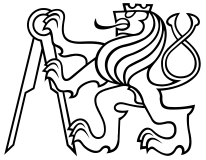
Oddělení aplikační a prezentační logiky Mustache

Martin Klíma

DCGI



Architktura MVC

M = Model

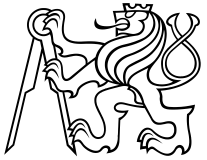
V = View

C = Controller

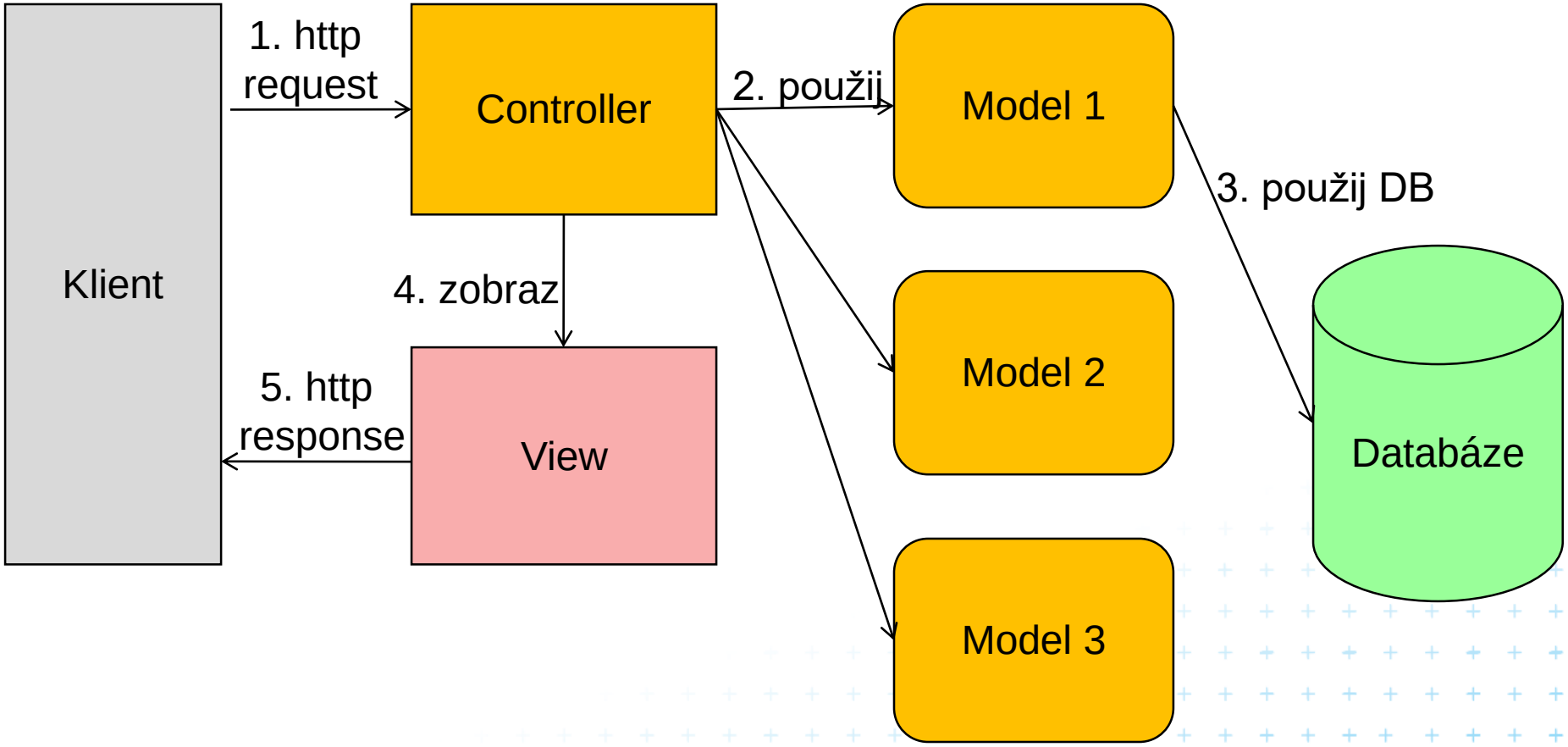
Model reprezentuje aplikační logiku

View reprezentuje prezentační logiku

Controller reprezentuje logiku, která to vše řídí



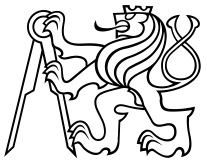
MVC


Klient
1. http request
Controller
2. použij
Model 1
3. použij DB
Databáze
4. zobraz
Model 2
5. http response
View
Model 3

Výhody MVC

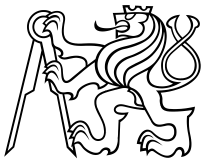
- Dobré oddělení aplikační (Model) a prezentační (View) logiky
- Přehlednost
- Objektovost
- Možnost dělby práce
  - view má na starosti grafik
  - model má na starosti programátor
- Lepší testování

- Co se v praxi nejvíc mění (u webové aplikace)
  - view



Nevýhody MVC

- Složitější projekt
- Více různých souborů
- Pro malé věci možná zbytečné
  - …. to je možná moc silné tvrzení ☺

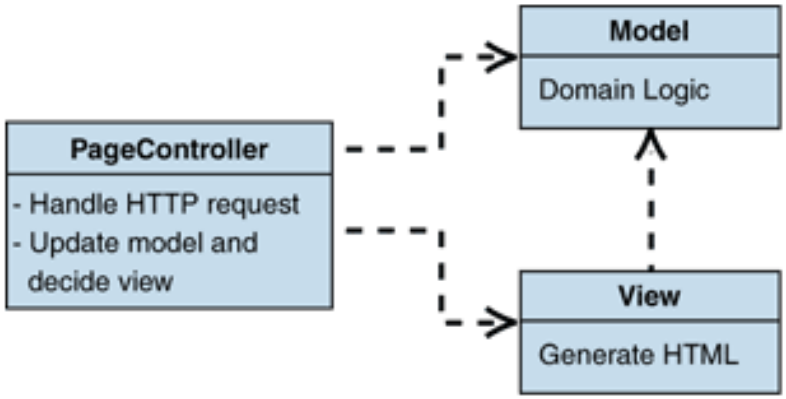


Varianty MVC

- Existuje řada variant
  - Page Controller
  - Front Controller
  - Composite View

Varianty MVC – Page Controller

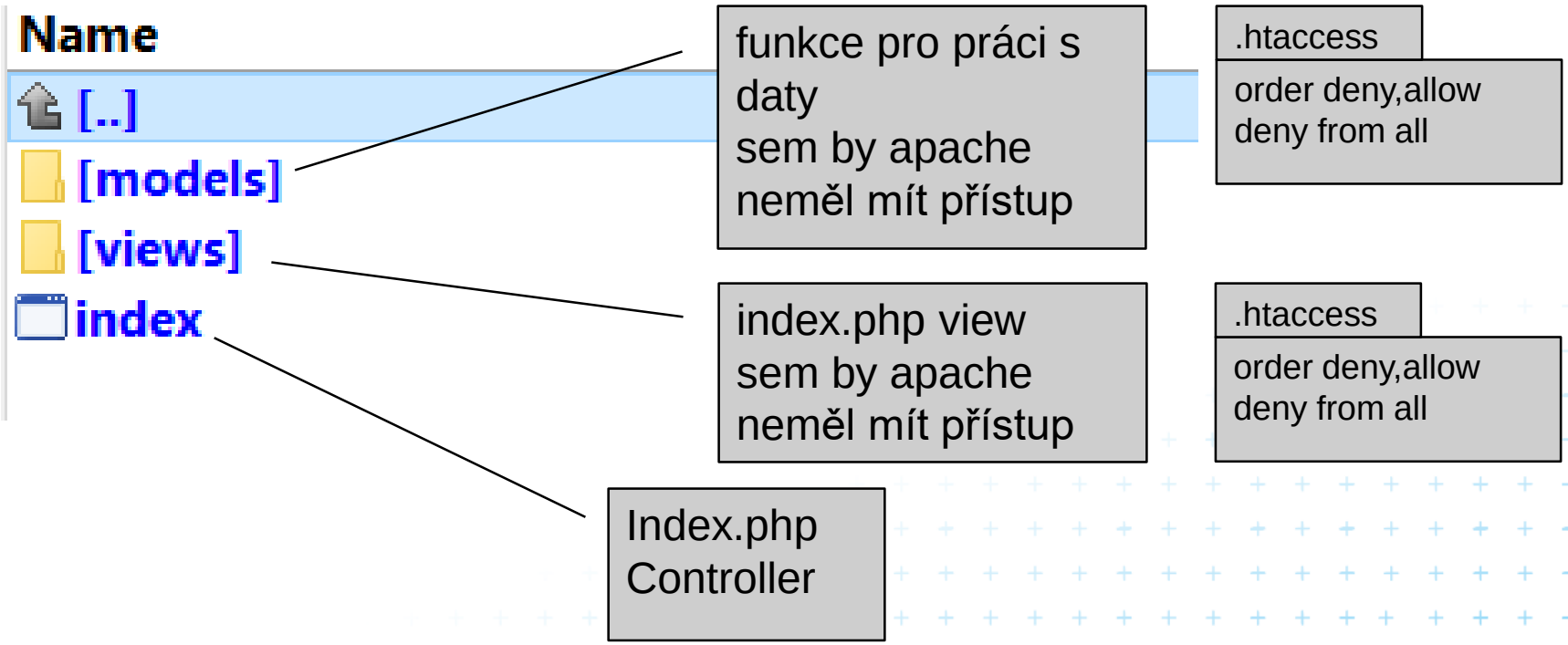
Každá adresa (v PHP skript) má svůj vlastní controller



Zdroj MSDN http://msdn.microsoft.com/en-us/library/ms978764.aspx

Page Controller

- Odpovídá nativnímu mapování URL na název skriptu.

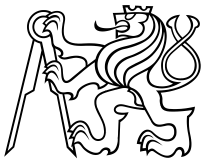


Controller

```php
include ("models/data.php");
const GROUP_PARAM = "group";
const BOYS = "boys";
const GIRLS = "girls";

if (!isset($_GET[GROUP_PARAM])) {
        $_GET[GROUP_PARAM]= GIRLS;
}

// podle parametru v URL zavolame prislusny model
switch ($_GET[GROUP_PARAM]) {
        case BOYS: $people = getBoys(); break;
        case GIRLS: $people = getGirls(); break;
        default: $people = getGirls();
}



// ted mame data v promenne $people
// rozhodne je nebudeme vypisovat v controlleru
// posleme je do view
include ("views/index.php");
```

Model

```php
<?php

function getBoys() : array {
	return array ("Martin", "Petr", "Milan", "David",
"Jan");
}

function getGirls() : array {
	return array ("Alice", "Lenka", "Ivana", "Simona",
"Emma");
}
```



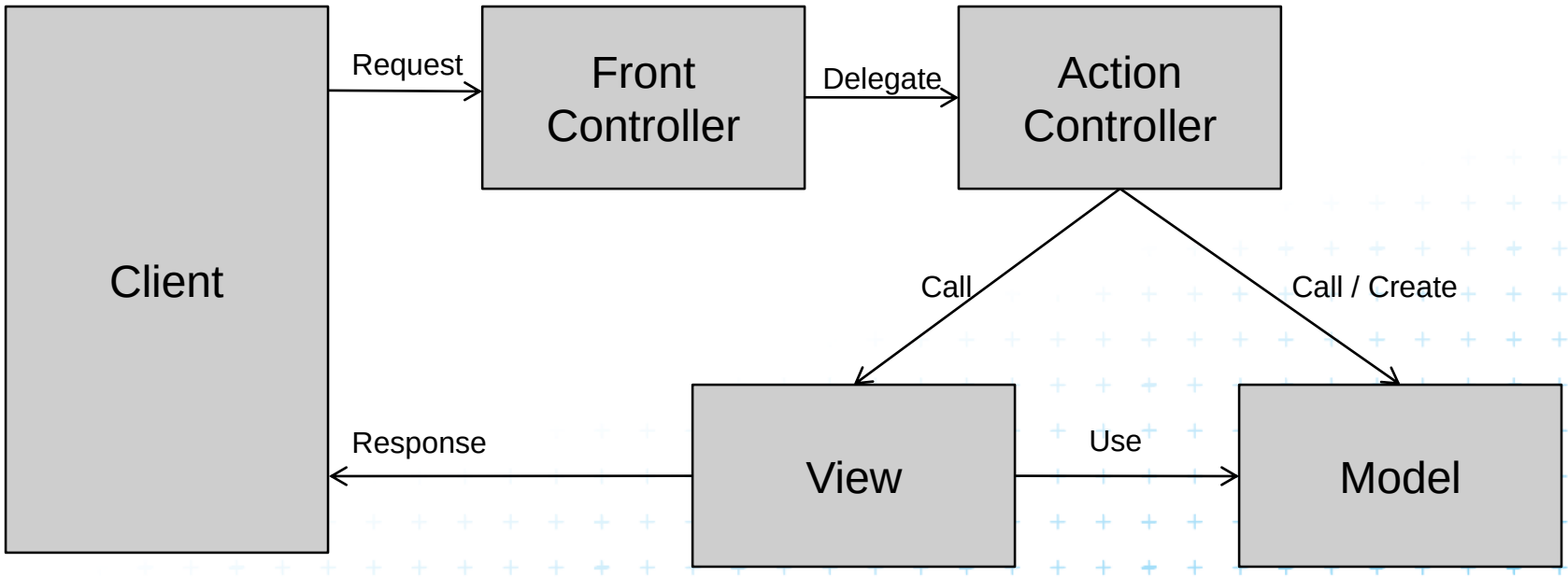
View

```
<!doctype html>
<html>
<head>
<title>Seznam lidí</title>
</head>
<body>
<h1>Skupina lidí</h1>
<table>
<tr><th>#</th><th>Jméno</th></tr>
<?php
$index = 1;
foreach ($people as $person) {
        echo "<tr>\n<th>".$index++;
        echo
"</th>\n<td>".htmlspecialchars($person)."</td>\n</tr>";
}
?>
</table>
</body>
</html>
```

Varianty MVC – Front Controller

Jeden Controller pro všechny stránky. V PHP se dá zajistit pomocí mod-rewrite v Apache.

Front Controller udělá globální práci nutnou pro všechny dotazy a deleguje dotaz na konkrétní implementaci



Mod Rewrite

```
Options +FollowSymLinks
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !(\.)
RewriteRule ^(.*)$ index.php?params=$1 [QSA,L]
```

Více o MOD_REWRITE například zde:
https://httpd.apache.org/docs/current/mod/mod_rewrite.html

Vyzkoušejte si, jak MOD_REWRITE funguje:
https://htaccess.madewithlove.be/

Front Controller

```php
// rekneme, ze prvni param je jmeno action controlleru
const PARAMS = 'params';
const ACTION_CONTROLLERS = "action_controllers";

$params = $_GET[PARAMS];
include ("libs/uri_libs.php");

$todo = (parseUri($params));

// ted vime, ktery action controller se ma zavolat
// je to $todo[0];
// a vime, jaka akce se ma zavolat
// je to $todo[1]
// a zname vsechny dalsi parametry, jsou to $todo s indexem > 1

$action_controller = ACTION_CONTROLLERS."/".$todo[0].".php";
if (file_exists($action_controller)) {
        include ($action_controller);
} else {
        echo ("Non existing action controller {$todo[0]} called.");
        exit();
}

if (function_exists($todo[1])) {
        $todo[1]();
} else {
        echo ("Non existing method {$todo[1]} called.");
        exit();
}
```

Action Controller

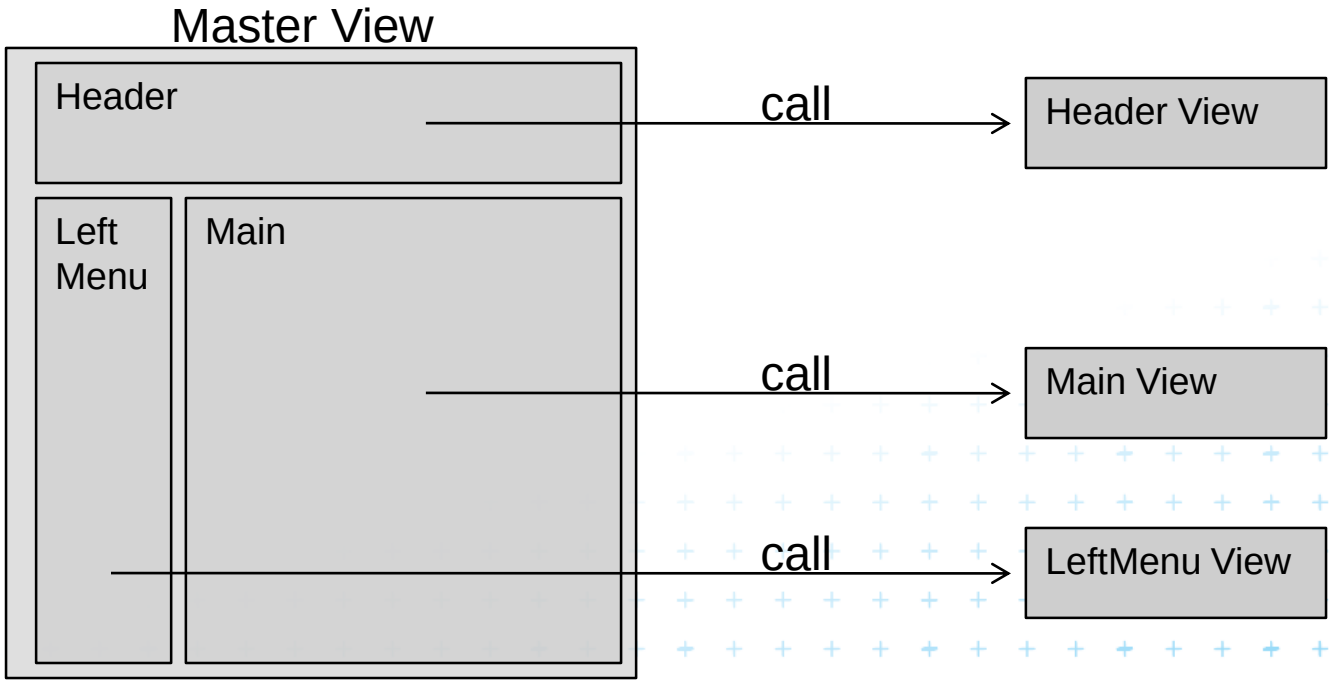
```php
<?php

include ("models/data.php");
const GROUP_PARAM = "group";
const BOYS = "boys";
const GIRLS = "girls";



function basic() : void {
        girls();
}

function boys () : void {
        $people = getBoys();
        include ("views/people.php");
}

function girls () : void {
        $people = getGirls();
        include ("views/people.php");
}
```

View

```
<!doctype html>
<html>
<head>
<title>Seznam lidí</title>
</head>
<body>
<h1>Skupina lidí</h1>
<table>
<tr><th>#</th><th>Jméno</th></tr>
<?php
$index = 1;
foreach ($people as $person) {
        echo "<tr>\n<th>".$index++;
        echo
"</th>\n<td>".htmlspecialchars($person)."</td>\n</tr>";
}
?>
</table>
</body>
</html>
```

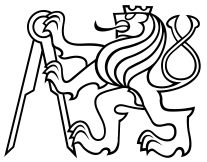
Composite View

- View se skládá z globální předlohy, která rozvrhuje menší části definované jinými View
- Skinovatelné aplikace, abstrakce



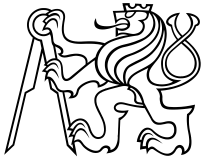
PHP a MVC

- PHP nemá nativní podporu pro MVC
- Nicméně je to možné "naroubovat"



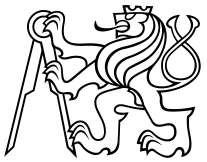
Template engine

MUSTACHE

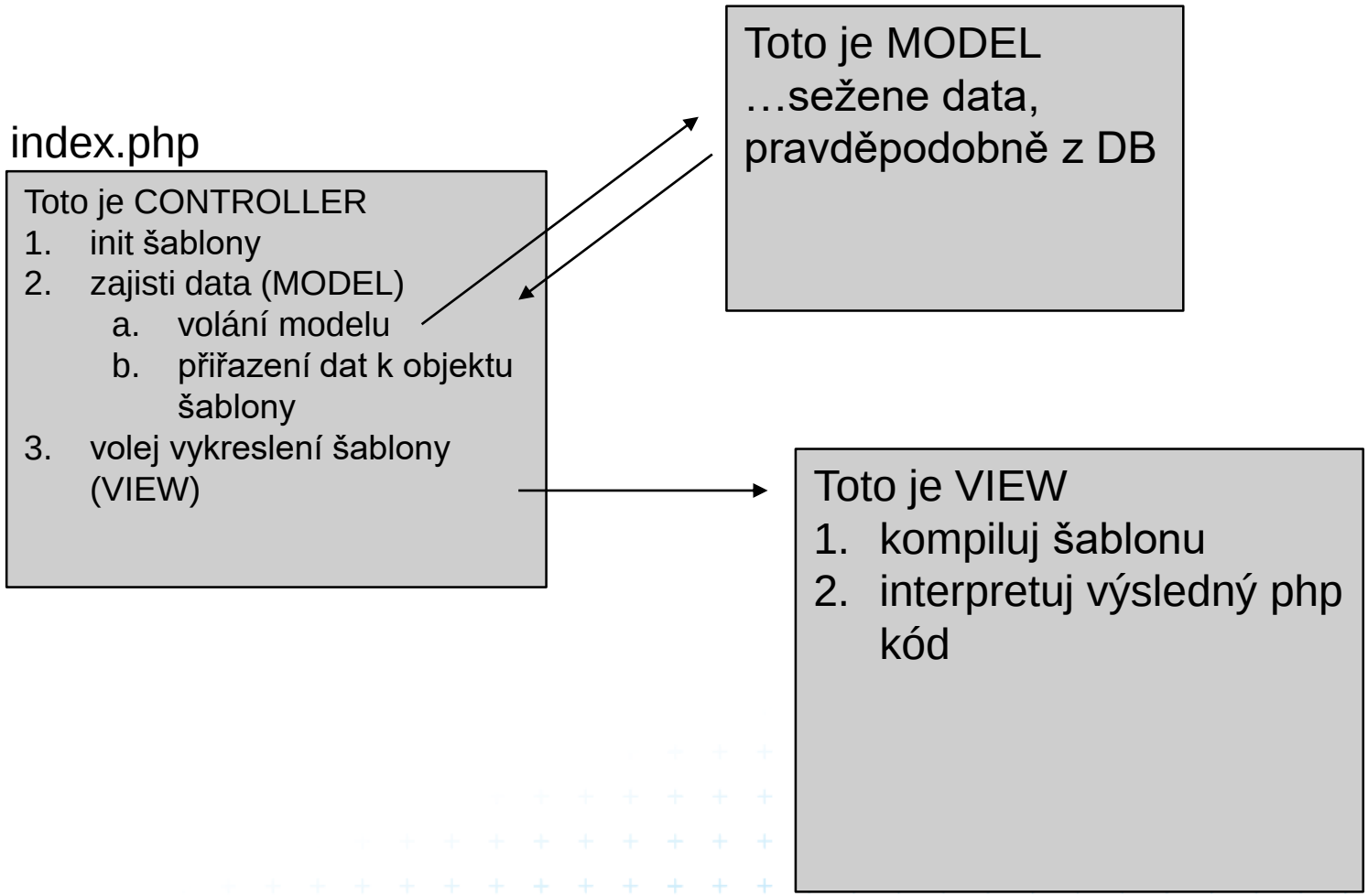


Mustache

- Šablonovací engine
- Poskytuje podporu pro MVC - primárně pro View část
- https://mustache.github.io/
- Multiplatformní
  - Včetně PHP



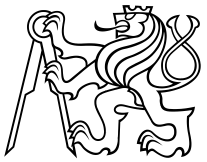
Základ


Toto je MODEL
…sežene data, pravděpodobně z DB
index.php
Toto je CONTROLLER
1. init šablony
2. zajisti data (MODEL)
a. volání modelu
b. přiřazení dat k objektu šablony
3. volej vykreslení šablony (VIEW)
Toto je VIEW
1. kompiluj šablonu
2. interpretuj výsledný php kód

Drobná ukázka Mustache

```php
<?php
// toto je controller

//příprava View datových struktur, inicializace
require_once('Mustache/Autoloader.php');
Mustache_Autoloader::register();
$mustache = new Mustache_Engine(array(
    'loader' => new Mustache_Loader_FilesystemLoader(dirname(__FILE__).'/templates')
));

$template = $mustache->loadTemplate('index');

// teď data - o to se má postarat model
// data si prozatím vymyslíme
$data = array ('firstname'=>'František', 'lastname'=>'Vomáčka');


// necháme to vykreslit tím, že zavoláme view renderování
echo $template->render($data);
```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Uživatel</title>
</head>
<body>
Uživatel {{firstname}} <strong>{{lastname}}</strong>
</body>
</html>
```



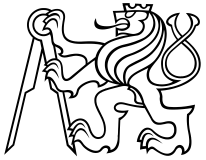
Cross Site Scripting?

- Je automaticky řešeno

```
$data = array ('firstname'=>'František', 'lastname'=>'<Vomáčka>');
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Uživatel</title>
</head>
<body>
Uživatel František <strong>&lt;Vomáčka&gt;</strong>
</body>
</html>
```

- Pokud si nepřejeme autmatický escaping, použijeme {{{ }}}



Iterace přes pole hodnot

```
//…stejné jako předchozí příklad

$data = array ('person'=> array(
        array('firstname'=>'František', 'lastname'=>'<Vomáčka>'),
        array('firstname'=>'Zdeněk', 'lastname'=>'Lakotík'),
        array('firstname'=>'Anna', 'lastname'=>'Skřivánková'))
);
```

Sekce

```
<!DOCTYPE html>
<html>
<head>
    <title>Uživatel</title>
</head>
<body>
<h1>Seznam uživatelů</h1>
<ol>
  {{#person}}
    <li>{{firstname}}, {{lastname}}</li>
  {{/person}}

</ol>
</body>
</html>
```


Modifikátory

```php
//příprava View datových struktur, inicializace
require_once('Mustache/Autoloader.php');
Mustache_Autoloader::register();
$mustache = new Mustache_Engine(array(
    'loader' => new Mustache_Loader_FilesystemLoader(dirname(__FILE__).'/templates')
));

// přidáme pomocné funkce
$mustache->addHelper('case', array(
    'lower' => function($value) { return strtolower((string) $value); },
    'upper' => function($value) { return mb_strtoupper((string) $value); },
));

$template = $mustache->loadTemplate('modifier');

// teď data - o to se má postarat model
// data si prozatím vymyslíme
$data = array ('firstname'=>'František', 'lastname'=>'<Vomáčka>', 'upper'=>"modify");

// necháme to vykreslit tím, že zavoláme view renderování
echo $template->render($data);
```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Uživatel</title>
</head>
<body>
{{%FILTERS}}
Uživatel
    {{firstname | case.upper}}
     <strong>{{lastname | case.lower}}</strong>
</body>
</html>
```