IPL
escola superior
de tecnologia e gestão
instituto politécnico
de leiria

MVC

Model-View Controller in PHP

Marco Monteiro

Contributors

- Author(s):
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

Summary

1. The problem
2. MVC Pattern
3. MVC Simple implementation

4. *MVC Framework typical code*
5. *MVC simple framework*

1 – THE PROBLEM

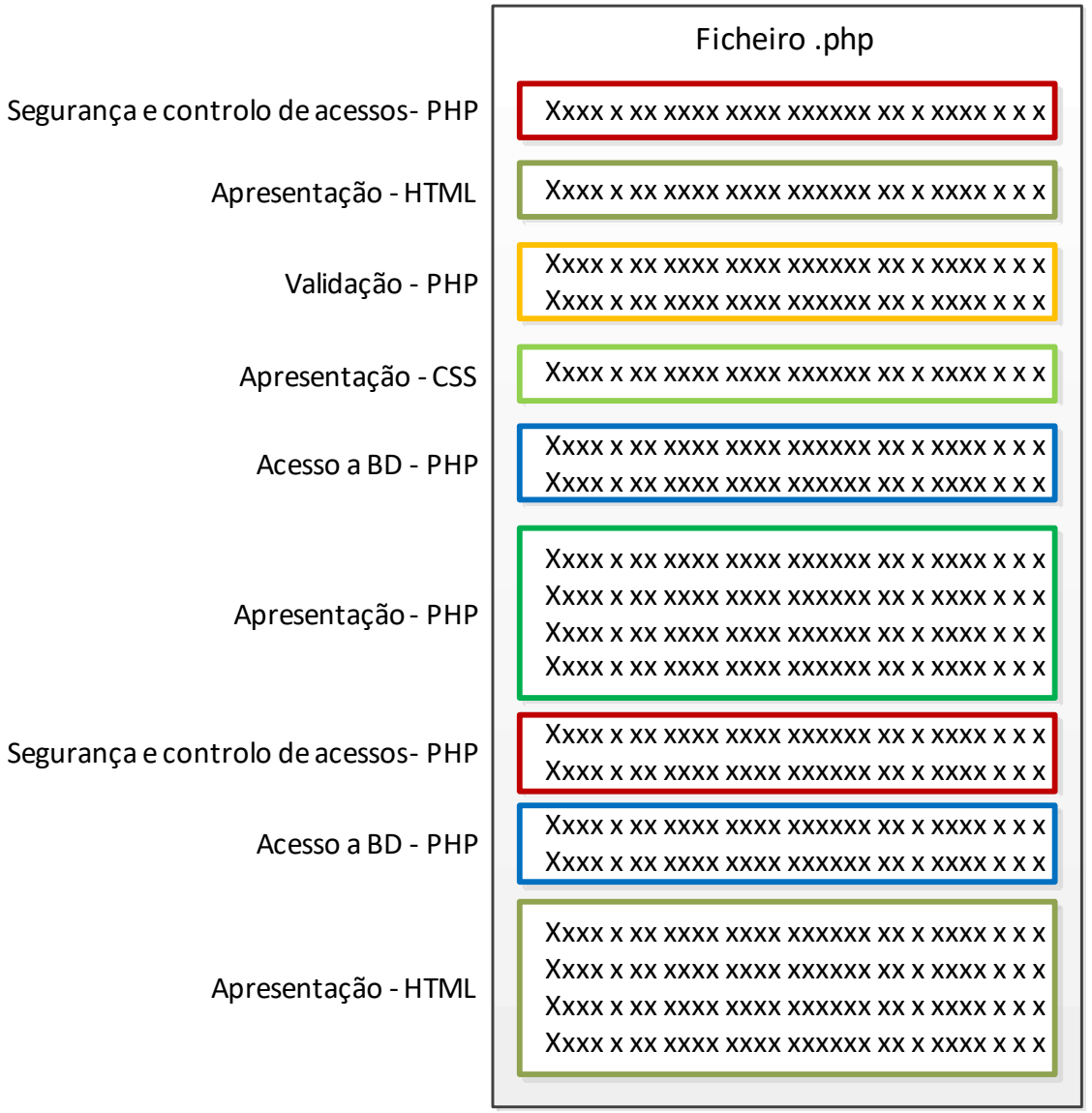
Spaghetti code

- The fact that PHP allows to mix HTML and PHP code on the same file (without enforcing any type of structure), can induce the production of spaghetti code

  - *Spaghetti code* - complex and disorganized code that mixes different aspects of the application. When one part is modified it may affect other, non-related, parts of the code

Spaghetti code

- .php pages can mix presentation code (html) with code related to database access, business operations, validation, authorization, etc…



Spaghetti code

- Complex and disorganized code
  - Same or similar features exist in several files – repeated code
  - Same or similar features may have different solutions on different files (inconsistent code)
  - Hard to know where everything is
  - Hard to understand what is affected when something is changed
- Hard to maintain
  - One feature may be fixed on file A but not on file B or C …
- Code is not easily reused
  - When each .php files have a lot of features, these features tend to be replicated on all files
    - When code is created with a lot of *"copy/past"*, it's a sign that code is not correctly reused
    - High probability of creating inconsistent code (same feature different solutions)

Spaghetti code

- Example ("EI" disciplines) (*complete code available on moodle*)

```
<?php function conn() { … //returns a database connection }   ?>
<!DOCTYPE html>
. . .
   <table>
       . . .
      <tbody>
<?php
  $bd = conn();
  $stmt = $bd->prepare('select name from disc where curso = ?');
  $stmt->execute(["EI"]);
  foreach ($stmt as $row) {
     echo "<tr> <td>" . $row['name'] . "</td> </tr>";
  }
?>
      </tbody>
  </table>
. . .
```

Spaghetti code

- Example ("JDM" disciplines) (*complete code available on moodle*)

```
<?php function conn() { … //returns a database connection }   ?>
<!DOCTYPE html>
. . .
   <table>
       . . .
      <tbody>
<?php
  $bd = conn();
  $stmt = $bd->prepare("select name from disc where curso = 'JDM' ");
  $stmt->execute();
  foreach ($stmt as $row) {
     echo "<tr> <td>" . $row['name'] . "</td> </tr>";
  }
?>
      </tbody>
  </table>
. . .
```

Spaghetti code

- Example (all disciplines) (*complete code available on moodle*)

```
<?php function conn() { … //returns a database connection }   ?>
<!DOCTYPE html>
. . .
   <table>
       . . .
       <tbody>
<?php
  $bd = conn();
  $stmt = $bd->prepare("select course, name from disc ");
  $stmt->execute();
  foreach ($stmt as $row) {
     echo "<tr> <td>" . $row['course'] . "</td>";
     echo "     <td>" . $row['name']   . "</td>    </tr>";
  }
?>
       </tbody>
  </table>
. . .
```

Spaghetti code

- Example analysis:
  - Page design (html) mixed with business related code
    - Code complex and disorganized
    - No separation of concerns - page designer must understand database and business-related code
  - Very similar pages with similar code - duplicated code
    - No code reusing
  - Although similar, there are some differences on the pages
    - Inconsistent code
  - If the database or page design changes, the code must be modified on 3 distinct files
    - Code hard to maintain – the probability of forgetting to modify one file or making an error on the modified code increases

Separated Model

- To minimize the problems of previous solution (spaghetti code), we can separate the implementation in 2 layers: Model and Page (View)
  - **Model** – centralizes all database and business-related code
  - **Page (view)** – page design. It uses the model to obtain data (it is dependent of the model)
- Previous example, but separated on 1 model file and 3 pages (views)

Separated Model

- Example - the **Model** (*complete code available on moodle*)

```
<?php function conn() { … //returns a database connection }   ?>

function getAllDisciplines()
{
    $bd = conn();
    $stmt = $bd->prepare('select course, name from disc');
    $stmt->execute();
    return $stmt->fetchAll();
}

function getDisciplinesCourse($course)
{
    $bd = conn();
    $stmt = $bd->prepare('select course, name from disc
                          where course = ?');
    $stmt->execute([$course]);
    return $stmt->fetchAll();
}
```

Separated Model

- Example – **View** ("EI" disciplines) (*complete code available on moodle*)

```
<?php require_once "model.php"; ?>
<!DOCTYPE html>
. . .
   <table>
       . . .
      <tbody>
<?php
  $disciplines = getDisciplinesCourse("EI");   // using the model
  foreach ($disciplines as $row) {
     echo "<tr> <td>" . $row['name'] . "</td> </tr>";
  }
?>
      </tbody>
  </table>
. . .
```

Separated Model

- Example – **View** ("JDM" disciplines) (*complete code available on moodle*)

```
<?php require_once "model.php"; ?>
<!DOCTYPE html>
. . .
   <table>
       . . .
      <tbody>
<?php
  $disciplinas = getAllDisciplines(); // using the model
  foreach ($disciplines as $row) {
     if ($row['course'] == 'JDM') {
       echo "<tr> <td>" . $row['name'] . "</td> </tr>";
     }
  }
?>
      </tbody>
  </table>
. . .
```

Although this page is similar to previous page, the model is incorrectly used

Separated Model

- Example – **View** (all disciplines) (*complete code available on moodle*)

```
<?php require_once "model.php"; ?>
<!DOCTYPE html>
. . .
   <table>
       . . .
      <tbody>
<?php
  $disciplinas = getAllDisciplines(); // using the model
  foreach ($disciplines as $row) {
     echo "<tr> <td>" . $row['course'] . "</td>";
     echo "     <td>" . $row['name']   . "</td>    </tr>";
  }
?>
      </tbody>
  </table>
. . .
```

Separated Model vs Spaghetti code

- Page design (html) uses almost no business-related code – it just uses the model
  - Code is simpler and more organized
  - Separation of concerns - page designer just uses the model
- Very similar pages with similar code
  - There is some code reuse - model code is reused
- Although similar, there are some differences on the pages
  - There is still some inconsistent code (compare EI disciplines with JDM disciplines)
- If the database design changes, code is modified on the model only
  - Code easier to maintain – for the model only

Separated Model - problems

- Separated model still has some problems
  - Separation of concerns - page designer still has to know how to use the model
  - Model code is reused, but the design (view) code is not reused
  - There can still be some inconsistent code (compare EI disciplines with JDM disciplines)
  - The page (view) code is still hard to maintain.
    - Changes on the database are easily updated (all modifications on one file), but changes on the design (HTML) still forces us to modify 3 distinct files (no code reuse on the page / view)

How MVC helps?

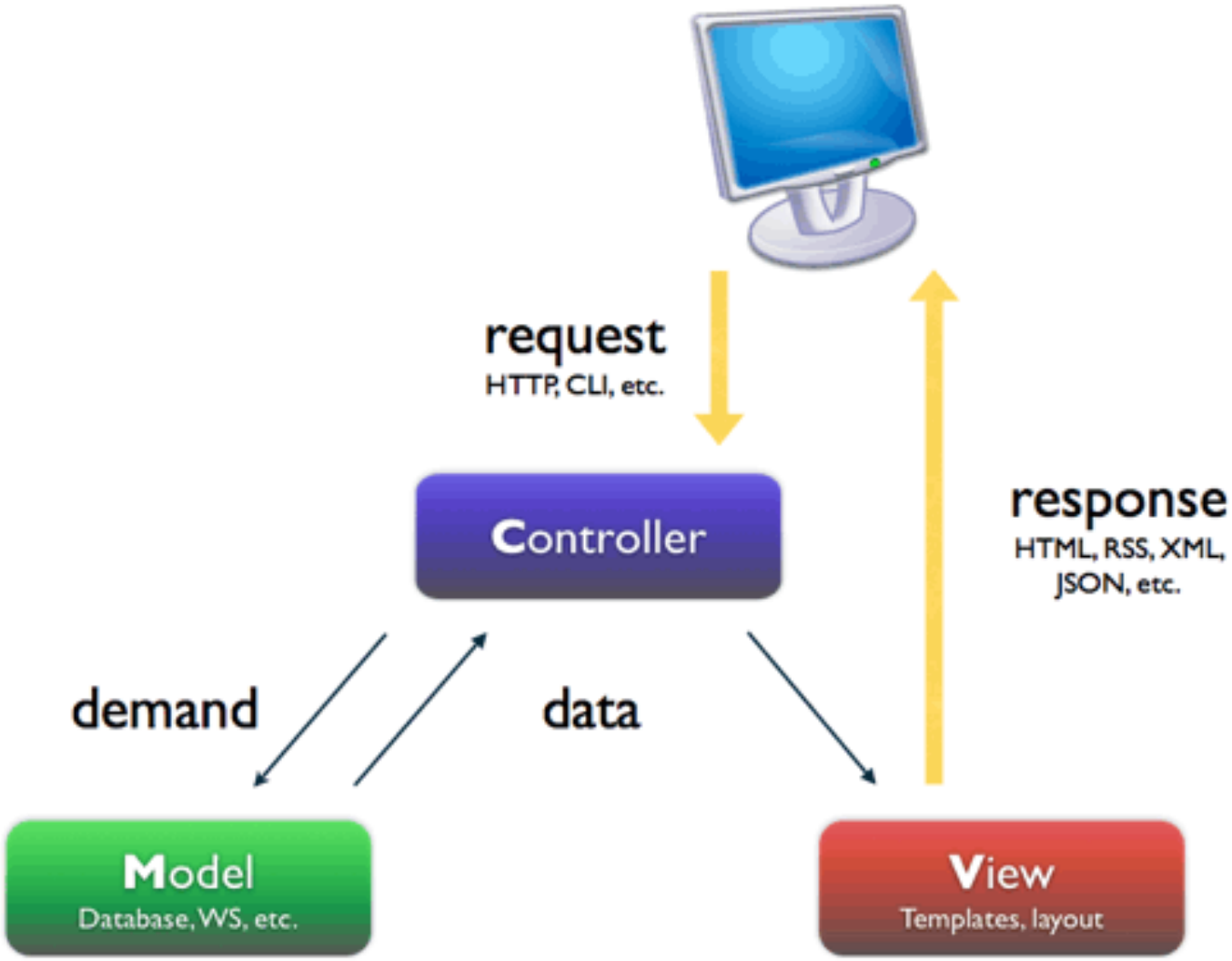
- MVC will force the **model** and **view** to be completely **independent** of other components
  - Separation of concerns
    - Page designer just handle design issues – view is independent of model (and of the controller)
    - Model just handles the database and business-related code – model is independent of the view (and of the controller)
  - Both the model and view code will be reused
  - Inconsistent code is reduced
    - Code repetition is highly reduced or non-existent
  - The model code and view code are easier to maintain
    - Model and view code is simpler and reused

2 – MVC PATTERN

MVC Pattern

- Model–view–controller (MVC) is an architectural pattern that isolates "domain logic" from user interface
  - Separation of concerns – user interface related code is separated from domain logic code
  - User Interface -> **view**
  - Domain logic -> **model**
  - MVC will force the **model** and **view** to be completely **independent** of other components
- Used in various platforms: Windows, MacOS, Web, etc.

MVC on the Web



Model

- The **model** is responsible for data management; it stores and retrieves data used by an application, usually from a database, and contains the application's domain logic
- It **does not depend** on the controller or view
- It can be reused, without modifications, by different controllers

View

- The **view** (presentation) is responsible for displaying data to the end user - generates content (usually HTML)
- Controller passes on data (variables) to the view, which can then use it to create dynamic content
- The view **does not depend** on the controller or model - it only depends on the data passed on to it
- It can be reused, without modifications (if the data structure is the same)
- Multiple views can exist for a single model for different purposes

Controller

- The **controller** handles the model and view layers to work together. The controller receives a request from the client (HTTP Request), invoke the model to perform the requested operations and send the data to the View
- Handles the HTTP Request.
  - When the server receives an HTTP request, it passes it to the controller (either directly or through a routing mechanism).

MVC on the Web

- **Summary** of typical responsibilities distribution:
- **Model**
  - Interact with the Database / business operations
- **View**
  - Consumes data (variables) passed on to it by the controller
  - Generates HTML (may include forms for user input)
- **Controller**
  - Access / validate data of the HTTP Request
    - GET or POST method
    - The **model** and the **view** should **never access** data from the **HTTP request**
  - Uses the model to read or store data
  - Creates the view, passing data (variables) to it

3 – MVC SIMPLE IMPLEMENTATION

A very simple implementation without any framework.

Also, it does not use any routing mechanism.

MVC – Simple implementation

- Example - the **Model** (*complete code available on moodle*)

  The same as on the "separated model" version

```
<?php function conn() { … //returns a database connection }   ?>

function getAllDisciplines()
{
    $bd = conn();
    $stmt = $bd->prepare('select course, name from disc');
    $stmt->execute();
    return $stmt->fetchAll();
}

function getDisciplinesCourse($course)
{
    $bd = conn();
    $stmt = $bd->prepare('select course, name from disc
                          where course = ?');
    $stmt->execute([$course]);
    return $stmt->fetchAll();
}
```

MVC – Simple implementation

- Example – **View** (*complete code available on moodle*)

```
. . .
 <h2><?= $title ?></h2>
. . .
   <table>
       . . .
       <tbody>
<?php
  foreach ($disciplines as $row) {
     echo "<tr> <td>" . $row['name'] . "</td> </tr>";
  }
?>
       </tbody>
  </table>
```

  - **Only one view** to show "All", "EI" or "JDM" disciplines
  - Data displayed by the view: **$title** and **$disciplines**
  - `$title` and `$disciplines` variables are not filled inside the view – the view only uses them

MVC – Simple implementation

- Example – **Controller** – ("all" disciplines)
  (*complete code available on moodle*)

```
<?php
  require_once "model.php";
  $title = "All Disciplines";
  $disciplines = getAllDisciplines();
  include "view_discipline.php";
```

- Responsible for filling the data (`$title` and `$disciplines` variables) and sending it to the view
  - *Note: when including the "view_disciplinas.php" file, all variables in the current file (the controller) will be available on the "view_disciplinas.php" file*
- Uses the model to get data: `$disciplines= getAllDisciplines()`
- It is **the controller** (not the view or the model) that **is invoked** by the client/browser. The controller handles the HTTP request.

MVC – Simple implementation

- Example – **Controller** – ("EI" disciplines)
  (*complete code available on moodle*)

```
<?php
  require_once "model.php";
  $title = "Disciplines of Computer Engineering";
  $disciplines = getDisciplinesCourse("EI");
  include "view_discipline.php";
```

- Code is the same as previous controller, only the variables (`$title` and `$disciplines`) are filled with different data
- Data is different, but the data structure is the same
- Uses the same model and the same view

MVC – Simple implementation

- Example – **Controller** – ("JDM" disciplines)
  (*complete code available on moodle*)

```
<?php
  require_once "model.php";
  $title = "Disciplines of Digital Games and Multimedia";
  $disciplines = getDisciplinesCourse("JDM");
  include "view_discipline.php";
```

- Code is the same as previous controller, only the variables (`$title` and `$disciplines`) are filled with different data
- Data is different, but the data structure is the same
- Uses the same model and the same view

MVC vs Separated Model

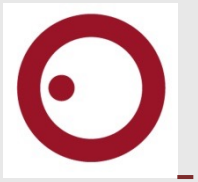
- The model is the same as the one on "Separated Model"
  - The model code is centralized and easily reused
- The view (page design - html) has no business-related code – it just depends on the data passed on to it
- The view is reused
  - Only one view file is used to represent 3 different versions of the page (3 different data sets)
- Also, the same data can be represented differently by different views
  - It is the view that "decides" how to represent the data

MVC vs Separated Model

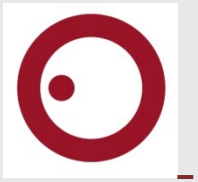
- It is the controller (not the view) that "decides" which data is presented – the view only decides how to present the data
  - 1 view is reused when resulting content has the same design but different data (with the same data structure)
  - The same data can be represented differently by creating different views
- It is the controller (not the view or the model) that handles the HTTP request.
  - The controller **is invoked** by the client/browser. The model and view are not directly accessible to the client/browser
  - Typical MVC frameworks implement **routing** mechanisms that map "arriving" HTTP Requests to controller methods.

PHP Frameworks

- Some PHP frameworks with an MVC based architecture:
  - Laravel (http://www.laravel.com/)
  - Zend Framework (http://framework.zend.com/)
  - Symfony (http://www.symfony-project.org/)
  - Yii Framework (http://www.yiiframework.com/)
  - FuelPHP (https://fuelphp.com/)
  - CakePHP (http://cakephp.org/)
- Typically, they support **routing**, **templates**, **ORM** (Object Relational Mapping) and other advanced features

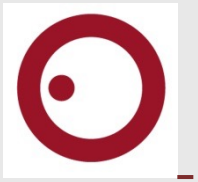


4 – MVC FRAMEWORK TYPICAL CODE



Model

- Model is usually implemented with an ORM technology
  - ORM – **O**bject **R**elational **M**apping
  - An ORM class represents a relation (table) on the DB. Example: class Product represents the products table
  - A model is an instance of the ORM class, and it represents a row of the relation (table)
  - The attributes (properties) of the model represent the columns (fields) of the row



Model


```
class Disciplina extends ORMClass {
    . . .
}
```

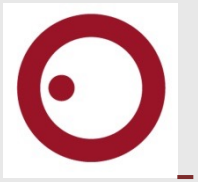
- When the model uses an ORM technology, it inherits a set of methods and properties from the ORMClass.

  ORMClass class name depends on the technology

  - It is the ORM technology that connects to the database and translate class methods to SQL commands

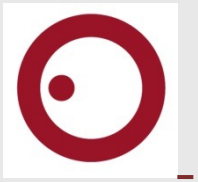
- Example of Model usage:

```
$disciplinas          = Disciplina::all();
$firstDisciplineName = $disciplinas[0]->name;
```



View

- View defines the content (HTML)
  - Mixes HTML with PHP
  - All PHP code should be related to content (HTML) generation
  - View code should depend only on data (variables) that is passed on to the view by the controller
    - These variables are filled "outside" the view (by the controller)
    - View developer only needs to use these variables – he does not need to get the data or execute any operation.
    - View does not call any function or use any class, except if the function or class is used to generate/manipulate the content

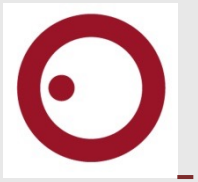


View

- Example that generates a table with disciplines

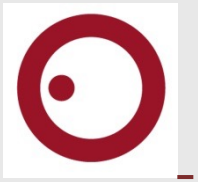
```
<!DOCTYPE html>
. . .
   <table>
       . . .
       <tbody>
<?php
  foreach ($disciplinas as $row) {
     echo "<tr> <td>" . $row['curso'] . "</td>";
     echo "     <td>" . $row['nome']  . "</td>    </tr>";
  }
?>
       </tbody>
  </table>
. . .
```

$disciplinas is created and filled by the controller.
The view only uses the variable



Routing

- MVC frameworks usually include a routing mechanism, that maps arriving HTTP requests to controller methods
- Each project includes a set of **routes**, that define the rules to map arriving HTTP requests to controller methods.
  Each route usually includes:
  - HTTP request method (GET; POST; DELETE; PUT; etc.)
  - URL pattern rule
  - Controller method
- When an HTTP request "arrives" to the server, the routing mechanism decides (analyzing the method and the URL of the HTTP request) which route will handle the request. Then, it will invoke the method of the controller associated to that route
  - If no route is selected, the routing mechanism sends a "404 Not Found" HTTP Response to the client



Routing

- Example of routes (using Laravel syntax):

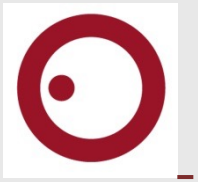
```
// ROUTES – Laravel Syntax – 3 routes:

Route::get("disciplines", [DisciplineController::class, "index"]);

Route::get("disciplines/create",
                          [DisciplineController::class, "create"]);

Route::post("disciplines", [DisciplineController::class, "store"]);
```

HTTP Method

URL Pattern

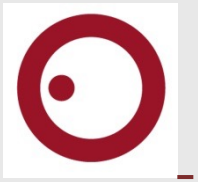
Controller Method

- Example (3rd route): when an HTTP request with the method **POST** and URL (relative URL) "**disciplines**" arrives, it will execute the method "**store**" of the class **DisciplineController**



Controller

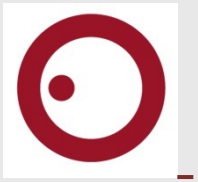
- Controller accesses the HTTP Request and handles the model and view
  - Gets data from the model or stores data through the model
  - Accesses / validates the HTTP request to read the user input/parameters – only the controller has access to the information of the HTTP request
  - Creates a view and passes data (variables) to it
    - Alternatively, it may create different type of content. For instance, it is common to create an HTTP Response with redirect header.



Controller

- Example to show a page with disciplines
  - Gets an array with disciplines from the model (model = Disciplina)
  - Passes that array (with the name 'disciplinas') to the **view**
  - The view (also named disciplinas) will have the variable $disciplinas pre-filled.

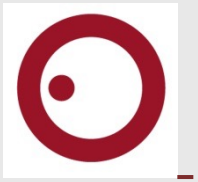
```
class DisciplinasController
{
    public function index()
    {
        $disc = Disciplina::all();
        view('disciplinas', ['disciplinas' => $disc]);
    }
}
```



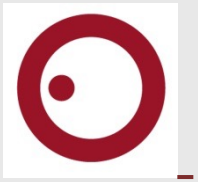
Controller

- Example to save data
  - Data received on the HTTP Request (POST method) is validated
    - If data is not valid, the execution is interrupted by the function validate(). Otherwise, that function returns an array with valid data
  - Validated data ($newDisc) is saved on the DB through the model
  - After operation (that saves data) is correctly executed, the controller returns a **redirect** header to the client.
  - The returned header "tells" the browser to redirect to another page (browser will "call" the page to show all "disciplinas")

```
class DisciplinasController  {
    public function store()  {
        $newDisc = validate($_POST);
        Disciplina::create($newDisc);
        return header('location: /disciplinas');
    }
}
```



5 – MVC SIMPLE FRAMEWORK

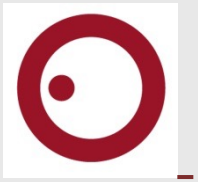


MVC Framework

- MVC based frameworks, like Laravel and others, provide a set of classes and other resources on a predefined structure, that simplifies and enforces the implementation of a Web Application using the MVC pattern

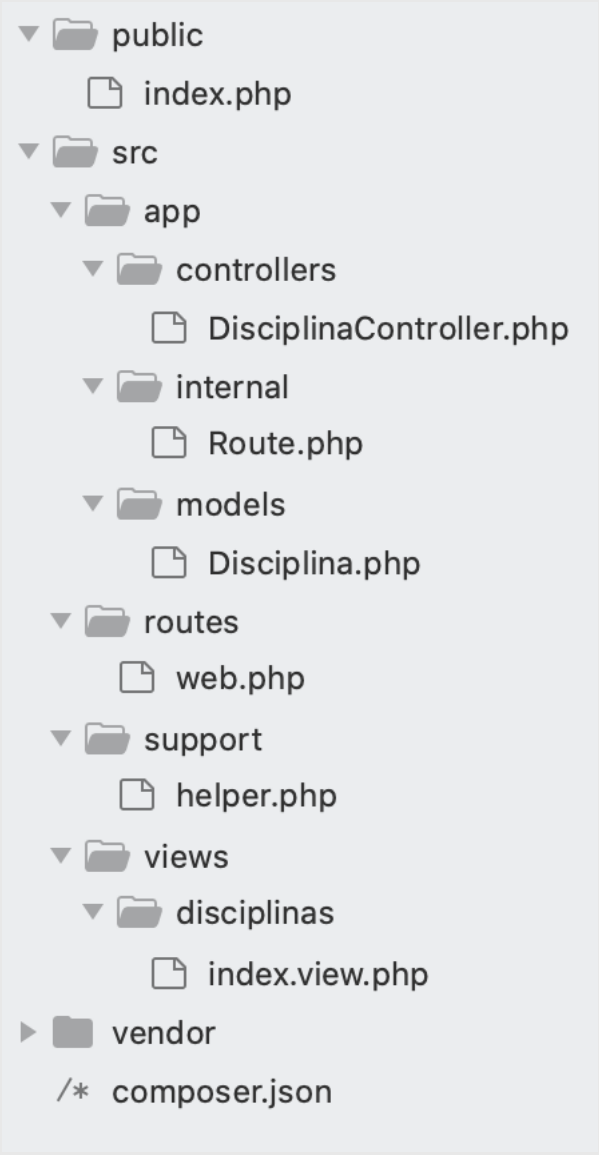
- Here, we will present our **own** and very **simple MVC framework**

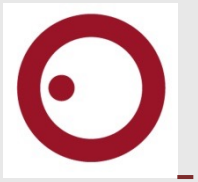
- Follow and implement the **tutorial of worksheet4** that creates a version of this MVC framework and a fully functional example that uses it.



MVC Framework

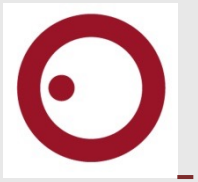
- Folder/File structure:
  - 1 Model :
    `src/app/models/Disciplina.php`
  - 1 Controller (with 3 methods):
    `src/app/controllers/DisciplinaController.php`
  - 1 View
    `src/views/disciplinas/index.view.php`
  - *"Entry point"*
    `public/index.php`
    What is really "called" by the client.
    Invokes the routes mechanism
  - 1 support file: `src/support/helper.php`
    Provides a function (view) to generate the view content
  - Route internal implementation
    `src/app/internal/Route.php`
  - Routes registration (mapping)
    `src/routes/web.php`





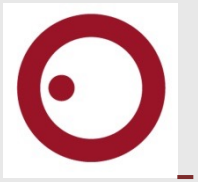
Model


```
<?php

namespace models;

class Disciplina  {
  private static function conn() {…}

  public static function all() {
    $bd = Disciplina::conn();
    $stmt = $bd->prepare('select curso, nome from disciplinas');
    $stmt->execute();
    return $stmt->fetchAll();
  }
  public static function whereCurso($curso) {
    $bd = Disciplina::conn();
    $stmt = $bd->prepare('select curso, nome from disciplinas '.
                         'where curso = ?');
    $stmt->execute([$curso]);
    return $stmt->fetchAll();
  }
}
```



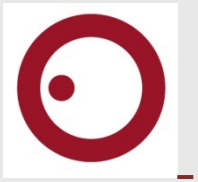
Controller (method index)


```
<?php

namespace controllers;

use \models\Disciplina;

class DisciplinaController
{
    public function index()
    {
        $disciplinas_to_show = Disciplina::all();
        $titulo_da_pagina = "Todas as Disciplinas";
        $mostraCurso = true;

        view('disciplinas.index', [
                        'disciplinas' => $disciplinas_to_show,
                        'titulo' =>      $titulo_da_pagina,
                        'mostraCurso' => $mostraCurso
            ]);
    }
    . . .
}
```



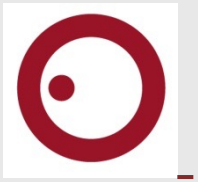
Controller (method ei)


```
<?php

namespace controllers;

use \models\Disciplina;

class DisciplinaController
{
    . . .
    public function ei()
    {
        $disciplinas_to_show = Disciplina::whereCurso("EI");
        $titulo_da_pagina = "Disciplinas de Engenharia Informática";
        $mostraCurso = false;
        view('disciplinas.index', [
            'disciplinas' => $disciplinas_to_show,
            'titulo' => $titulo_da_pagina,
            'mostraCurso' => $mostraCurso
         ]);
    }
}
```



Controller (method jdm)

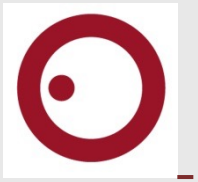

```
<?php

namespace controllers;

use \models\Disciplina;

class DisciplinaController
{
    . . .
 public function jdm()
    {
        $disciplinas_to_show = Disciplina::whereCurso("JDM");
        $titulo_da_pagina = "Disciplinas de Jogos Digitais e ".
                            "Multimédia";
        $mostraCurso = false;
        view('disciplinas.index', [
            'disciplinas' => $disciplinas_to_show,
            'titulo' => $titulo_da_pagina,
            'mostraCurso' => $mostraCurso
        ]);
    }
}
```



view function

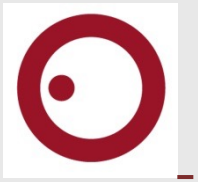
File: src/support/helper.php

```
<?php

function view($viewName, $vars)
{
    // Declares a local variable for each pair inside $vars
    foreach ($vars as $name => $value) {
        $$name = $value;
    }

    include 'views/'.str_replace('.', '/', $viewName).'.view.php';
}
```



View


```
<!DOCTYPE html>
. . .
<h2><?= $titulo ?></h2>
<table>
  <thead>
     <tr>
        <?= $mostraCurso ? "<th>Curso</th>" : "" ?>
        <th>Nome</th>
     </tr>
  </thead>
  <tbody>
     <?php
         foreach ($disciplinas as $row) {
            echo "<tr>";
            echo $mostraCurso ? "<td>".$row['curso']."</td>" : "";
            echo "<td>" . $row['nome'] . "</td>";
            echo "</tr>";
         }
     ?>
  </tbody>
</table>
. . .
```

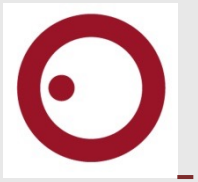


Entry Point

File: public/index.php

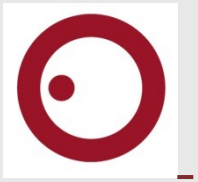
```
<?php
require __DIR__ . '/../vendor/autoload.php';

use \internal\Route;

include __DIR__ . '/../src/routes/web.php';

Route::invoke();
```

- File that is called by the client – the only one
- Includes the routes registration (file src/routes/web.php) that associates URL patterns to controller methods to execute
- Uses the routing mechanism (class \internal|Route) to invoke the proper controller method for the incoming HTTP request



Routing mechanism

- MVC frameworks usually implement a routing mechanism that, when an incoming HTTP request "arrives" to the server, decides which method will handle it and invokes that method (a method of a controller)
- Usually, it supports some type of mapping (e.g. using routes registration) that associates an URL pattern to a method of a controller.
- If the incoming HTTP request URL (and HTTP method) is recognized (registered), the associated controller is instantiated, and the associated method is executed. Otherwise, the routing mechanism send a "404 Not Found" HTTP response to the client



Route Registration

File: src/routes/web.php

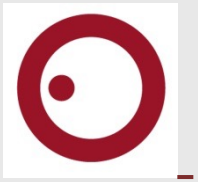
```
<?php

use \internal\Route;

Route::get("/disciplinas",     "DisciplinaController@index");
Route::get("/disciplinas/ei",  "DisciplinaController@ei");
Route::get("/disciplinas/jdm", "DisciplinaController@jdm");
```

URL patterns

ControllerClassName@methodName

Method

- Maps URL patterns to controller methods
  - URL "/disciplinas" -> index method of the DisciplinaController
  - URL "/disciplinas/ei" -> ei method of the DisciplinaController
  - URL "/disciplinas/jdm" -> jdm method of the DisciplinaController
- Example
  - When the incoming HTTP request has the method GET with the URL "/disciplinas", then the routing mechanism of the framework will execute the method index of the DisciplinaController



Routes

File: src/app/internal/Route.php

```
<?php
namespace internal;
class Route
{
    public static function get($url, $controllerMethod) {…}
    public static function post($url, $controllerMethod)  {…}
    public static function invoke()  {…}
}
```

- ***Check demos*** *full code to view the internal code*
- Methods "get" and "post" register the routes (used on src/routes/web.php file)
- Method "invoke()" automatically executes the method of the controller associated to the incoming HTTP Request
- URL format: hostname/index.php?routeurl
  - For pretty URLs (e.g. hostname/routeurl), check the **worksheet4** tutorial