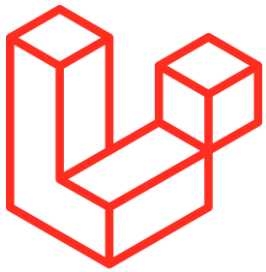


Laravel

MVC web framework for PHP

Radni okvir za veb (eng. *Web framework*)

- Objektno orijentisan (OO) sistem konstruisan kako bi ga programeri proširili na takav način da obezbede funkcionalnosti koje im se zahtevaju.

- Najbolje programerske prakse su već ugrađene u samom radnom okviru.

Loš pristup

- Jedna PHP skripta, koja radi:
  - generisanje dinamičkog HTML-a,
  - poslovnu logiku (kada se kakvi postupci i akcije pokreću),
  - komunikaciju sa bazom podataka,
  - izdvajanje parametara iz _GET i _POST promenljivih,
  - rukovanje svim greškama,
  - komunikaciju sa drugim (veb) servisima,
  - ...
- Nešto kao božanska klasa u OO uzorcima
- Teško za održavanje, teško za razumevanje

Bolji pristup - *separation of concerns (SoC)*

- Razdvajanje odgovornosti
- Monolitna aplikacija → Raslojena aplikacija
- Svaki sloj i komponenta rade samo jednu stvar i ništa osim toga
  - Bolja čitljivost programskog koda
  - Bolji uslovi za testiranje koda
- Primenjujemo razne projektne uzorke:
  - MVC (*Model-View-Controller*)
  - MVVM (*Model-View-ViewModel*)
  - MVP (*Model-View-Presenter*)

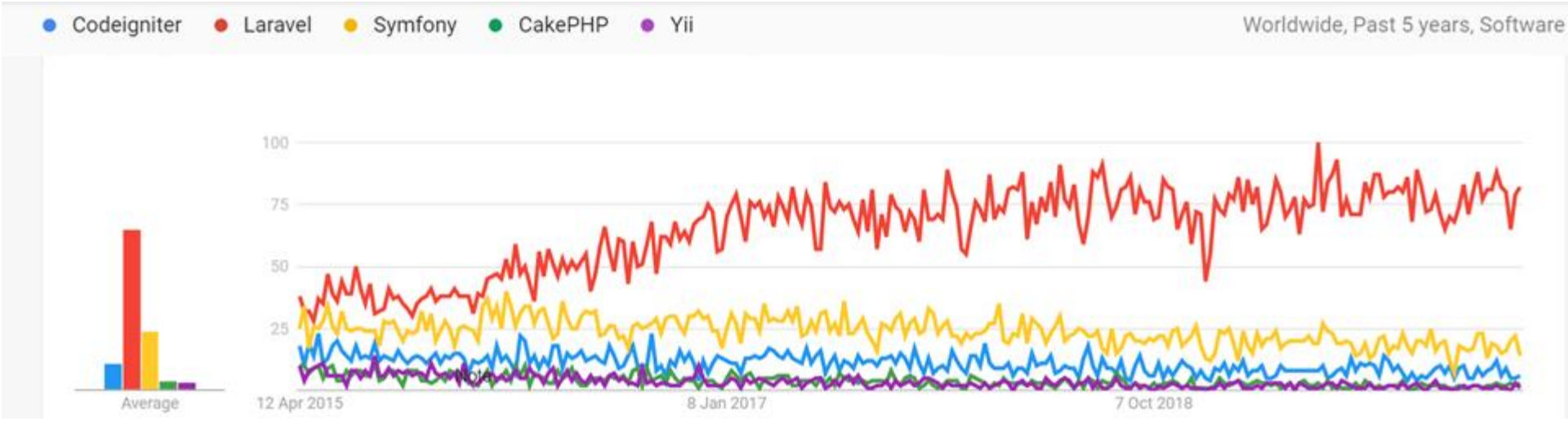
Princip dizajna u veb programiranju

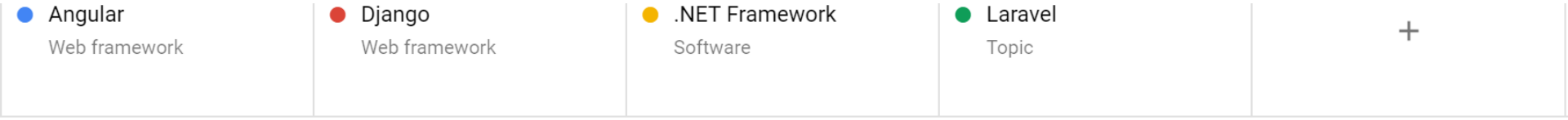
- Celu veb aplikaciju najčešće delimo u 3 celine:
  - Stil i prezentacija: vizuelni izgled veb aplikacije, korisnički interfejs (UI)
  - Poslovna (biznis) logika: način na koji se veb aplikacija ponaša kao odgovor na interakciju od strane korisnika (korisničke radnje)
  - Sadržaj: stvarni podaci koji se prezentuju, najčešće čitanjem iz neke baze podataka (podaci o studentima, predmetima, vesti, itd.)

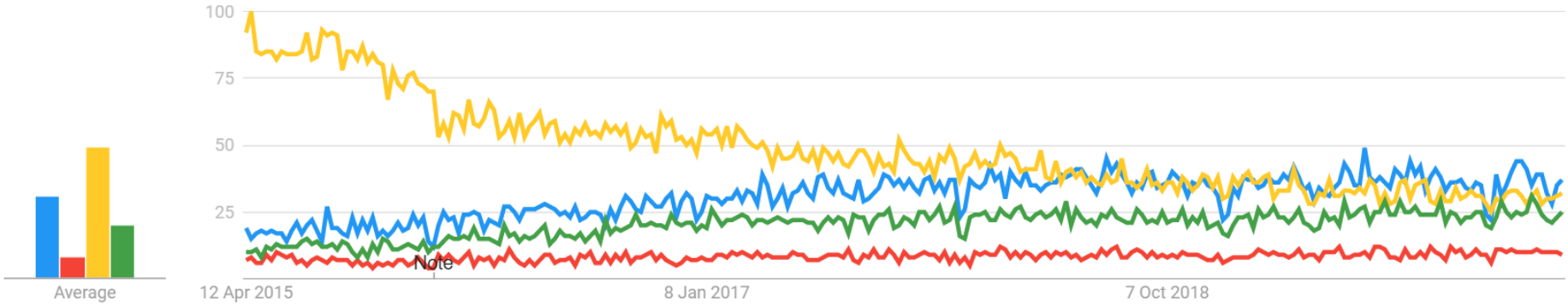
Danas

- Radni okviri postoje i za veb, ali i desktop aplikacije
- Neki su komercijalni, neki otvorenog koda
  - Java Web: JSP, Struts2, Grails, Play, Vaadin, **JSF, Spring,**...
  - Java Desktop: Swing, Griffon, **JavaFX,**...
  - Java Script: **Angular 2+**, **React**, Vue.js, Ember.js, **Node.js**, Meteor, Backbone.js,...
  - .NET Web: ASP .Net, **ASP MVC,**...
  - .NET Desktop: **WPF**, Windows Forms
  - Php: **Codeigniter, Laravel**, Symfony, CakePHP, Yii, Zend,…
  - Python: **Django**, Flask
  - Ostali: Ruby on Rails, Xamarin,...
  - CSS: **Bootstrap,** Gumby, Foundatin, YAML...
  - Ima ih još MNOGO!
- Odabir? Pametno odabrati, loš izbor košta!!!

Popularnost radnih okvira







Odabir tehnologije / radnog okvira

- Programski jezik: Java, Scala, Groovy, PHP, Ruby, Python, C#, JavaScript,...
  - Šta članovi tima dobro poznaju?
  - Šta nije teško za učenje?
  - Šta je produktivno za programiranje?
    - Tipiziranost, postojeće biblioteke...
- Platforma: JVM, PHP, RVM, .NET,...
  - Šta je robusno?
  - Šta je skalabilno?
  - Koja je namena i kakva je planirana upotreba aplikacije? Koliko brzo treba „izbaciti“ prvu verziju?
- Radni okvir: utiču prethodne dve odluke
  - Šta članovi timova znaju?
  - Kakva je kriva učenja?
  - Koliko je podržan (zajednica, biblioteke, dodaci)?
  - Koliko je produktivno za programiranje?
  - Iskustva drugih?
  - Kakva je namena aplikacije? (nema *one-fits-all* rešenja)

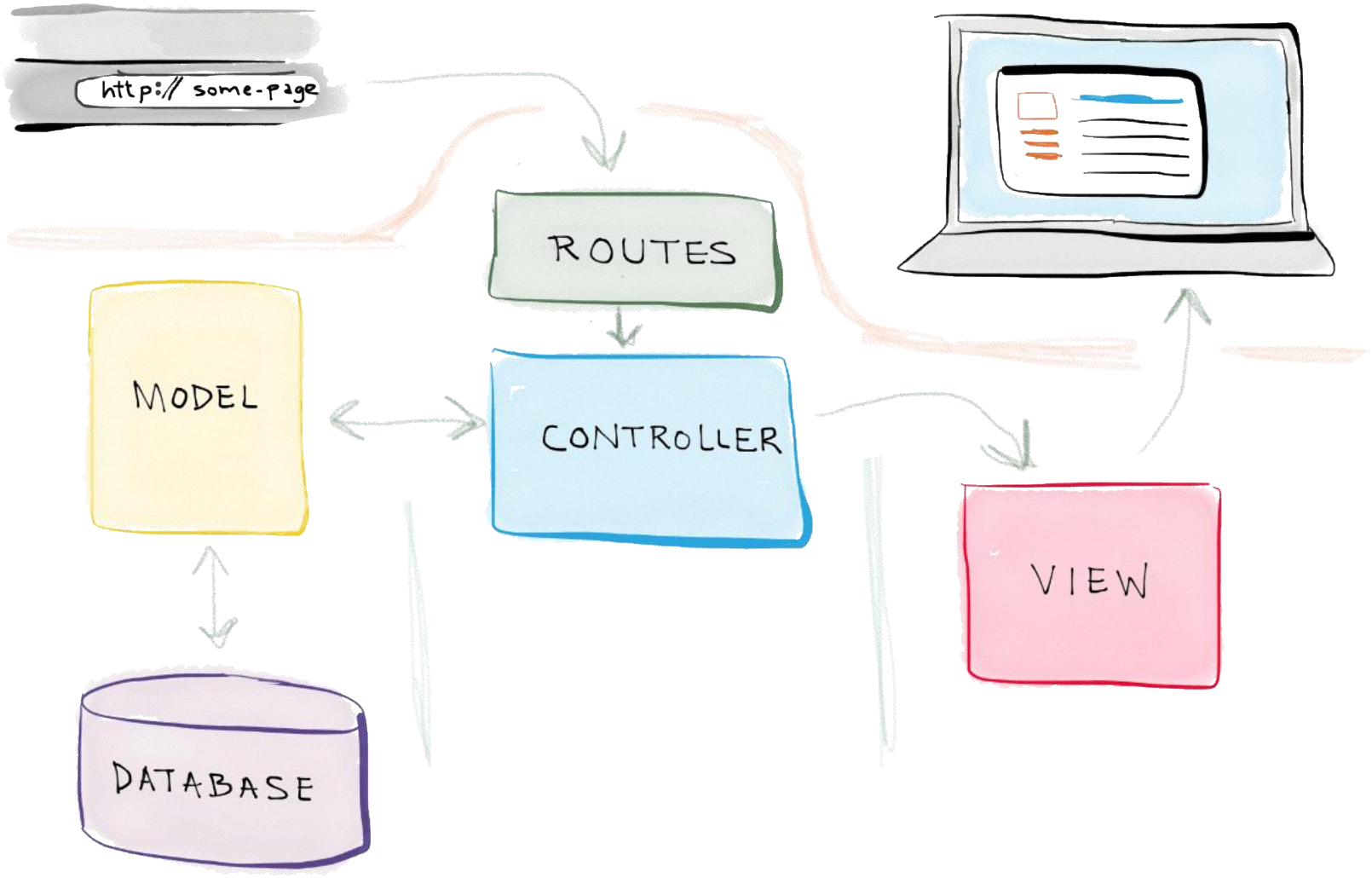
Dogovor umesto konfigurisanja

- Princip *Convention over configuration* je danas jako popularan
- Projekti najčešće imaju identičnu, logičnu strukturu
- Ranije – sve kroz konfiguracione fajlove
  - Java aplikacije: kroz serije XML fajlova, od Java 1.5 kroz anotacije
- Danas: usvojena je razumna pretpostavka
- Aplikacija se strukturirana po unapred definisanim pravilima, uz mogućnost konfigurisanja neočekivanih odluka (*override*)
- Radni okvir forsira određenu strukturu da bi kod obavljao ono što želite
- Posledica: svi učesnici projekta znaju kako su unutar projekta razvrstane komponente; lako je nastaviti tuđi započeti rad, lako pronaći mesto na kome se pojavljuje defekat (*bug*)
- Don’t reinvent the wheel!

MVC ideja

- Razdvojiti model od prezentacije
  - Generisanje dinamičkog HTML-a, putem echo: **view**
  - Manipulacija podacima: **controller**
  - Podaci koji se razmenjuju između C i V: **model**
    - Tu „pripada“ i dobavljanje podataka iz neke DB
- Grupisati tešku poslovnu logiku na jedno mesto, veb orijentisane stvari na drugo (*controller*)

Projektni uzorak MVC



Projektni uzorak MVC

- Model
  - Podaci kojima se manipuliše kroz aplikaciju
  - Domen problema (Ljude, Knjige,...)
  - Služi da nosi informacije između *Controller*-a i *View*-a
  - Zadužen da perzistira podatke (sama komunikacija sa DB)
  - Zadužen da dobavlja podatke iz baze podataka
- View
  - Komponenta koja prikazuje podatke (*model*) na određeni način
- Controller
  - Komponenta koja prima zahteve (korisnika), pokreće određenu poslovnu logiku, odlučuje šta se čuva u bazi podataka i kada

Dodatak: Services

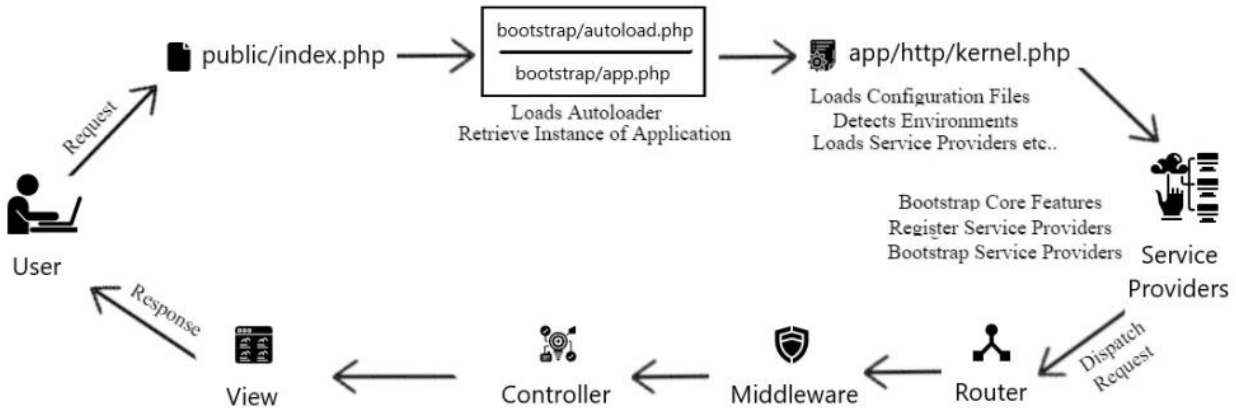
- Ideja: ne opteretiti kontrolere business logikom
- Kontroler odlučuje koji view prikazati, i koje podatke tada prikazivati, ili kome proslediti izvršavanje
- Ne bi trebalo da sadrži čitav proces i donošenje odluka oko domenskog problema
- To (mozganje) ostaviti servisima
  - Posebne klase koje sadrže kod koji ostvaruje business logiku, ali bez svesti o tome da je business logika potrebna web aplikaciji
- Tada kontroler samo “diriguje”

LARAVEL

MVC unutar Laravel-a

- Svaki radni okvir ima sopstvene mehanizme na osnovu kojih ostvaruje MVC
- Pored MVC, u Laravel-u se ostvaruje:
  - *Loose coupling*: povezivanje komponenti na labav način – ne referencirati direktno klase međusobno, kako bi sistem bio modularan
  - *Dynamic loading*: komponente se učitavaju na eksplicitan zahtev, po potrebi za korišćenjem
    - Štedi se na memoriji i vremenu izvršavanja zahteva

MVC i Laravel



- Zahtev (*request*) ide do index.php, a zatim:
  - analizira se URL (*routing*);
  - zahtev se procesira zbog sigurnosnih razloga;
  - instancira se odgovarajući kontroler i poziva se odgovarajuća metoda (akcija);
  - kontroler izvršava potrebnu obradu uz pomoć modela, biblioteki, pomoćnih funkcija i svih drugih resursa potrebne za obradu specifičnog zahteva;
  - definiše se odgovarajući pogled koje će korisniku biti prikazan.

Priprema radnog okruženja

- Priprema radnog okruženja moguća je na sledeći način:
  - Korišćenjem composer-a
    - Composer-a možete preuzeti sa https://getcomposer.org/
    - Potrebno je izvršiti sledeću komandu iz direktorijuma u kome želite da vam se nalazi root projekta:

```
composer create-project laravel/Laravel
example-app
```

    - Composer omogućava jednostavan prelazak na noviju verziju radnog okvira, pa se preporučuje njegovo korišćenje

Priprema radnog okruženja

- Sadržaj početne stranice se može videti na linku:
  http://localhost/etfvesti/public/index.php

- Laravel sadrži lokalni server za razvoj koji koristi ugrađeni PHP server i radi mapiranje ruta.
  Pokreće se izvršavanjem komande:
  ```
  php artisan serve
  ```
  i tada projektu možete pristupiti i putem linka:
  http://localhost:8000/

Režim razvoja aplikacije (eng. *Development mode*)

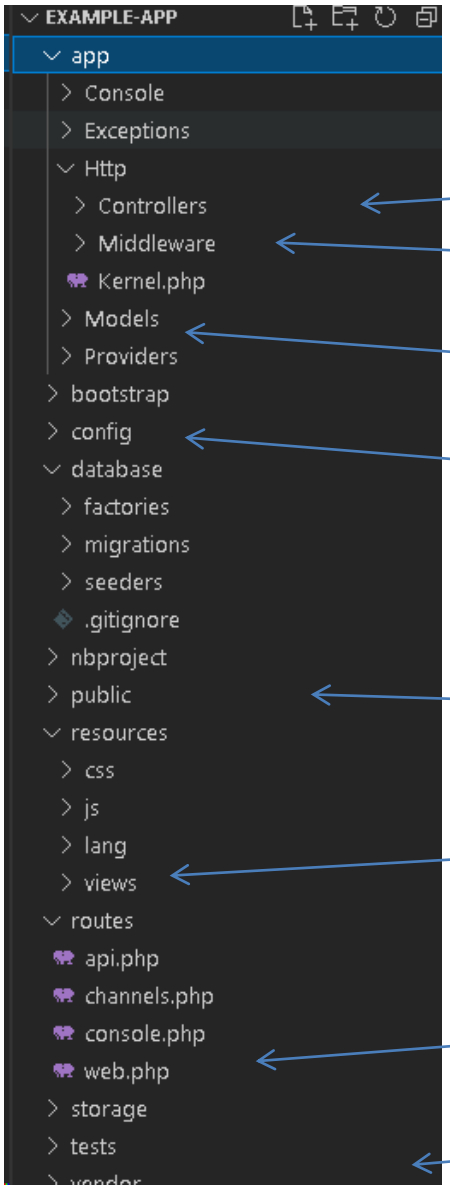
- Projekat se podrazumevano pokreće u produkcionom režimu (eng. *Production mode*).

- Projekat je moguće pokrenuti u režimu razvoja, ako se u *root* direktorijumu napravi fajl pod nazivom **.env** i doda sledeća linija:

```
APP_DEBUG=true
APP_ENV=local
```

- Razvojni režim omogućava prikazivanje svih nastalih grešaka u pretraživaču, dok se u produkcionom režimu one ne prikazuju.

Struktura aplikacije



Klase koje implementiraju poslovnu logiku

Klase za filtriranje zahteva

Klase koje omogućavaju rad sa podacima

Konfiguracioni fajlovi

Ovde se nalazi index.php fajl

PHP/Blade skripte koje generišu HTML

Mesto gde se definišu rute za mapiranje URL-ova ka kontrolerima

Sistemski fajlovi

Kako funkcioniše?

- Kliknete na link
  http://localhost/etfvesti/public/index.php/vest/3
  ili
  http://localhost:8000/vest/3

- U web.php je definisana sledeća linija:

```
Route::get('/vest/{id}', [\App\Http\Controllers\BaseController::class, 'vest'])->name('vest');
```

- Server vidi da treba doći do index.php i predaje joj URL
- Laravel analizira URL na osnovu konfiguracije ruta
  - Kontroler: BaseController
  - Akcija: vest
  - Parametar: 3
  - Pravi objekat klase BaseController, poziva se metoda vest() sa parametrom 3

CONTROLLER I VIEW

Controller

- Izveden je iz klase App\Http\Controllers\Controller
- Naziv kontrolera - Prvo veliko slovo i ostala mala
- Fajl u kome se nalazi kontroler potrebno je da se zove identično kao i klasa kontroler
- Logički grupisan skup akcija
- Akcija - metoda unutar kontrolera
  - Obavlja nekakvu obradu (možda sa BP)
  - Rezultuje prikazom (dinamički) generisanog HTML

View

- PHP skripta koja generiše HTML
- Kontroler odlučuje koji pogled (*view*) se koristi i prosleđuje mu podatke
- Elementi prosleđenog niza su dostupni u navedenom pogledu kao obične php promenljive
- Putanja do pogleda je relativna u odnosu na *views* direktorijum

- Primer:

```
return view('vesti',
    ['poruka'=>'Prikaz svih vesti',
    'vesti'=>$niz_vesti]);
```

Putanja do view-a bez ekstenzije

Niz sa podacima koji se šalju stranici

Modularnost view-ova

- Često se ponavljaju delovi veb stranica
  - header, footer, menu, navigation,...
- Napraviti šablonsku (*template*) stranicu koji sadrži header.php, footer.php i mesto za centralni deo stranice
  - Stranica koju želimo da prikažemo će proširivati šablonsku stranicu
  - Moguće je raditi učitavanje (*incude*) jedne stranice u drugoj
  - Podaci prosleđeni pogledu su vidljivi i u svim ostavim fajlovima
- Kontroler učitava više *view* fajlova redosledom kojim oni treba da se nadovezuju:
  - Na primer, kontroler učitava header.php, vesti.php i footer.php

Šablonska stranica - primer

default.php:

```
@include('header')
@yield('content')
    <center>Copyright 2020</center>
</body>
</html>
```

pocetna.php:

```
@extends('default')
@section('content')
     <h1>Hello World!</h1>
@endSection
```

Poziv iz kontrolera:

```
return view("pocetna", $data);
```

Blade

- Za potrebe generisanja view-ova, Laravel koristi Blade template engine
- Za ugrađivanje PHP koda se koriste {{ }}
- Neke od osnovnih direktiva Blade-a:
  - @if / @endif – za uslovno prikazivanje dela stranice između ove dve direktive
  - @for / @endfor; @foreach / @endforeach – za prikazivanje istog dela koda između ove dve direktive
  - @csrf – ubacuje se u forme, radi zaštite od CSRF napada
  - @auth / @endauth – za prikazivanje dela stranice između ove dve direktive samo u slučaju da je korisnik autentifikovan
  - @guest / @endguest – za prikazivanje dela stranice između ove dve direktive samo u slučaju da korisnik nije autentifikovan

Parametri zahteva

- Parametri zahteva se prosleđuju kroz $_GET i $_POST nizove

- Informacije o pristiglom zahtevu se smeštaju u objekat Request klase.
- Kontroler ima polje $request koje će se inicializovati prilikom inicijalizacije kontrolera

- $request->ime
  $request-> input('ime')
  - vraća vrednost elementa iz $_GET , $_POST i $_REQUEST niz
  - vraća null vrednost ukoliko element ne postoji

- $request**->**isMethod ('post')
  - vraća informaciju da li je zahtev jednak onoj metodi koja je prosleđena

- $request**->**ajax()
  - vraća informaciju da li je AJAX zahtev

SERVICE PROVIDERS

Service Providers

- Skup klasa koje se koriste za “bootstrapping” aplikacije
- Izvode se iz Illuminate\Support\ServiceProvider
- Postoje 2 metode koje se mogu nadjačati prilikom dodavanja novog Service Provider-a:
  - register()
  - boot()
- Obe se koriste za inicijalizaciju aplikacije – register koristiti ako nisu potrebni drugi service provider-i, dok u boot metodi je to dozvoljeno
- Moguće je menjati i implementaciju postojećih service provider-a

Dodavanje Service Provider-a

- Sve service provider-e dodavati u config/app.php, inače se neće pozvati prilikom inicijalizacije aplikacije

- Primer izgleda sekcije za service provider-e u app.php

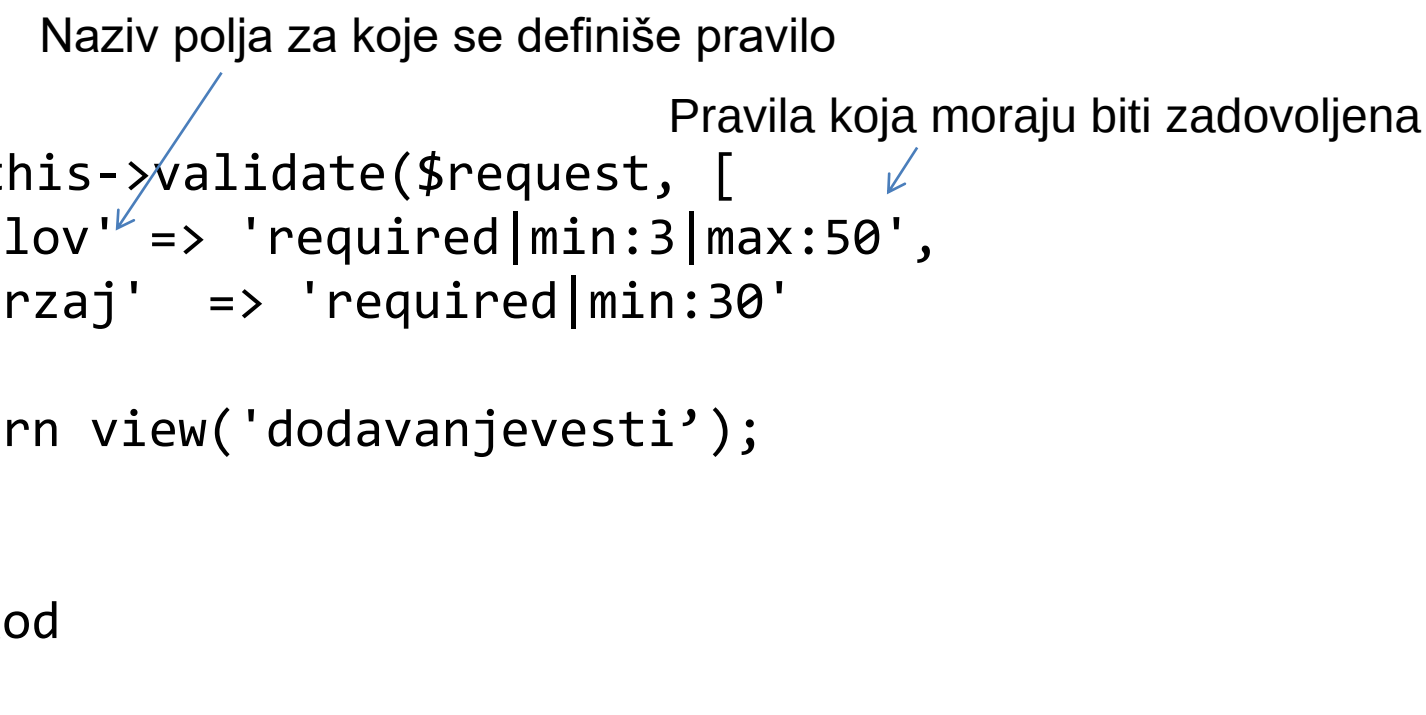
```
'providers' => [

    /*
     * Laravel Framework Service Providers...
     */
    Illuminate\Auth\AuthServiceProvider::class,
    Illuminate\Broadcasting\BroadcastServiceProvider::class,
    Illuminate\Bus\BusServiceProvider::class,
    Illuminate\Cache\CacheServiceProvider::class,
    Illuminate\Foundation\Providers\ConsoleSupportServiceProvider::class,
    Illuminate\Cookie\CookieServiceProvider::class,
    Illuminate\Database\DatabaseServiceProvider::class,
    Illuminate\Encryption\EncryptionServiceProvider::class,
    Illuminate\Filesystem\FilesystemServiceProvider::class,
    Illuminate\Foundation\Providers\FoundationServiceProvider::class,
    Illuminate\Hashing\HashServiceProvider::class,
    Illuminate\Mail\MailServiceProvider::class,
    Illuminate\Notifications\NotificationServiceProvider::class,
    Illuminate\Pagination\PaginationServiceProvider::class,
    Illuminate\Pipeline\PipelineServiceProvider::class,
    Illuminate\Queue\QueueServiceProvider::class,
    Illuminate\Redis\RedisServiceProvider::class,
    Illuminate\Auth\Passwords\PasswordResetServiceProvider::class,
    Illuminate\Session\SessionServiceProvider::class,
    Illuminate\Translation\TranslationServiceProvider::class,
    Illuminate\Validation\ValidationServiceProvider::class,
    Illuminate\View\ViewServiceProvider::class,

    /*
     * Package Service Providers...
     */

    /*
     * Application Service Providers...
     */
    App\Providers\AppServiceProvider::class,
    App\Providers\AuthServiceProvider::class,
    // App\Providers\BroadcastServiceProvider::class,
    App\Providers\EventServiceProvider::class,
    App\Providers\RouteServiceProvider::class,
```

FORM VALIDATION

Validation library

- Omogućava validaciju formi
- Validacija podrazumeva semantičku proveru unetih vrednosti

Primer:



Validation library

- Prikaz ispisa greške u view-u:

```
@error('naslov')
        <td><font color='red'>{{ $message }} </font></td>
@enderror
```

- Blade direktiva @error/@enderror za definisanje uslovne sekcije u slučaju greške (u zagradama se specificira na koje polje se odnosi)
- $message je promenljiva koja postoji u okviru @error/@enderror sekcije koja služi za ispis greške

Validation library

- Pravila se navode u okviru jednog stringa
- Pravila su nazivi funkcija
- Parametri se pravilima prosleđuju nakon :

- Pored ugrađenih pravila, moguće je pisati i sopstvene funkcije koje validiraju vrednost podatka (kompleksne provere)
  - Klasa u kojoj se nalazi pravilo treba da se izvodi iz interfejsa Illuminate\Contracts\Validation\Rule
  - Tu treba definisati metodu passes, koja vraća bool
  - Prvi parametar je atribut koji se validira, a drugi njegova vrednost
  - Metodu message definisiati za definisanje poruke koja se ispisuje u slučaju greške

Validation library

- Biblioteka za validaciju ima predefinisane generičke poruke
  - Primer: require pravilo ima poruku "The %s field is required.”
- Predefinisane poruke je moguće izmeniti:
- Primer:

```
$this->validate($request,[
            'korime' => 'required|min:3',
            'lozinka' => 'required'
        ], [
            'required' => 'Polje :attribute je obavezno',
            'min' => 'Polje :attribute je neophodno da bude minimalne
duzine :min karaktera'
        ]);
```

- Informacije o greškama se mogu dohvatiti pomoću sledećih metoda u okviru view-a:
  - $errors->all() - vraća niz grešaka
- Polja forme se mogu popuniti unetim vrednostima koristeći funkciju old()

SESIJE

Session library

- Dohvatanje deljenog objekta sesije i inicijalizacija:

```
$session = session(); //poziv pomoćne funkcije
```

- Rad sa podacima unutar sesije:
  - `$user = $session->get('user')`
    - vraća vrednost promenljive ili null, ako promenljiva ne postoji
  - `$session->has('user')`
  - `$session->put('user', 'tasha')`
  - `$session->forget('user');`

Session library

- *Flashdata* su podaci koji će biti dostupni prilikom sledećeg zahteva, a zatim će nestati.
  - Pogodno je za ispis poruka.
- Rad sa *Flashdata*:
  - `$val = $session->get('item');`
  - `$session->flash('item', 'value');`
  - `$session->keep ('item');`
  - `$session->now ('item');`

MODEL I RAD SA BAZOM PODATAKA

Model

- Koristi se Eloquent u Laravelu za komunikaciju sa bazom
- Moguće koristiti i DB fasadu
- Model predstavlja klasu zaduženu za perzistiranje i pribavljanje potrebnih podataka
- Izveden je iz klase Illuminate\Database\Eloquent\Model

- Model sadrži sledeća polja:
  - `$connection`
  - `$table`
  - `$primaryKey`
  - `$keyType`
  - `$fillable`
  - `$guarded`
  - `$timestamps`

    …

Model

- Model sadrži sledeće predefinisane metode:
  - `find($id)`
  - `all()`
  - `get()`
  - `first()`
  - `where($name, $value)`
  - `create($data)`
  - `update($data)`
  - `save($data)`
  - `delete()`
  - `destroy($id)`
  - …

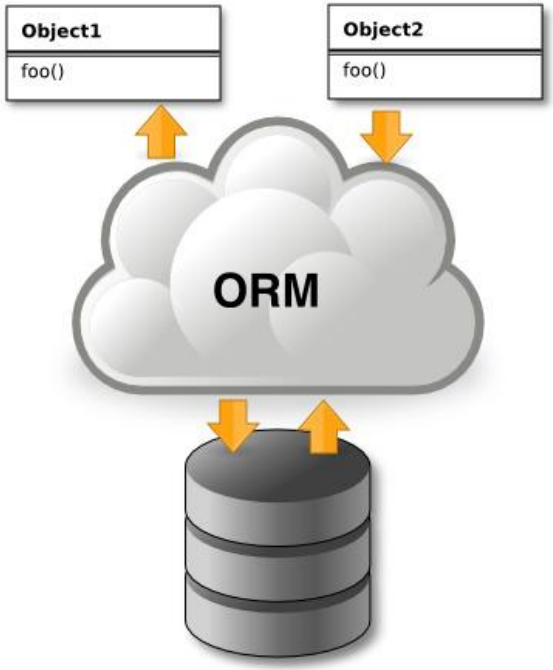
Model - Primer

```
class VestModel extends Model
{
    protected $table      = 'vest';
    protected $primaryKey = 'id';
    protected $keyType = 'object';

    protected $fillable = [ 'naslov', 'sadrzaj',
        'autor', 'datum'];

    public function dohvatiVestiAutora($autor){
        return $this->where('autor', $autor)->get();
    }

    public function pretraga($tekst){
        return $this->where('naslov', 'like',"%$tekst%")
                ->orWhere('sadrzaj','like',"%$tekst%")->get();
    }
}
```

ORM

- ORM: *Object Relational Mapping*
  - Mehanizam koji omogućuje rad sa objektima na aplikativnom nivou, uz automatski rad sa BP
  - Skup klasa/funkcija koje generišu odgovarajući SQL kod

- Postoje različita rešenja
  - Kod nekih se podrazumeva da je na programeru da promene održava i u BP i u modelu
  - Kod nekih, postoje alati koji generišu tabele i promene u tabelama, na osnovu izmena u klasama modela
  - Kod nekih, neophodno je da se, kroz konfiguracione fajlove specificira kako se model mapira na tabele u BP; drugi analiziraju strukturu tabela i samostalno „shvataju“



Laravel model i baza podataka

- CI nudi implementaciju *Active Record* projektnog uzorka za perzistiranje podataka
  - Nije „čist“ *Active Record*
  - Nudi metode kojima se podaci sadržani u objektima ili mapama čuvaju u BP, na takav način da se ne piše SQL kod
    - Taj posao obavlja *ActiveRecord* klasa
  - Potrebno je konfigurisati parametre pristupa bazi (server, username, password)
  - Nije potrebno konfigurisati nikakve dodatne parametre
- CI ne proverava da li svaka klasa modela ima odgovarajuću tabelu samostalno
  - nema nikakve automatizovane sinhronizovanosti => povećana šansa za probleme i greške
  - Prednost: „lagano“ ORM rešenje
  - Mana: dosta odgovornosti na programeru – ručno specificiranje migracija

Prednosti ORM

- Objektno orijentisani kod, čak i pri pisanju upita
- Bez SQL i preterane brige o valjanosti korisničkih podataka
- Olakšano jedinično testiranje rada kontrolera i modela – „mock“-ovati bazu podataka

- Primer:

```
$vestModel->save([
    'naslov' => 'Tema cetvrta lab vezba',
    'sadrzaj'=> 'Tema ove lab vezbe je ORM',
    'autor'  => 'tasha' ]);
$id=$vestModel->id;
```

Definisanje relacija između modela

- Definiše se kao metoda koja vraća podatke sa druge strane relacije
- One-To-One relacija
  - Kod “vlasnika” relacije dodati u njegovoj metodi:
    - $this->hasOne(<model-klasa-sa-druge-strane>)
  - Kod “posedovanog modela” relacije dodati u njegovoj metodi:
    - $this->belongsTo(<model-klasa-sa-druge-strane>)
- One-To-Many relacija
  - Kod “vlasnika” relacije dodati u njegovoj metodi:
    - $this->hasMany(<model-klasa-sa-druge-strane>)
  - Kod “posedovanog modela” relacije dodati u njegovoj metodi:
    - $this->belongsTo(<model-klasa-sa-druge-strane>)

Definisanje relacija između modela

- Many-To-Many relacija
  - Sa obe strane relacije dodati u njihovim metodama:
    - $this->belongsToMany(<model-klasa-sa-druge-strane>)
- Korišćene metode imaju podrazumevane vrednosti, koje je moguće zameniti:
  - To su vrednosti naziva primarnog/stranog ključa
  - U many-to-many relaciji moguće definisati koja tabela je tzv. pivot tabela gde se radi povezivanje relacije

Rad sa bazom podataka

- Konekcija se definiše unutar .env fajla
- Migracija baze:
  - Migraciju baze je potrebno odraditi ukoliko dođe do nekih promena u njenoj strukturi
  - Moguće je (ručno) specificirati izmene koje su potrebno odraditi
  - Forge predstavlja klasu koja nam omogućava lakši rad sa bazom

OSTALI KONCEPTI

Form Resubmission

- Pojava koja se događa kada je nakon potvrđivanja (submit) forme na neku akciju to moguće učiniti ponovo, osvežavanjem (refresh) veb stranice
- **Operacija Submit**, tj. potvrđivane forme i njeno slanje na server je slanje HTTP zahteva koji sadrži vrednosti polja (u URL kod GET formi ili unutar HTTP zahteva, kod POST formi)
- Osvežavanje u veb pregledaču (F5, refresh dugme) dovodi do slanja istog zahteva preko kog se došlo na tekuću stranicu
- Kada se na stranicu dođe putem potvđene forme, osvežavanje je ekvivalentno ponovnom slanju forme na server, uz slanje istih podataka
- Nakon što se u akciji obradi sadržaj POST forme, neophodno je poslati **redirect** veb pregledaču (preusmeriti ga na drugu akciju)
- Time se veb pregledaču da instrukcija da izvrši neku drugu akciju, čime se rešava ovaj problem

Redirect

- `redirect()->route($imerute);`
  - Preusmeravanje na neku drugu stranicu
  - $imerute može biti bilo koja ruta koja je definisana
- `redirect()->back();`
  - Preusmeravanje na prethodno učitanu stranicu
- `redirect()->back()->withInput();`
  - Preusmeravanje na prethodno učitanu stranicu uz čuvanje informacija o prethodno unetim podacima u formu
  - Pomoću old() funkcije može se pristupiti prethodno unetim podacima

Middleware

- Implementiraju metodu handle(), koja govori da li se propušta HTTP zahtev
- Middleware je moguće dodati u konstruktor kontrolera radi provere
- Moguće je dodati i u app/Kernel.php u slučaju da treba da se primeni na veći broj kontrolera
- Primeri middleware-a:
  - VerifyCsrfToken – za CSRF zaštitu
  - Authenticate – za proveru da li korisnik pokušava neautentifikovan da pristupi nekom resursu
  - RedirectIfAuthenticated – za proveru da li korisnik pokušava autentifikovan da pristupi nekom resursu

    ...

Localization

- Bitan aspekt svake ozbiljne aplikacije. Sadržaj koji nije dinamički (ne čuva se u bazi) a prikazuje se (kao npr. poruke o greškama, stavke navigacije,...) bi trebalo lako „prevesti“ na neki jezik
- Pri formiranju dinamičke stranice, takve elemente ne pisati direktno (ručno) na govornom jeziku
- __($key) - vraća vrednost elementa niza iz odgovarajućeg rečnika
- Prevođenje se vrši promenom odgovarajućih fajlova u lang direktorijumu
- Postavljanje jezika u config/app.php