



## Code & Go

Rush MVC

*Teaching managers*

[pedago-coding@epitech.eu](mailto:pedago-coding@epitech.eu)

# Table of contents

---

Administrative details .....	3
Introduction.....	3
Structure of your website.....	4
Phase 1: Initialization .....	5
Phase 2: Users .....	8
Phase 3: Login and Registration .....	9
Phase 4: Administration of users.....	10
Phase 5: Articles .....	11
Phase 6: Comments.....	12
Phase 7: Administration of articles .....	12
Phase 8: Categories and tags.....	13

# 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\_Rush\_MVC**

## Introduction

---

Expect this rush to be an introduction to how frameworks like CakePHP, Symfony, and etc. work.  
Now you'll see more **advanced notions like dispatcher**.

Here is the situation: **a client wants you to create a blog with these specifications**

- Your site must respect the **MVC** design pattern (Model – View – Controller).
- It has to respect **OOP** (Object Oriented Programming) paradigm.
- The models must contain only functions to manage database requests exclusively with the help of **PDO**.
- Your views will be managed by **TWIG**.
- The file **db.php** must contain all connection information to the database(s).
- The file **core.php** initializes all components (configurations) you need.
- Your libraries must be located in the **Vendors** directory.
- The only page called by the user is **index.php**.
- **Webroot folder** is the only directory that can be accessed by the user. It's then up to the **index.php** file to make the **necessary includes**.
- **session.php** contains a class Session that will implement methods read, write, delete, destroy and load. These methods are static.
- **Session class** overlays PHP sessions, which means that a session variable is accessible by: `$_SESSION['key'];` or `Session::read('key');` (key represents a path with a dot separator).
  - Example: `Session::read('Auth.User.id') == $_SESSION['Auth']['User']['id'];`
- The **router.php** file retrieves information from the URL and parses it. The **dispatcher.php** file executes it.

# Structure of your website

---

Your work must follow the structure below:

## Models/

Article.php

...

## Controllers/

ArticlesController.php

UsersController.php

ArticlesController.php

...

## Views/

### Layouts/

theme1.tpl

...

### Articles/

action.tpl

...

## Config/

configuration.php

core.php

db.php

## Src/

session.php

router.php

## Vendors/

...

## Webroot/

.htaccess

index.php

### js/

...

### Css/

...

### Img/

...

dispatcher.php

.htaccess

Code & Go

Rush MVC

---

# Phase 1: Initialization

---

Htaccess files provide some security to your website. You should search how they accomplish their role.

**First of all, you have to change your apache configuration files:**

- Go on /etc/apache2/apache2.conf
- Modify the line AllowOverride None to AllowOverride All in the Directory /home/your\_login/Rendu.
- Then, enable the rewrite module.

**Then, you have to create two htaccess files with the following emplacements:**

- current\_work\_folder/.htaccess with this text inside:

```
RewriteEngine on
RewriteBase /PHP_Rush_MVC
RewriteRule ^$ /webroot/ [L]
RewriteRule (.*) /webroot/$1 [L]
```

- current\_work\_folder/webroot/.htaccess with this text inside:

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php?$1 [QSA,L]
```

Depending on systems, sometimes you have to add or delete slashes, or even add path from previous folder.

You must use TWIG to do all of you views. So, basically, what is it? It's a frontend template, to help us to design our HTML pages with the PHP content.

**The structure of the site follows the MVC pattern. This pattern is very important because it's the most used pattern nowadays. So, you have to understand that structure to be able to understand and recreate it.**

Before to begin the MVC pattern, you have to design all of your website. **You have to create the db.php, core.php, configuration.php, route.php and dispatcher.php and use them in your conception.**

- The core.php initializes all of your Controllers, Database, Dispatcher classes, and your router which is a map;
- The db.php is where you have your Database class which is performing all operations with the database;
- The configuration.php contains all of your configurations concerning the project;
- route.php contains your route map which is composed by your routes and the methods and controllers where they are pointing;
- dispatcher.php contains the class Dispatcher which will redirect datas received by the client side to the corresponding method with the router map informations.

The Controllers and Database classes are singletons, search what it means.

**They are the core of your project and you must design well this part, if you don't want to have hazardous results. So, take a big look to understand why these classes and map are very important and do it in your conception (again).**

**Now, we have to create the ApplicationController. It is simply the controller by who the other controllers will inherit. And you know, these Controllers will all need some basic methods:**

- public function loadModel(\$model);
  - o Loads the Database class so that it can be accessed in the controller by using \$this->\$model.
- public function render(\$file = null) ;
  - o Method calling the view file. In the case where the parameter is null, we load the view file associated to the method. So without parameter, we get Views/<nom\_du\_controller>/<nom\_de\_l'action>.html.twig
- public function beforeRender();
  - o Method called right before the call to the view. This is to do some actions before the rendering.
- protected function redirect(\$param);
  - o Redirects a user from a method of the router to another method of the router.
  - o \$param is an array with the URL of the route.
  - o You have to use Dispatcher class.

#### **Authorized external libraries**

- JQuery
- TWIG

**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.**

## Phase 2: Users

---

You have to create user entities which will contain:

- Username;
- Hashed password;
- Email;
- Group;
- Status of the user (if he/she is banished);
- His creation date;
- His last modification date.

There is a hierarchy between users. Indeed, they can be administrators, writers or normal users. Each user group inherits the rights of lesser groups (Example: ADMINISTRATOR can perform all actions of WRITER who can then perform all actions of USER). You decide what rights to each group, but it has to remain consistent.



## Phase 3: Login and Registration

---

Obviously, you have to create a connection / registration management. Your users have to register to connect.

**You have to handle their errors and build them like that:**

- Registration:
  - Username: Between 3 and 10;
  - Email: Handle this part with regex;
  - Password and password confirmation:
    - Between 8 and 20 characters;
    - Have to be the same;
  - The message has to be “Invalid username”, or “Invalid password” or “Invalid email”.
  
- Connection:
  - Invalid username;
  - Invalid password;
  - The message has to be “Invalid username and/or password.”.
  
- Logout
  
- Account deletion:
  - You can delete your account, but not the others’;
  - The message has to be “The account has been deleted” and you will be redirected on the index page.

## Phase 4: Administration of users

---

In this part, you have to create the administration interface for user management. It's different of the normal user interface, because there, an administrator will manage a lot of things.

**But for now, we only will create few of them:**

- This interface is accessible only by an administrator;
  - o A writer and a user can't access to this interface.
- The activation / deactivation of an account: Sometimes, an administrator has to kick somebody of the site, and deactivating his account is the best way to do this
- The creation of a user: it's very important for a lot of reasons. For example, when you have to create a bunch of users to test a feature or to create other administrators;
- The modification of a user, including his promotion to an administrator or a writer;
- The destruction of a user.

# Phase 5: Articles

---

You will now create some articles because we want to publish a lot of them on our website.

**So, there are their characteristics:**

- Only the users who are connected and have writer rights can write articles. The writer's interface is accessible by writers and administrators.;
- Articles are listed by date. It means that you will have the most recent articles on the top of your page.
- Users can see all articles:
  - o Their content;
  - o Their title;
  - o By who it was created;
  - o When it was created;
  - o When was the last modification.
- Users can select articles by their:
  - o Title.
  - o Creation date.
- We can see the comments of the article;
- They can be identified by tags : look at the associative table;
- They belong to a category, but it's none of your business at now.

## Phase 6: Comments

---

Our articles should have comments, because it's a good and simple way to communicate with the others about their articles.

**However, there are some rules for them.**

- Only the users who are connected can write comments on articles;
- We can see when the comment was published and by who;
- In an article, we can see the comments of this article;
- An article can contain many comments;
- A comment belongs to only one article.

## Phase 7: Administration of articles

---

Writer of an article and administrators manage the administration side of articles and comments.

**Here are the managements to have for them in the administrator's interface:**

- An article can be created, modified and deleted;
- A comment can be created (by everyone) and deleted;
- When the article is deleted, all his relative comments are deleted too;

Code & Go

Rush MVC

## Phase 8: Categories and tags

---

For this part, you have to create categories and tags for articles. They are very important because they regroup articles with some criteria. So, what's the difference between them?

It is that the category regroups articles by their principal domain / description, and a tag is like a #hashtag on Twitter, and is defining only one domain / description of the article. For example, an apple and a banana can be placed in the "fruits" category while they can have the following hashtags: #fruit #healthy #red #yellow, etc.

**So, here are the features to manage in the writer's interface:**

- An administrator can create, modify, and delete a category. The modification is the fact of adding or deleting articles in a category, and modifying his name;
- Tags can be created and deleted by writers and administrators;
- Tags can be associated to an article by an administrator and the writer of that article.
- You have to implement the research of articles which can be done by:
  - o Author;
  - o Title;
  - o Date;
  - o Category;
  - o Tag.

# Bonuses

---

- Good designed views <3
- Global searching
- Autocomplete
- ORM creation
- ...

Code & Go

Rush MVC