C1- Code & Go

PHP_Intro_Rush_MVC

Intro MVC

Introduction to MVC

CODING ACADEMY
BY EPITECH

Intro MVC

Administrative Details

- The project is to be done in teams of two.
- Sources must be handed in with BLIH.
- Location of files to hand in: PHP_Intro_Rush_MVC

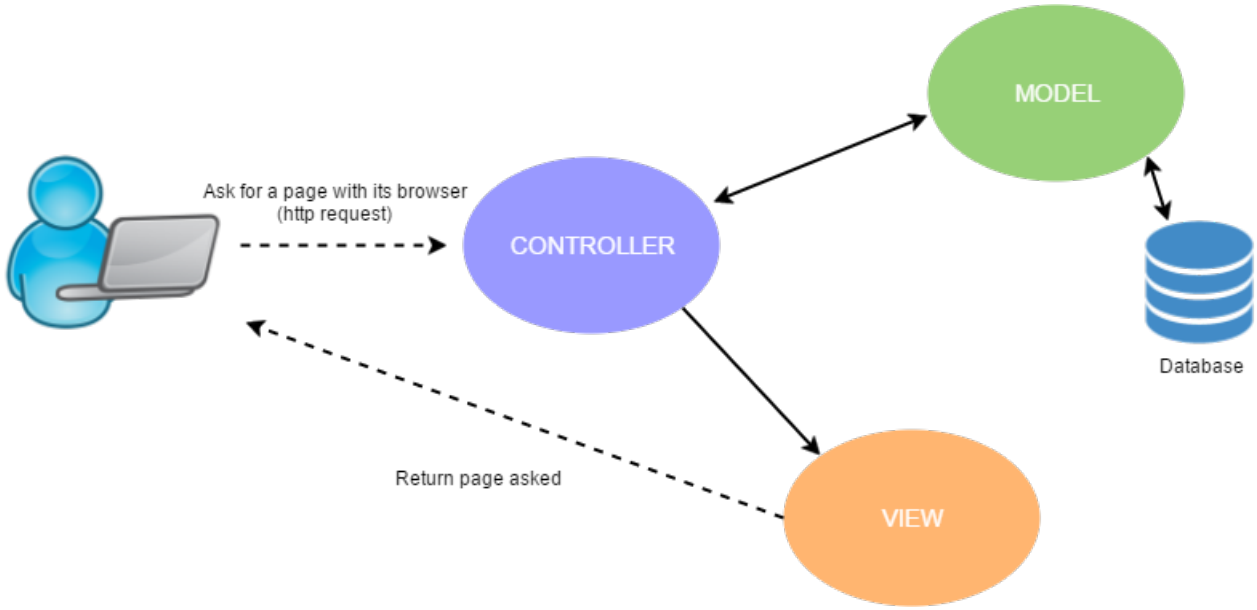
Introduction

You already know how to do a simple website with PHP and JQuery. Now, it's time to raise the speed by using a conceptual manner of coding: the **MVC design pattern** (Model – View – Controller)! This day is a bit special, you'll have more theory. Today's goal is to make you understand the rudiments of MVC.

MVC is **THE** design pattern used nowadays in production. By using MVC, your **code will be clearer** for you and it also allows to **work with other people easier**.

MVC's goal is to **separate your code's logic in 3 parts:**

- **Model:** this part **manage your website's data.** Its goal is to get "raw" data from database(s), and give it back to controller. For example, you'll find SQL requests here
- **View:** this is the **display part.** It gets variables and displays it, (nearly) without doing any calculation. It's only HTML code, but sometimes you'll use PHP loops or conditions, for example if you want to display posts of a user
- **Controller:** this part is the brain of your application. **It contains the logic, and take decisions.** It's the intermediary between the Model and the View: the controller asks for data from Model, analyze it, take decisions and give data to display to the View. Controller is only in PHP. For example, it also determines if a visitor has the right to see a page or not (right accesses management)



For this introduction, you'll have to make a simple web application: the famous **todo app.**

You'll implement these features, also known as **CRUD** (Create – Read – Update – Delete):

- **Tasks list:**
  - Get all tasks from database (DB) and display its title and description (optional)
- **Create task:**
  - Below tasks list, display a form with 2 input fields:
    * Title
    * Description (optional)
    * Submit button to create the task
  - Save this in input field and create it in DB
  - Refresh page after saving, to display tasks list again
- **Read task:**
  - Clicking on task goes on this task page
  - Displaying its title AND its description
- **Update task:**
  - 2 ways to update a task in DB:
    * On tasks list, task title input field has to be editable
    * On task page, task title and task description has to be editable
- **Delete task:**
  - 2 ways to delete a task in DB:
    * On tasks list, tasks can be deleted individually
    * On task page, task can be deleted

You'll handle errors for all of these.

Intro MVC

Restrictions

This project must, of course, respect the MVC design pattern.
Your work must follow the structure below:

```
PHP_Intro_Rush_MVC/
    Models/
        Db.php
        TodoList/
            Task.php
    Controllers/
        TodoList/
            tasksController.php
            taskController.php
    Views/
        TodoList/
            tasks.php
            task.php
            style.css
    Vendors/
        jquery.min.js
    index.php
    .htaccess
```

- **Db model** must contain only function to manage database requests exclusively with the help of **PDO**.
- Your **db object must be a global** object type to avoid doing connection to db again and again. In production, you'd use a "Singleton" design pattern, so try to use it. But if not, just use a global.
- Your libraries must be located in the Vendors directory.
- The only page called by the user is index.php.
- All your project MUST be Object Oriented.

Authorized external libraries:

- JQuery

Non-exhaustive and minimal MYSQL structure:

The tasks must have an id, a title, a description, a creation date, and a modification date.
For the rest, it's up to you, but your structure must be coherent.
Of course, your code must be clear and understandable. Don't forget to write comments.

Step 0

Database creation

First of all, you must create a database named **todo_php**.
This database contains one table, named **tasks**.

This table must respect this structure:

Fields	Type
id	UNSIGNED BIGINT, AUTO_INCREMENT
title	VARCHAR
description	TEXT, can be NULL
creation_date	DATETIME
edition_date	DATETIME

Step 1

Models

You'll start by creating the models.

For this step, you have to create **Db.php** and **Task.php** files in Models/ folder.
Remember that your models must be thought in a object oriented manner.

Db.php:

Db.php contains only function to manage database requests exclusively with the help of **PDO**.

Task.php:
First of all, **include Db.php**, and use a **global** db object to connect to DB.
Do all the necessary functions to **get / post / put / delete** tasks data to DB:

Functions	Description
`function get_tasks()`	Get all tasks and return them in an associative array.
`function get_task($id)`	Get a specific task by its id, and return it.
`function post_task($title, $description = null)`	Create a new task with a title and an optional description.
`function put_task($id, $title = null, $description = null)`	Update a task by its id, with optional title and description.
`function delete_task($id)`	Delete a task by its id.

For GET functions, task(s) must be returned fetched (fetchAll()).

Step 2

Controller

For this step, you have to create **tasksController.php** in Controllers/ folder.

tasksController.php:

First of all, you have to **include Task.php** model file.

TaskController must be a class implementing all the methods that should enable you to check the inputs (user to model(db)) and render outputs (model(db) to view).
Then, save the return result of get_tasks() in a $tasks array.
Now, you can **loop through the associative array** returned and secure it like this:

```
...
foreach($tasks as $key => $task)
{
    $tasks[$key]['title'] = htmlspecialchars($task['title']);
    $tasks[$key]['description'] = nl2br(htmlspecialchars($task['description']));
}
...
```

Please have the curiosity to see what are htmlspecialchars() and nl2br().

You'll note that we do operations on $tasks keys instead of $task directly. It's because $task is only a copy of $tasks array made in foreach. $task exists only in foreach, and will be deleted after. To avoid XSS vulnerabilities, you have to act directly on the array used in display, so $tasks.

One of the controller's goal is to secure display. If there was some **operations to do on right accesses**, it would be the good time to do it. Also, you have to know that sometimes, to secure the display, operations are done in the view (rarely), and some frameworks use to do it with a transitional layer.

An interesting thing here is that **get_tasks() can be reused** in other controllers.

Of course, here was only get_tasks() function, **you'll have to do the same for other functions**, like **check if the input fields are empty or not, secure its field...** Here is an example of securing input fields:

```
…
if(isset($_POST["task_title"])) // check empty field(s) in form
{
  $title = secure_input($_POST["task_title"]); // secure input form
  if($_POST["description"]) // because description isn't mandatory
  {
    $description = secure_input($_POST["task_description"]);
  }

  // post the task (for example), and test if it worked (depending on how you
handled errors in your model…)
  if(post_task($title, $description) == -1)
  {
    … // error handling
  }
}

function secure_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
…
```

Again, see what are trim(), stripslashes().

When you are over with this controller, at the end of the file, you have to **include tasks.php** view file.

Step 3

View

For this step, you have to create **tasks.php** in Views/ folder.

In this file, you just have to **display tasks, and forms needed**. No logic, no security, everything has been prepared before.

Using a foreach, you can display all tasks from the $tasks associative array created in controller.
Create forms needed for posting a new task, and put buttons for deleting and updating tasks.

And that's it for the view!

Good thing is that you can give this file to a web designer who doesn't necessary know PHP to work on page lay-out.

Step 4

Router and URL rewriting

Final step!

You'll finish by the router, and the URL rewriting.

For now, URL are accessible by the user this way:
http://localhost/Controllers/TodoList/tasksController.php, or
http://localhost/Controllers/TodoList/tasksController.php?id=5
Erk! That's so ugly. Not intuitive at all, and very bad for security.
An example of what we want instead: http://localhost/todolist, or http://localhost/todolist/5
Let's resolve this.
Because you're using Apache now, you'll need to create a .htaccess in your root directory.

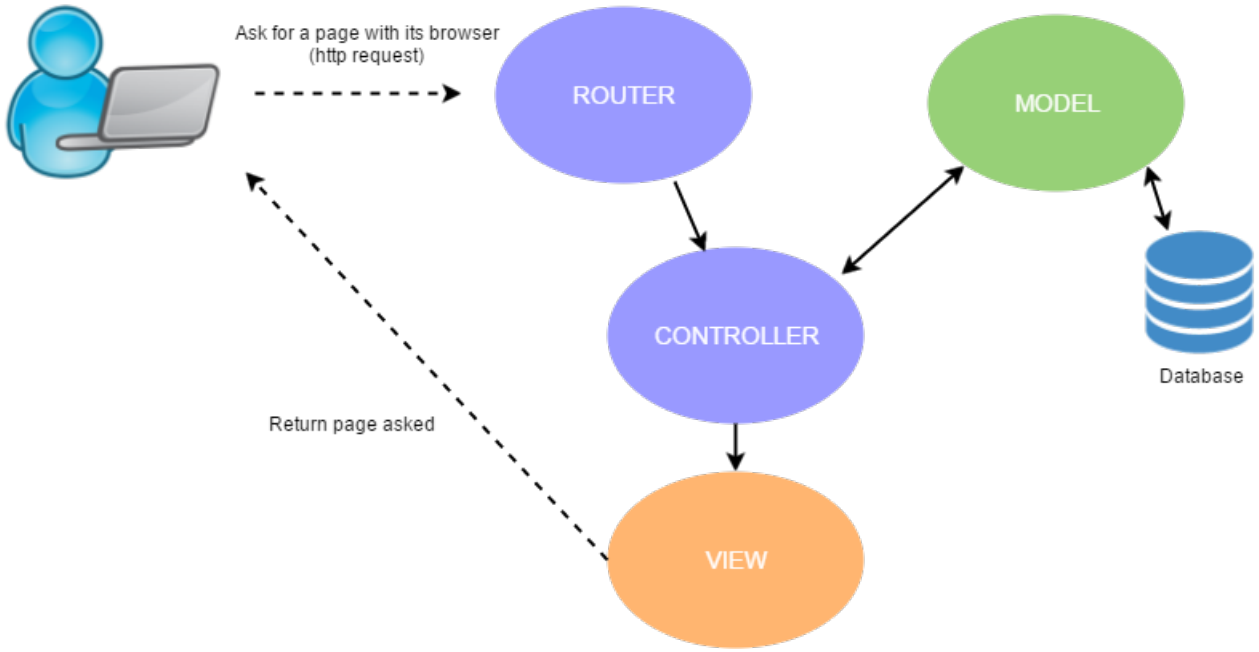
In your .htaccess file, write this line:

FallbackResource /index.php

This line will **redirect ALL the URL entered by a user on your server to your index.php file.**

Before Apache 2.2.16, you had to put lot more things in your .htaccess, it's not necessary yet right now, this line does the work. If you want to use Nginx later, you'll have to create a config file instead.

Now that all our files redirect to our index.php, let's parse all the URL given!
Guess what, you'll use index.php as a router.
A router is a system which will parse the request done by the user and determine which route it corresponds to. In other terms, using a router:
If a user calls http://yourwebsite/todolist, http://yourwebsite/todolist/55, or even http://yourwebsite/account/186479, the router will lead your user to the good controller, which include the good view.



As you see, now the router is your entry point.
index.php:

```php
$path = ltrim($_SERVER['REQUEST_URI'], '/');  // Trims leading slash(es)
$params = explode('/', $path);                // Splits path on slashes
if($params[0] === NULL)                       // No path params means home page
{
    require_once('./Controllers/TodoList/tasksControllers.php');
} else
    array_shift($params);                     // Pops off '/' from params

$rules = array(
    '{todolist} => './Controllers/TodoList/tasksControllers.php',
    '{todolist/(?P<id>\d+)/?}' => './Controllers/TodoList/taskControllers.php'
);
$found = false;
foreach($rules as $pattern => $target) {
    if(preg_match($pattern, $path, $params)) {
        require_once($target);
        $found = true;
        break;
    }
}
if (!$found) {
    header('HTTP/1.1 404 Not Found');
    require_once('./Controllers/TodoList/404Controller.php');
}
```

You can still access to $params array in your controller, so you can use it as you wish!
Don't' forget that now, you can do the one task view and controller (available at http://localhost/todolist/5 for example).