escola superior
de tecnologia e gestão
instituto politécnico
de leiria

PHP - MVC

Model-View Controller in PHP

Vitor Carreira & Marco Monteiro

Contributors

- Author(s):
  - Vitor Carreira (vitor.carreira@ipleiria.pt)
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

- Contributor(s):
  - Fernando Silva(Fernando.silva@ipleiria.pt)

Summary

1. MVC Pattern
2. MVC Example
3. References

1 – MVC PATTERN

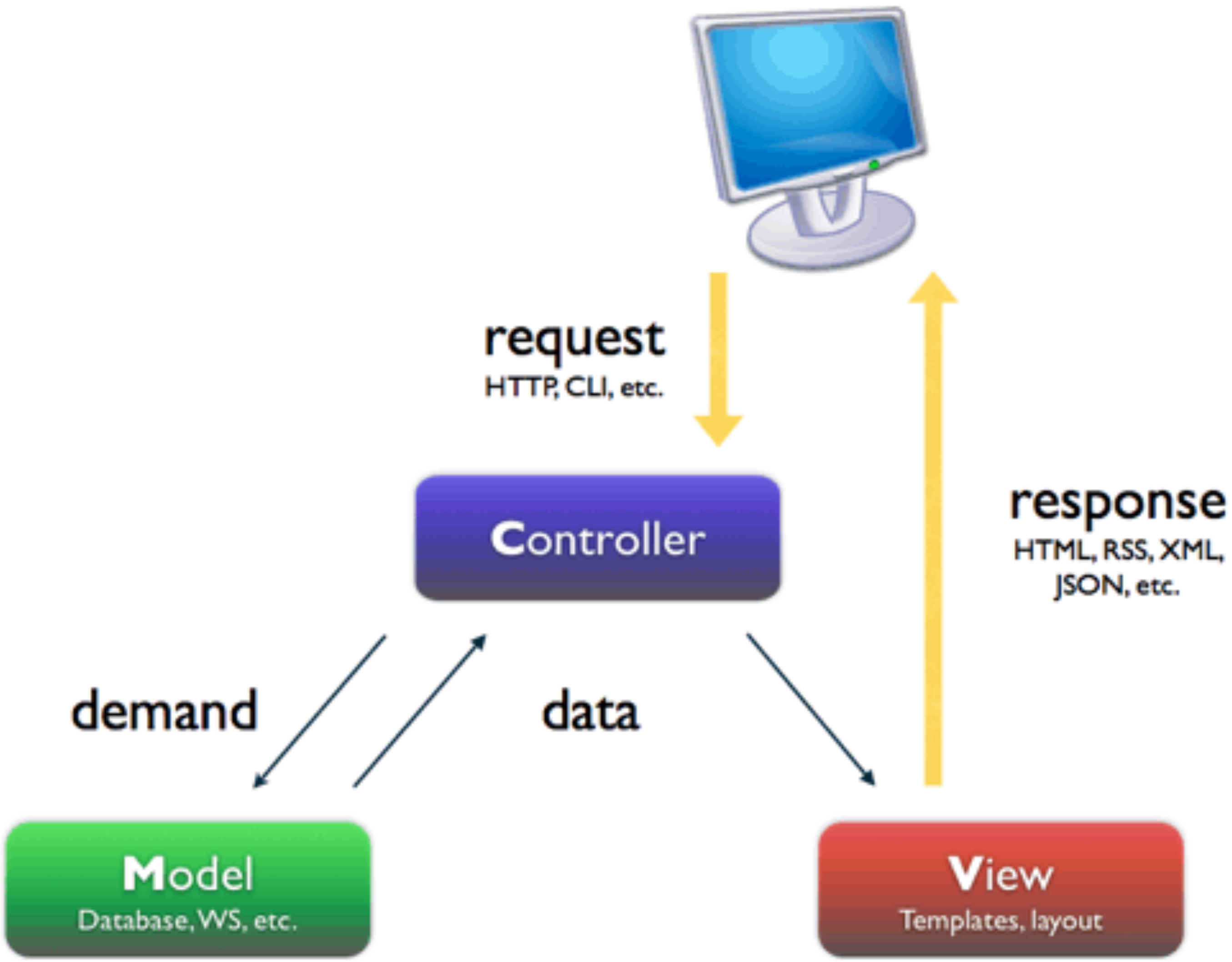
MVC Pattern

- 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
- Used in various types of platforms: Windows, MacOS, Web, etc.

MVC on the Web

- Why?
  - Typical unstructured PHP code mixes presentation code (HTML), with all other types of code (data access; validation; security; authentication and authorizations) etc..
- The MVC pattern clearly separates different types of code (different concerns) in distinct files, each with its own responsibility

MVC on the Web



Model

- **The model** is responsible for data managing; it stores and retrieves entities used by an application, usually from a database, and contains the application’s domain logic
- It **<u>does not depend</u>** of the controller or view
- It can be reused, without modifications, by different controllers and views

View

- **The view** (presentation) is responsible for displaying the data provided by the model in a specific format (html, xml, etc).
- 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, 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
- View
  - Generates HTML (may include forms for user input)
- Controller
  1. Access data of the HTTP Request
     - GET or POST method ($_GET; $_POST)
  2. Uses the model to read or store data
  3. Creates a view, passing it the data obtained from the model

2 – MVC EXAMPLE

Practical Example

- Create a web page that shows a list of articles stored on a array
  - Note that the data could also be on a database
- Solution 1 - No MVC
- Solution 2 - MVC architecture

Solution 1: No MVC

```
<?php
    $articles = ['Article Title' => 'Article content …', … ];
    $pagetitle = "List of Articles (No MVC)";
?>
<!DOCTYPE html>
<html lang="en">
  <head> . . .  <title><?= $pagetitle ?></title> . . .  </head>
  <body>
    <table>
      <thead><tr><th>Title</th><th>Content</th> </tr></thead>
      <tbody>
         <?php foreach ($articles as $title => $content) : ?>
           <tr><td><?= htmlspecialchars($title) ?></td>
               <td><?= htmlspecialchars($content) ?></td></tr>
         <?php endforeach; ?>
      </tbody>
    </table>
  </body>
</html>
```

Solution 1: No MVC

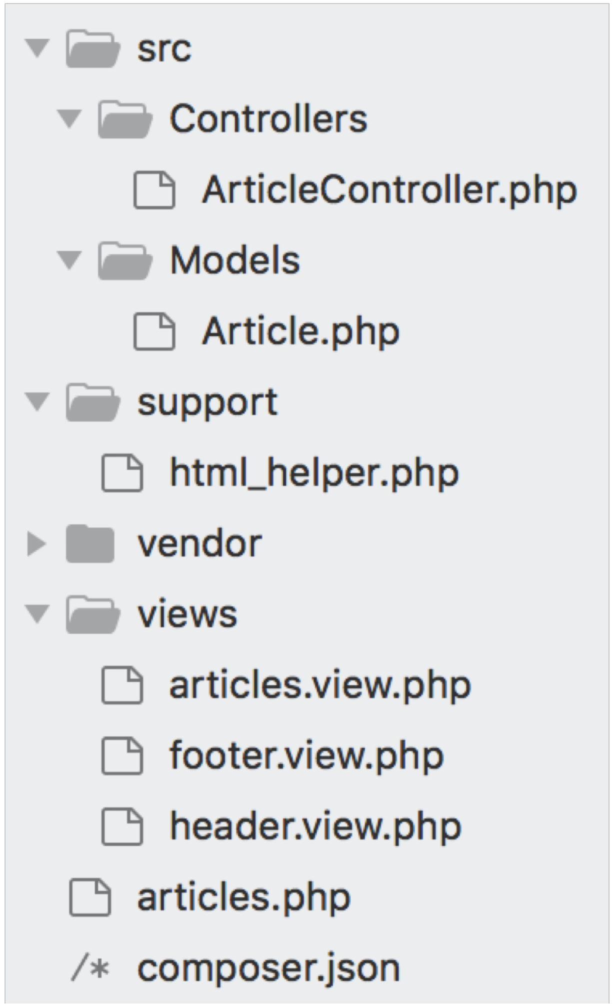
- Issues:
  - Hard to maintain (data access code will be scattered across the page)
  - If more than one page displays articles, all the code must be duplicated and kept consistent

Solution 2: MVC Architecture

- Solution 2 - Transforming to MVC
  - **Model** - A class with the code that accesses the data (array). This class can be reused when required.
  - **View** - Loops the article list and creates the HTML page.
  - **Controller** - separates the code that fetches the data (model) from the code that writes the HTML (view).

Solution 2: MVC Architecture

- Folder/File structure:
  - 1 Model :
    `src/Models/Article.php`
  - 1 Controller:
    `src/Controllers/ArticleController.php`
  - 1 view : partitioned in 3 files:
    - Content view: `views/articles.view.php`
    - Template views: `views/footer.view.php`
      `views/header.view.php`
  - "Entry point": `articles.php`
    - The "URL" address called by the client
  - 1 support file: `support/html_helper.php`
    - Provides a function (render_view) to generate (render) the view content



```
src
  Controllers
    ArticleController.php
  Models
    Article.php
support
  html_helper.php
vendor
views
  articles.view.php
  footer.view.php
  header.view.php
articles.php
composer.json
```

Model

File: src/Models/Article.php

```php
<?php

namespace Models;

class Article
{
    public $title;
    public $content;

    public function __construct($title = null, $content = null)
    {
        $this->title = $title;
        $this->content = $content;
    }

    public static function all()
    {
        return ['Article Title' => 'Article content …', … ];
    }
}
```

Controller

File: src/Controllers/ArticleController.php

```
<?php
namespace Controllers;

use Models\Article;

class ArticleController
{
    public function getArticles()
    {
    // Reads data from the Model
        $articles = Article::all();
        $pagetitle = "List of Articles";

    // render_view will render the view articles
    // passing data ($articles & $pagetitle) to it
        render_view('articles', compact('articles', 'pagetitle'));
    }
}
```

View

Files: views/header.view.php ; views/articles.view.php; views/footer.view.php

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title><?= $pagetitle ?></title>
    </head>
    <body>
        <h1><?= $pagetitle ?></h1>
```

```
        <table>
            <thead> <tr><th>Title</th><th>Content</th></tr> </thead>
            <tbody>
            <?php foreach ($articles as $article) : ?>
                <tr>
                <td><?= htmlspecialchars($article->title) ?></td>
                <td><?= htmlspecialchars($article->content) ?></td>
                </tr>
            <?php endforeach; ?>
            </tbody>
        </table>
```

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

Entry Point

File: articles.php

```
<?php
require_once "vendor/autoload.php";

use Controllers\ArticleController;

$controller = new ArticleController;
$controller->getArticles();
```

- File (“URL”) that will be called by the client
  - what the HTTP Request refers to.
- Creates the controller and invokes the respective controller method: getArticles()
- On a typical MVC based framework, this will be replaced by a routing mechanism

render_view function

File: support/html_helper.php

```
<?php

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

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

- Function that will render a view.
- Accepts the view name and a set (array) of variables (data)
- Creates the complete HTML document, including header and footer templates

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 also support routing, templates, ORM (Object Relational Mapping) and other advanced features

3 – REFERENCES

References

- Official (PHP)
  - http://www.php.net/
  - http://php.net/manual/en/
  - http://php.net/manual/pt_BR/

- PSR - PHP Standard Recommendations
  - https://www.php-fig.org
  - https://www.php-fig.org/psr/psr-4/

- PHP and MySQL Web Development (4th Edition)
  - Luke Welling and Laura Thomson, Addison-Wesley 2009
- PHP Objects, Patterns, and Practice (2nd Edition)
  - Matt Zandstra, APress 2008
- Object Oriented PHP Concepts Techniques and Code
  - Peter Lavin, No Starch Press 2006