Napredno Web programiranje

Laravel

Rutiranje

- Rute se koriste da povežu URL-ove sa kontrolerima ili funkcijama koji se trebaju izvesti kad korisnik pristupi nekoj stranici
- Po defaultu sve rute su definirane u datoteci routes/web.php koja je uključena u svaki novi projekt
- Ruta za prikaz svih zadataka, ruta za dodavanje novog zadatka i ruta za brisanje zadatka
- Sve rute se mogu postaviti u međusloj (eng. Middleware)

Rutiranje

```
use App\Task;
use Illuminate\Http\Request;
/*Prikaz svih zadataka*/
Route::get('/', function () {
//
});
/* Novi zadatak*/
Route::post('/task', function (Request $request) {
//
});
/*Obriši zadatak*/
Route::delete('/task/{id}', function ($id) {
//
 });
```

Rutiranje

- Grupni atribut prefix može se koristiti da svakoj ruti stavimo prefiks određene grupe sa danim URL-om

```
Route::group(['prefix' => 'admin'], function () {
        Route::get('students', function () {
        // Odgovara URL-u "/admin/students"
        });
});
```

- Također možemo grupi kontrolera dodijeliti isti PHP namespace

```
Route::group(['namespace' => 'Admin'], function() {
// Kontroleri u namespace "App\Http\Controllers\Admin"
        Route::group(['namespace' => 'User'], function() {
// Kontroleri u namespace  "App\Http\Controllers\Admin\User"
        });
});
```

Rutiranje

- Kako bi svim rutama u grupi dodijelili programski međusloj koristi se ključna riječ middleware

```
Route::group(['middleware' => 'auth'], function () {
        Route::get('/', function () {
        // Koristi Auth međusloj
        });
        Route::get('/students', function () {
        // Koristi Auth međusloj
        });
});
```

Rutiranje

- Poddomene također mogu biti dodijeljene kao parametri rute, tako možemo koristiti dijelove poddomene u ruti ili kontroleru. Koristi se ključna riječ domain

```
Route::group(['domain' => '{account}.mdome.com'], function ()
{       Route::get('student/{id}', function ($account, $id) {
        // Koristi Auth međusloj
        });
});
```

- Ostale ključne riječi možete vidjeti u Laravel dokumentaciji

Rutiranje

- Laravel omogućava zaštitu od cross-site request forgeries (CSRF)
- To je prijevara u kojoj se izvode nedozvoljene naredbe u ime prijavljenog korisnika
- Laravel stvara CSRF "token" za svakog aktivnog korisnika
- Token se koristi za provjeru je li korisnik uistinu onaj koji stvara zahtjev
- Za stvaranje sakrivenog (eng. Hidden) input polja koje sadrži CSRF token koristi se csrf_field helper funkcija:

```
echo csrf_field();
```

Rutiranje

- Pojedine stranice mogu se isključiti iz CSRF zaštite
- Pojedine URI-e možemo isključiti dodajući ih u svojstvo $except koje pripada VerifyCsrfToken međusloju (eng. Middleware)

```
namespace App\Http\Middleware;
class VerifyCsrfToken extends BaseVerifier {
        protected $except = [ 'stripe/*', ];
}
```

Rutiranje

- HTML forme ne podržavaju PUT, PATCH i DELETE akcije
- Stoga kad koristimo PUT, PATCH ili DELETE rute iz HTML forme moramo ih nekako simulirati
- Stoga dodajemo sakriveno _method polje u formu koje će se koristiti kao metoda u HTTP zahtjevu

```
<form action="/foo/bar" method="POST">
<input type="hidden" name="_method" value="PUT">
<input type="hidden" name="_token" value="{{ csrf_token()
}}">
</form>
```

Međusloj (eng. Middleware)

- Međusloj omogućava filtriranje HTTP zahtjeva
- Na primjer međusloj koji provjerava je li korisnik prijavljen, ako nije preusmjerava ga na login
- Mogu se napisati različiti međuslojevi kao što je CORS (Cross-Origin Resource Sharing) međusloj za dodavanje zaglavlja u sve odgovore iz aplikacije
- U Laravel su uključeni međuslojevi kao što su maintenance, authentication, CSRF protection i drugi
- Svi međuslojevi nalaze se u direktoriju

  app/Http/Middleware

Međusloj (eng. Middleware)

- Za stvaranje novog međusloja koristi se Artisan naredba make:middleware:

```
php artisan make:middleware OldMiddleware
```

- Naredba će stvoriti novu OldMiddleware klasu unutar app/Http/Middleware direktorija
- U ovom sloju ćemo dozvoliti pristup ruti samo za one čije su godine veće od 18

```
namespace App\Http\Middleware; use Closure;
class OldMiddleware {
        public function handle($request, Closure $next) {
                if ($request->input('age') < 18) {
                        return redirect('home');
                }
                return $next($request);
        } }
```

Međusloj (eng. Middleware)

- Međusloj se može izvesti prije ili poslije zahtjeva
- Međusloj koji se izvodi prije zahtjeva

```
namespace App\Http\Middleware;
use Closure;
class BeforeMiddleware {
        public function handle($request, Closure $next) {
                // Izvedi međusloj
                return $next($request);
        }
}
```

Međusloj (eng. Middleware)

- Međusloj koji se izvodi poslije zahtjeva

```
namespace App\Http\Middleware;
use Closure; class AfterMiddleware {
        public function handle($request, Closure $next) {
                $response = $next($request);
                // Izvedi međusloj
                return $response;
        }
}
```

Međusloj (eng. Middleware)

- Ako želimo da se međusloj pokrene pri svakom HTTP zahtjevu treba dodati međusloj u svojstvo $middleware u klasi app/Http/Kernel.php
- Ako želimo da se međusloj pokrene pri određenoj ruti moramo međusloju dodati ključ u app/Http/Kernel.php datoteci
- Po defaultu svojstvo $routeMiddleware ove klase sadrži unose za međuslojeve uključene u Laravel
- Za dodavanje svog međusloja samo ga dodate u listi i ključu date ime po izboru kao u sljedećem primjeru:

Međusloj (eng. Middleware)

- Primjer dodavanja ključa međusloja:

```
// Unutar App\Http\Kernel klase
protected $routeMiddleware = [
'auth' => \App\Http\Middleware\Authenticate::class,
'auth.basic' =>
\Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class, ];
```

- Nakon što se doda ključ međusloj možete koristiti

```
Route::get('admin/profile', ['middleware' => 'auth', function () {
//
}]);
```

Međusloj (eng. Middleware)

- Možemo dodati više međuslojeva za rutu i to na dva načina:

```
Route::get('/', ['middleware' => ['first', 'second'], function () {
//
}]);
```

- Ili na drugi način

```
Route::get('/', function () { // })
        ->middleware(['first', 'second']);
```

Međusloj (eng. Middleware)

- Međusloj može primati i parameter
- Na primjer možemo provjeriti ima li korisnik određenu ulogu stvaranjem RoleMiddleware koji prima ulogu kao dodatni argument

```
namespace App\Http\Middleware;
use Closure; class RoleMiddleware {
public function handle($request, Closure $next, $role) {
        if (! $request->user()->hasRole($role)) {
                // Preusmjeri...
        }
        return $next($request);
}
}
```

Međusloj (eng. Middleware)

- Međusloj može obaviti i određene radnje nakon što je HTTP odgovor već poslan pregledniku
- Na primjer "session" međusloj upisuje svoje podatke
- To se postiže dodavanjem metode terminate u međusloj

```
namespace Illuminate\Session\Middleware;
use Closure;
class StartSession { public function handle($request, Closure
$next) {
        return $next($request); }
        public function terminate($request, $response) {
                // Spremi podatke iz sesije
        }
}
```

KONTROLERI

- Ako ne želimo svu logiku držati u jednoj datoteci sa rutama (web.php) možemo koristiti klase Controller. Svi Laravel kontroleri nasljeđuju osnovnu Controller klasu.

```
namespace App\Http\Controllers;
use App\User;
use App\Http\Controllers\Controller;
class UserController extends Controller {
        public function showProfile($id) {
                return view('user.profile', [
                        'user' => User::findOrFail($id)]);
        }
}
```

- Upotreba preko rute je
  Route::get('user/{id}',UserController@showProfile');

KONTROLERI

- Također možemo imenovati rute kontrolera

```
Route::get('foo', ['uses' => 'FooController@method', 'as' =>
'name']);
```

- A route helper se koristi za stvaranje URL-a do imenovane rute kontrolera

```
$url = route('name');
```

- A action helper za stvaranje URL-a pomoću imena klase kontrolera i imena metode

```
$url = action('FooController@method');
```

- Imenu akcije kontrolera se pristupa pomoću metode currentRouteAction

```
$action = Route::currentRouteAction();
```

KONTROLERI

- Međusloj možemo dodijeliti ruti kontrolera

```
Route::get('profile', [ 'middleware' => 'auth', 'uses' =>
'UserController@showProfile' ]);
```

- No, najbolje je dodijeliti međusloj u konstruktoru kontrolera. Čak možete dodijeliti međusloj određenoj metodi

```
class UserController extends Controller {
public function __construct() {
        $this->middleware('auth');
        $this->middleware('log', ['only' => ['fooAction',
'barAction']]);
        $this->middleware('subscribed', ['except' =>
['fooAction', 'barAction']]);
 }
}
```

KONTROLERI

- Resurs kontroleri omogućavaju izradu RESTful kontrolera oko resursa
- Na primjer možemo napraviti kontroler koji će obrađivati HTTP zahtjeve oko slika u aplikaciji
- Artisan naredom make:controller se stvara takav kontroler

```
php artisan make:controller PhotoController
```

- Stvoriti će se kontroler u datoteci app/Http/Controllers/PhotoController.php
- Kontroler će imati metode za svaku operaciju sa resursom

KONTROLERI

- Potrebno je povezati rutu sa kontrolerom

```
Route::resource('photo', 'PhotoController');
```

- Gornja deklaracije rute stvara višestruke rute za obratu različitih RESTful akacija

Verb	Path	Action	Route Name
GET	`/photo`	index	photo.index
GET	`/photo/create`	create	photo.create
POST	`/photo`	store	photo.store
GET	`/photo/{photo}`	show	photo.show
GET	`/photo/{photo}/edit`	edit	photo.edit
PUT/PATCH	`/photo/{photo}`	update	photo.update
DELETE	`/photo/{photo}`	destroy	photo.destroy

KONTROLERI

- Pri deklaraciji rute resursa možemo reći koje akcije želimo obavljati

```
Route::resource('photo', 'PhotoController',
        ['only' => ['index', 'show']]);
Route::resource('photo', 'PhotoController',
        ['except' => ['create', 'store', 'update', 'destroy']]);
```

- Sve akcije u kontroleru imaju imena, ako ih želimo promijeniti dodajući niz names

```
Route::resource('photo', 'PhotoController',
        ['names' => ['create' => 'photo.build']]);
```

KONTROLERI

- Mogu se dodavati i rute do ugniježđenih resursa
- Slike mogu imati višestruke komentare

```
Route::resource('photos.comments', 'PhotoCommentController');
```

- Ruta će stvoriti ugniježđene resurse kojima se može pristupiti sa

  photos/{photos}/comments/{comments}

```
class PhotoCommentController extends Controller {
/* Prikaži komentar sa id-em i ide-em slike*/
        public function show($photoId, $commentId) {
                //
        }
}
```

KONTROLERI

- Ako je potrebno dodati nove rute osim već postojećih nužno je definirati rute prije nego se pozove Route::resource;
- U suprotnome ruta koji definira resurs može imati prednost pred zamjenskom rutom

```
Route::get('photos/popular', 'PhotoController@method');
```

```
Route::resource('photos', 'PhotoController');
```

KONTROLERI

- Laravel nudi mogućnost da se definira jedna ruta koja će obraditi svaku akciju u kontroleru
- Najprije se definira ruta pomoću metode Route::controller koja prima dva argumenta
- Prvi je bazni URI dok je drugi ime klase kontrolera

```
Route::controller('users', 'UserController')
```

- Potrebno je dodati metode u kontroler

```
class UserController extends Controller {
public function getIndex() { //GET /users }
public function getShow($id) { //GET /users/show/1 }
public function getAdminProfile() { // GET /users/admin-profile}
public function postProfile() { //GET /users/profile }
}
```

KONTROLERI

- Laravel ima mogućnost korištenja dependecy injection u konstruktor koje će biti ubačene u instance kontrolera

```
namespace App\Http\Controllers;
use Illuminate\Routing\Controller;
use App\Repositories\UserRepository;
class UserController extends Controller {
 /*Instanca repozitorija user */
protected $users;
/*Stvaranje nove instance kontrolera */
public function __construct(UserRepository $users) {
        $this->users = $users;
}
}
```

KONTROLERI

- Osim u konstruktor injection se može dodati i u metode kontrolera
- Ubaciti ćemo instancu Illuminate\Http\Request u jedno od metoda

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Routing\Controller;
class UserController extends Controller {
public function store(Request $request) {
        $name = $request->input('name');
        //
}
}
```

KONTROLERI

- Ako aplikacija koristi samo rute bazirane na kontrolerima možete koristiti Laravel rute cache
- To će značajno smanjiti vrijeme registracije svih ruta u aplikaciji
- U nekim slučajevima i do 100 puta
- Za stvaranje rute cache koristi se Artisan naredba:

```
php artisan route:cache
```

- Cache ruta sada se koristi umjesto web.php
- Ako se doda nova ruta potrebno je napraviti novi rute cache pa je najbolje to raditi na kraju
- Rute cache ne radi sa Closure rutama

Zahtjevi (eng. Requests)

- Za dohvaćanje instance HTTP zahtjeva upotrebom dependency injection potrebno je koristiti Illuminate\Http\Request klasu u kontroleru ili metodi
- Instanca zahtjeva će biti ubačena u kontroler

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Routing\Controller;
class UserController extends Controller {
/* Spremi novog korisnika */
public function store(Request $request) {
        $name = $request->input('name');
}
}
```

Zahtjevi (eng. Requests)

- Ako metoda očekuje unos iz parametara rute jednostavno se unesu argumenti rute nakon ovisnosti. Ako je ruta definirana sa:

```
Route::put('user/{id}', 'UserController@update');
```

- U metodi pristupamo parametru:

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Routing\Controller;
class UserController extends Controller {
        public function update(Request $request, $id) {
                //
        }
}
```

Zahtjevi (eng. Requests)

- Dohvat URI-a od zahtjeva, metoda path

```
$uri = $request->path();
```

- Pomoću is metode provjerava se uzorak u URI-u:

```
if ($request->is('admin/*')) { // }
```

- Za puni URL, a ne samo putanju

```
$url = $request->url();
```

- Pomoću method saznajemo metodu HTTP zahtjeva, a isMethod provjera odgovara li zahtjev određenom stringu

```
$method = $request->method();
if ($request->isMethod('post')) { // }
```

Zahtjevi (eng. Requests)

- Dohvat korisničkog unosa za sve metode

```
$name = $request->input('name');
```

- Ili pomoću svojstva

```
$name = $request->name;
```

- Možemo dati i defaultnu vrijednost

```
$name = $request->input('name', 'Pero');
```

- Provjera postoji li unos:

```
if ($request->has('name')) { // }
```

- Dohvat svih input podataka u niz, ili samo nekih

```
$input = $request->all();
$input = $request->only(['username', 'password']);
$input = $request->except('credit_card');
```

Zahtjevi (eng. Requests)

- Metoda flash će spremiti input u sesiju

```
$request->flash();
```

- Mogu se koristiti flashOnly i flashExcept metode

```
$request->flashOnly('username', 'email');
$request->flashExcept('password');
```

- Isto tako može se spremiti u sesiju i preusmjeriti na neku stranicu sa withInput metodom

```
return redirect('form')->withInput();
return redirect('form')->withInput($request->except('password'));
```

- Dohvat starog podatka iz sesije

```
$username = $request->old('username');
```

Zahtjevi (eng. Requests)

- Svi kolačići (eng. Cookies) u Laravelu su kriptirani i potpisani sa autentifikacijskim kodom
- Za dohvat se koristi cookie metoda

```
$value = $request->cookie('name');
```

- Dodavanje kolačića u odgovor

```
$response = new Illuminate\Http\Response('Hello World');
$response->withCookie(cookie('name', 'value', $minutes));
return $response;
```

- Za vječni kolačić koji nema rok trajanja koristi se metoda cookie bez argumenata i forever metoda

```
$response->withCookie(cookie()->forever('name', 'value'));
```

Zahtjevi (eng. Requests)

- Dohvat uploadanih datoteka metodom file
- Objekt koji vrati metoda file je instanca klase Symfony\Component\HttpFoundation\File\UploadedFile koja nasljeđuje klasu PHP SplFileInfo

```
$file = $request->file('photo');
```

- Provjera postoji li datoteka

```
if ($request->hasFile('photo')) { // }
```

- Provjera je li upload uspješno obavljen

```
if ($request->file('photo')->isValid()) { // }
```

- Za prebacivanje datoteke iz privremene lokacije

```
$request->file('photo')->move($destinationPath);
$request->file('photo')->move($destinationPath, $fileName);
```

Odgovori (eng. Responses)

- Rute i kontroleri vraćaju odgovore pregledniku
- Laravel može vratiti odgovore na nekoliko načina
- Najjednostavniji način je vraćanje stringa

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

- Laravel će string automatski pretvoriti u HTTP odgovor
- U većini slučajeva vraćati će se instanca klase Illuminate\Http\Response ili pogled
- Imate mogućnost postavljati status kodove i zaglavlja

Odgovori (eng. Responses)

- Vraćanje odgovora preko instance klase Response

```
use Illuminate\Http\Response;
Route::get('home', function () {
        return (new Response($content, $status))
        ->header('Content-Type', $value);
});
```

- Isto tako možemo koristiti response helper

```
Route::get('home', function () {
        return response($content, $status)
        ->header('Content-Type', $value);
});
```

Odgovori (eng. Responses)

- Metode u odgovoru se mogu ulančavati

```
return response($content)
            ->header('Content-Type', $type)
            ->header('X-Header-One', 'Header Value')
            ->header('X-Header-Two', 'Header Value');
```

- U odgovor se mogu dodati i kolačići

```
return response($content)->header('Content-Type', $type)
            ->withCookie('name', 'value');
```

- Metoda može imati i dodatne parameter

```
->withCookie($name, $value, $minutes, $path, $domain,
$secure, $httpOnly)
```

Odgovori (eng. Responses)

- Za vraćanje pogleda koristi se metoda view

```
return response()->view('hello', $data)->header('Content-Type', $type);
```

- Za vraćanje JSON koristi se metoda json

```
return response()->json(['name' => 'Abigail', 'state' => 'CA']);
```

- Za preuzimanje datoteka koristi se metoda download

```
return response()->download($pathToFile);
return response()->download($pathToFile, $name, $headers);
```

Odgovori (eng. Responses)

- Preusmjereni odgovori su instance klase Illuminate\Http\RedirectResponse
- Sadrže zaglavlja nužna za preusmjeravanje korisnika na drugi URL
- Preusmjeravanje se može obaviti na nekoliko načina, a najjednostavnije je preko metode redirect

```
Route::get('dashboard', function () {
        return redirect('home/dashboard');
});
```

Odgovori (eng. Responses)

- Za vraćanje korisnika na prethodnu lokaciju

```
Route::post('user/profile', function () {
        // Obrada zahtjeva
        return back()->withInput();
});
```

- Preusmjeravanje na imenovanu rutu, i rutu koja ima parametre

```
return redirect()->route('login');
return redirect()->route('profile', [1]);
```

Odgovori (eng. Responses)

- Preusmeravanje na akcije kontrolera preko metode action

```
return redirect()->action('HomeController@index');
return redirect()->action('UserController@profile', [1]);
```

- Preusmjeravanje i spremanje podataka u sesije

```
Route::post('user/profile', function () {
// Izmjena profila i spremanje statusa u sesiju
return redirect('dashboard')->with('status', 'Profile updated!');
});
```

- Dohvat varijable u pogledu

```
@if (session('status'))
<div class="alert alert-success"> {{ session('status') }}
</div>
@endif
```

Odgovori (eng. Responses)

- Ako želite stvoriti vlastiti odgovor koristi se metodu macro klase Illuminate\Contracts\Routing\ResponseFactory

```
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
use Illuminate\Contracts\Routing\ResponseFactory;
class ResponseMacroServiceProvider extends ServiceProvider
{
  public function boot(ResponseFactory $factory) {
   $factory->macro('caps', function ($value) use ($factory) {
            return $factory->make(strtoupper($value));
    });
  }
}
```

- Poziv makroa `return response()->caps('foo');`

Pogledi (eng. Views)

- Pogledi sadrže HTML koji poslužuje vaša aplikacija i odvajaju aplikacijsku logiku u kontroleru od prezentacijske logike
- Pogledi se nalaze u direktoriju resources/views
- Najjednostavniji pogled

```
<!– Pogled resources/views/pozdrav.php -->
<html>
  <body>
        <h1>Dobro došao <?php echo $ime; ?></h1>
  </body>
</html>
```

Pogledi (eng. Views)

- Gornji pozdrav možemo vratiti sa sljedećim kodom

```
Route::get('/', function () {
        return view('pozdrav', ['ime' => 'Umberto']);
});
```

- Provjera postoji li pogled

```
if (view()->exists('pozdrav')) { // }
```

- Osim slanja podataka na gornji način pogledu možemo poslati podatke i metodom with

```
$view = view('pozdrav')->with('ime', 'Umberto');
```

Pogledi (eng. Views)

- Ako želimo podijeliti podatke sa svim pogledima u aplikaciji koristi se share metoda
- Metoda share se uobičajeno poziva unutar service provider boot metode

```
namespace App\Providers;
class AppServiceProvider extends ServiceProvider {
        public function boot() {
                view()->share('key', 'value');
        }
        /* Registriraj service provider. */
        public function register() { // }
}
```

Pogledi (eng. Views)

- Kompozitori pogleda (eng. View composers) su funkcije povratnog poziva (eng. Callbacks) ili metode klase koje se pozivaju kad se izrenderira pogled
- Koriste se ako imate podatke za koje želite da se povežu sa pogledom pri svakom pozivu pogleda
- Potrebno je registrirati kompozitore u service provider. Koristi se view helper za pristup Illuminate\Contracts\View\Factory
- Laravel ne uključuje defaultni direktorij za kompozitore pogleda pa možete koristiti direktorije koje želite, npr. App\Http\ViewComposers

Pogledi (eng. Views)

```
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
class ComposerServiceProvider extends ServiceProvider {
/*Registriraj povezivanja*/
   public function boot() {
            // Upotreba kompozitora baziranih na klasama
            view()->composer( 'profile',
            'App\Http\ViewComposers\ProfileComposer' );
            // Upotreba kompozitora baziranih na Closure
            view()->composer('dashboard', function ($view) { });
   }
   /*Registriraj service provider*/
   public function register() { // }
}
```

Pogledi (eng. Views)

- Ako napravite novi service provider za vaše kompozitore morate dodati service provider u providers niz u config/app.php configuration
- Nakon što smo registrirali kompozitor metoda ProfileComposer@compose će se izvršiti svaki put kad se renderira pogled profile
- Sljedeće što je potrebno je definirati klasu za kompozitor na sljedećem slajdu

Pogledi (eng. Views)

```
namespace App\Http\ViewComposers;
use Illuminate\Contracts\View\View;
use Illuminate\Users\Repository as UserRepository;
class ProfileComposer {
/*Repozitorij user*/
protected $users;
/* Stvaranje kompozitora  za novi profil */
public function __construct(UserRepository $users) {
// Dependency injection za korisnike
        $this->users = $users;
}
/*Poveži podatke sa pogledom, napomena: metoda compose
poziva se prije renderiranja pogleda*/
public function compose(View $view) {
        $view->with('count', $this->users->count());
} }
```

Pogledi (eng. Views)

- Kompozitor pogleda možete dodati za višestruke poglede odjednom pomoću metode composer

```
view()->composer( ['profile', 'dashboard'],
'App\Http\ViewComposers\MyViewComposer' );
```

- Ili ga dodati svim pogledima

```
view()->composer('*', function ($view) { // });
```

- Kreatori pogleda su slični kompozitorima, ali se pokreću odmah i ne čekaju da se pogled renderira u potpunosti. Za registriranje se koristi creator metoda

```
view()->creator('profile',
'App\Http\ViewCreators\ProfileCreator');
```

Blade

- Blade je jednostavni engine za predloške
- Blade za razliku od drugih PHP engina za predloške omogućava korištenje običnog PHP-a u pogledima
- Svi se Blade pogledi kompajliraju u obični PHP i spremaju u cache sve dok se ne modificiraju
- To znači da Blade ne usporava aplikaciju
- Blade datoteke imaju eksenziju .blade.php i spremaju se u direktorij resources/views

Blade

- Dvije najveće mogućnosti Blade predložaka su nasljeđivanje i sekcije
- Budući da većina stranica ima identičan glavni raspored pomoću Blade-a se definira glavni raspored (eng. Layout)

```
<!-- resources/views/layouts/master.blade.php --> <html>
<head>
<title>Naziv aplikacije - @yield('title')</title> </head>
<body>
        @section('sidebar') Master sidebar.
        @show
        <div class="container">
                @yield('content')
        </div>
</body> </html>
```

Blade

- U glavnom rasporedu vidimo direktivu @section koja definira sekciju sadržaja
- Direktiva @yield koristi se za prikaz sadržaja sekcije dane kao parameter
- Slijedi primjer pogleda koji ga nasljeđuje

```
<!-- resources/views/child.blade.php -->
@extends('layouts.master')
@section('title', 'Page Title')
@section('sidebar')
        @parent
<p>Ovo se dodaje u master sidebar.</p>
@endsection
@section('content')
        <p>Sadržaj same stranicep>
@endsection
```

Blade

- U gornjem primjeru sidebar sekcija koristi direktivu @parent koja će umjesto da prebriše roditeljski sidebar na njega samo nadodati svoj sadržaj
- Direktiva @parent će pri renderiranju biti zamijenjena sadržajem dječjeg pogleda
- Blade pogledi isto kao i obični PHP pogledi iz ruta mogu biti vraćeni upotrebom view helper funkcije

```
Route::get('blade', function () {
        return view('child');
});
```

Blade

- Podatke koje predamo Blade pogledu možemo dohvaćati pomoću varijable u vitičastim zagradama. Tako za rutu

```
Route::get('pozdrav', function () {
        return view('dobrodosli', ['ime' => 'Marko']);
});
```

- U pogledu sadržaj varijable ima možemo prikazati sa Pozdrav {{ $ime }}
- Osim varijabli predanih pogledu možemo ispisati i rezultat bilo koje PHP funkcije kao što je funkcija za prikaz vremena {{ time() }}

Blade

- Blade ima skraćenice za PHP naredbe kontrole toka

```
@if (count($records) === 1)
@elseif (count($records) > 1)
@else
@endif
```

- Također ima i @unless direktivu:

```
@unless (Auth::check())
Vi niste prijavljeni
@endunless
```

Blade

- Blade ima skraćenice za PHP petlje

```
@for ($i = 0; $i < 10; $i++)
Vrijednost od i je {{ $i }}
@endfor

@foreach ($users as $user)
<p>Trenutni korisnik je {{ $user->id }}</p>
@endforeach

@while (true)
<p>Beskonačna petlja <p>
 @endwhile
```

Blade

- Blade ima mogućnost ubacivanja u imenovane stogove koji mogu biti renderirani negdje drugo, npr. u drugom pogledu ili layoutu

```
@push('scripts')
<script src="/example.js"></script> @endpush
```

- @inject direktiva se koristi za dohvat servisa iz Laravel service container
- Prvi argument predan direktivi @inject je ime varijable u koju će se smjestiti servis, drugi argument je ime klase ili interfejsa koji želite koristiti

```
@inject('metrics', 'AppServicesMetricsService')
```

Blade

- Blade nudi mogućnost stvaranja vlastitih direktiva pomoću metode directive
- Kad Blade kompajler naiđe na direktivu poziva se callback. Sljedeći primjer stvara direktivu @datetime($var) koja formatira varijablu $var

```
namespace App\Providers;
use Blade; use Illuminate\Support\ServiceProvider;
class AppServiceProvider extends ServiceProvider {
  public function boot() {
        Blade::directive('datetime', function($expression) {
              return "<?php echo with{$expression}
              ->format('m/d/Y H:i'); ?>";
        });
  }}
```

- Konačni izlaz iz direktive je

```
<?php echo with($var)->format('m/d/Y H:i'); ?>
```

Povezivanje rute i modela

- Laravel povezivanje rute i modela (eng. route model binding provides) na jednostavan način umeće instance modela u rute
- Umjesto umetanja korisničkog ID-a možemo umetnuti čitavu instancu korisničkog modela za dani ID
- Laravel će implicitno povezati Eloquent modele definirane u rutama ili kontrolerima čija imena odgovaraju imenu rute:

```
Route::get('api/users/{user}', function (App\User $user) {
        return $user->email;
});
```

Rad sa bazom podataka

- Laravel se jednostavno može povezati s bazom podataka i pokretati upite nad različitim SUBP koristeći obični SQL, fluent query bulder i Eloquent ORM
- Laravel može raditi sa sljedećim SUBP:
  - MySQL
  - Postgres
  - SQLite
  - SQL Server

Rad sa bazom podataka

- Konfiguracija je jednostavna

```
'mysql' =>
[ 'read' =>
            [ 'host' => '192.168.1.1', ],
'write' =>
            [ 'host' => '196.168.1.2' ],
'driver' => 'mysql',
'database' => 'database',
'username' => 'root',
'password' => '',
'charset' => 'utf8',
'collation' => 'utf8_unicode_ci',
 'prefix' => '',
],
```

Rad sa bazom podataka

- Upotreba običnih SQL upita
- Select upit
- $results = DB::select('select * from users where id = :id', ['id' => 1]);
- Insert

  DB::insert('insert into student (id, ime) values (?, ?)', [1, 'Matej']);
- Update

  $affected = DB::update('update student set godina = 1 where ime = ?', ['Matej']);
- Delete

  $deleted = DB::delete('delete from student');

Rad sa bazom podataka

- Ako želimo obaviti više naredba kao jednu transakciju u bazi podataka možemo koristiti metodu transaction
- Ako se tijekom transakcije dogodi iznimka transakcija će biti poništena
- Ako se sve provede ispravno transakcija će biti potvrđena

```
DB::transaction(function () {
        DB::table('users') ->update(['votes' => 1]);
        DB::table('posts') ->delete();
});
```

Rad sa bazom podataka

- Ako želimo koristiti više spojeva na bazu podataka svakom spoju možemo pristupiti preko metode connection
- Ime koje predamo metodi connection mora odgovarati imenu spojeva na bazu u datoteci config/database.php

```
$users = DB :: connection('foo') ->select(...);
OR
$users = DB :: connection() ->getPdo();
```

Rad sa bazom podataka

- Query builder koristi PDO kako bi spriječio SQL injection napade
- Sve redove iz tablice možemo dohvatiti na sljedeći način

```
namespace App\Http\Controllers;
use DB; use App\Http\Controllers\Controller;
class StudentController extends Controller {
/*Prikaži sve studente iz tablice students*/
    public function index() {
        $students = DB::table('students')->get();
        return view('student.index', ['students' => $students]);
    }
}
```

Rad sa bazom podataka

- Jedan redak iz tablice možemo dohvatiti na sljedeći način

```
$student = DB::table('students')->where('name', 'Ante')->first();
echo $ student >name;
```

- Ako ne želimo čitav redak nego samo jedan atribut možemo koristiti metodu value

```
$student = DB::table('students')->where('name', 'Ante')-
>value('lastname');
```

- Ako radimo s puno zapisa možemo koristiti chunk metodu

```
DB::table('students')->chunk(100, function($students) {
        foreach ($students as $student) {
        // }
});
```

Rad sa bazom podataka

- Query builder omogućava korištenje agregatnih funkcija kao što su count, max, min, avg i sum

```
$students = DB::table('students')->count();
$ects = DB::table('students')->max('ects');
```

- Ove funkcije se mogu kombinirati s drugim naredbama

```
$price = DB::table('orders')
        ->where('finalized', 1)
        ->avg('price');
```

Rad sa bazom podataka

- Inner join (možemo koristiti i leftJoin)

```
$users = DB::table('users')
        ->join('contacts', 'users.id', '=', 'contacts.user_id')
        ->join('orders', 'users.id', '=', 'orders.user_id')
         ->select('users.*', 'contacts.phone', 'orders.price')
        ->get();
```

- Advanced join omogućava da predamo Closure kao drugi argument u join metodu, i Closure će primiti JoinCaluse objekt pa možemo dodati ograničenja na join

```
DB::table('users') ->join('contacts', function ($join) {
        $join->on('users.id', '=', 'contacts.user_id')->orOn(...);
        })
        ->get();
```

Rad sa bazom podataka

- Query builder omogućava da napravimo uniju dva upita
- Možemo napraviti prvi upit i pomoću union metode napraviti uniju s drugim upitom

```
$first = DB::table('users')
            ->whereNull('first_name');

$users = DB::table('users')
            ->whereNull('last_name')
            ->union($first)
            ->get();
```

Rad sa bazom podataka

- Također možemo dodati where

```
DB::table('users')
        ->join('contacts', function ($join) {
                $join->on('users.id', '=', 'contacts.user_id')
                        ->where('contacts.user_id', '>', 5);
        })
        ->get();
```

- Možemo dodati i više uvjeta

```
$user = DB::table('users')
        ->where([
                ['status', '1'],
                ['subsribed', '<>','1'],
                ]) ->get();
```

Rad sa bazom podataka

- Različiti upiti

```
$users = DB::table('users')
        ->whereBetween('votes', [1, 100])->get();

$users = DB::table('users')
        ->whereNotBetween('votes', [1, 100]) ->get();

$users = DB::table('users') ->whereIn('id', [1, 2, 3]) ->get();

$users = DB::table('users') ->whereNotIn('id', [1, 2, 3]) ->get();

$users = DB::table('users') ->whereNull('updated_at') ->get();

$users = DB::table('users')
        ->whereNotNull('updated_at') ->get();
```

Rad sa bazom podataka

- Također moguće je raditi kompliciranije upite sa podupitima
- Opet orWhere funkciji predajemo Closure

```
DB::table('users')
        ->where('name', '=', 'John')
        ->orWhere(function ($query) {
                $query->where('votes', '>', 100)
                ->where('title', '<>', 'Admin'); }) ->get();
```

- Gornji primjer dati će rezultat

```
select * from users where name = 'John' or (votes > 100 and
title <> 'Admin')
```

Rad sa bazom podataka - migracije

- Migracije (eng. Migrations) kako kontrola verzije (eng. version control) za bazu podataka
- Na jednostavan način možete mijenjati i dijeliti shemu baze podataka
- Migracije se najčešće koriste s Laravel schema builder za izgradnju sheme baze podataka
- Migracije su spremljene u direktoriju database/migrations
- Možemo koristiti opcije --table i --create

```
php artisan make:migration add_votes_to_users_table --table=users
php artisan make:migration create_users_table --create=users
```

Rad sa bazom podataka - migracije

- Klasa migration ima dvije metode up i down
- Metoda up se koristi za dodavanje novih tablica i indeksa, a down radi obrnuto. U obje metode koristi se Laravel schema builder za izmjenu tablica

```
class CreateFlightsTable extends Migration {
public function up() {
        Schema::create('flights', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->string('airline');
        $table->timestamps(); });
}
public function down() { Schema::drop('flights'); } }
```

Rad sa bazom podataka - migracije

- Migracije se pokreću sa

  php artisan migrate

- Možemo poništiti migraciju i vratiti se na prethodnu migraciju

  php artisan migrate:rollback

  php artisan migrate:reset

- Za stvaranje nove tablice koristiti create table metodu Laravel Schema

```
Schema::create('users', function (Blueprint $table) {
    $table->increments('id');
});
```

Rad sa bazom podataka - migracije

- Laravel nudi mogućnost dodavanja stranih ključeva zbog referencijalnog integriteta

```
Schema::create('posts', function ($table) {
$table->integer('user_id') ->unsigned();
$table->foreign('user_id')->references('id') ->on('users');
});
```

- Može se odrediti i akcija na on delete i on cascade

```
$table->foreign('user_id')
        ->references('id') >on('users')
        ->onDelete('cascade');
```

Eloquent ORM

- Eloquent ORM (eng. Object relational model) omogućava jednostavan rad sa bazom podataka
- Svaka tablica u bazi podataka ima odgovarajući model koji vrši interakciju s tom tablicom
- Modeli omogućavaju upite nad tablicom kao i unos podataka
- Najjednostavniji način za stvaranje modela je:

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

- Ako želite napraviti migraciju nakon stvaranja modela možete koristiti artisan naredbu:

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

ELOQUENT ORM

- Imena tablica: 'snake case', ime tablice biti će množina imena modela, osim ako se eksplicitno ne navede drugo ime:

```
class Flight extends Model {
/* Tablica za model s novim imenom*/
        protected $table = 'my_flights';
}
```

- Eloquent će pretpostaviti da svaka tablica ima primarni ključ imena id, može se koristiti svojstvo $primaryKey da to promijenimo
- Po defaultu Eloquent pretpostavlja da imate stupce created_at i updated_at u tablici, ako to želimo promijeniti postavimo:

```
public $timestamps = false;
```

Eloquent ORM

- Svi Eloquent modeli koriste defaultni spoj na bazu podataka. Ako to želite promijeniti:

```
class Flight extends Model {
/*Naziv spoja za model */
protected $connection = 'connection-name';
}
```

- Možete koristiti metodu create kako biste snimili novi model u jednoj liniji. Unesena instanca modela biti će vam vraćena iz metode
- Trebate definirate koji atribut će biti „mass assignable” pomoću svojstva $fillable

```
class Flight extends Model {
        protected $fillable = ['name'];
}
```

Eloquent ORM

- Umjesto trajnog brisanja iz baze Eloquent može napraviti "soft delete" modele
- To znači da nisu uklonjeni iz baze podataka
- Umjesto toga postavlja se atribut deleted_at na model i unosi se u bazu podataka
- Za omogućavanje trebate dodati navedeni stupac deleted_at u tablicu i u klasu Modela:

```
class Flight extends Model {
        use SoftDeletes;
        protected $dates = ['deleted_at'];
}
```

Eloquent ORM

- Stupac deleted_at možete dodati pomoću:

```
Schema::table('flights', function ($table) {
            $table->softDeletes();
});
```

- Za provjeru je li model soft deleted:

```
if ($flight->trashed()) {
 //
}
```

- Ako želite i obrisane uključiti u upite:

```
$flights = App\Flight::withTrashed()
            ->where('account_id', 1)
            ->get();
```

Eloquent ORM

- Eloquent možemo koristiti za relacije između tablica
- Eloquent ima nekoliko tipova relacija:
  - One to One
  - One to Many
  - Many to Many
  - Has Many Through
  - Polymorphic Relations
  - Many To Many Polymorphic Relations

Eloquent ORM

- One to One
- Na primjer model User može biti povezan s jednim modelom Phone. Za definiranje relacije postavljamo metodu phone u model User
- Metoda phone treba vratiti rezultat metode hasOne u osnovnu Eloquent model klasu. Parametar metode hasOne je naziv modela

```
class User extends Model {
/*Dohvati telefon povezan s korisnikom*/
        public function phone() {
                return $this->hasOne('App\Phone');
        }
}
```

Eloquent ORM

- One to Many
- Post na blogu može imati neograničen broj komentara. Ova relacija se radi pomoću funkcije hasMany u Eloquent modelu:
- Parametar funkcije je naziv modela

```
class Post extends Model {
/*Dohvati komentare za blog */
        public function comments() {
                return $this->hasMany('App\Comment');
        }
}
```