Napredno Web programiranje

Laravel

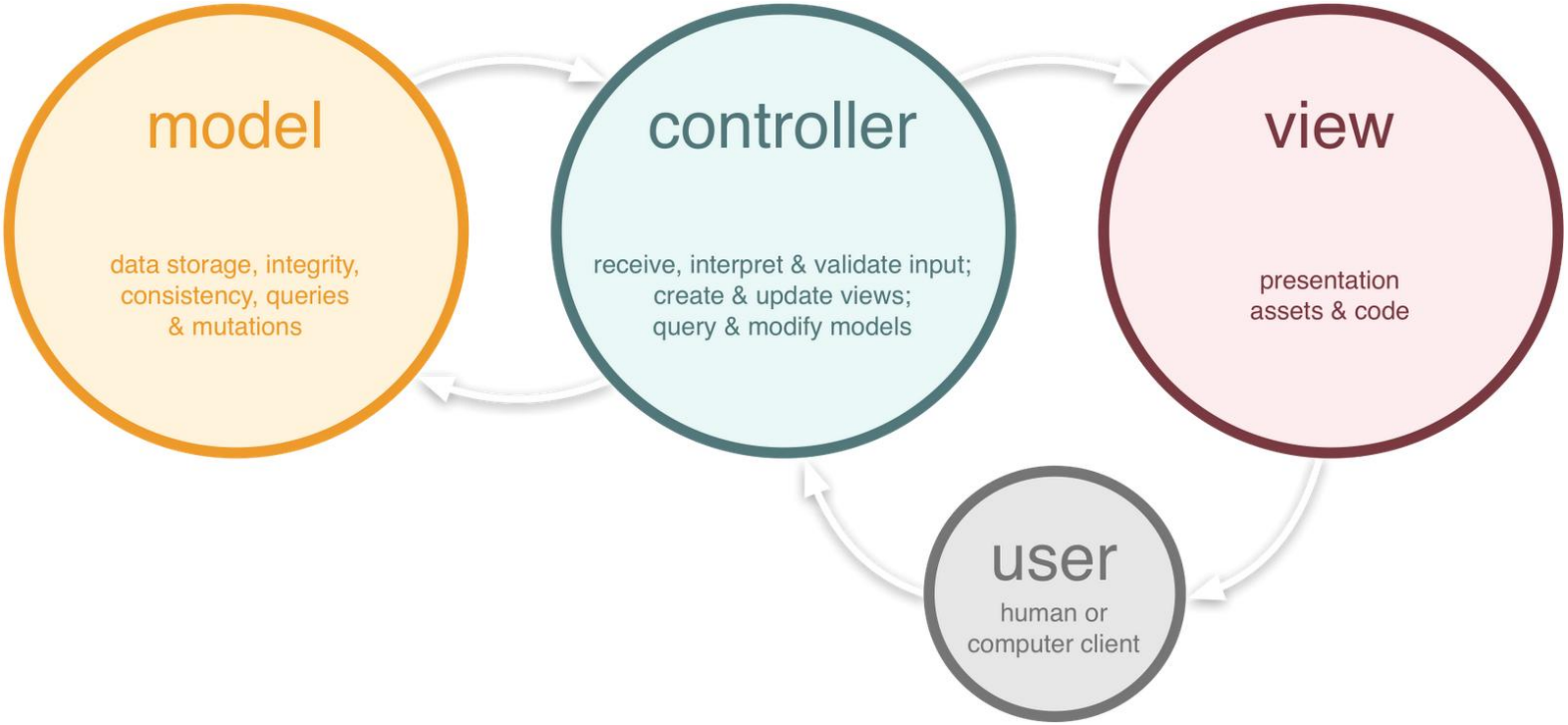
Model-View-Controller (MVC)

- MVC (eng. Model-View-Controller) je uzorak za dizajn softvera (eng. software design pattern)
- Sastoji se od tri komponente modela, pogleda i kontrolera
- Model je dio u kojem se drži poslovna logika aplikacije. To je način na koji aplikacija sprema podatke
- Ako aplikacija sprema podatke u bazu podataka u modelu će se nalaziti kod za rad sa bazom podataka, to jest kod za dohvat i spremanje podataka

Model-View-Controller (MVC)

- U pogledu se nalazi korisničko sučelje aplikacije
- Najčešće uključuje HTML, CSS i JavaScript datoteke
- Sve ono sa čime korisnik komunicira nalazi se u pogledu, a to je najčešće kombinacija više pogleda
- Kontroler je komponenta koja povezuje model i pogled. On odvaja poslovnu logiku iz modela od korisničkog sučelja i određuje kako će aplikacija odgovoriti na korisničke zahtjeve (eng. requests)
- Kontroler je prva komponenta koja se poziva jer zahtjev se prvo predaje kontroleru koji će zatim povezati traženi model i pogled

Model-View-Controller (MVC)



Model-View-Controller (MVC)

- MVC uzorak omogućuje podjelu aplikacije na više dijelova
- Ako se aplikacija razvija u većem timu posao se može podijeliti tako da svatko radi dio aplikacije
- U uobičajenim aplikacijama ti se dijelovi isprepliću i teško ih je razdvojiti
- Ako se dijelovi isprepliću teško je uočiti što će se dogoditi u drugom dijelu aplikacije ako nešto promijenimo u prvom dijelu
- Na primjer, MVC omogućuje da mijenjamo model, a da kontroler i pogled to ne primijete ako im ostavimo isto sučelje

Model-View-Controller (MVC)

- Osim MVC uzorka postoje i drugi uzorci u programiranju kao što su:

1. Singleton – samo jedna instanca klase je aktivna u jednom trenutku
2. Registry – uzorak u kojem se spremaju instance klase i pozivaju kad su potrebne
3. Factory – sučelje u kojem možete stvarati brojne instance bez definiranja o kojoj se klasi radi
4. Observer – uzorak u kojem postoji pošiljatelj i primatelj, kad se nešto promijeni pošiljatelj obavještava primatelje

Model-View-Controller (MVC)

- Da zaključimo, MVC provodi razdvajanje između poslovne logike i prezentacijske logike koja se prikazuje na GUI (eng. Graphical User Interface)
- Model – domena oko koje je sagrađena aplikacija. Zasnovan na stvarnom svijedu i njegovim objektima. Ako pravite blog Vaš model će biti post i komentar
- View – visualni prikaz modela u danom kontekstu. Obično je krajnji prikaz u pregledniku u HTML obliku. Odgovoran je za prikaz podataka korisniku.
- Controller – koordinator koji je veza između pogleda i modela. Odgovoran je za procesiranje ulaza i određivanje potrebnih akcija kao što je renderiranje pogleda i prebacivanje na drugu stranicu

Što je Laravel

- Laravel je open-source PHP web razvojno okruženje (eng. Framework)
- Napravio ga je Taylor Otwell
- Namijenjen je za razvoj web aplikacija upotrebom model-view-controller (MVC) arhitekture
- Laravel je razvojno okruženje sa elegantnom i izražajnom sintaksom kako bi olakšao razvoj web aplikacija

Zašto koristiti Laravel

- Jednostavna i izražajna sintaksa
- Laravel je dizajniran za jednog programera ili za čitav tim programera
- Laravel omogućava sinkronizaciju koristeći migracije baze podataka (eng. Migrations) i schema builder
- Koristi moderne alate kao što je ORM, rutiranje, jednostavnu autentifikaciju i brojne druge biblioteke
- Omogućava jednostavno održavanje stranice

Instalacija

- Kako bi mogli koristiti Laravel moramo ispuniti sljedeće zahtjeve
- Zahtjevi na server:
  - PHP >= 8.2
  - BCMath PHP Extension
  - Ctype PHP Extension
  - Fileinfo PHP extension
  - JSON PHP Extension
  - Mbstring PHP Extension
  - OpenSSL PHP Extension
  - PDO PHP Extension
  - Tokenizer PHP Extension
  - XML PHP Extension

Instalacija

- Najprije je potrebno instalirati composer
- Linux / Unix / OSX
  - Pokrenuti sljedeće komande u terminalu kako bi instalirali composer
    - php composer-setup.php --install-dir=bin --filename=composer
    - mv composer.phar /usr/local/bin/composer
- Windows
  - Preuzeti i instalirati composer exe https://getcomposer.org/Composer-Setup.exe
- Preuzeti Laravel instalaciju koristeći composer

  composer global require "laravel/installer"

Instalacija

- Kako bi mogli koristiti izvršne naredbe Laravela potrebno je u PATH dodati direktorij ~/.composer/vendor/bin directory
- Nakon instalacije naredba laravel new će instalirati novi projekt u direktoriju koji navedemo sa svim potrebnim direktorijima i datotekama
- Na primjer

  laravel new naziv_projekta

- Ovaj način instalacije je jednostavniji nego koristeći Composer

composer create-project --prefer-dist laravel/laravel:^12.0 naziv_projekta

Stvaranje novog projekta

- Napraviti ćemo novi projekt kako bismo pokazali strukturu direktorija i konfiguraciju projekta
- Laravel naredba new će stvoriti novu Laravel instalaciju u direktoriju koji navedete
- Upotreba naredbe:

```
laravel new zadaci
```

Konfiguracija

- Dozvole nad direktorijem (eng. Directory Permissions)
- Direktoriji za spremanje i bootstrap/cache direktoriji moraju dozvoliti pisanje web serveru ili Laravel neće raditi
  - chmod 775 /storage
  - chmod 775 /bootstrap/cache
- Ključ aplikacije (eng. Application Key) treba biti postavljen ako želimo da korisničke sesije i drugi kriptirani podaci budu sigurni
  - php artisan key:generate

Konfiguracija

- Laravel koristi DotEnv PHP biblioteku (.env)
- U novoj Laravel instalaciji unutar root direktorija nalaziti će se datoteka .env.example
- Ako instalirate Laravel koristeći Composer datoteka će automatski biti preimenovana u .env, inače ćete to morati napraviti ručno
- Ova datoteka sadrži sql host, korisničko ime i lozinku. Laravel će iz ove datoteke koristiti $_env superglobal pa u ovoj datoteci postavljate svoje okruženje
- Mod za održavanje (eng. Maintenance Mode) – kako bi ga omogućili pokrenite artisan down naredbu

  php artisan down
- Za ukidanje moda naredbu php artisan up

Struktura direktorija

- app
  - Commands
  - Console
  - Events
  - Handlers
    - Commands
    - Events
  - Http
    - Controllers
    - Middleware
    - Requests
  - Providers
  - Services
- Bootstrap
- config
- database
  - migrations
  - seeds
- public
  - package
- resources
  - lang
  - views
- routes
- storage
  - cache
  - logs
  - sessions
  - views
  - work
- tests

Struktura direktorija

- App – glavni dio aplikacije nalazi se u ovom direktoriju. Po defaultu ovaj direktorij se nalazi u namespace App i Composer ga automatski pokreće koristeći PSR-4 auto loading standard. Možete promijeniti namespace koristeći Artisan naredbu:

  app:name

- Config – sadrži konfiguracijske datoteke
- Database – sadrži migracije baze podataka i seeds za punjenje testnim podacima

Struktura direktorija

- Public – sadrži index.php datoteku koji je ulazna točka u aplikaciju. Također sadrži slike, JavaScript i CSS
- Resources – sadrži podatke za poglede (eng. Views), nekompajlirane podatke za LESS i SASS, jezične datoteke i slično
- Routes – sadrži definicije ruta aplikacije
- Storage – sadrži kompajlirane blade predloške (eng. Templates), file based sesije, cache i ostale datoteke koje stvara framework

Izrada aplikacije

- U ovom primjeru dati ćemo uvod u Laravel
- Koristiti ćemo:
  - Migracije baze podataka
  - Eloquent ORM
  - Rutiranje
  - Validaciju
  - Poglede i Blade predloške
  - Autentifikaciju
- Napraviti ćemo aplikaciju u kojoj će korisnik voditi evidenciju zadataka

Izrada aplikacije – baza podataka

- Aplikaciju možete stvoriti koristeći Composer ili kao što smo prethodno rekli sa laravel new:

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

- Prije starta potrebno je postaviti konfiguraciju za spoj na bazu podataka
- Migracije baze podataka omogućuju jednostavan način mijenjanja strukture baze podataka pomoću PHP-a
- Stvaranje tablice u bazi podataka za zadatke:

```
php artisan make:migration create_tasks_table --create=tasks
```

- Naredba make:migration stvara novu migraciju za tablicu tasks u direktoriju database/migrations

Izrada aplikacije – baza podataka

- Možete stvorit projekt s gotovim Laravel starter kitom i već ugrađenom autentifikacijom
- Za Breeze sa blade predlošcima projekt se stvara na sljedeći način:

```
composer create-project laravel/laravel moj-projekt
cd moj-projekt
composer require laravel/breeze --dev
php artisan breeze:install blade
php artisan migrate
php artisan serve
```

- U posebnom terminalu pokrenuti sljedeću naredbu za instalaciju frontenda i Vite server:

```
npm install && npm run dev
```

Izrada aplikacije – baza podataka

- Za Livewire sa blade predlošcima projekt se stvara na sljedeći način:

```
composer create-project laravel/laravel moj-projekt
cd moj-projekt
composer require laravel/breeze --dev
php artisan breeze:install livewire
php artisan migrate
php artisan serve
```

- Za Vue i React samo se umjesto livewire napiše vue ili react
- U posebnom terminalu pokrenuti sljedeću naredbu za instalaciju frontenda i Vite server:

```
npm install && npm run dev
```

Izrada aplikacije - Artisan

- Artisan je CLI uključen u Laravel
- On sadrži brojne naredbe koje možete koristiti pri razvoju aplikacije
- Artisan je pogonjen sa komponentom Symfony Console
- Kako biste vidjeli dostupne Artisan naredbe utipkajte:
  - php artisan list
- Svaka naredba uključuje help ekran koji opisuje dostupne argumente i opcije u naredbi
- Za help ekran migrate naredbe:
  - php artisan help migrate

Izrada aplikacije – baza podataka

- Korisnicima ćemo dozvoliti da stvaraju svoje račune u aplikaciji
- Trebati ćemo tablicu za spremanje korisnika
- Laravel već dolazi sa migracijom za stvaranje osnovne tablice users
- Stoga je mi ne moramo ručno stvarati
- Migracija se nalazi u direktoriju database/migrations kao i ostale migracije koje stvorimo

Izrada aplikacije – baza podataka

- Naredba make:migration stvara novu migraciju za tablicu tasks u direktoriju database/migrations
- U migraciju su već dodani atributi auto- increment i timestamps
- Mi ćemo dodati atribute name za naziv zadatka i atribut user_id koji će povezati tablice tasks i users
- Migracija je prikazana na sljedećem slajdu

IZRADA APLIKACIJE – BAZA PODATAKA

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class CreateTasksTable extends Migration {
/*Funkcija koja pokreće migraciju */
    public function up() {
      Schema::create('tasks ', function (Blueprint $table) {
          $table->increments('id');
          $table->integer('user_id')->unsigned()->index();
          $table->string('name');
          $table->timestamps();
       });
    }
/*Poništava migraciju*/
     public function down() {
          Schema::drop('tasks');
      }
}
```

Izrada aplikacije – baza podataka

- Za pokretanje migracije koristi se Artisan naredba migrate:

  ```
  php artisan migrate
  ```

- Ova naredba će stvoriti sve tablice u bazi podataka, to jest users i tasks
- Ako sada pogledate bazu podataka vidjeti ćete stupce koje smo dodali u tablice
- Iduće što je potrebno je napraviti Eloquent ORM model za naše zadatke

Izrada aplikacije – Eloquent ORM

- Eloquent je Laravel-ov ORM (object-relational mapper)
- Eloquent omogućava da na jednostavan način dohvaćate i spremate podatke u bazu podataka koristeći modele
- Svaki Eloquent model komunicira direktno sa jednom tablicom u bazi podatka
- Trebamo definirati model Task koji će komunicirati s tablicom tasks
- Model User je već stvoren
- Koristimo Artisan naredbu make:model:

```
php artisan make:model Task
```

Izrada aplikacije – Eloquent ORM

- Model će biti u app direktoriju
- Po defaultu model klasa je prazna
- Ne moramo joj reći sa kojom tablicom komunicira jer je naziv tablice množina naziva modela
- U ovom slučaju model Task komunicira sa tablicom tasks
- Ovako treba izgledati prazni model:

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Task extends Model {
//
}
```

Izrada aplikacije – Eloquent ORM

- Za model Task trebamo reći da je atribut name „mass-assignable” kako bismo ga mogli popuniti metodem create u Eloquent ORM-u

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Task extends Model {
/*Atributi koji su mass assignable trebaju biti dodani u niz.*/
        protected $fillable = ['name', 'description'];
}
```

Izrada aplikacije – Eloquent ORM

- Nakon što napravimo modele potrebno je napraviti relacije među modelima
- Korisnik može imati više zadataka, ali zadatak može biti dodijeljen samo jednom korisniku
- Ako ispravno definiramo relacije vrlo lako možemo pretražiti zadatke za pojedinog korisnika

```
$user = App\User::find(1);
foreach ($user->task as $task) {
        echo $task->name;
}
```

Izrada aplikacije – Eloquent ORM

- Za model User ćemo definirati relaciju za zadatke pomoću metode hasMany iz Eloquent-a
- Definiramo da jedan student može imati više zadataka

```
namespace App;
use Illuminate\Foundation\Auth\User as Authenticatable;
class User extends Authenticatable {
        public function tasks() {
                return $this->hasMany(Task::class);
        }
}
```

Izrada aplikacije – Eloquent ORM

- Za model Task ćemo definirati relaciju za User pomoću metode belongsTo iz Eloquent-a
- Definiramo da zadatak može pripadati samo jednom korisniku

```
namespace App;
use App\User;
use Illuminate\Database\Eloquent\Model;
class Task extends Model {
        protected $fillable = ['name'];

        /*user kome pripada task*/
        public function user() {
                return $this->belongsTo(User::class);
        }
}
```

Izrada aplikacije – 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 file koja je uključena u svaki novi projekt
- Ruta za prikaz svih zadataka, ruta za dodavanje novog zadatka i ruta za brisanje zadatka
- Sve ćemo rute postaviti u međusloj (eng. Middleware) tako da imaju svoje sesijsko stanje i CSRF zaštitu

Izrada aplikacije – 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) {
//
 });
```

Izrada aplikacije – autentifikacija

- Korisnici trebaju moći stvoriti svoj račun i prijaviti se na stranicu
- U Laravel-u se to može napraviti vrlo jednostavno
- U aplikaciju je uključen kontroler

  app/Http/Controllers/Auth/AuthController
- Kontroler koristi svojstvo

  AuthenticatesAndRegistersUsers
- Ono sadrži svu potrebnu logiku za stvaranje i autentifikaciju korisnika

Izrada aplikacije – autentifikacija

- Na korisnicima je da naprave predloške za registraciju i prijavu, te definiraju rute koje će voditi do kontrolera za autentifikaciju
- Sve ovo se može napraviti pomoću naredbe

  make:auth Artisan
- Potrebno je napraviti rute za autentifikaciju u route datoteci
- To možemo obaviti pomoću metode auth koja će registrirati sve rute za registraciju, login i reset lozinke:

  Route::auth();

Izrada aplikacije – autentifikacija

- Nakon što su rute registrirane potrebno je postaviti svojstvo $redirectTo u kontroleru app/Http/Controllers/Auth/LoginController da je jednako:

```
protected $redirectTo = '/tasks';
```

- I dodati putanju za preusmjeravanje u datoteku app/Http/Middleware/RedirectIfAuthenticated.php:

```
return redirect('/tasks');
```

- Ovisno o načinu instalacije koji se koristi neke verzije nemaju LoginController pa treba postavit u kontroleru app/Http/Controllers/Auth/AuthenticatedSessionController u funkciji store() postavi:

```
return redirect()->intended(route('tasks', absolute: false));
```

Izrada aplikacije – autentifikacija

- Za dohvat i spremanje zadataka potrebno je napraviti TaskController upotrebom Artisan CLI

php artisan make:controller TaskController

- Kontroler će se nalaziti u app/Http/Controllers direktoriju
- U datoteci app/Http/routes.php treba dodati rute koje će pokazivati na kontroler za starije verzije Laravela te u routes/web.php za novije verzije:

```
Route::get('/tasks', 'TaskController@index');
Route::post('/task', 'TaskController@store');
Route::delete('/task/{task}', 'TaskController@destroy');
```

Izrada aplikacije – autentifikacija

- Sve rute za dohvat zadataka moraju proći autentifikaciju, to jest samo im mogu pristupati prijavljeni korisnici. Laravel to obavlja pomoću međusloja. Svi međuslojevi definirani su u datoteci app/Http/Kernel.php
- Svim akcijama u kontroleru dajemo auth međusloj

```
namespace App\Http\Controllers;
use App\Http\Requests;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
class TaskController extends Controller {
        public function __construct() {
                $this->middleware('auth');
}}
```

Izrada aplikacije – raspored (layout)

- Većina aplikacija dijeli isti izgled na većini stranica
- Na primjer sve stranice imaju identičnu navigaciju
- Laravel olakšava dijeljenje zajedničkih osobina na svim stranicama upotrebom Blade Layouta
- Svi Laravel pogledi nalaze se u resources/views
- Napraviti ćemo layout u resources/views/layouts/app.blade.php
- .blade.php ekstenzija govori da se koristi Blade za izradu predložaka iako možemo koristiti obični PHP

Izrada aplikacije – raspored (layout)

- Konačni izgled app.blade.php

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <title> Lista zadataka za obaviti </title>
                <!-- CSS And JavaScript -->
        </head>
        <body>
                <div class="container">
                        <nav class="navbar navbar-default">
                        <!-- Navbar sadržaj -->
                        </nav>
                </div>
                @yield('content')
        </body>
</html>
```

Izrada aplikacije – raspored (layout)

- Bitno je napomenuti da @yield('content') unutar layouta predstavlja Blade direktivu koja govori gdje će se ubaciti sadržaj stranica koje nasljeđuju layout
- Potrebno je napraviti pogled koji koristi ovaj layout i u njega smješta svoj sadržaj
- Ako odemo na pogled vidjeti ćemo da sadrži sljedeće dvije linije kojima nasljeđuje layout i govorimo odakle kreće sadržaj koji se ubacuje u layout

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

Izrada aplikacije – View

- Sada želimo prikaz koji će imati formu za dodavanje novog zadatka i listu za prikaz svih zadataka
- U Laravelu svi HTML predlošci su spremljeni u direktoriju resources/views
- Davanje pogleda kao parametra funkciji task će stvoriti View objekt instancu koja odgovara predlošku u resources/views/tasks/index.blade.php.
- Za dohvat predložaka koristi se view helper

```
Route::get('/', function () {
        return view('tasks.index');
});
```

Izrada aplikacije – View

- Za dohvat predložaka osim gornje get rute može se koristiti i index metoda u TaskController koju ćemo i mi koristiti

```
public function index(Request $request) {
        return view('tasks.index');
}
```

Izrada aplikacije – View

- Sljedeće što je potrebno je napraviti pogled
- U pogledu treba biti forma za dodavanje novih zadataka i lista za ispis svih zadataka
- Na sljedećim slajdovima je izgled pogleda

```
@extends('layouts.app')
@section('content')
<div class="container">
    <div class="col-sm-offset-2 col-sm-8">
        <div class="panel panel-default">
            <div class="panel-heading">
                Dodaj novi zadatak
            </div>
```

Izrada aplikacije – View

```
<div class="panel-body">
    <!– Prikaz pogreški-->
    @include('common.errors')
    <form action="{{ url('task')}}" method="POST"
class="form-horizontal">
        {{ csrf_field() }}
        <div class="form-group">
            <label for="task-name" class="col-sm-3 control-
label">Naziv zadatka</label>
            <div class="col-sm-6">
                <input type="text" name="name" id="task-
name" class="form-control" value="{{ old('task') }}">
            </div>
        </div>
```

Izrada aplikacije – View

```
<div class="form-group">
                    <div class="col-sm-offset-3 col-sm-6">
                      <button type="submit" class="btn btn-
default">
                       <i class="fa fa-btn fa-plus"></i>Dodaj zadatak
                      </button>
                    </div>
                </div></form></div> </div>
```

Izrada aplikacije – View

```
<!-- Trenutni zadatak -->
@if (count($tasks) > 0)
<div class="panel panel-default">
    <div class="panel-heading">
        Svi zadaci
    </div>
    <div class="panel-body">
        <table class="table table-striped task-table">
            <thead>
            <th>Naziv zadatka</th>
             <th> </th>
            </thead>
```

Izrada aplikacije – View

```
<tbody>
@foreach ($tasks as $task)
<tr>
<td class="table-text"><div>{{ $task->name }}</div></td>
<!-- Gumb za brisanje -->
<td>
<form action="{{ url('task/'.$task->id) }} method="POST">
 {{ csrf_field() }}
 {{ method_field('DELETE') }}
<button type="submit" class="btn btn-danger">
   <i class="fa fa-btn fa-trash"></i>Obriši</button>
</form></td> </tr>@endforeach
</tbody></table></div></div>@endif</div></div>@endsection
```

Izrada aplikacije – dodaj zadatak

- Potrebno je napraviti kod koji će se pokrenuti za POST /task rutu i dodati novi zadatak
- Zbog sigurnosti potrebno je validirati unos kroz input polje
- Zbog jednostavnosti postaviti ćemo da name ne može biti duži od 255 znakova
- Ako unos ne prođe validaciju korisnik će biti preusmjeren na / URL, isto tako će se upisati stari unos i pogreška u sesijske varijable
- To nam omogućava da zadržimo korisnikov unos i ako se dogode pogreške pri validaciji

Izrada aplikacije – dodaj zadatak

- Kod u routes/web.php
- U rute se može dodati kod koji odrađuje validaciju i vrši spremanje u bazu
- Preporuka je koristiti Controller

```
Route::post('/task', function (Request $request) {
    $validator = Validator::make($request->all(), [
            'name' => 'required|max:255'
            ]);
    if ($validator->fails()) {
            return redirect('/')
            ->withInput()
            ->withErrors($validator);
    }
});
```

Izrada aplikacije – dodaj zadatak

- Kod umjesto u rutu dodajemo u kontroler
- Slijedi kod za metodu TaskController@store

```
public function store(Request $request) {
        $this->validate($request, [
          'name' => 'required|max:255', ]);
        // Napravi novi zadatak
}
```

- Kod za validaciju je drukčiji jer u kontroleru možemo iskoristiti svojstvo ValidatesRequests koje je uključeno u base Laravel controller
- Ne moramo ručno provjeriti je li propala validacija i raditi preusmjeravanje
- Korisnik se preusmjerava odakle je došao i ispisuje poruka o pogrešci

Izrada aplikacije – dodaj zadatak

- Poziv validatora - >withErrors($validator) će spremiti pogrešku iz trenutne instance validatora u sesijske varijable
- Vrijednostima spremljenim u sesiju može se pristupiti preko $errors varijable u pogledu
- U ranijim pogledima je uključena direktiva @include('common.errors') kako bi se mogle prikazati pogreške u validaciji
- common.errors direktiva omogućava da se prikažu pogreške validacije na svim stranicama
- $errors varijabla je prazna ako nema pogrešaka, to jest prazna instanca klase ViewErrorBag

Izrada aplikacije – dodaj zadatak

- Kod za pogled koji prikazuje pogreške u resources/views/common/errors.blade.php

```
@if count ( ($errors) > 0)
<div class="alert alert-danger">
        <strong>Dogodila se nekakva pogreška!</strong>
        <br><br>

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

Izrada aplikacije – dodaj zadatak

- Nakon što se obavi validacija potrebno je dodati novi zadatak
- To se može obaviti pomoću save metode nakon što postavimo svojstva novog Eloquent modela u routes/web.php
- Kod je prikazan na sljedećem slajdu

IZRADA APLIKACIJE – DODAJ ZADATAK

```
Route::post('/task', function (Request $request) {
   $validator = Validator::make($request->all(), [
            'name' => 'required|max:255',
]);
   if ($validator->fails()) {
            return redirect('/')
            ->withInput()
            ->withErrors($validator);
   }
$task = new Task;
$ task >name = $request->name;
$ task >save();
return redirect('/');
});
```

Izrada aplikacije – dodaj zadatak

- Nakon što se doda novi zadatak korisnik se preusmjerava na URL /tasks
- Za stvaranje zadatka koristimo Eloquent relacije
- Laravel relacije imaju create metodu koja prima niz atributa i automatski stvara strani ključ prema povezanom modelu prije spremanja u bazu
- U ovom slučaju strani ključ će biti user_id trenutno prijavljenog korisnika kojem pristupamo sa $request->user():

Izrada aplikacije – dodaj zadatak

```
/*Stvaranje novog zadatka*/
public function store(Request $request) {
   $this->validate($request, [
            'name' => 'required|max:255', ]);
            $request->user()->tasks()->create([ 'name' =>
$request->name, ]);
            return redirect('/tasks');
}
```

Izrada aplikacije – prikaz zadataka

- Potrebno je urediti rutu / za prikaz svih zadataka u pogledu
- View funkcija prihvaća i drugi argument koji je u biti niz podataka koji će biti dostupni pogledu
- Svaki ključ u upitu će postati varijabla u pogledu

```
Route::get('/', function () {
        $tasks = Task::orderBy('created_at', 'asc')->get();
        return view('tasks', [
                'tasks' => $tasks
        ]);
});
```

Izrada aplikacije – prikaz zadataka

- Nakon što se podaci prebace pogledu moguće je dobiti sve zadatke u tasks/index.blade.php pogledu i prikazati ih u tablici
- @foreach petlja nam u Blade-u dozvoljava da na jednostavan način dohvatimo sve podatke pomoću PHP-a
- Sljedeća stvar koju je potrebno napraviti je brisanje zadataka
- Gumb za brisanje je već ranije dodan u pogled tasks/index.blade.php

Izrada aplikacije – prikaz zadataka

- Isto se preko kontrolera obavlja uređivanjem TaskController@index metode koja prosljeđuje sve zadatke u pogled
- Prvi parametar view metode je pogled, a drugi niz zadataka koje prosljeđujemo pogledu

```
/*Prikaz svih zadataka */
public function index(Request $request) {
        $tasks = $request->user()->tasks()->get();
        return view('tasks.index', [ 'tasks' => $tasks, ]);
}
```

Izrada aplikacije – prikaz zadataka

- Laravel ima mogućnost upotrebe dependency injection
- Može se ubaciti TaskRepository u kontroler TaskController koji ćemo koristiti za pristup podacima
- TaskRepository će čuvati svu logiku za pristup podacima za Task model što postaje korisno rastom aplikacije kad moramo dijeliti upite cijelom aplikacijom
- Treba u direktorij app/Repositories dodati klasu TaskRepository

Izrada aplikacije – prikaz zadataka

```
namespace App\Repositories;
use App\User;
class TaskRepository {
/*Dohvati sve zadatke za danog korisnika.  Vraća se kolekcija,
@return Collection */
public function forUser(User $user) {
   return $user->tasks() ->orderBy('created_at', 'asc') ->get(); }
}
```

Izrada aplikacije – prikaz zadataka

- Repozitorij možemo ga koristiti u kontroleru

```
namespace App\Http\Controllers;
use App\Task;
use App\Http\Requests; use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Repositories\TaskRepository;
class TaskController extends Controller {
protected $tasks;
/*Nova instanca kontrolera*/
public function __construct(TaskRepository $tasks) {
 $this->middleware('auth');
 $this->tasks = $tasks; }
 /*Prikaz svih zadataka*/
public function index(Request $request) {
            return view('tasks.index', [ 'tasks' => $this->tasks-
>forUser($request->user()), ]);
} }
```

Izrada aplikacije – brisanje zadataka

- Kad se klikne na gumb za brisanje biti će pozvan zahtjev DELETE /task
- Forma u kojoj je gumb za brisanje koristi POST iako mi odgovaramo na zahtjev kroz Route::delete route
- HTML forme dozvoljavaju samo GET i POST HTTP zahtjeve pa moramo lažirati DELETE zahtjev preko forme. To možemo napravit slanjem izlaza funkcije method_field('DELETE') Funkcija stvara hidden input polje koje će Laravel prepoznati i prebrisati HTTP zahtjev

```
<input type="hidden" name="_method" value="DELETE">
```

Izrada aplikacije – brisanje zadataka

- Sljedeće što je potrebno je dodati logiku u rutu kako bi obrisali željeni zadatak. Možemo koristiti povezivanje s modelom (eng. Model binding) kako bi automatski dohvatili Zadatak model koji odgovara {zadatak} parametru u ruti
- Unutar callback-a u našoj ruti koristiti ćemo delete metodu za brisanje zadatka. Nakon što obrišemo zadatak korisnika preusmjeravamo na / URL

```
Route::delete('/task/{id}', function ($id) {
        Task::findOrFail($id)->delete();
        return redirect('/');
});
```

Izrada aplikacije – brisanje zadataka

- Umjesto brisanja u ruti to možemo napraviti u kontroleru pomoću destroy metode. Prvo treba pogledati deklaraciju rute i metodu u kontroleru koja odgovara na rutu
- Varijabla {task} u ruti odgovara varijabli $task u metodi kontrolera pa će Laravelov implicit model binding automtaski ubaciti odgovarajuću instancu modela Task

```
Route::delete('/task/{task}', 'TaskController@destroy');

public function destroy(Request $request, Task $task) {
        // Obriši zadatak
}
```

Izrada aplikacije – brisanje zadataka

- Iako smo ubacili instancu modela u destroy metodu nismo napravili provjeru je li korisnik vlasnik zadatka kojeg želi obrisati
- Korisnik može obrisati tuđi zadatak
- Potrebno je provesti autentifikaciju
- Laravel koristi "policies" za autentifikacijsku logiku gdje svaka policy odgovara jednom modelu
- Mi trebamo napraviti TaskPolicy pomoću Artisan CLI u app/Policies

```
php artisan make:policy TaskPolicy
```

Izrada aplikacije – brisanje zadataka

- Dodajemo destroy metodu u policy koja provjerava ima li korisnik pravo brisanja

```
namespace App\Policies;
use App\User; use App\Task;
use Illuminate\Auth\Access\HandlesAuthorization;
class TaskPolicy {
use HandlesAuthorization;
/*Smije li korisnik obrisati zadatak, vraća bool*/
   public function destroy(User $user, Task $task) {
            return $user->id === $task->user_id;
   }
}
```

Izrada aplikacije – brisanje zadataka

- Na kraju se povezuje Task model sa TaskPolicy dodavanje linije koda u datoteku app/Providers/AuthServiceProvider.php i njeno svojstvo $policies
- Ovo obavještava Laravel koja se policy koristi za autentifikaciju instance Task

```
protected $policies = [ 'App\Task' =>
'App\Policies\TaskPolicy', ];
```

Izrada aplikacije – brisanje zadataka

- Jednom kad je policy definiran može se koristiti u destroy metodi
- Svi kontroleri mogu pozvati metodu authorize koja je izložena u svojstvu AuthorizesRequest

```
public function destroy(Request $request, Task $task) {
        $this->authorize('destroy', $task);
        //Ako nije prošla autentifikacija baciti će se
        //iznimka 404 i ispisati poruka o pogrešsci
        // Obriši zadatak ako je prošla autentifikacija
        $task->delete();
        return redirect('/tasks.index');
}
```