Codeigniter

MVC

Web framework

- OO sistem,
  konstruisan kako bi ga programeri proširili na takav način da obezbede funkcionalnosti koje im se zahtevaju
  - half baked
- Best practices – ugrađeno u samom framework-u

Loš pristup

- Jedna php skripta, koja radi
  - Generisanje dinamičkog HTML-a
  - Business 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 (web) servisima
  - ....
- Nešto kao *božanska klasa* u OO patternima
- Teško za održavanje, teško za razumevanje

Bolji pristup – separation of concerns

- Razdvajanje odgovornosti
- Monolitna aplikacija → Raslojena aplikacija
- Svaki sloj i komponenta rade samo jednu stvar i ništa osim toga
  - Bolja čitljivost
  - Bolji uslovi za testiranje koda
- Razni patterni
  - MVC
  - MVVM
  - MVP

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 (business) logiku, odlučuje šta se čuva u bazi i kada...

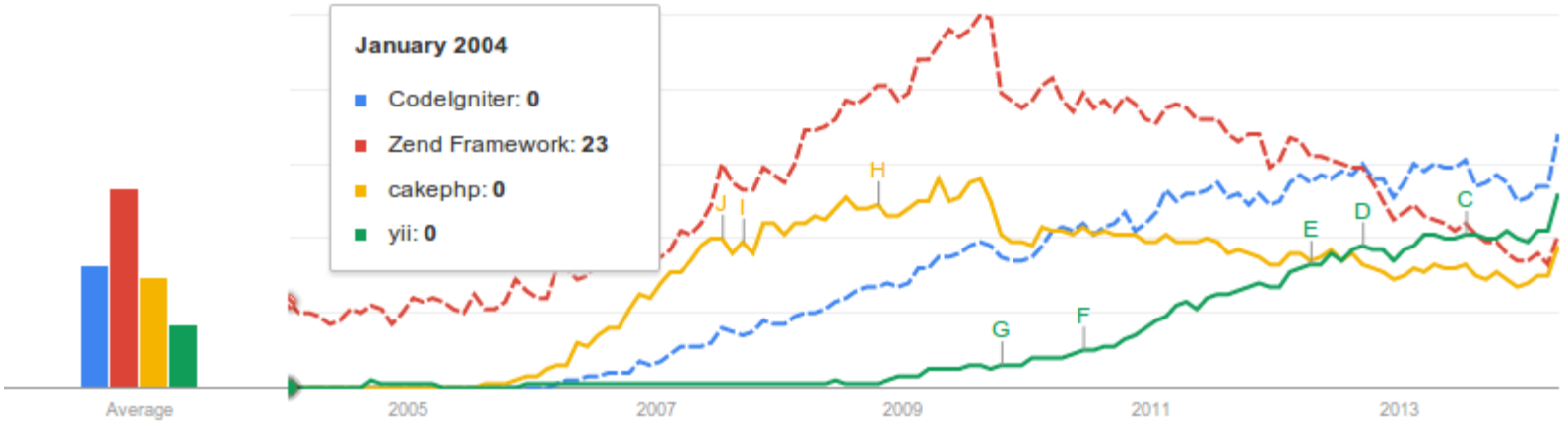
Dodatak: Services

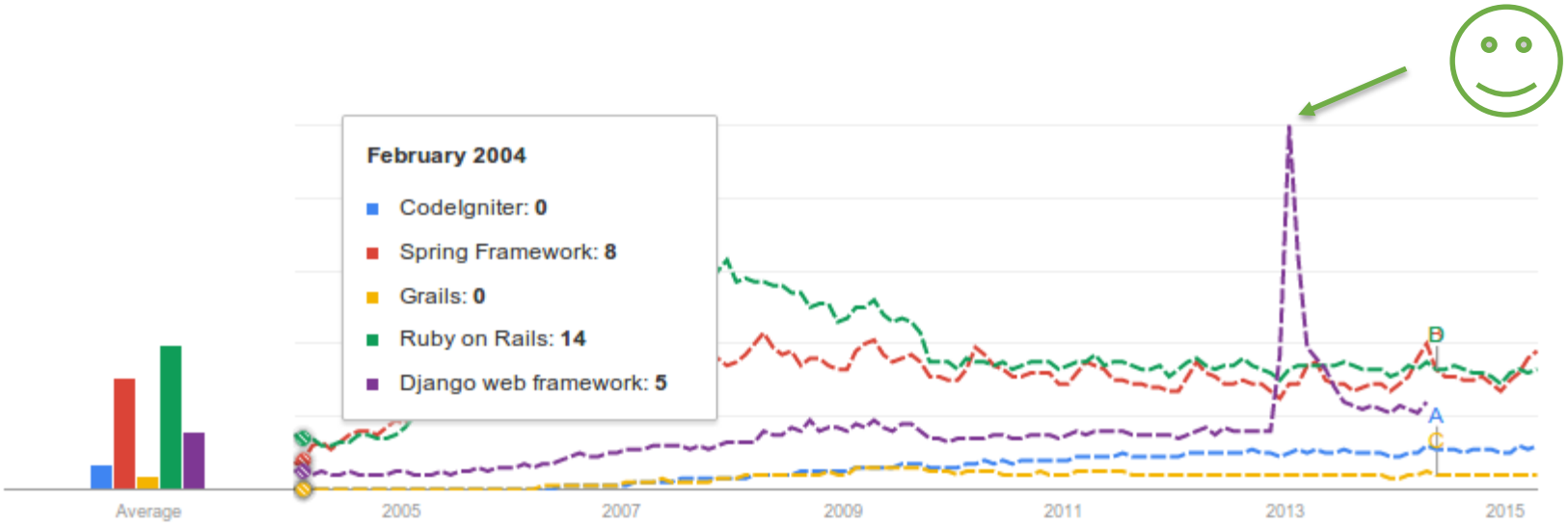
- Ideja: ne opteretiti kontrolere business logikom
- Kontroler odlučuje koji *view* prikazati, i koje podatke tada prikazivati, ili kome proslediti izvršavanje
- Ne bi trebalo da sadrži čitav proces i donošenje odluka oko domenskog problema
- To (mozganje) ostaviti servisima
  - Posebne klase koje sadrže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

Danas

- Frameworks za web, ali i desktop aplikacije
- Neki su komercijalni, neki open-source
  - Java web: JSP, Struts2, **Grails**, **Play**, Vaadin, **Spring**...
  - Java desktop: Swing, Griffon, JavaFX...
  - .net web: asp .net, **asp mvc**...
  - .net desktop: **WPF**, widnows forms
  - Php: YII, Zend, **Codeigniter**, CakePHP,...
  - Ostali: Ruby on Rails, Django, NodeJS, Angular...
  - CSS: Bootstrap (from twitter), Gumby, Foundatin, YAML…
  - Ima ih još MNOGO!
- Odabir? Pametno odabrati, loš izbor košta!!!

Popularnost web framework-a





Odabir tehnologije/framework-a

- Programski jezik: Java, Scala, Groovy, php, Ruby, Python, C#, VB, JS…
  - 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?
- Framework: utiču prethodne dve odluke
  - Šta članovi timova znaju?
  - Kakva je kriva učenja?
  - Koliko je podržan (zajednica, biblioteke, plugin-i)?
  - Koliko je produktinvno za programiranje?
  - Iskustva drugih?
  - Kakva je namena aplikacije? (nema *one-fits-all* rešenja)



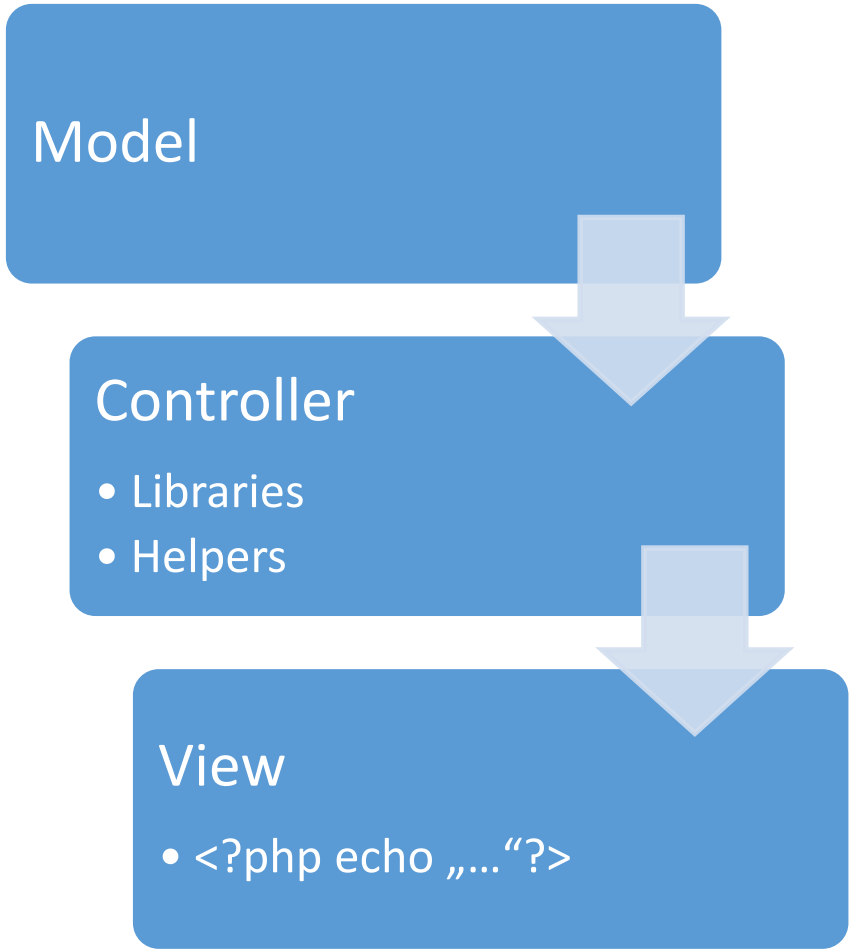


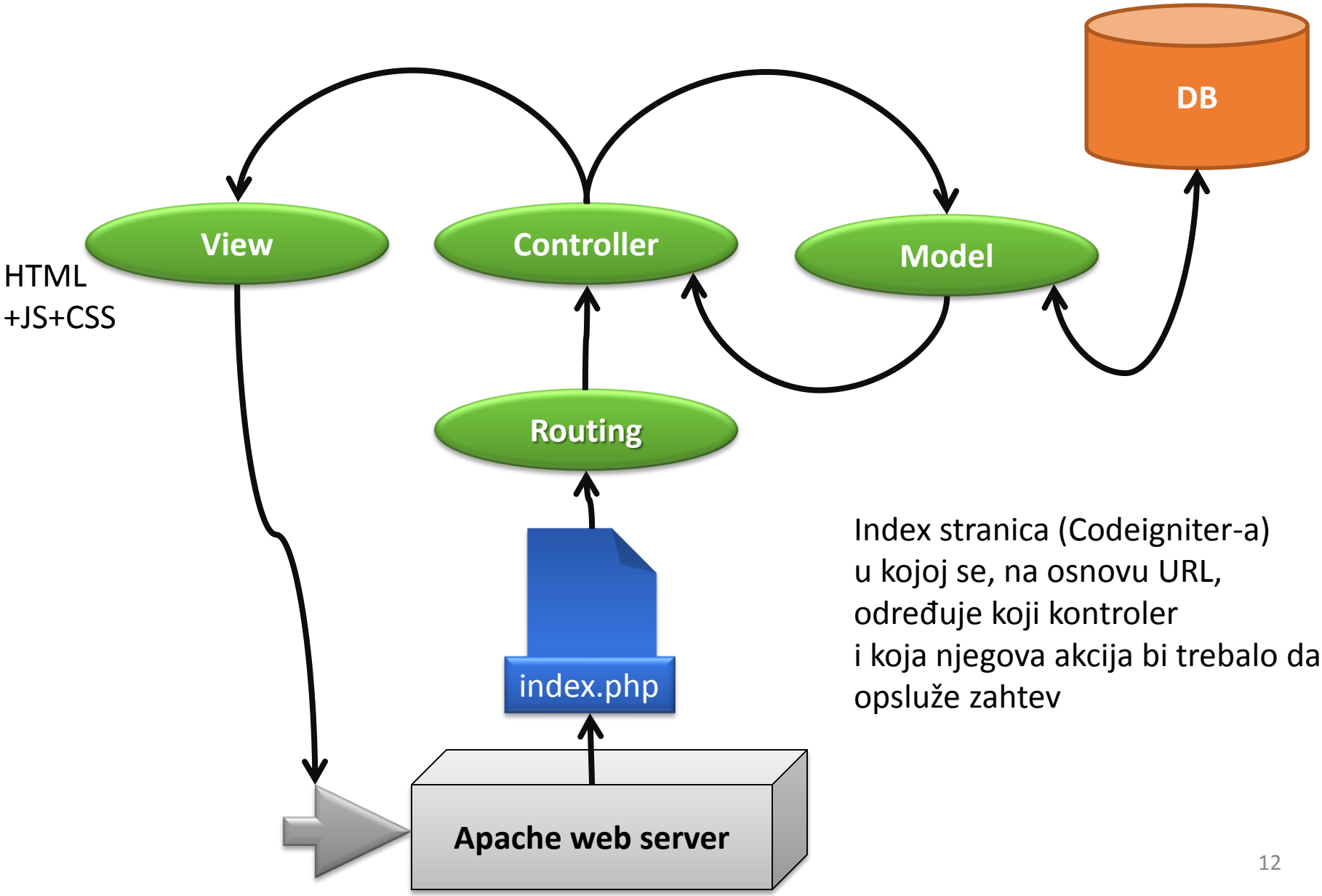
Dogovor umesto konfigurisanja

- 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 strukturira po unapred definisanim pravilima, uz mogućnost konfigurisanja neočekivanih odluka (override)
- Framework 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 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 business logiku na jedno mesto, web-related stvari na drugo (controller)



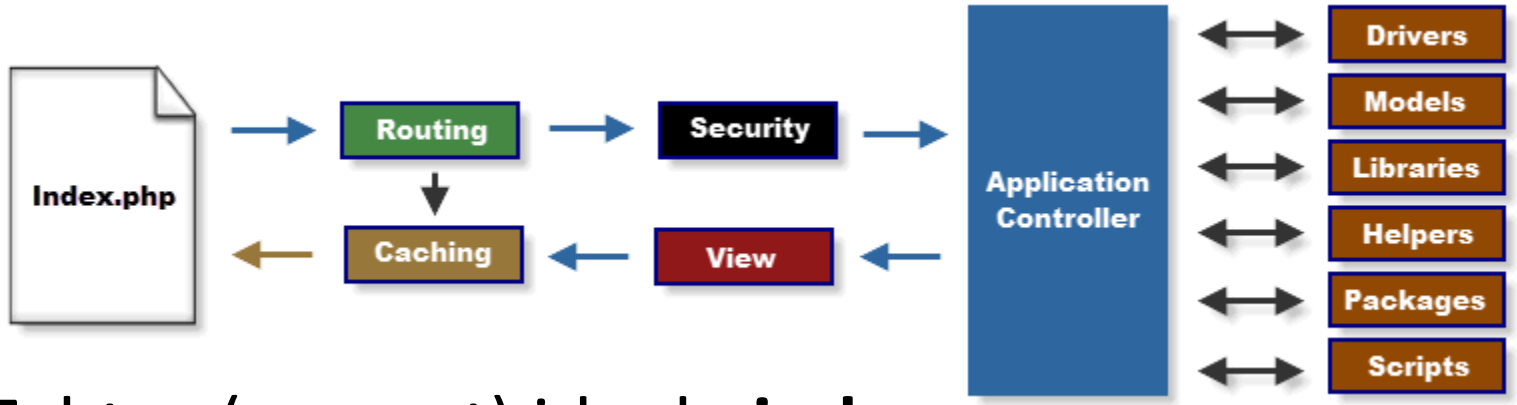


Index stranica (Codeigniter-a)
u kojoj se, na osnovu URL,
određuje koji kontroler
i koja njegova akcija bi trebalo da
opsluže zahtev

MVC i Codeigniter

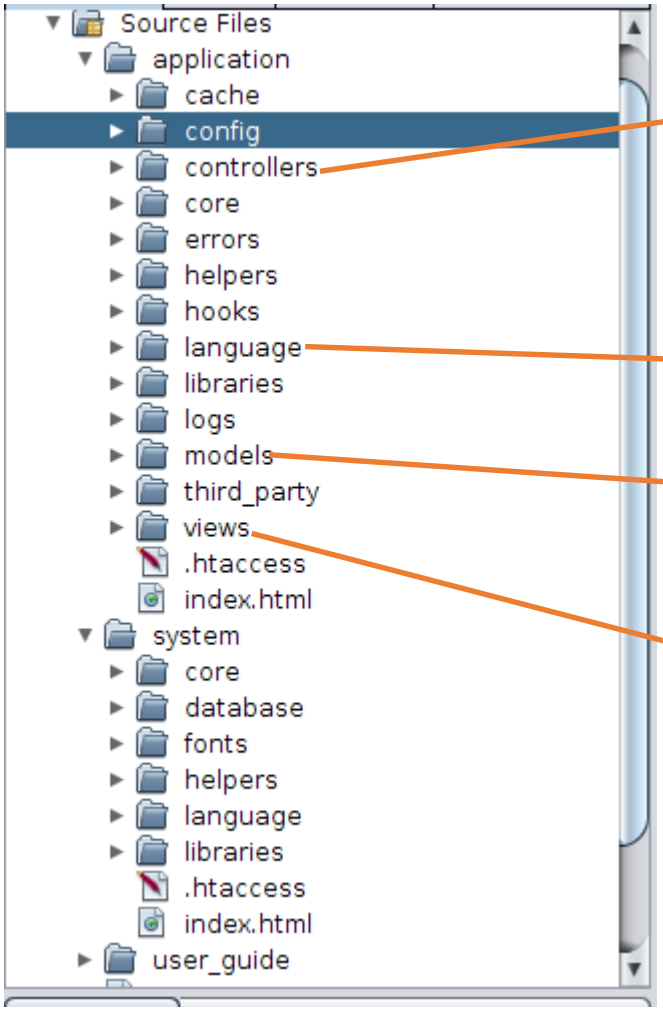
- Svaki framework ima sopstvene mehanizme na osnovu kojih ostvaruje MVC
- Pored MVC, ostvaruju se i:
  - *Loose coupling*: povezivanje komponenti na labav način – ne referencirati direktno klase međusobno, kako bi sistem bio modularan
  - *Dynamic loading*: komponente se učitavaju na eksplicitan zahtev,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

MVC i Codeigniter



- Zahtev (request) ide do **index.php**
  - Povezuju se komponente (preko Loader-a)
  - Analizira se URL (routing)
  - Zahtev se procesira zbog sigurnosnih razloga
  - Instancira se odgovarajući kontroler i poziva se odgovarajuća metoda (akcija)
    - Kontroler je već povezan sa loader-om, preko koda dobavlja modele, helper-e, biblioteke i sl.

Struktura aplikacije



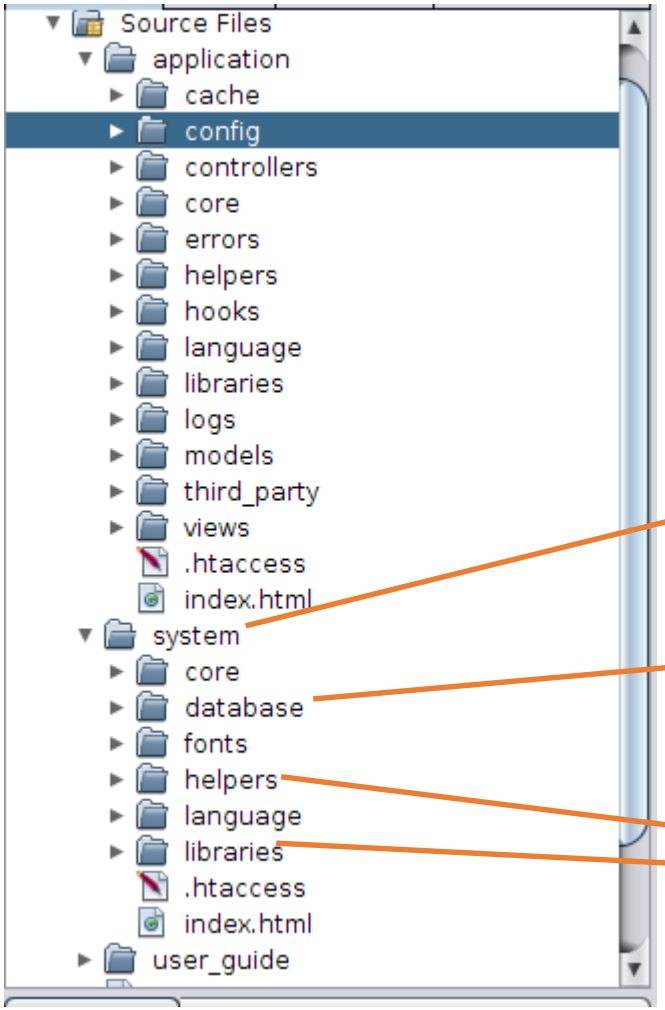
Kontroleri
Klase koje extenduju
CI_Controller

Namenjeno lokalizaciji
aplikacije
(radi lakog prevoda)

Modeli
(CI_Model)

Views
PHP skripte u kojima se
generiše HTML.
Poziva ih odgovarajući
kontroler

Jezgro Codeigniter-a



Sadrži sve bazne klase i sve što se tiče rada sa CI

Klase kroz koje CI ostvaruje rad sa bazom podataka

Ugrađeni helper-i i biblioteke

CI – način funkcionisanja

Index.php

```
/*
* Set a bunch of system and variables information,
* Routing information etc.
*/

/*
* LOAD THE BOOTSTRAP FILE
*
* And away we go...
*
*/
require_once BASEPATH.'core/CodeIgniter.php';

/* End of file index.php */
/* Location: ./index.php */
```

Codeigniter.php

```
/*
*  Security check
*/
$class  = $RTR->fetch_class();
$method = $RTR->fetch_method();

  if (...){...}
else{show_404("{$class}/{$method}");}...
/*
*  Instantiate the requested controller
*/
$CI = new $class();

/*
*  Is there a "post_controller_constructor" hook?
*/
$EXT->_call_hook('post_controller_constructor');

/*
*  Call the requested method
*/
// Is there a "remap" function? If so, we call it
instead
if (method_exists($CI, '_remap')){
 $CI->_remap($method, array_slice($URI->rsegments, 2));
}
else{...}

// Call therequested
methodcall_user_func_array(array(&$CI, $method),
array_slice($URI->rsegments, 2));
...
```

Kako funkcioniše?

- Kliknete na link http://localhost/psitut/index.php/news/showLatest
- Apache server vidi da treba doći do **index.php**
- Predaje joj url
- CodeIgniter analizira URL
  - Kontroler: news
  - Akcija: showLatest
    - Prema routes konfiguracij
  - I pravi se objekat klase News, poziva se metoda showLatest()

Controller
View
Helpers
Libraries
Forms – validation and creation (via libraries and helpers)
Model

KONTROLER I POGLED (CONTROLLER AND VIEW)

CI - Kontroler

- Naziv kontrolera – malim slovima
- 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

CI - View

- PHP skripta koja generiše HTML
- Kontroler odlučuje **koji** view se koristi i prosleđuje mu **podatke**
- Elementi prosleđenog niza su dostupni u navedenom view-u kao obične php promenljive

Mapa sa podacima koji se šalju stranici

Putanja do view-a, relativno, u odnosu na **views** direktorijum

```
$this->load->view('templates/page',
  array(
  'body'=> 'formTesting/index',   'pageTitle'=>'primer
forme',
  'title'=>'Primer forme'
));
```

Modularnost view-ova

- Često se ponavljaju delovi stranica
  - Header, footer, menu, navigation…
- Napraviti templates, koji sadrže `header.php`, `footer.php` i `body.php`
  - Unutar body se layout-uje php stranica koja predstavlja glavni sadržaj, a putanja do nje je prenešena body.php template-u
  - Jedno od mogućih rešenja, moguće je i drugačije raditi

Template (šablon) za stranice

views/hello/index.php

```
<! <!DOCTYPE html>
<html lang="en">
<head>
  <title><?= $pageTitle ?></title>
</head>
<body>

<div id="container">
  <h1>Welcome to CodeIgniter!</h1>
  <div id="body">
```

2

```
  <p>The page you are looking at is being
generated dynamically by CodeIgniter.</p>

  <p>If you would like to edit this page you'll
  find it located at:</p>

  <p>The corresponding controller for this
   page</p>
<code>application/controllers/welcome.php</code>
```

```
 </div> <!-- body end-->

  <p class="footer">Page rendered in
   <strong>{elapsed_time}</strong> seconds</p>
</div>
</body>
</html>
```

views/templates/*

header.php 1

```
<! <!DOCTYPE html>
<html lang="en">
<head>
  <title><?= $pageTitle ?></title>
</head>
<body>

<div id="container">
  <h1>Welcome to CodeIgniter!</h1>
  <div id="body">
```

footer.php 3

```
</div> <!-- body end-->

  <p class="footer">Page rendered in
   <strong>{elapsed_time}</strong> seconds</p>
</div>
</body>
</html>
```

page.php

```
<?php
$this->load->view('templates/header');
$this->load->view($body);
$this->load->view('templates/footer');
```

```
$this->load->view('templates/page', array('body'=> 'hello/index'));
```

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 helperi:
  - url_helper, form_helper, date_helper, download_helper…
- Upotreba: u kontroleru ili view-u, potrebno učitati helper:

```
$this->load->helper('form_helper');
```

- Nakon toga, fajlovi su učitani, i funkcije su dostupne za upotrebu
  - Zapravo se pri `load` izvrši `include` fajlova sa defnicijama funkcija koje čine helper

Navigacija – linkovi

- Linkovi se mogu postavljati ručno u view
- Moguće i koristiti `anchor()` iz helper-a **url_helper**.

```
anchor($putanja, $tekst, $opcije)
```

- Link može voditi na apsolutnu ili relativnu putanju
- Relativna putanja:

```
ime_kontrolera/ime_akcije/par1/par2/…/parn
```

- Potpis akcije:

```
function ime_akcije($p1, $p1, …, $pn)
```

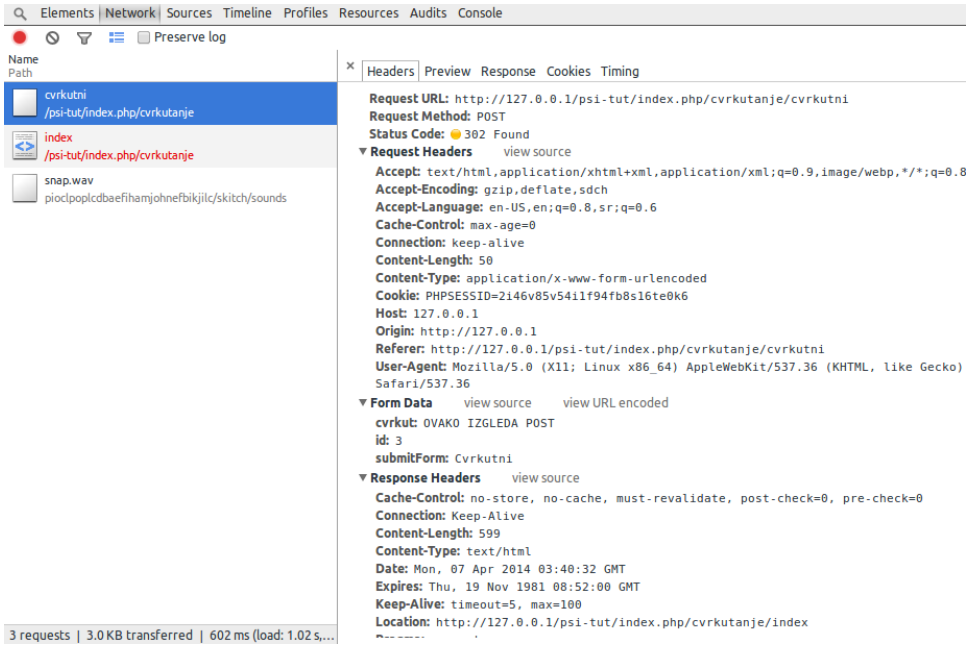
- Sve nakon imena akcije se „pakuje“ u parametre akcije

Parametri akcija

- Akcije mogu primati parametre preko HTML komponenti za unos podataka (input, textarea, select, checkbox, radio)
- Svaka input komponenta moda imati **`name`** atribut
- Unetoj vrednosti se pristupa koristeći vrednost navedenu u **`name`** atributu
- `<input type='text' name='ime' />`
- php: koristeći 'ime' pristupa vrednosti unetoj u to tekst polje:
  `$_GET['ime']`
- CI: unutar kontrolera, preko **`input`** objekta:
  `$this->input->get('ime')`
- Pristup preko POST parametara:
  `$this->input->post('ime')`
- Pristup nezavisno na način dostavljanja:
  `$this->input->get_post('ime')`

POST vs GET

- **GET** parametri su parametri koji se zadaju kroz sam URL
- **POST** parametri su oni koji se šalju serijalizacijom forme
(unutar tela HTTP zahteva)
- **GET**: kada se neki resursi „dohvataju“ (pretrage, dovlačenje stranica i sl.)
- **POST**: kada se šalju informacije koje dovode do promene stanja aplikacije (čuvanje podataka u BP npr.),
i kada ne bi trebalo ostaviti korisniku URL za slobodnu upotrebu



`http://localhost/psi-tut/index.php/formTesting/process?search=trazimo`

Rad sa formama u CI

- CI pruža podršku generisanju formi, u vidu helpera: **form_helper**
  - Primereno je koristiti samo na view-u, ne i unutar akcija kontrolera, budući da ovaj helper generiše HTML kôd za forme
- Postoji i podrška za validaciju unetih podataka
  - U vidu biblioteke (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

- Otvaranje forme

```
echo form_open("formTesting/process", array("method"=>"get"));
```

  - Podrazumevana metoda je PUT

- Dodavanje komponenata (deo view skripta):

```
echo form_label("Pretraga", "usernameID");
// generiše <label for=„usernameID“>Pretraga</label>
// labela se vezuje za element sa ID jednakim for atributu labele
// labela se vezuje za
echo form_input(array(
        'name' => 'search', // name of element => parameter name
        'id' => 'usernameID'
));
// generiše <input type=’text’ name=„search“ id=„usernameID“/>
```

- Zatvaranje forme

```
echo form_close();
```

- Razdvajanje logike i prikaza? –tu je.
- Ostale komponente funkcionišu slično.

Rad sa bibliotekama (library)

- Biblioteke predstavljaju kompleksnije pomoćne mehanizme u realizaciji čestih operacija
  - Moguće koristiti bilo u View, Model ili Controller
- Ugrađene biblioteke se nalaze u `system/libraries`
- Dodatne (vaše) biblioteke se smeštaju u `application/libraries`
- Biblioteku je potrebno učitati (moguće u M, V, C)
  - Koristi se naziv direktorijuma u kom je smeštena: system/**calendar**

```
$this->load->library('calendar');
```

- Nakon toga, u objekat kontrolera se „dodaje“ polje koje nosi isti naziv kao biblioteka, pa je moguće na taj način pristupiti metodama koje ona nudi:

```
echo $this->calendar->generate(2014, 4);
                        // generiše kalendar za 4. 2014.
```

- Dodato polje predstavlja fasadu ka funkcionalnostima koje biblioteka pruža (*façade* uzorak)
- Kontroler ne zna i ne poznaje internu organizaciju biblioteke

Validacija formi

- Validacija podrazumeva semantičku proveru unetih vrednosti
- U CI se ostvaruje pomoću biblioteke `form_validation`

Naziv polja za koje se definišu pravila (na formi postoji polje `username`)

Pravila koja moraju biti zadovoljena

```
$this->form_validation->set_rules('username', 'Korisnicko ime',
'trim|required|max_length[30]');
```

```
$this->form_validation->set_rules('id', 'ID', 'required');
if ($this->form_validation->run() == FALSE) {
// nekorektni podaci, prikazati (nekako) ponovo formu uz poruku
// o greški
} else {
// podaci su korektni, nastaviti obradu (npr. dodati nešto u BP)
// potom OBAVEZNO uraditi redirect (šta se desi ako se ne uradi?)
}
```

Validacija formi

- Pravila se navode u okviru jednog stringa
- Zapravo su nazivi funkcija
- Pored pravila, postoje i „obrade“

```
'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)
  - Metoda kontrolera se koristi za validaciju
  - U pravilu se referiše kao `callback_<ime_metode>`
    - Neophodan prefiks `callback_`
  - Prvi parametar je uvek polja (koja se validira), a drugi parametar može biti predat preko `[]`
  - Povratna vrednost metode: `TRUE/FALSE`

Validacija formi

- Nakon sprovedenog validiranja (metoda `run`), potrebno je prikazati poruke o greškama.
- Biblioteka za validaciju ima „spremne“ generičke poruke

```
$lang['required'] = "The %s field is required.";
```

- Parametar poruke je naziv polja, koji je proizvoljan, i podešava se prilikom podešavanja pravila

```
$this->form_validation->set_rules('id', 'ID', 'required');
```

- Poruke se mogu prikazati na view-u preko funkcije `validation_errors()`.
- Polja forme se mogu popuniti unetim (nekorektnim) vrednostima koristeći funkciju

```
set_value("id")
```

Validacija formi

- Moguće je sačuvati pravila o validaciji u eksternom izvoru: application/config/form_validation.php
- Moguće je grupisati pravila, tako da se odnose na različite forme

```
$config = array(
   'signup' => array(array(
                'field'   => 'username',
               'label'   => 'Username',
               'rules'   => 'required'),…
  ),
   'email' => array(…)
);
```

- Tada se, prilikom validacije, referiše skup pravila po imenu:

```
$this->form_validation->run('signup');
```

Form Resubmission

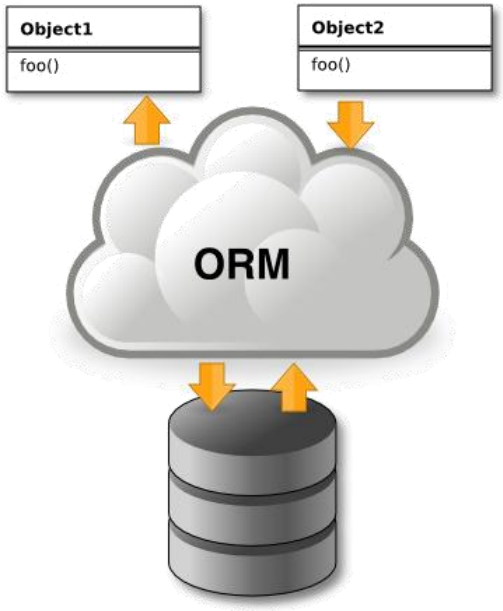
- Pojava koja se događa kada je nakon submit forme na neku akciju to moguće učiniti ponovo, refresh-om stranice
- ***Submit forme*** je HTTP zahtev koji sadrži vrednosti polja (u URL kod GET formi ili unutar HTTP zahteva, kod POST formi)
- Refresh u browseru (F5, refresh dugme) dovodi do slanja istog zahteva preko kog se došlo na tekuću stranicu
- Kada se na stranicu dođe putem subtmit-a forme, *refresh je ekvivalentan ponovnom submit-u*, uz slanje istih podataka
- Nakon što se u akciji obradi sadržaj POST forme, neophodno je poslati **redirect** browser-u (preusmeriti ga na drugu akciju)
  - U PHP pomoću `header( 'Location: <putanja>' ) ;`
  - CI: `url_helper` funkcija `redirect(<akcija>);`
- Time se browseru da instrukcija da poseti stranicu `<akcija>`, čime se rešava ovaj problem

Model

- Model predstavlja klasu zaduženu za perzistiranje i pribavljanje potrebnih podataka
- Neophodno da extenduje `CI_Model`
- Nije obavezno da koristi bazu podataka, niti je obavezno da koristi ugrađeni ORM mehanizam
- Sadržaj klase:
  - Relevantna svojstva objekata sa kojima se radi (ime, datum rođenja i sl)
  - Metode koje vrše perzistiranje objekata
  - Metode koje vrše pretrage objekata, dohvatanje svih postojećih objekata i sl.

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 kôd
- 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“



CI Modeli i baza podataka

- CI nudi implementaciju ActiveRecord pattern-a 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 kôd
    - 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 izmena (*migracije*)

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 – mockovati bazu podataka
- Dalje o ORM, na primeru Doctrine:
  na narednoj lab vežbi

```
$query = $this->db->get_where('mytable',
                              array('id' => $id), $limit, $offset);
```

```
$query = $this->db->query("YOUR QUERY");

foreach ($query->result() as $row)
{
   echo $row->title;
   echo $row->name;
   echo $row->body;
}
```

Model

Model –domenska klasa

Svojstva – šta čuvati za posmatrani objekat

Učitavanje *neke* biblioteke za rad sa bazom podataka ili drugim načinom perzistiranja objekata

Metode za pretragu

Metoda za perzistiranje

```
class Person extends CI_Model{
    var $name = "";
    var $id = 0;
    function __construct(){
        parent::__construct();
        $this->load->library('sessionds');
    }

    public function findPersonByName($name) {
        return $this->sessionds->select("persons", array("name" => $name));
    }

    public function listPersons() {
        return $this->sessionds->select("persons");
    }

    public function findPerson($id) {
        return $this->sessionds->select("persons", array("id" => $id));
    }

    public function addPerson($name){
        $p = new Person(); $p->name=$name;
        $this->sessionds->insert("persons", $p);
    }

}
```

OSTALI KONCEPTI I LINKOVI

Ostali koncepti

- ***Rutiranje***
  - Ako podrazumevano tumačenje URL-ova (`kontroler/akcija/param`) nije zadovoljavajuće, konfigurisati specifične putanje kroz `config/routing.php`
- Komunikacija sa ***BP***: `config/database.php`
- ***Internacionalizacija*** (i18n): `languages` direktorijumu
  - 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, već pomoću CI podrške: `$this->lang->line('username');` umesto `'Korisnicko ime:'`
  - Prevođenje se vrši promenom odgovarajućih fajlova u languages direktorijumu

Ostali koncepti

- ***Hooks***: pre i posle odgovarajućih akcija koje CI sprovodi, moguće „ubaciti“ sopstveni kôd:
  - `pre_controller`
  - `post_controller`
  - Navode se klasa i metoda koju treba pozvati
- ***Prikazivanje grešaka*** u obradi: postoje template-i za svaki (`application/errors`)
  - Statusi: 404, 505, 500, 403…
- Rad sa ***sesijom***: preko `session` biblioteke je preporučeni pristpup. Tada je neophodno defnisati session key (proizvoljan string) u `config/config.php`.
- ***Migracije***: pri radu sa modelom, moguće je (ručno) specificirati izmene koje su napravljene i način na koji bi trebalo ažurirati sanje u bazi podataka (v. linkove)

Korisni linkovi - CI

- http://ellislab.com/codeigniter/user-guide/index.html
- http://www.slideshare.net/Chirag2411/code-igniter-documentation
- http://www.phpeveryday.com/articles/CodeIgniter-Framework-Basic-Tutorial-P841.html
- http://ellislab.com/codeigniter/user-guide/libraries/form_validation.html
  Validacija formi
- http://ellislab.com/codeigniter/user-guide/libraries/sessions.html
  Rad sa sesijom
- http://ellislab.com/codeigniter/user-guide/libraries/migration.html
  Migracije - promene načinjene nad modelom i njihova sinhronizacija sa bazom
- http://ellislab.com/codeigniter/user-guide/helpers/form_helper.html

Korisni linkovi - opšte

- http://martinfowler.com/eaaCatalog/
  Katalog uzoraka za kompleksne aplikacije
- http://www.sitepoint.com/best-php-frameworks-2014/
  Poređenje PHP web framework-a
- http://java.dzone.com/articles/comparing-jvm-web-frameworks
  Poređenje Java web framework-a, uopšte o poređenju i odabiru framework-a
- http://en.wikipedia.org/wiki/Convention_over_configuration
  O konvenciji nad konfuguracijom
- Preporučeno pogledati:
  - PHP: CodeIgniter, Larvel, Yii
  - Java: Vaadin, Grails, Spring, Spring MVC, GWT