

**DCGI**

**KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE**

# Tvorba automatické dokumentace

PHPDoc

# O co jde?

- Zdrojový kód je třeba dokumentovat
  - Nepracujeme sami
  - Ke kódu je třeba se za čas vrátit
  - Dílo je třeba předat
  - Dílo se může rozšiřovat

# Triviální způsob komentování

```
<?php

/*
  Tento formulář demonstruje, jak jednoduše zpracovat data z formuláře.
  Ukázkový formulář obsahuje základní informace, jako jsou jméno, příjmení, adresa,
  stav (svobodný, ženatý/vdaná)
*/

include("helpers/form_functions.php");
include("models/data_functions.php");

// definuji vychozi hodnoty poli formulare

if (isSubmittedForm()) {
    $errors = validate();
    $data = array();
    if (sizeof($errors) > 0) {
        // formular byl odeslan, ale byly objeveny chyby, zpet na pohled formulare.
        include "views/formular.php";
    } else {
        // formular byl odeslan a nebyly objeveny zadne chyby
        // data by bylo dobre nejak zuzitkovat, napriklad zapsat do databaze
        insertOrUpdateStudent(
            $_POST["id"], $_POST['jmeno'], $_POST['prijmeni'], $_POST['ulice'], $_POST['cislo'],
isset($_POST['zenaty'])
        );
    }
}
```

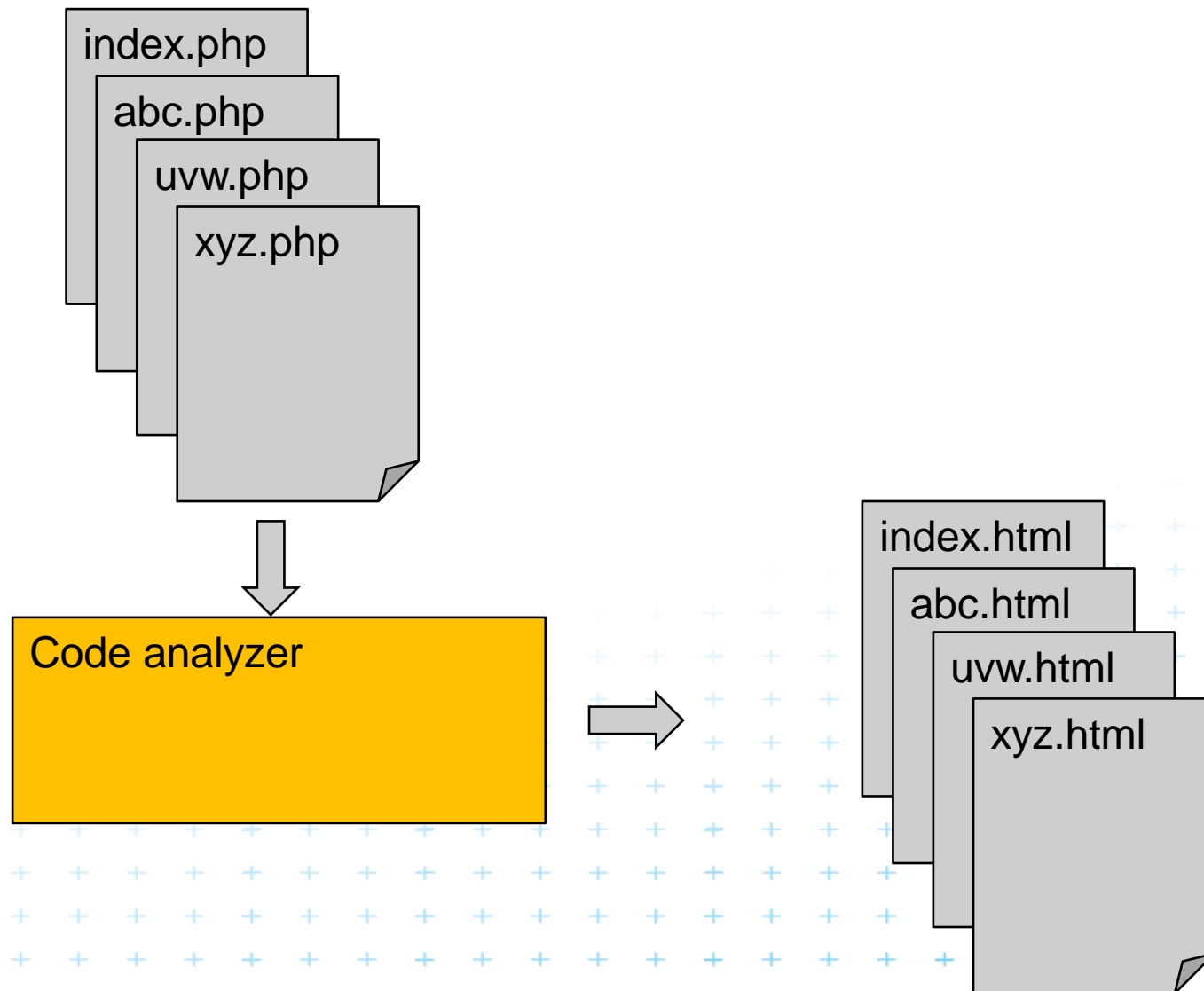
Blokový  
komentář

Řádkový  
komentář

# Lze zavést systém

- Co komentovat – k čemu se komentáře váží?
  - Celý soubor
  - Funkce a metody
  - Třídy a Interface
  - Proměnné
  - Konstanty
  - `include(_once)`

# Automatický dokumentátor



# Jaké existují dokumentátory?

- PHP Documentor
- JavaDoc
- Doxygen
- DocX
- Swagger io

# PHP Documentor - phpdoc <https://www.phpdoc.org/>

```
/**
 * This file is the model for data Students
 * @author Martin Klima
 */

/**
 * @param $dataName Name of the verified element.
 * @param $defaultValues Default values.
 * @return Returns the value for the wanted element. The value is HTML safe.
 */
function getFormTextData(string $dataName, $defaultValues): string {
    if (isset($_POST[$dataName])) {
        $value = $_POST[$dataName];
    } else {
        if (isset($defaultValues[$dataName])) {
            $value = $defaultValues[$dataName];
        } else {
            $value = "";
        }
    }
    return htmlspecialchars($value);
}
```

Komentář k souboru

@author

je klíčové slovo

Komentář k funkci

@param

@return

jsou klíčová slova

# Jak začít?

- Do kořene projektu umístěte konfigurační soubor phpdoc.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpdocumentor
  configVersion="3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="https://www.phpdoc.org"

  xsi:noNamespaceSchemaLocation="https://raw.githubusercontent.com/phpDocumentor/phpDocumentor
/master/data/xsd/phpdoc.xsd"
>
  <title>Forms</title>
  <paths>
    <output>docs/api</output>
    <cache>docs/cache</cache>
  </paths>
  <version number="3.0.0">
    <api>
      <source dsn=".">
        <path>.</path>
      </source>
    </api>
  </version>
</phpdocumentor>
```

Název dokumentace

Adresář s výsledkem

Adresář pro cache

Adresář, který se bude indexovat





# Spuštění ve windows a linuxu

v linuxu spusťte příkazem

**phpdoc**

ve windows

stáhněte soubor phpdocumentor.phar například do c:\xampp\php  
vyrobte si .bat soubor phpdoc.bat s tímto obsahem

**php.exe c:\xampp\php\phpdocumentor.phar %\***

poté stačí v spustit dokumentaci příkazem

**phpdoc**

# Výsledek

## Packages

Application

## Reports

Deprecated

Errors

Markers

## Indices

Files

Application

## data\_functions.php

*This is an implementation of the Model. It manipulates all data.*

The data can be taken from any source like a database, file, or be defined statically.

## Tags

author

Martin Klíma

version

1.0

## Table of Contents

## Functions

- M** [getAllStudents\(\)](#): array<string|int, mixed>  
*Returns all data from a static data definition. Here the loading from database could be implemented. Something like `SELECT * FROM student ORDER BY prijmeni, jmeno;`*
- M** [getStudentById\(\)](#): array<string|int, mixed>  
*Function that looks for a student in all students by given ID in the param. It can be implemented by querying a database like `SELECT * FROM student WHERE student.id = $id;`*

## On this page

Table Of Contents

[Functions](#)

Functions

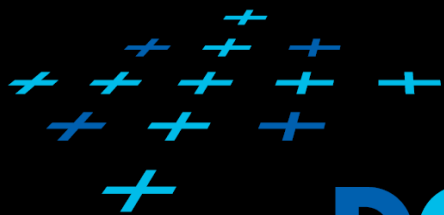
[getAllStudents\(\)](#)

[getStudentById\(\)](#)

[getStudentsByAttribute\(\)](#)

[insertOrUpdateStudent\(\)](#)





**DCGI**

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

# Oddělení aplikační a prezentační logiky Mustache

Martin Klíma

# Architektura MVC

M = Model

V = View

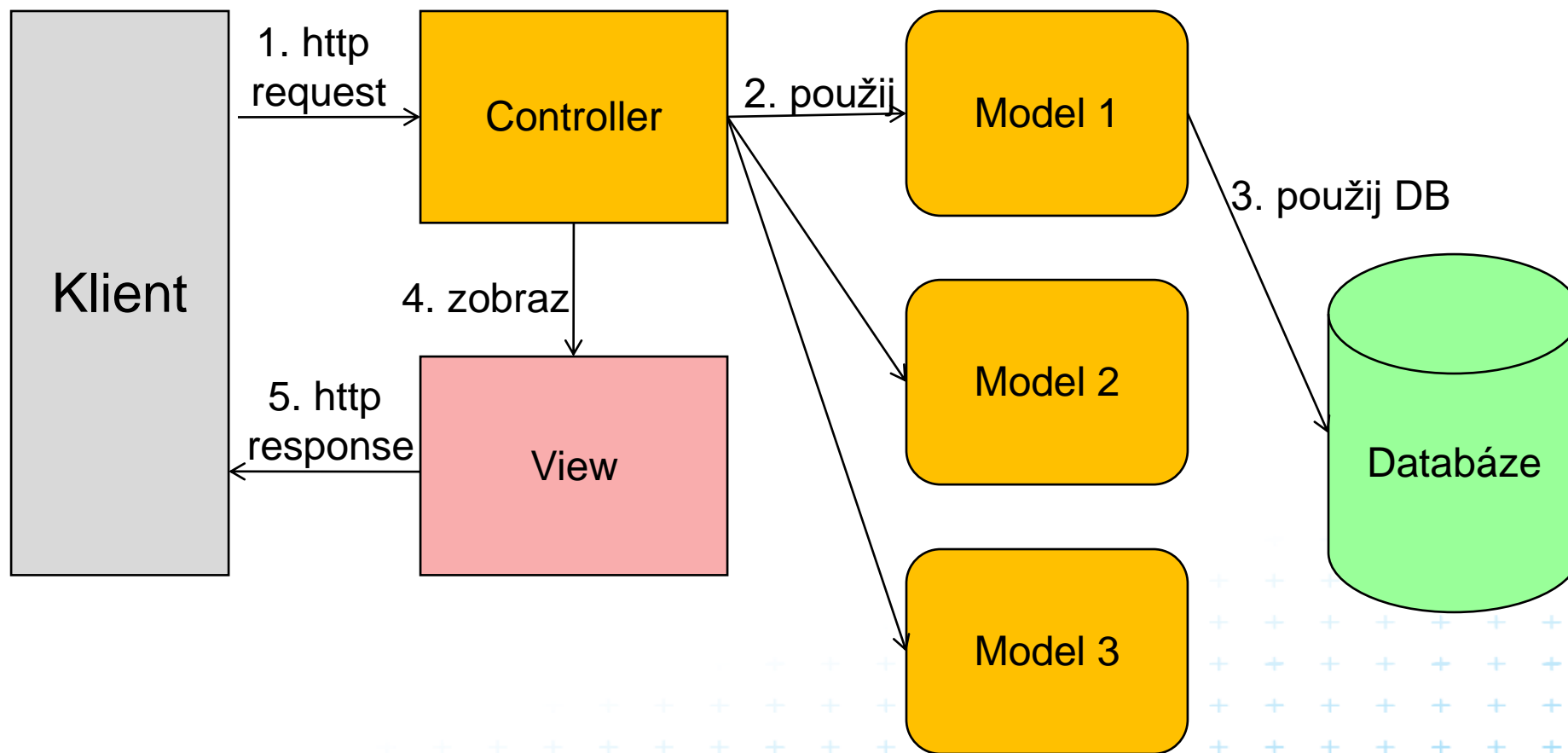
C = Controller

Model reprezentuje aplikační logiku

View reprezentuje prezentační logiku

Controller reprezentuje logiku, která to vše řídí

# MVC



# 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 😊

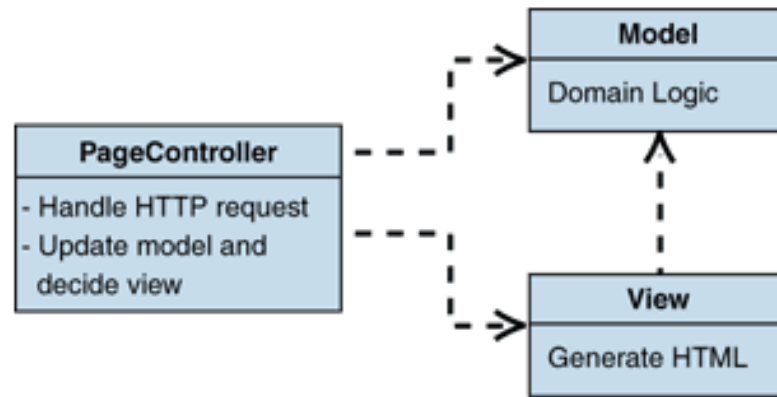
# Varianty MVC

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



# Varianty MVC – Page Controller

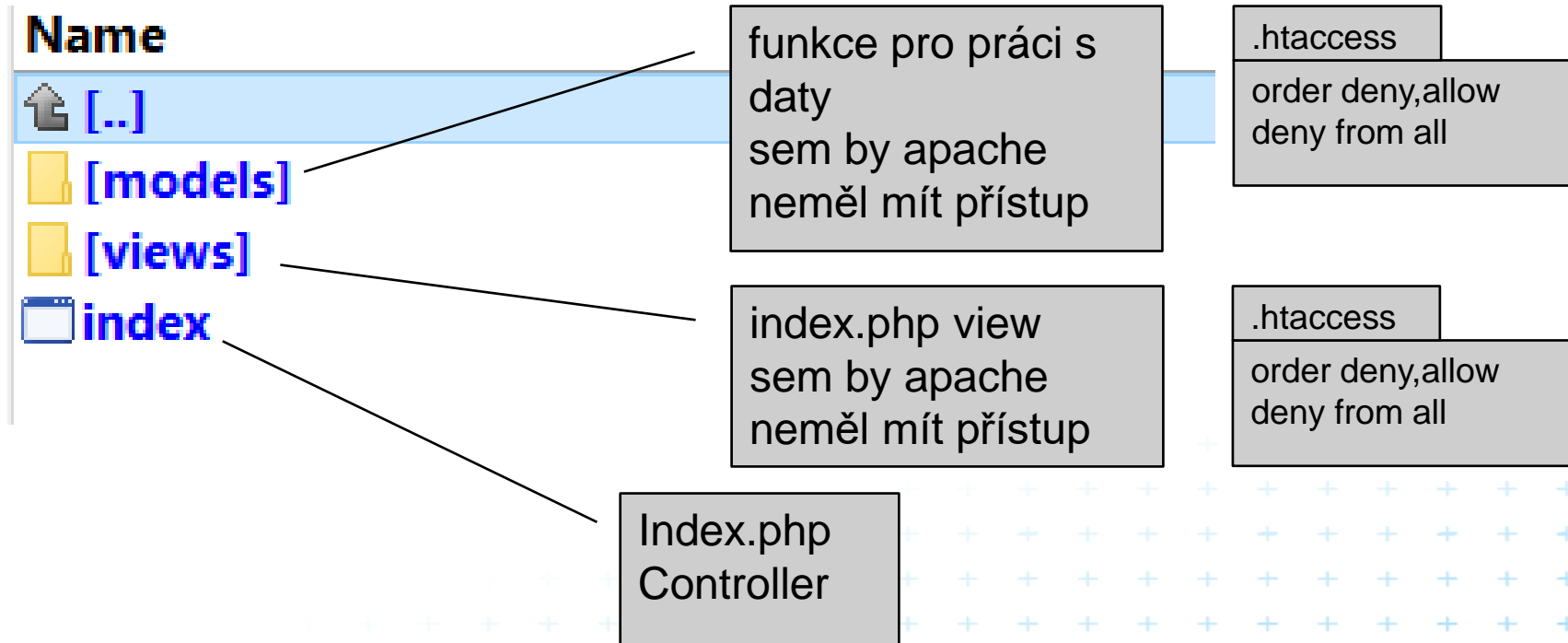
Každá adresa (v PHP skript) má svůj vlastní controller



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

# Page Controller

- Odpovídá nativnímu mapování URL na název skriptu.



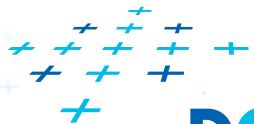
# Controller

```
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

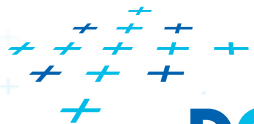
```
<?php
```

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

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

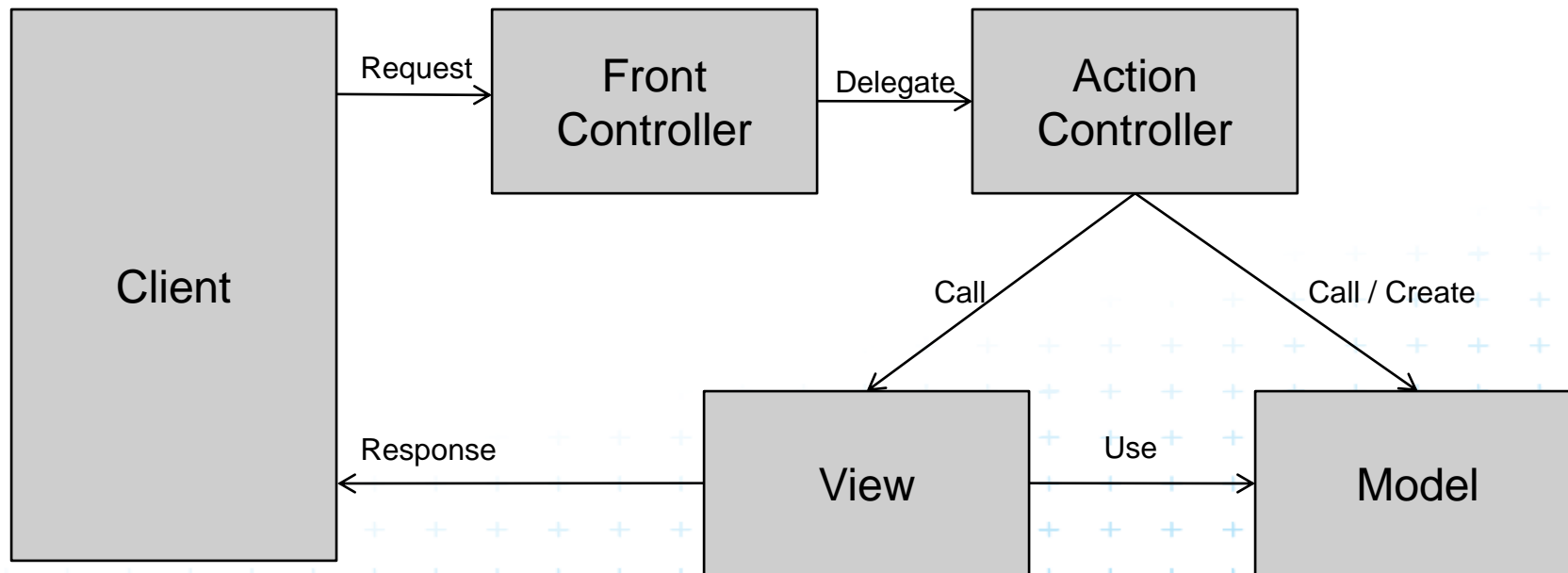
# View

```
<!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](https://httpd.apache.org/docs/current/mod/mod_rewrite.html)

Vyzkoušejte si, jak MOD\_REWRITE funguje:

<https://htaccess.madewithlove.be/>

# Front Controller

```
// rekname, 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 znam 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

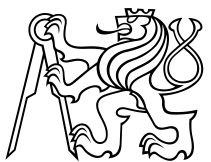
```
<?php

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

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

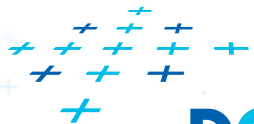
function boys () : void {
    $people = getBoys();
    include ("views/people.php");
}

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



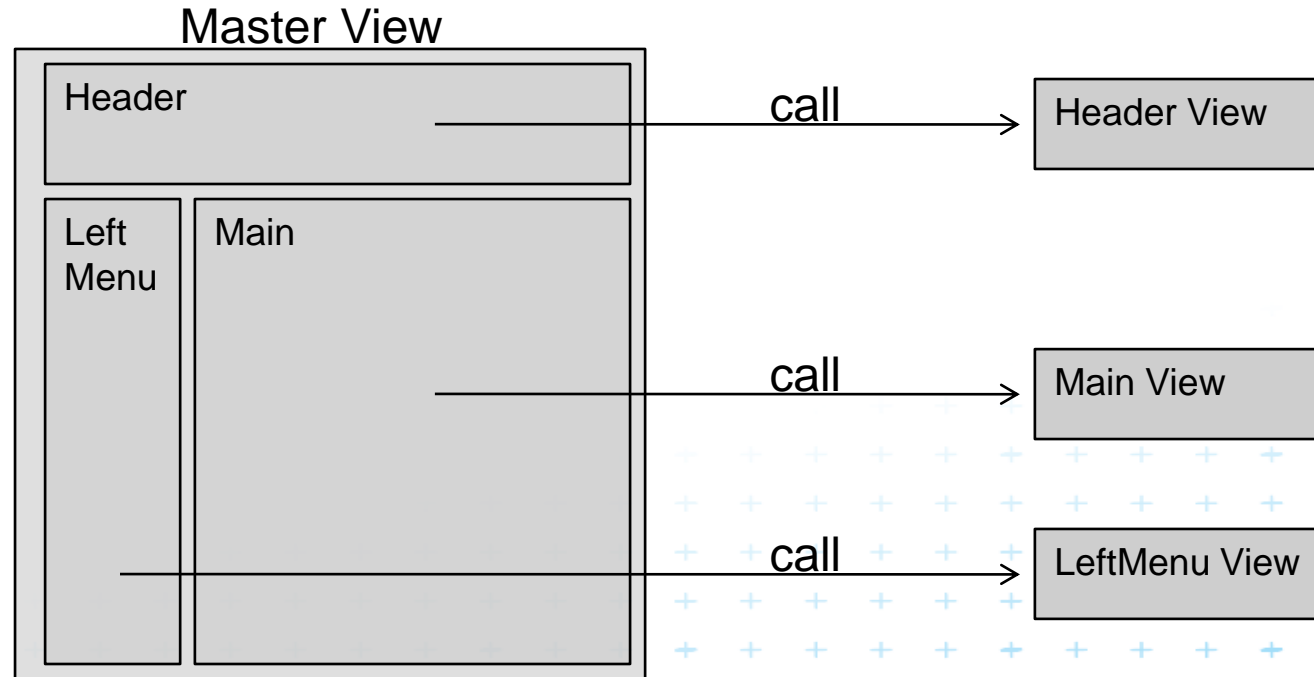
# View

```
<!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

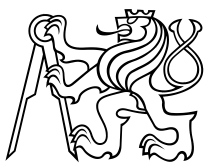


# PHP a MVC

- 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

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 MODEL  
...sežene data,  
pravděpodobně z DB

Toto je VIEW  
1. kompiluj šablonu  
2. interpretuj  
výsledný php kód

# Drobná ukázka Mustache

```
<?php
// toto je controller

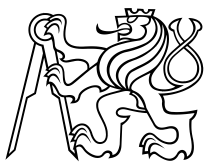
//příprava View datových struktur, inicializace
require_once('Mustache/Autoloader.php');
Mustache_Autoloader::register();
$mustache = new Mustache_Engine(array(
    'loader' => new Mustache_Loader_FileSystemLoader(dirname(__FILE__).'/templates')
));

$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);
```

```
<!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 automatický escaping, použijeme {{{}}}

# Iterace přes pole hodnot

```
//...stejně jako předchozí příklad

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

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

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

Sekce

# Modifikátory

```
//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);
```

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