Codeigniter 4

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*)

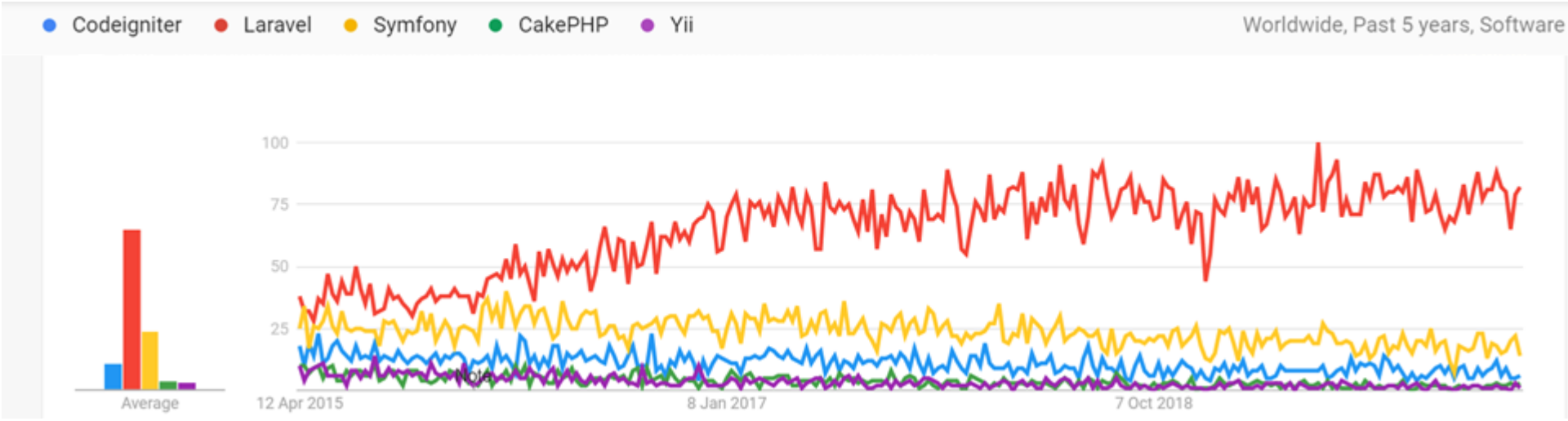
Princip dizajna u veb programiranju

- 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



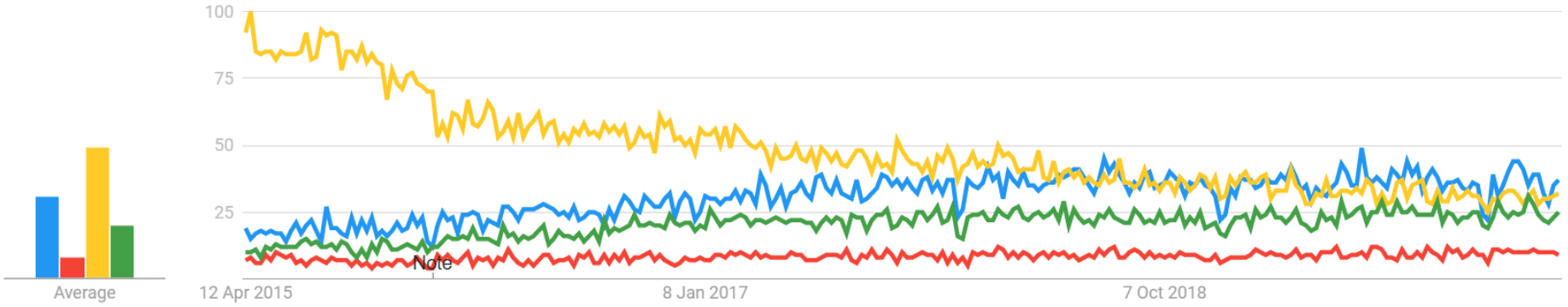
Angular
Web framework

Django
Web framework

.NET Framework
Software

Laravel
Topic

+



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)

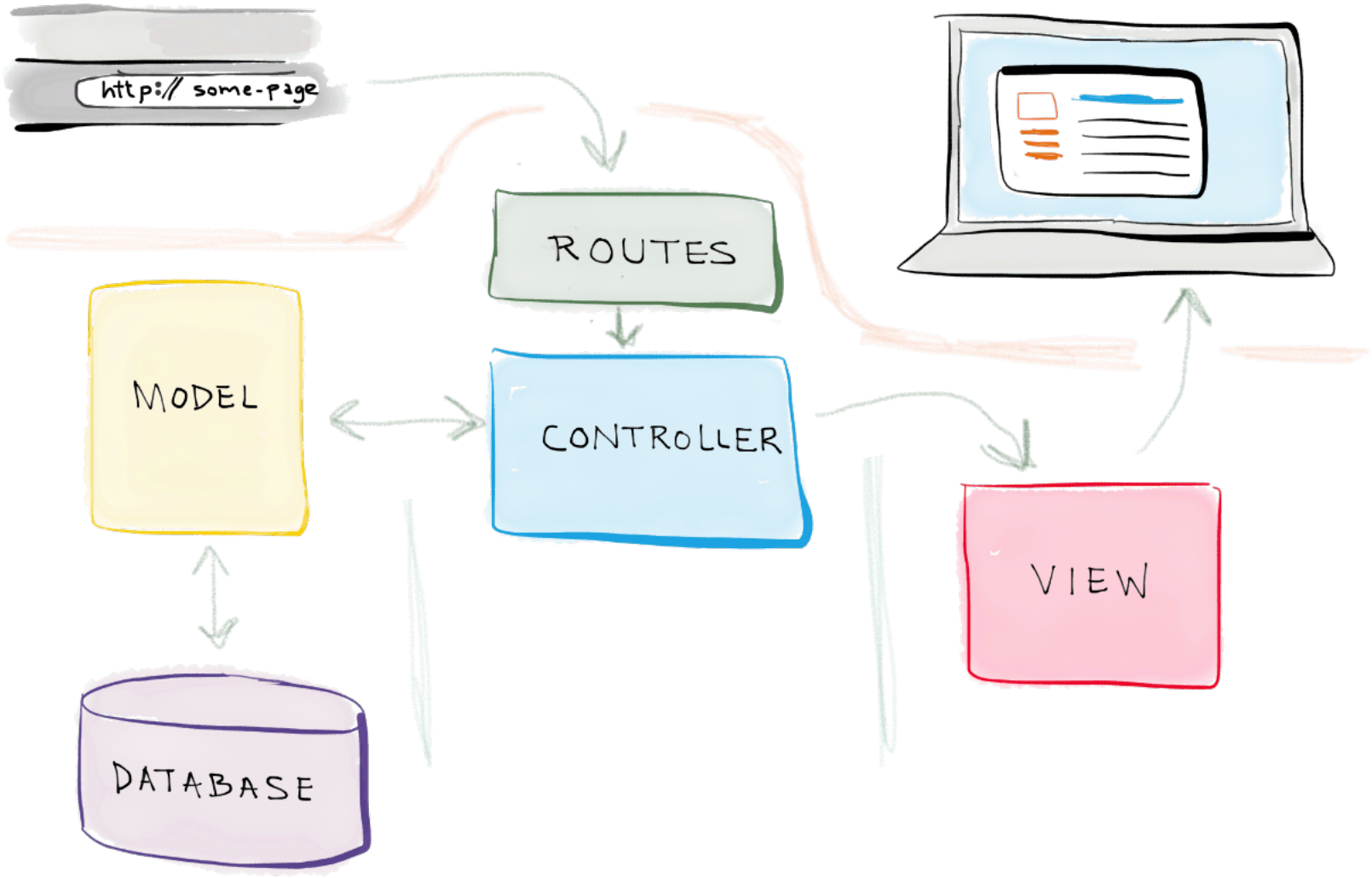
Dogovor umesto konfigurisanja

- 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

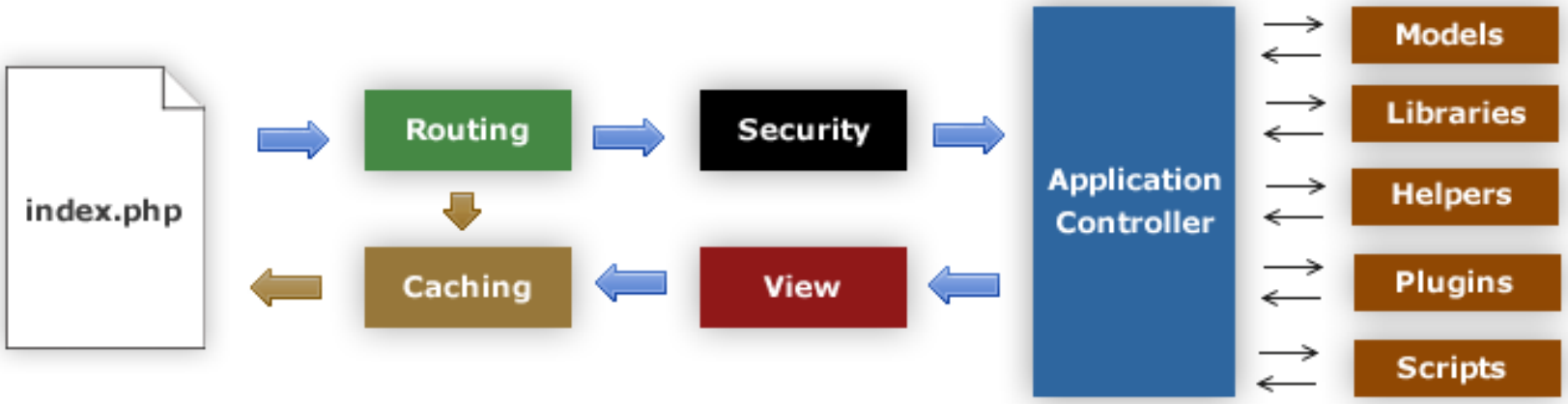
- Ideja: ne opteretiti kontrolere business logikom
- Kontroler odlučuje koji view prikazati, i koje podatke tada prikazivati, ili kome proslediti izvršavanje
- Ne bi trebalo da sadrži čitav proces i donošenje odluka oko domenskog problema
- To (mozganje) ostaviti servisima
  - Posebne klase koje sadrže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

CODEIGNITER

MVC unutar Codeigniter-a

- Svaki radni okvir ima sopstvene mehanizme na osnovu kojih ostvaruje MVC
- Pored MVC, u Codeigniter-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 Codeigniter



- 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 sledeća tri načina:
  - Preuzimanjem trenutne verzije radnog okvira sa sajta
    - link: https://codeigniter.com/download
  - 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
codeigniter4/appstarter project-root
```

    - Composer omogućava jednostavan prelazak na noviju verziju radnog okvira, pa se preporučuje njegovo korišćenje
  - Preuzimanje trenutne verzije sa GIT-a

Priprema radnog okruženja

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

- Codeigniter 4 sadrži lokalni server za razvoj koji koristi ugrađeni PHP server i radi mapiranje ruta.
  Pokreće se izvršavanjem komande:
  ```
  php spark serve
  ```
  i tada projektu možete pristupiti i putem linka:
  http://localhost:8080/

Režim razvoja aplikacije (eng. *Development mode*)

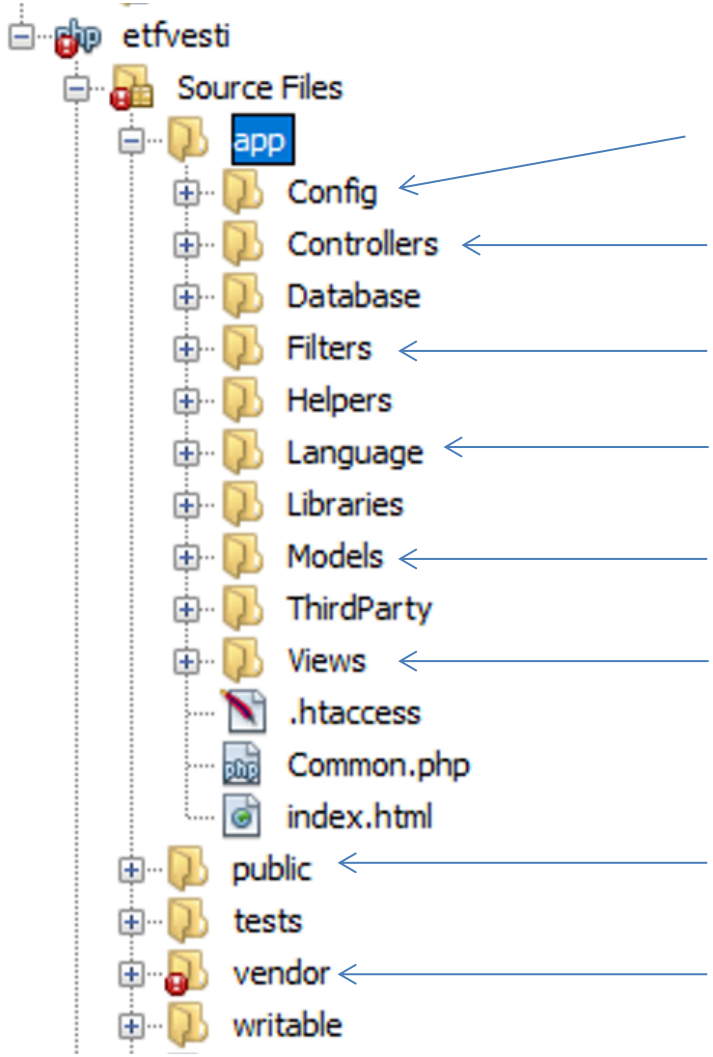
- Projekat se podrazumevano pokreće u produkcionom režimu (eng. *Production mode*).

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

```
CI_ENVIRONMENT = development
```

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

Struktura aplikacije



Konfiguracioni fajlovi

Klase koje implementiraju poslovnu logiku

Klase za filtriranje zahteva

Namenjeno za lokalizaciju aplikacije

Klase koje omogućavaju rad sa podacima

PHP skripte koje generišu HTML

Ovde se nalazi index.php fajl

Sistemski fajlovi

Kako funkcioniše?

- Kliknete na link
  http://localhost/etfvesti/public/index.php/Korisnik/vest/3
  ili
  http://localhost:8080/Korisnik/vest/3

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

CONTROLLER I VIEW

Controller

- Izveden je iz klase CodeIgniter\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
- Inicijalizaciju kontrolera ne treba raditi u konstruktoru, već se može preklopiti postojeća metoda *initController*

- Logički grupisan skup akcija
  - URL-ovi se odnose na akcije kontrolera
- 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:

```
echo 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:

```
<?= $this->include('header') ?>
<?= $this->renderSection('content') ?>
    <center>Copyright 2020</center>
</body>
</html>
```

pocetna.php:

```
<?= $this->extend('default') ?>
<?= $this->section('content') ?>
    <h1>Hello World!</h1>
<?= $this->endSection() ?>
```

Poziv iz kontrolera:

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

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

- $request**->**getMethod()
  - vraća informaciju da li je zahtev POST ili GET

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

HELPER

Helper - skup pomoćnih funkcija

- Namenjene kao pomoć u pisanju koda,
  bilo pri generisanju HTML-a, bilo pri radu kontrolera
- Kolekcija jednostavnih funkcija (nije OO)
- Neki važni *helper*-i:
  - URL Helper, Form Helper, Date Helper, Filesystem Helper,...

- Pre upotrebe potrebno je učitati *helper* pozivanjem funkcije:

```
helper('form');
```

  ili dodavanjem elementa niza u polje kontrolera $helpers
  odakle se helper-i automatski učitavaju.
  - Nakon toga, fajlovi su učitani i funkcije su dostupne za upotrebu
  - Na ovaj način se izvrši *include* fajlova sa definicijama funkcija koje čine *helper*.

URL Helper

- Pomaže pri kreiranju linkova i generiše HTML kod

- **site_url**([*$uri=''*[, *$protocol=NULL*[, *$altConfig = NULL*]]])
  - Generiše kompletan url na osnovu zadatog kontolera i metode
  - Dodaje kao prefix putanju do index.php fajla
  - Primer:

    ```
    echo site_url('news/local/123');
    ```

    Ispisuje:

    ```
    http://example.com/index.php/news/local/123
    ```

- **anchor**([*$uri=''*[, *$title=''*[, *$attributes=''*
  [, *$altConfig=NULL*]]]])
  - Generiše link na osnovu zadatog url-a i naslova
  - Primer:

    ```
    echo anchor('news', 'Click');
    ```

    Ispisuje:

    ```
    <a href="http://example.com/index.php/news">Click</a>
    ```

Form Helper

- Primereno je koristiti samo na view-u, ne i unutar akcija kontrolera
- Pomaže pri kreiranju formi i generiše HTML kod

- Postoji i podrška za validaciju unetih podataka:
  - Implementirano u vidu biblioteke (*Validation library*)
  - Moguće definisanje uslova koje uneti podaci moraju da zadovolje
  - Ukoliko su uneti podaci neodgovarajući, prikazuju se poruke o grešci, i ne dozvoljava se nastavak (serverska validacija)

Form Helper - Primer

```
echo form_open("Korisnik/novaVest","method=post");
// <form method="post"
// action="http://localhost/etfvesti/Korisnik/novaVest" >

echo form_label("Naslov: ", "naslov");
// <label for="naslov"> Naslov: </label>

echo form_input("naslov",set_value("naslov"));
// <input type="text" name="naslov" value="PSI"/>

// set_value – vraća vrednost elementa $_POST niza
// ili prazan string ukoliko on ne postoji

echo form_label("Sadrzaj: ", "sadrzaj");
echo form_textarea("sadrzaj",set_value("sadrzaj"));
echo form_submit("dodaj", "Dodaj");

echo form_close(); // </form>
```

LIBRARY

Library (Biblioteka)

- Biblioteke predstavljaju kompleksnije pomoćne mehanizme u realizaciji čestih operacija
- Moguće koristiti bilo u View, Model ili Controller
- Neki biblioteke:
  - Session, Validation, Pagination, Email Class,Time, Image, File,...

- Ugrađene biblioteke su realizovane kao servisi i korisnik ne zna i ne poznaje internu organizaciju biblioteke.
- Pozivanjem servisa se dohvata deljeni objekat biblioteke
  - Objekat predstavlja fasadu ka funkcionalnostima koje biblioteka pruža (fasada uzorak)
  - Primer:

```
$validation = \Config\Services::validation();
```

Validation library

- Omogućava validaciju formi
- Validacija podrazumeva semantičku proveru unetih vrednosti

Primer:

Naziv polja za koje se definiše pravilo

Pravila koja moraju biti zadovoljena

```
if (!$this->validate([
   'naslov' => 'required|min_length[3]|max_length[50]',
   'sadrzaj'  => 'required|min_length[30]'
])){
   echo view('dodavanjevesti',
      ['errors' => $this->validator->listErrors()]);
}
else {
   // kod
}
```

Validation library

- Pravila se navode u okviru jednog stringa
- Pravila su nazivi funkcija
- Pored pravila, postoje i „obrade“
  - Primer:
    'trim|required|max_length[30]'
- Parametri se pravilima prosleđuju unutar []

- Pored ugrađenih pravila, moguće je pisati i sopstvene funkcije koje validiraju vrednost podatka (kompleksne provere)
  - Klasa u kojoj se nalazi pravilo je potrebno dodati u niz $ruleSets u fajla App/Config/Validation.php
  - Prvi parametar je uvek polja koja se validira, a drugi parametar može biti poruka greške
  - Povratna vrednost metode je bool

Validation library

- Validaciju je moguće raditi na više načina
- Prvo se definišu se sva pravila nad svim željenim poljima pozivanjem metoda $validation->setRule() i $validation->setRules(), a zatim se poziva $validation->run() koja radi validaciju podataka
- Controller i Model sadrže metode validate() koje enkapsuliraju pozive prethodnih metoda u poziv jedne metode

- U klasi Validation u fajlu App/Config/Validation.php se mogu definisati pravila za validaciju koje se pomoću metode $validation**->**getRuleGroup() mogu dohvatati.

Validation library

- Biblioteka za validaciju ima predefinisane generičke poruke
  - Primer: require pravilo ima poruku "The %s field is required.”
- Predefinisane poruke je moguće izmeniti.
- Parametar poruke je podrazumevano naziv polja, a moguće ga je definisati prilikom definisanja pravila
  - Primer:
    ```
    $validation->setRule('korime', 'User', 'required');
    ```

- Informacije o greškama se mogu dohvatiti pomoću sledećih metoda:
  - $validator->getErrors() - vraća niz grešaka
  - $validation->listErrors() – vraća html kod
- Polja forme se mogu popuniti unetim vrednostima koristeći funkciju set_value() iz form helper-a

Session library

- Dohvatanje deljenog objekta sesije i inicijalizacija:

```
$session = \Config\Services::session($config);
```

ili

```
$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')`
  - `$session->set('user', 'tasha')`
  - `$session->remove('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 = $session->getFlashdata('item');`
  - `$session->setFlashdata('item', 'value');`
  - `$session->keepFlashdata('item');`
  - `$session->markAsFlashdata('item');`

- *Tempdata* su podaci koji se čuvaju u sesiji određeni vremenski period.
- Rad sa *Tempdata*:
  - `$session->markAsTempdata('item', 300);`

MODEL I RAD SA BAZOM PODATAKA

Model

- Model predstavlja klasu zaduženu za perzistiranje i pribavljanje potrebnih podataka
- Izveden je iz klase CodeIgniter\Model

- Model sadrži sledeća polja:
  - `$DBGroup`
  - `$table`
  - `$primaryKey`
  - `$returnType`
  - `$useSoftDeletes`
  - `$allowedFields`
  - `$useTimestamps`
  - Konfiguracija za validaciju

    …

Model

- Model sadrži sledeće predefinisane metode:
  - `find($id)`
  - `findAll()`
  - `findAll($limit, $offset)`
  - `first()`
  - `where($name, $value)`
  - `insert($data)`
  - `update($id, $data)`
  - `save($data)`
  - `delete($id)`
  - `getInsertId()`
  - `asArray()`
  - `asObject()`

    …

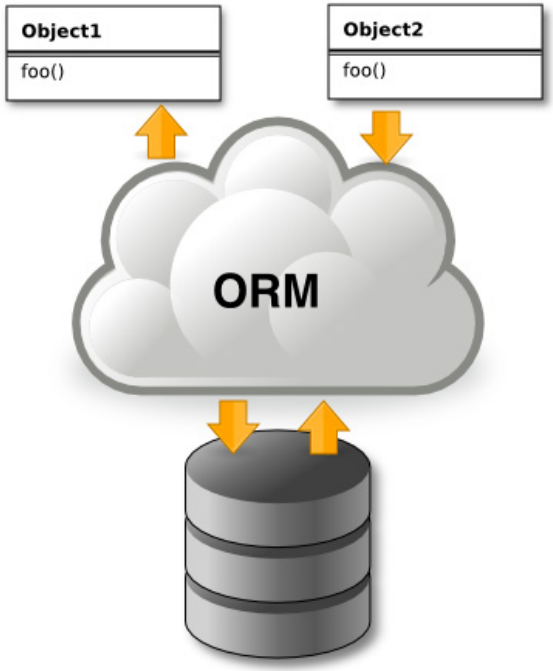
Model - Primer

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

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

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

    public function pretraga($tekst){
        return $this->like('naslov',$tekst)
                ->orLike('sadrzaj',$tekst)->findAll();
    }
}
```

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“



Codeigniter 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->getInsertId();
```

Entity class

- Predstavlja red u bazi
- Izveden je iz klase CodeIgniter\Entity
- Sadrži polja koja predstavljaju redove u bazi
- Može da sadrži dodatne metode koje implementiraju poslovnu logiku reda

- Njeno korišćenje nije obavezno
- Metode modela mogu da dohvataju podatke u ovom obliku
  - Potrebno je definisati $returnType

Rad sa bazom podataka

- Konekcija se definiše unutar klasi Database u fajlu App/Config/Database.php
  - Podrazumevano se koristi $default konekcija

- 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()->to($url)`
  - Preusmeravanje na neku drugu stranicu
  - Potrebno je zadati celu $url
- `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

Routes

- U fajlu App/Config/App.php definisana su polja $baseURL i $indexPage.
  - Vrednosti ovih polja se koriste prilikom generisanja URL.
- U fajlu App/Config/Routes.php je definisan podrazumevani kontroler i podrazumevana metoda.

- Podrazumevano tumačenje URL-a: kontroler/akcija/param
- Ako podrazumevano tumačenje URL-ova nije zadovoljavajuće, rute se mogu konfigurisati u App/Config /Routes.php fajlu

- Primer:

```
$routes->add(‘Gost/(:num)', 'Gost::vest/$1’);
$routes->match(['get', 'put’],
    ‘Gost/(:num)', 'Gost::vest/$1’);
$routes->get(‘Gost/(:num)', 'Gost::vest/$1');
```

Filter

- Implementiraju interface FilterInterface i sadrže metode before() i after()
- Njegove metode se izvršavaju pre i posle odgovarajućih akcija kontrolera
- Filter klasi je obavezno dodeljivanje aliasa
  - U fajlu App/Config/Filters.php u polje $aliases potrebno je dodati novi element
- Korišćenjem aliasa se definiše kada će se ona pozivati
  - U fajlu App/Config/Filters.php u polja $filters, $globals i $methods moguće je dodavati nove elemente

- Primer:

```
$filters=['user'=>['before'=>['Korisnik','Korisnik/*']]];
```

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
- lang($key) - vraća vrednost elementa niza iz odgovarajućeg rečnika
- Prevođenje se vrši promenom odgovarajućih fajlova u language direktorijumu