

## Content

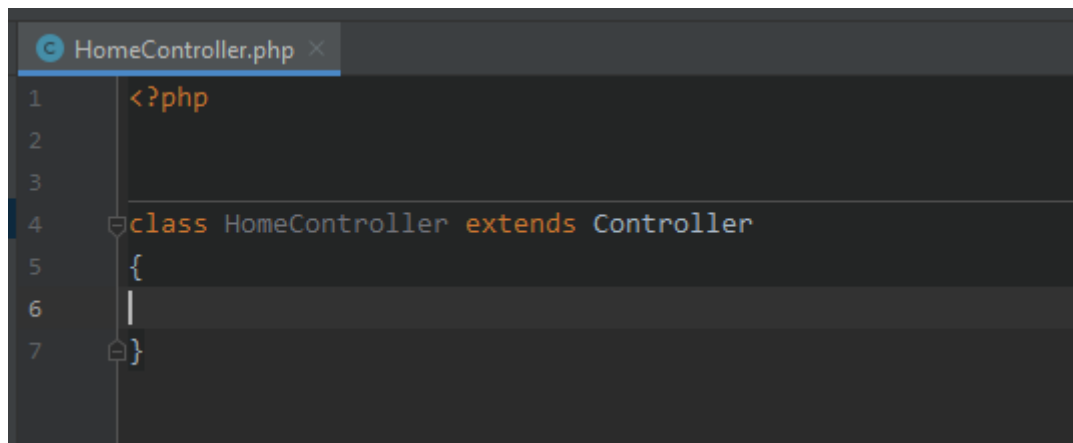
1 Website structure.....	1
2 Controllers.....	1
2.1 Create controller .....	1
2.2 Controller Actions.....	2
2.3 Return a View or Redirect .....	2
3 Models.....	2
3.1 Create a model .....	2
3.2 Set table and define Table-Fields .....	3
4 Views .....	3
4.1 Creating a View.....	3
4.2 Page data / View data .....	3
5 Configuration.....	4
6 Frontend .....	4
7 Query Builder / ORM .....	4
7.1 Database Connection .....	4
7.2 Save a record/entity .....	4
7.3 Update record/entity .....	5
7.4 Delete a record/entity .....	5
7.5 Selecting records from one table .....	5
7.6 Select records with Join.....	6
8 Closing words .....	6

## 1 Website structure

## 2 Controllers

### 2.1 Create controller

For creating a controller you need to create a class in the folder app/Controllers. It should be named like [WORD]Controller. For example HomeController. The created controller needs to extend the class Controller.



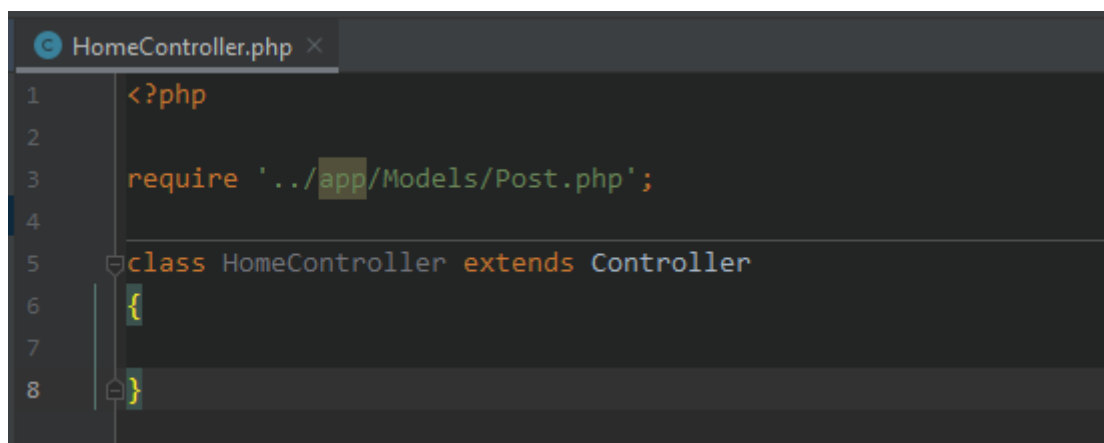
```

1  <?php
2
3
4  class HomeController extends Controller
5  {
6
7

```

## 2.2 Controller Actions

Controller methods (actions) should be public and take no parameters. For using models in the controller you need to require the needed Model before the codeblock starts where the class gets defined.



```

1  <?php
2
3  require '../app/Models/Post.php';
4
5  class HomeController extends Controller
6  {
7
8

```

A controller action can actually return everything but actually should return a View or Redirect.

## 2.3 Return a View or Redirect

When a controller action is called and the function returns a View the method View(\$data, \$layout = null, \$view = null) will search a file in the directory app/Controllername/ActionName.php and include it if it exists. If the parameter layout isnt null it will load the header with navbar and the footer. The View-File in app/ControllerName/ActionName.php needs to be called exactly like the called action if its on a Linux OS or an Casesensitive OS. For example if the called URL is :

« <http://mywebsite.com/Home/Index> » it will call the method Index from the Controller (class)

HomeController. And it will search the View in app/Views/Home/Index.php. The parameter \$data needs to be called \$data under every circumstances. If you want to return a redirect you need to call the method Helper ::Redirect(\$controller, \$action, \$id = null). This could be for example :

Helper ::Redirect('Dashboard', 'Index').

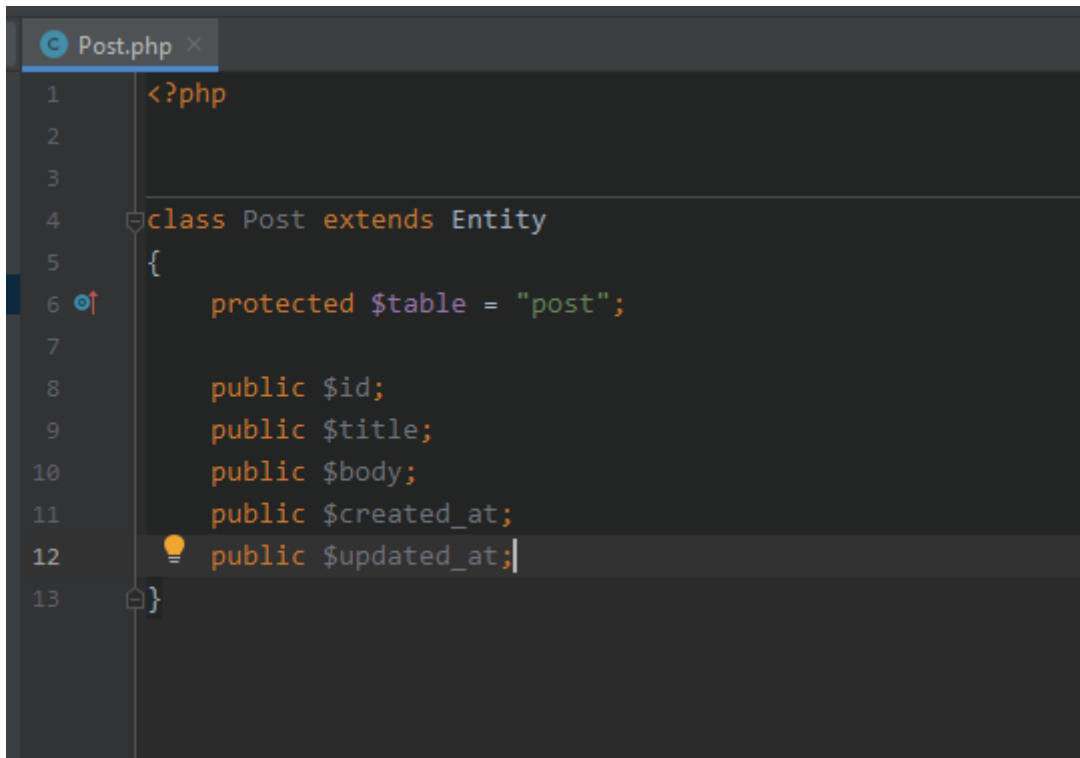
# 3 Models

## 3.1 Create a model

For creating a model you need to create a class in the directory app/Models. Its of benefits if the Model is called like the Database-Table it relates to.

### 3.2 Set table and define Table-Fields

First of all the model needs to extend the class Entity which is the QueryBuilder/ORM for the MVC.



```
1 <?php
2
3
4 class Post extends Entity
5 {
6     protected $table = "post";
7
8     public $id;
9     public $title;
10    public $body;
11    public $created_at;
12    public $updated_at;
13 }
```

The table-fields must be public properties because the ORM is gonna get the fields by public properties. It must be all correct or the queries won't work.

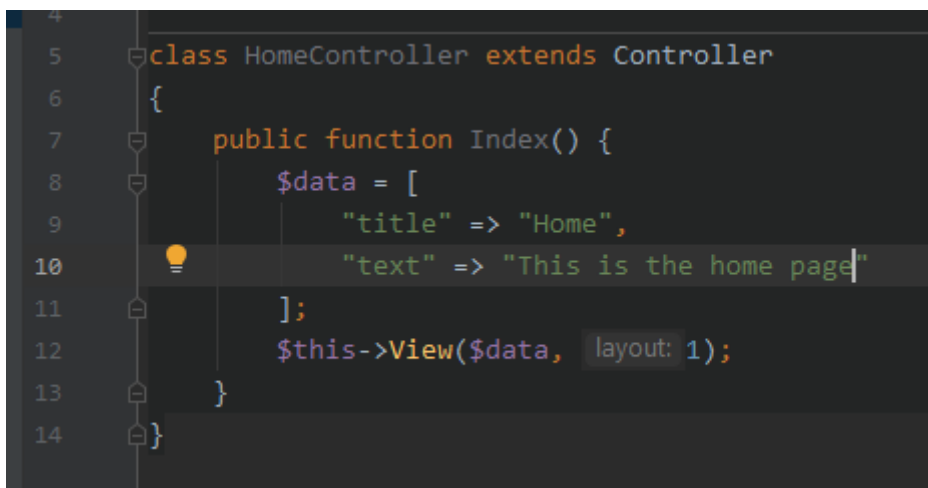
## 4 Views

### 4.1 Creating a View

To create an empty view you need to create a file with php extension in the directory app/Views/ControllerName/ViewName.php. For example Index.php in app/Views/Home. The variable to access data passed by the controller is \$data which is an array which contains the data.

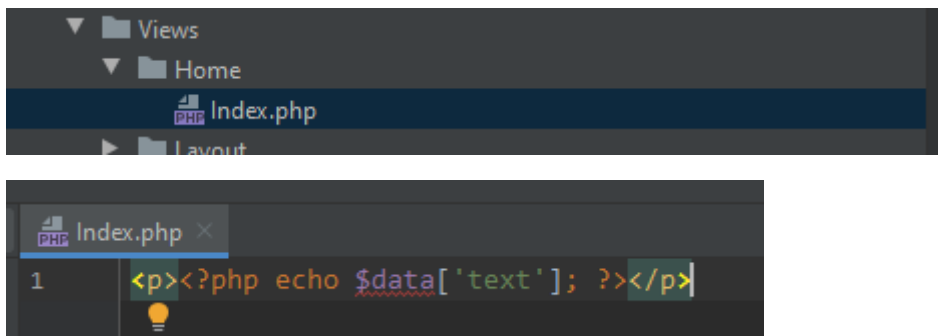
### 4.2 Page data / View data

For setting the page title the array \$data needs a key «title» with the value of the title. For example \$data['title'] = 'Home'.

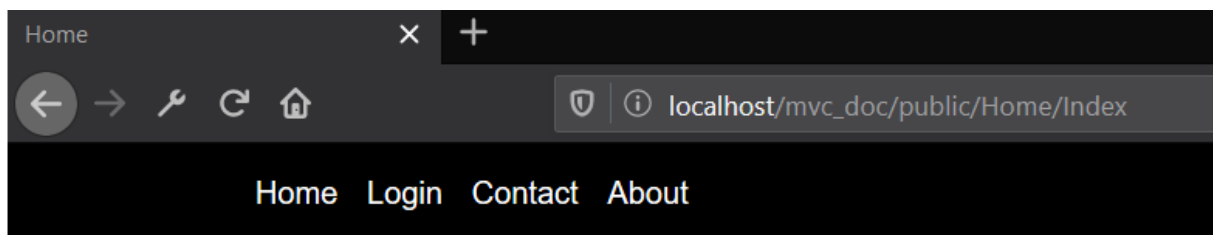


```
5 class HomeController extends Controller
6 {
7     public function Index() {
8         $data = [
9             "title" => "Home",
10            "text" => "This is the home page"
11        ];
12        $this->View($data, layout: 1);
13    }
14 }
```

This will output a view with the default navbar and footer and the given data.



The View with following code and passed data will output this:



This is the home  
page

## 5 Configuration

To use the framework you just gotta download it from github.

## 6 Frontend

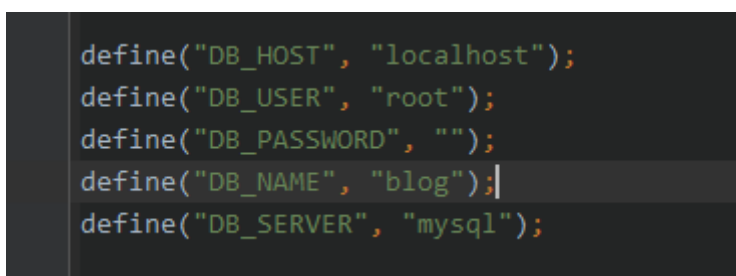
For the frontend you can use whatever library or ur own files. The CSS files are in /public/styles/ and the JS files are in /public/js/. The default CSS for the app is styles.css in /public/styles/.

## 7 Query Builder / ORM

The Query Builder is in the class entity.

### 7.1 Database Connection

The Database connection is by default in Databasehandler. The Data for connection will be received from constants defined in /config/config.php.



### 7.2 Save a record/entity

Here is an example how to save an record/entity with the created model from before.

```

public function CreatePost() {
    $post = new Post();
    $_POST['title'] = "Not empty";
    if(!empty($_POST['title'])) {
        $post->post_id = null;
        $post->title = "Title of post";
        $post->body = "Body of post";
        $post->created_at = date( format: "Y-m-d H:i:s");
        $post->updated_at = null;
        $post->save();
        Helper::Redirect( controller: "Home", action: "Index");
    }
}

```

### 7.3 Update record/entity

Here is an example how to update an entity with the model created before.

```

public function UpdatePost() {
    $post = new Post();
    $_POST['title'] = "Not empty";
    if(!empty($_POST['title'])) {
        $post->post_id = 27;
        $post->title = "Title of post#1";
        $post->body = "Body of post#1";
        $post->Update(["title" => $post->title, "body" => $post->body])->Where(["post_id" => $post->post_id]);
        Helper::Redirect( controller: "Home", action: "Index");
    }
}

```

The method needs to be called with Where because you rarely update all records and Update doesn't execute a query.

### 7.4 Delete a record/entity

Here is an example how to delete an entity/record with the model created before.

```

public function DeletePost() {
    $post = new Post();
    $post->post_id = 27;
    $post->Delete()->Where(["post_id" => $post->post_id]);
}

```

The method needs to be called with Where because you rarely delete all records and Delete doesn't execute a query.

### 7.5 Selecting records from one table

Get all records:

```
$post->Select()->Get();
```

Limit amount of records to get :

```
$post->Select()->Take(5);
```

Select records with a Where condition :

```
$post->Select()->Where(["post_id" => $post->post_id])->Get();
```

Get amount of results in Query:

```
$post->Select()->Where(["post_id" => $post->post_id])->Count();
```

Order records :

```
$post->Select()->OrderBy(["ASC" => "created_at"])->Get();
```

The key of the parameter in the method OrderBy must be the direction ASC or DESC and the value must be the field which it will be sorted by.

## 7.6 Select records with Join

Get all records with Join :

```
$post->Join(["x" => "category.title"], ["category.fk_category_id" => "post.fk_category_id"])->Get();
```

The first parameter is the field or the fields which want to be joined. It only works with one field from each joined table. The second parameter is the join condition.

## 8 Closing words

Have fun using the framework. I will maybe make some updates on it in future.