREDJI

Vježbe 11/12 – Povezivanje tabela Vozaci i Auta kroz tabelu Voznja i vršenje složenijih upita

Na isti način kao i prethodne dodat ćemo tabelu **rides**

php artisan make:model -mcr Ride

Dakle, dodali smo model Ride, kontroler RideController i migraciju za kreiranje tabele rides.

Sad ćemo izmijeniti model i migraciju da odgovaraju sljedećoj tabeli:

RIDES

#id

*** code (integer) unique**

*** date (date)**

*** grade (double)**

o description (text)

*** driver (integer) FK**

*** car (integer) FK**

Kao u prethodnim vježbama izmjenit ćemo migraciju i model na način da dodamo sve željene attribute i njihove tipove. Model Drive izgleda kao:

```
app > Models > Ride.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Ride extends Model
9  {
10     public $timestamps = false;
11
12     use HasFactory;
13
14     /**
15      * The attributes that are mass assignable.
16      *
17      * @var string[]
18      */
19     protected $fillable = [
20         'code',
21         'date',
22         'grade',
23         'description',
24         'driver',
25         'car',
26     ];
27 }
```

Migracija izgleda kao:

```
14     public function up()
15     {
16         Schema::create('rides', function (Blueprint $table) {
17             $table->id();
18             $table->integer('code')->unique();
19             $table->datetime('date');
20             $table->integer('grade');
21             $table->text('description')->nullable();
22             $table->integer('driver');
23             $table->integer('car');
24         });
25     }
```

Sada ćemo kreirati **RideFactory** i **Seeder** kako bi mogli tabelu popuniti novim podacima koje ćemo koristiti za testiranje.

php artisan make:factory RideFactory --model=Ride

Pored RideFactory kreiramo i Seeder naredbom:

php artisan make:seed RideTableSeeder

RideTableSeeder koji se nalazi u fajlu database>seeders se izmjeni kao:

```
database > seeders > RideTableSeeder.php
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use App\Models\Ride;
7
8  class RideTableSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         Ride::factory()->count(10)->create();
18     }
19 }
20
```

RideFactory ćemo izmjeniti kao:

```
database > factories > RideFactory.php
1  <?php
2
3  namespace Database\Factories;
4
5  use Illuminate\Database\Eloquent\Factories\Factory;
6  use App\Models\Ride;
7  use Carbon\Carbon;
8
9  class RideFactory extends Factory
10 {
11     protected $model = Ride::class;
12
13     /**
14      * Define the model's default state.
15      *
16      * @return array
17      */
18     public function definition()
19     {
20         return [
21             'code' => $this->faker->unique()->numberBetween(1,1000),
22             'date' => Carbon::now(),
23             'grade' => $this->faker->numberBetween(1,5),
24             'description' => 'Obicni opis',
25             'driver' => $this->faker->numberBetween(1,10),
26             'car' => $this->faker->numberBetween(1,10),
27         ];
28     }
29 }
```

Kako bi što bolje odradili testiranje i imali mogućnost za više upita. Uzet ćemo i kreirati factory i seedere i za vozače i brendove obzirom da imamo sad samo za auta i za vožnju.

Factory za brendove se kreira sa naredbom:

php artisan make:factory BrandFactory --model=Brand

Seeder se pravi naredbom:

php artisan make:seed BrandTableSeeder

Factory za brendove se izmjeni na sljedeći način:

```
database > factories > BrandFactory.php
1  <?php
2
3  namespace Database\Factories;
4
5  use Illuminate\Database\Eloquent\Factories\Factory;
6  use App\Models\Brand;
7
8  class BrandFactory extends Factory
9  {
10     protected $model = Brand::class;
11
12     /**
13      * Define the model's default state.
14      *
15      * @return array
16      */
17     public function definition()
18     {
19         return [
20             'name' => $this->faker->randomElement(['Mercedes', 'Audi', 'Citroen']),
21             'country' => $this->faker->randomElement(['DE', 'FR', 'AU', 'BA']),
22         ];
23     }
24 }
```

Seeder za brendove je sljedeći:

```
database > seeders > BrandTableSeeder.php
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use App\Models\Brand;
7
8  class BrandTableSeeder extends Seeder
9  {
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         Brand::factory()->count(3)->create();
18     }
19 }
```

Factory za vozače je sljedeći:

```
database > factories > DriverFactory.php
1  <?php
2
3  namespace Database\Factories;
4
5  use Illuminate\Database\Eloquent\Factories\Factory;
6  use App\Models\Driver;
7
8  class DriverFactory extends Factory
9  {
10
11     protected $model = Driver::class;
12
13     /**
14      * Define the model's default state.
15      *
16      * @return array
17      */
18     public function definition()
19     {
20         return [
21             'name' => $this->faker->name(),
22             'lastname' => $this->faker->lastname(),
23             'level' => $this->faker->numberBetween(1,5),
24             'description' => 'Obicni opis vozaca',
25         ];
26     }
27 }
```

Seeder za vozače je sljedeći:

```
database > seeders > DriverTableSeeder.php
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use App\Models\Driver;
7  class DriverTableSeeder extends Seeder
8  {
9
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         Driver::factory()->count(10)->create();
18     }
19 }
```

Factory za auta ćemo izmijeniti tako da name auta reprezentuje neko moguće ime, a to je riješeno na sljedeći način (samo se 'name' zamjeni sa random elementom iz nekog niza (npr. Malo Auto, Limuzina...)).

```
23
24     public function definition()
25     {
26         return [
27             'name' => $this->faker->randomElement(['Malo Auto', 'Limuzina', 'Karavan', 'Dzip']),
28             'year' => now(),
29             'engine' => $this->faker->randomDigit(),
30             'code' => $this->faker->unique()->numberBetween(1,1000),
31             'air_condition' => $this->faker->numberBetween(0,1),
32             'brand' => $this->faker->numberBetween(1,3),
33         ];
34     }
```

Sada ćemo odraditi rollback svih migracija i resetirati bazu, te je ponovno migrirati i poslati Seedere za sve tabele, tako da imamo lijepo posložene dummy podatke.

php artisan migrate:rollback --step=10

Ponovno ćemo odraditi migraciju:

php artisan migrate

Sada se idemo ponovno registrirati i logirati na sistem. Zatim ćemo poslati sve seedere u tabele.

php artisan db:seed --class=BrandTableSeeder

php artisan db:seed --class=CarTableSeeder

php artisan db:seed --class=DriverTableSeeder

php artisan db:seed --class=RideTableSeeder

Sada su sve tabele popunjene sa dummy podacima i možemo kreirati novi view za vožnje i odgovarajuću rutu sa kojom ćemo prosljeđivati rezultate određenih upita. Prije svega kreirajmo novi folder unutar views-a naziva **rides**. Tu kreirajmo fajl **index.blade.php** i u njega kopirajmo sve iz index.blade.php koji se nalazi u folderu **brands**, izbaciti sve što uključuje **foreach** petlju.

```
resources > views > rides > index.blade.php
1  <x-app-layout>
2      <x-slot name="header">
3          <h2 class="font-semibold text-xl text-gray-800 leading-tight">
4              {{ __('Početna-Auta') }}
5          </h2>
6      </x-slot>
7
8      <div class="py-12">
9          <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10             <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11                 <div class="p-2">
12
13                 </div>
14             </div>
15         </div>
16     </div>
17 </x-app-layout>
```

Zatim u *routes>web.php* dodajmo rutu koja vodi na kontroler **RideController.php** i metodu **index**. Paziti da se na vrhu fajla uključi *RideController.php*.

U *navigation-menu.blade.php* dodati novi nav-link sa rutom **rides** i naziva Voznje.

Sada ćemo postaviti nekoliko upita koje želimo ispisati i koje ćemo vraćati na *index.blade.php* od vožnji.

UPIT 1:

Ispisati auta koja su se najčešće testirala u vožnji.

UPIT 2:

Ispisati države čija auta su se najčešće testirala u vožnji.

UPIT 3:

Ispisati broj vožnji odrađenih u periodu od 1.12.2021 do 31.12.2021

UPIT 4:

Ispisati imena i prezimena vozača, naziv auta njemačke proizvodnje koji su voženi u periodu od 10.12.2021 do 31.12.2021.

Sve te informacije vratiti na *index.blade.php* od vožnji iz metode **index**.

```
9      <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10      <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11      <div class="grid grid-cols-4 gap-4 p-4 justify-items-center">
12      <div>
13          <h1>Upit 1.</h1>
14          <hr/>
15          <p>1. ALa ALa</p>
16      </div>
17      <div>
18          <h1>Upit 2.</h1>
19          <hr/>
20          <p>1. ALa ALa</p>
21      </div>
22      <div>
23          <h1>Upit 3.</h1>
24          <hr/>
25          <p>1. ALa ALa</p>
26      </div>
27      <div>
28          <h1>Upit 4.</h1>
29          <hr/>
30          <p>1. ALa ALa</p>
31      </div>
32      </div>
33      </div>
34  </div>
```

Obzirom da se radi o četiri upita izmjenit ćemo i **index.blade.php** od vožnji, na način da dodamo četiri **div** elementa u koje ćemo ispisivati rezultate upita. To ćemo uraditi na sljedeći način:

Upit 1.

```
// Ispisati auta koja su se najčešće testirala u vožnji.  
  
$most_common_cars = DB::table('cars')  
    ->select('cars.*', DB::raw('count(*) as brojac'))  
    ->groupBy('cars.id')  
    ->join('rides', 'cars.id', '=', 'rides.car')  
    ->orderByRaw('COUNT(*) DESC')  
    ->get();
```

Upit 2.

```
27 // Ispisati države čija auta su se najčešće testirala u vožnji.  
28  
29 $most_common_car_countries = DB::table('cars')  
30     ->select('brands.*', DB::raw('count(*) as brojac'))  
31     ->groupBy('brands.id')  
32     ->join('rides', 'cars.id', '=', 'rides.car')  
33     ->join('brands', 'cars.brand', '=', 'brands.id')  
34     ->orderByRaw('COUNT(*) DESC')  
35     ->get();
```

Upit 3.

```
37 // Ispisati broj vožnji odradenih u periodu od 1.12.2021 do 31.12.2021  
38  
39 $from = '2021-12-01 00:00:00';  
40 $to = '2021-12-31 23:59:59';  
41  
42 $number_of_rides = DB::table('rides')  
43     ->whereBetween('date', [$from, $to])  
44     ->count();
```

Upit 4.

```
47  
48 $german_car_drivers = DB::table('drivers')  
49     ->select('drivers.name as driver_name', 'drivers.lastname as driver_lastname')  
50     ->groupBy('drivers.id')  
51     ->join('rides', 'drivers.id', '=', 'rides.driver')  
52     ->join('cars', 'rides.car', '=', 'cars.id')  
53     ->join('brands', 'cars.brand', '=', 'brands.id')  
54     ->where('brands.country', 'DE')  
55     ->whereBetween('date', [$from, $to])  
56     ->get();
```

To sve vraćamo sa

```
58     return view('rides.index',
59         ['most_common_cars' => $most_common_cars,
60         'most_common_car_countries' => $most_common_car_countries,
61         'number_of_rides' => $number_of_rides,
62         'german_car_drivers' => $german_car_drivers
63     ]);
```

Podatke ispisujemo na **index.blade.php**:

```
11 <div class="grid grid-cols-4 gap-4 p-4 justify-items-center">
12 <div>
13     <h1>Upit 1.</h1>
14     <hr/>
15     @foreach($most_common_cars as $most_common_car)
16     <p>{{ $loop->iteration }}. {{ $most_common_car->name }} - {{ $most_common_car->brojac }}</p>
17     @endforeach
18 </div>
19 <div>
20     <h1>Upit 2.</h1>
21     <hr/>
22     @foreach($most_common_car_countries as $most_common_car_country)
23     <p>{{ $loop->iteration }}. {{ $most_common_car_country->country }} - {{ $most_common_car_country->brojac }}</p>
24     @endforeach
25 </div>
26 <div>
27     <h1>Upit 3.</h1>
28     <hr/>
29     <p>{{ $number_of_rides }}</p>
30 </div>
31 <div>
32     <h1>Upit 4.</h1>
33     <hr/>
34     @foreach($german_car_drivers as $german_car_driver)
35     <p>{{ $loop->iteration }}. {{ $german_car_driver->driver_name }} - {{ $german_car_driver->driver_lastname }}</p>
36     @endforeach
37 </div>
38 </div>
```

LOGO

Početna

Auta

Brendovi

Voznje

Zinaid ▾

Početna-Voznja

Upit 1.

1. Dzip - 3
2. Limuzina - 3
3. Limuzina - 2
4. Malo Auto - 1
5. Dzip - 1

Upit 2.

1. BA - 4
2. DE - 4
3. FR - 2

Upit 3.

10

Upit 4.

1. Ezekiel Fay DDS - Hessel
2. Jarrod Russel - Will
3. Juanita Murphy - Halvorson

Vježbe 13 – Rad sa fajlovima koristeći Laravel Storage

Laravel pruža moćnu apstrakciju sistema datoteka zahvaljujući Flysystem PHP paketu. Integracija laravel Flysystem daje jedinstavne drajvere za rad sa lokalnim sistemom datoteka, SFTP-om i Amazon S3. Jednostavno je prebacivanje između ovih opcija pohrane.

Laravelov konfiguracioni fajl za sistem datoteka se nalazi u folderu **config/filesystems.php**. Unutar ove datoteke se mogu konfigurirati svi „diskovi“ našeg sistema datoteka. Svaki disk predstavlja određeni upravljački program za spremanje i mjesto spremanja. Lokalni upravljački program stupa u interakciju sa datotekama pohranjenim lokalno na poslužitelju koji pokreće Laravel aplikaciju, dok S3 upravljački program se koristi za pisanje u Amazonovu S3 uslugu pohrane u cloudu.

Kreirajmo novi model, kontroler i migraciju za fajlove naziva **CarFiles**, a koja će se koristiti za unos fajlova vezanih za auto na sistem, a spremat će fajlove kao što su png, jpg, pdf, doc itd.

Prije svega odradimo naredbu:

php artisan make:model -mcr CarFile

Izmjenit ćemo migraciju i model na sljedeći način:

```
app > Models > CarFile.php
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class CarFile extends Model
9  {
10     public $timestamps = false;
11
12     use HasFactory;
13
14     /**
15      * The attributes that are mass assignable.
16      *
17      * @var string[]
18      */
19     protected $fillable = [
20         'car',
21         'type',
22         'file',
23     ];
24 }
```

```
14     public function up()
15     {
16         Schema::create('car_files', function (Blueprint $table) {
17             $table->id();
18             $table->integer('car');
19             $table->integer('type');
20             $table->string('file');
21         });
22     }
23 }
```

Zatim to migriramo u bazu i napravimo novi view u folderu **cars** naziva **file_add.blade.php**.

Na fajlu **cars>index.blade.php** dodati novo dugme koje vodi na ovaj novi fajl **file_add.blade.php** i šalje **id** odabranog auta.

```
40 <div class="flex-1">
41 <form method="POST" action="{{ route('file_add') }}">
42 @csrf
43 <input type="hidden" name="id" value="{{ $car->id }}">
44 <div class="p-2">
45 <button class="ml-4 inline-flex items-center px-4 py-2 bg-green-700 border border-transparent rounded-md
46 font-semibold text-xs text-white uppercase">
47 {{ __('Dodaj fajl') }}
48 </button>
49 </div>
50 </form>
51 </div>
```

Potrebno je definirati rutu **file_add** u **web.php**.

```
29 Route::middleware(['auth:sanctum', 'verified'])->get('cars', [CarController::class, 'index'])->name('cars');
30 Route::middleware(['auth:sanctum', 'verified'])->get('add_car', [CarController::class, 'create'])->name('add_car');
31 Route::middleware(['auth:sanctum', 'verified'])->post('store_car', [CarController::class, 'store'])->name('store_car');
32 Route::middleware(['auth:sanctum', 'verified'])->post('delete_car', [CarController::class, 'delete'])->name('delete_car');
33 Route::middleware(['auth:sanctum', 'verified'])->post('edit_car', [CarController::class, 'edit'])->name('edit_car');
34 Route::middleware(['auth:sanctum', 'verified'])->post('update_car', [CarController::class, 'update'])->name('update_car');
35 Route::middleware(['auth:sanctum', 'verified'])->post('file_add', [CarController::class, 'file_add'])->name('file_add');
36
37 Route::middleware(['auth:sanctum', 'verified'])->get('brands', [BrandController::class, 'index'])->name('brands');
38
39 Route::middleware(['auth:sanctum', 'verified'])->get('rides', [RideController::class, 'index'])->name('rides');
```

Sada u **CarController** dodati metodu **file_add** u kojoj ćemo pokupiti id poslanog auta i proslijediti ga na view **file_add.blade.php**.

Dodaj auto

Ovdje će biti izlistana auta			
Dzip - 2021-12-14	OBRISI	URED	DODAJ FAIL
Limuzina - 2021-12-14	OBRISI	URED	DODAJ FAIL
Malo Auto - 2021-12-14	OBRISI	URED	DODAJ FAIL
Limuzina - 2021-12-14	OBRISI	URED	DODAJ FAIL
Dzip - 2021-12-14	OBRISI	URED	DODAJ FAIL
Dzip - 2021-12-14	OBRISI	URED	DODAJ FAIL
Dzip - 2021-12-14	OBRISI	URED	DODAJ FAIL
Dzip - 2021-12-14	OBRISI	URED	DODAJ FAIL
Limuzina - 2021-12-14	OBRISI	URED	DODAJ FAIL
Malo Auto - 2021-12-14	OBRISI	URED	DODAJ FAIL

Fajl `file_add.blade.php` urediti na sljedeći način:

```
1 <x-app-layout>
2   <x-slot name="header">
3     <h2 class="font-semibold text-xl text-gray-800 leading-tight">
4       {{ __('Početna-Auta-Dodaj fajl') }}
5     </h2>
6   </x-slot>
7
8   <div class="py-12">
9     <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
10      <div class="bg-white overflow-hidden shadow-xl sm:rounded-lg">
11        <div class="p-2">
12          <form action="/process" enctype="multipart/form-data" method="POST">
13            @csrf
14            <input type="hidden" value="{{ $id }}" name="id">
15            <p>
16              <label for="file">
17                <input type="file" name="file" id="file">
18              </label>
19            </p>
20            <button class="ml-4 inline-flex items-center px-4 py-2 bg-green-700 border border-transparent rounded-md font-semibold text-xs text-white uppercase float-right m-2">Upload
21          </form>
22        </div>
23      </div>
24    </div>
25  </div>
26</x-app-layout>
```

Dodati rutu **process** u `web.php`:

```
34 Route::middleware(['auth:sanctum', 'verified'])->post('update_car', [CarController::class, 'update'])->name('update_car');
35 Route::middleware(['auth:sanctum', 'verified'])->post('file_add', [CarController::class, 'file_add'])->name('file_add');
36 Route::middleware(['auth:sanctum', 'verified'])->post('process', [CarController::class, 'process'])->name('process');
```

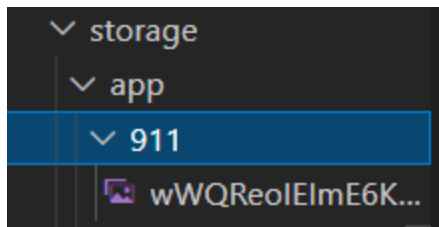
Dodati metodu `process` u `CarController.php`:

```
35 public function process(Request $request)
36 {
37     $id=$request->id;
38
39     $cars = Car::find($id);
40
41     $folder_to_save = $cars->code;
42
43     $path = $request->file('file')->store($folder_to_save);
44
45     return redirect()->route('cars');
46 }
47 }
```

Dakle u ovoj metodi kupimo poslani **id**, na osnovu njega dobijamo informacije o autu preko Laravel Eloquent upita. Deklariramo varijablu `$folder_to_save` koja ima vrijednost unikatnog koda auta. Zatim ugrađena funkcija `store` sprema pokupljeni fajl iz poslanske forme i sprema ga u folder `$folder_to_save`.

Sad se taj fajl nalazi u folderu:

Storage>app>“code“



Vidimo da je fajl spremljen pod random kreiranim imenom. To ime možemo i mi odabrati na sljedeći način:

```
35     public function process(Request $request)
36     {
37         $id=$request->id;
38
39         $cars = Car::find($id);
40
41         $folder_to_save = $cars->code;
42
43         $file = $request->file('file');
44
45         $filename = $cars->id . time() . '.' . $file->getClientOriginalExtension();
46
47         $path = $file->storeAs($folder_to_save, $filename);
48
49         return redirect()->route('cars');
50
51     }
```

Dakle, u isti folder koji odgovara code-u auta spremamo poslani fajl koji je privremeno spremljen u varijablu \$file, a naziv fajla \$filename sačinjava **id** odabranog auta + vrijeme + ekstenzija poslanog fajla. Funkcija storeAs dozvoljava dodjelu naziva i foldera u koji spremamo fajl.

Poslani fajl možemo i validirati tako što koristimo poznatu funkciju validate.

npr.

```
$request->validate([  
  
    'photo' => 'required|file|image|mimes:jpeg,png,gif,webp|max:2048'  
  
]);
```

Postoji ogroman broj funkcija koje su omogućene unutar Laravela, a naročito unutar Laravelovog Facade Storaža.

use Illuminate\Support\Facades\Storage;

Lista nekih od ključnih funkcija su sljedeće:

Odabir diska za upload fajla. Ako disk nije specificiran, Laravel unutar filesystems.php pronalazi koji je to defaultni disk:

Storage::disk('local')->exists('file.txt');

Kreiranje novog fajla sa sadržajem:

Storage::put('file.txt', 'Contents');

Prepend nekom fajlu:

Storage::prepend('file.txt', 'Prepended text');

Append nekom fajlu:

Storage::append('file.txt', 'Appended text');

Uzmi sadržaj fajla:

Storage::get('file.txt');

Provjeri da li postoji fajl:

Storage::exists('file.txt');

Download fajla:

Storage::download('file.txt', \$name, \$headers); \$name i \$headers su opcionalni

Generiši javni dostupan URL:

Storage::url('file.txt');

Generiši privremeni javni URL:

Storage::temporaryUrl('file.txt', now()->addMinutes(10));

Dobij veličinu fajla:

Storage::size('file.txt');

Dobij zadnji datum modificiranja:

Storage::lastModified('file.txt');

Kopiraj fajl:

Storage::copy('file.txt', 'shared/file.txt');

Premjesti fajl:

Storage::move('file.txt', 'secret/file.txt');

Obriši fajl:

Storage::delete('file.txt');

Obriši više fajlova:

Storage::delete(['file1.txt', 'file2.txt']);

Vježbe 14 – Postavljanje Laravel aplikacije na Github koristeći Git

Prije svega potrebno je instalirati **Git**.

<https://git-scm.com/downloads>

Zatim je potrebno napraviti račun na platformi Github


<https://github.com/>

Kreirati repositorij:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner *

 zinaid ▾

Repository name *

/

Great repository names are short and memorable. Need inspiration? How about [silver-broccoli](#)?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

Otvoriti git bash terminal u folderu Laravel projekta:



```
MINGW64:/c/Users/WWTDev - Zinaid/Desktop/AutoShop
WWTDev - Zinaid@DESKTOP-JGT060C MINGW64 ~
$ cd Desktop
WWTDev - Zinaid@DESKTOP-JGT060C MINGW64 ~/Desktop
$ cd AutoShop
WWTDev - Zinaid@DESKTOP-JGT060C MINGW64 ~/Desktop/AutoShop
$ |
```

git init

git add .

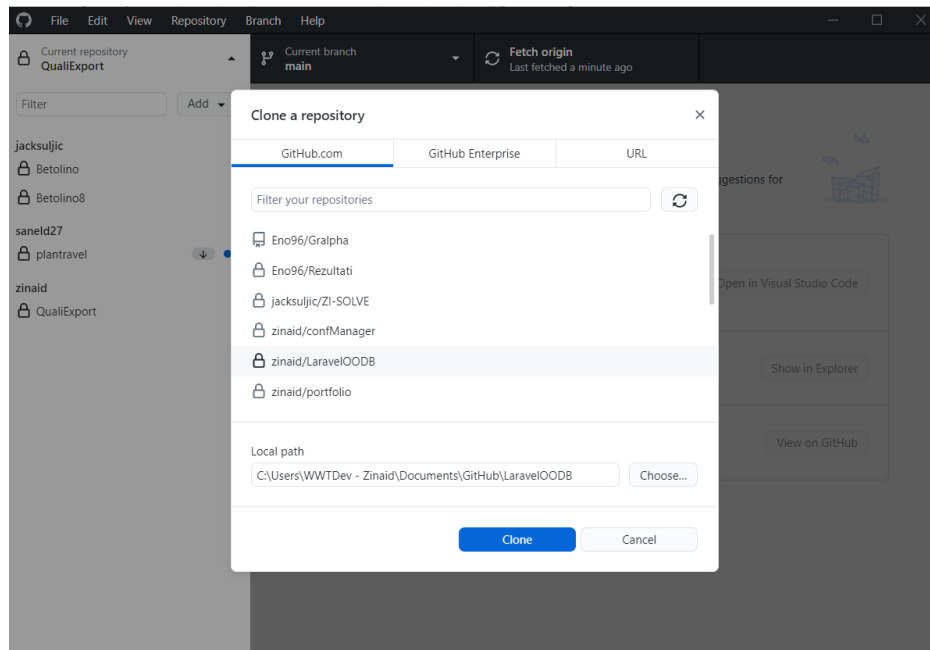
git commit -m 'Comment'

git remote add origin <https://github.com/zinaid/LaravelOODB>

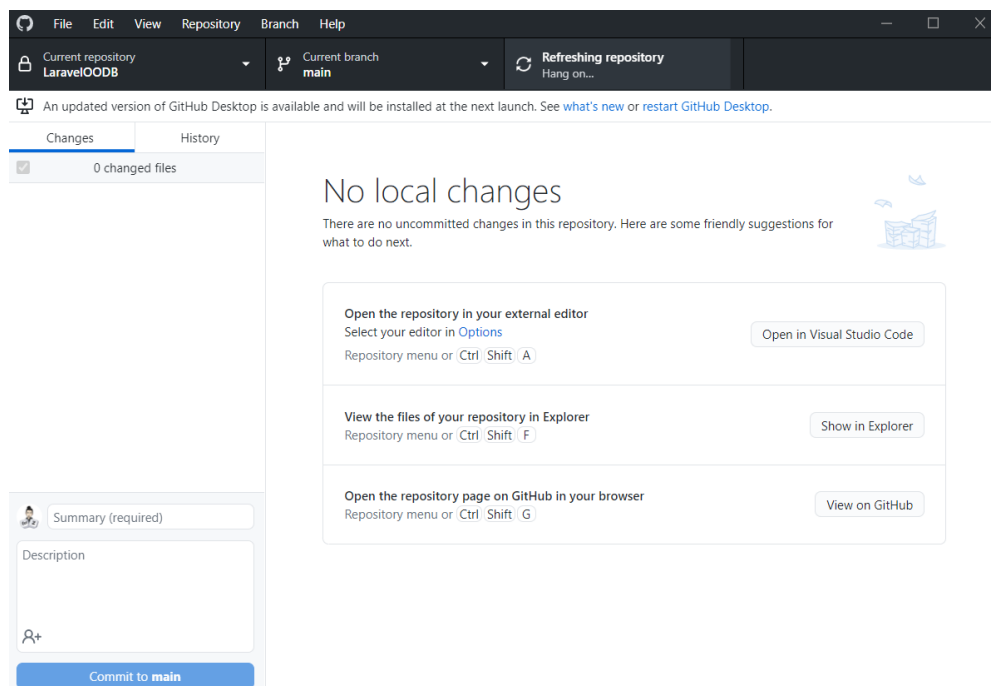
git push -u -f origin master

Na ovaj način je uploadan naš projekat na github.

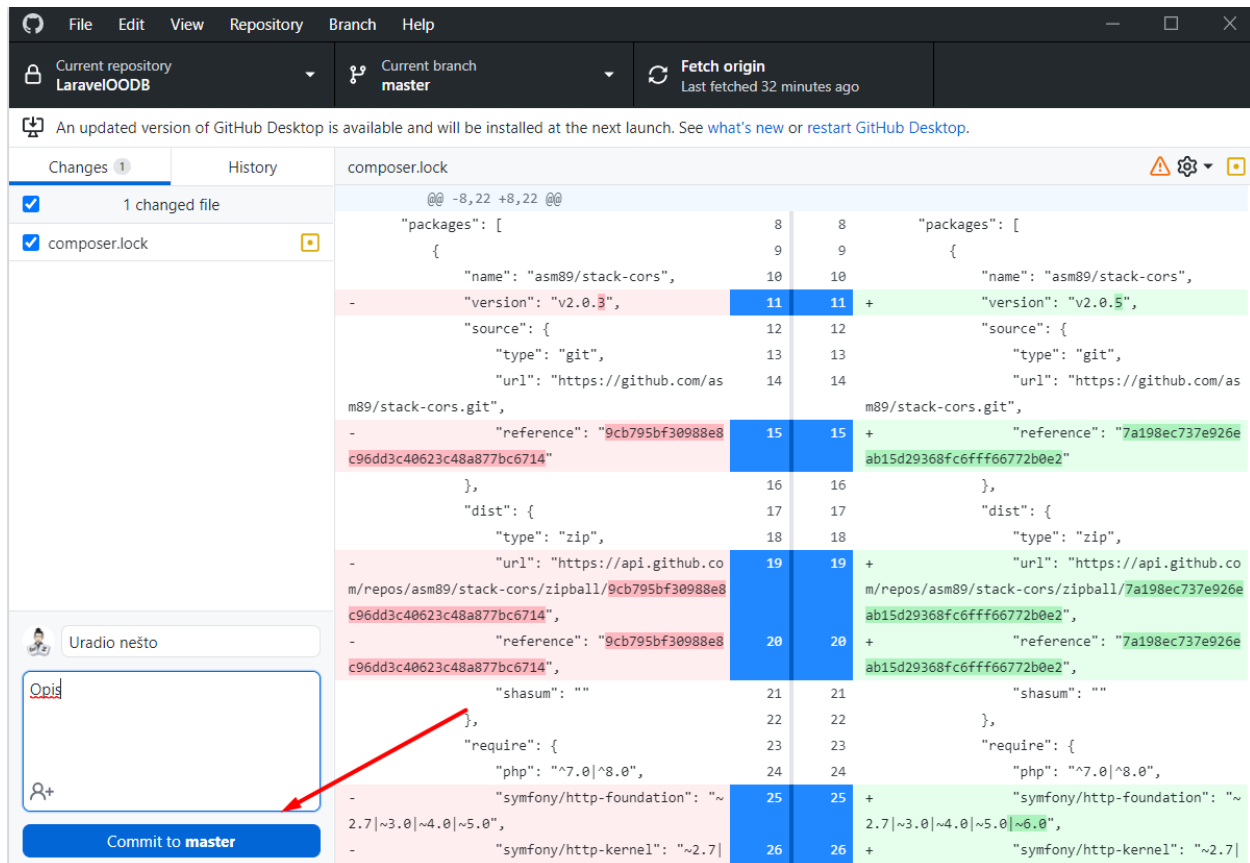
Taj projekat se sad najlakše otvori koristeći softver Github Desktop koji je došao instaliran skupa sa Git-om. Odemo na File i Clone repository koji će nam dalje ponuditi da odaberemo neki repository na github-u.



Naravno mi ćemo odabrati novokreirani LaravelOODB. Isti taj LaravelOODB možemo otvoriti koristeći Visual Studio Code.



Prije toga promjenimo branch jer nam je na branchu master sadržan kod aplikacije. Sad taj branch master otvorimo koristeći Visual Studio Code i pokrenemo naredbu u terminalu **composer update**. Zatim je potrebno napraviti novi fajl **.env** i kopirati sadržaj **.env.example** u njega i porediti naredbu **php artisan key:generate**. Porektanjem **php artisan serve** vidjet ćemo da je to ista aplikacija koju smo dosad razvijali i čuvali na našem lokalnom kompjuteru. Osnovne radnje koje možemo vršiti sa GitHub Desktopom su svakako **commit**, **pull** i **merge**. Sa commit šaljemo promjene, sa pull povlačimo najnoviju verziju aplikacije i sa merge spajamo različite verzije aplikacije.



Tutorijali za git i github:

<https://www.youtube.com/watch?v=RGOj5yH7evk>

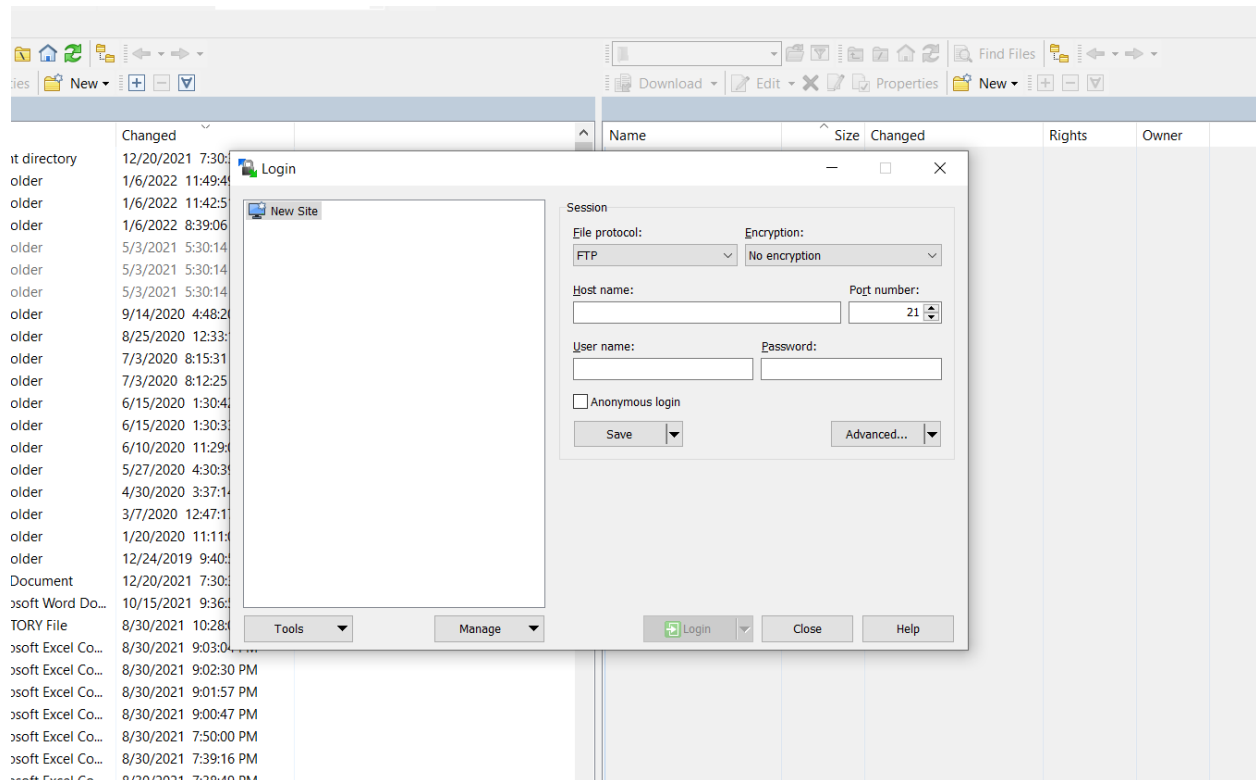
<https://www.youtube.com/watch?v=T4i1-7kT4xQ>

<https://medium.com/@ujalajha/connecting-the-laravel-project-on-github-73acf55bbd63>

<https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners>

Vježbe 15 – Podizanje Laravel aplikacije na server koristeći FTP

Aplikacija koja se razvila koristeći Laravel jednostavno se postavi na FTP server. Uz pretpostavku da prethodno imamo zakupljen host i domen, te da smo dobili pristupne FTP podatke aplikaciju ćemo najlakše podići na server koristeći neki od FTP poslužitelja kao što su WinSCP ili FileZilla. Koristeći pristupne podatke, a to su: naziv servera (ili često IP adresa), username FTP klijenta, password FTP klijenta i odabirom porta 21 koji je port za FTP komunikaciju, logiramo se na server. Ovako to izgleda koristeći WinSCP.



Root direktoriji servera i raspored fajlova na serveru zavisi od vrste servera. Prilikom objave Laravel aplikacije sve fajlove osim **public** fajla treba staviti na root servera (u neki folder npr. naziv projekta) ili u **private** folder ukoliko je takva konstrukcija servera (isto u neki folder npr. naziv projekta). Public folder Laravel aplikacije se stavlja u public folder servera npr. ako se radi o Apache/Linux serveru onda je to public_html folder.

Potrebno je u .env folderu promjeniti APP_ENV na production, a APP_DEBUG na false kako svako ne bi mogao vidjeti errore koje Laravel izbacuje jer bi se na taj način mogle izdati osjetljive informacije. APP_URL se isto izmjeni da odgovara URL-u aplikacije. Pristupni za bazu podataka se postavke da odgovaraju pristupnim za bazu podataka.

Od fajlova potrebno je izmjeniti informacije unutar public index.php fajla i referencirati ga prema folderu unutar root-a gdje smo stavili aplikaciju. Dalje dodajemo sljedeću konfiguraciju na .htaccess fajl u public folderu.


```
<IfModule mod_rewrite.c>
```

```
    <IfModule mod_negotiation.c>
```

```
        Options -MultiViews -Indexes
```

```
    </IfModule>
```

```
RewriteEngine On
```

```
# Handle Authorization Header
```

```
RewriteCond %{HTTP:Authorization} .
```

```
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
```

```
# Redirect Trailing Slashes If Not A Folder...
```

```
RewriteCond %{REQUEST_FILENAME} !-d
```

```
RewriteCond %{REQUEST_URI} (.+)/$
```

```
RewriteRule ^ %1 [L,R=301]
```

```
# Handle Front Controller...
```

```
RewriteCond %{REQUEST_FILENAME} !-d
```

```
RewriteCond %{REQUEST_FILENAME} !-f
```

```
RewriteRule ^ /index.php [L]
```

```
</IfModule>
```

Na taj način smo postavili našu aplikaciju na server koristeći FTP konekciju i sada je ona dostupna na internetu.