Пример MVC в php. Вторая статья. Маршрутизация, контролеры, экшены, шаблоны и модели.

В этой статье мы напишем «каркас» нашего проекта. Под словом «каркас» я подразумеваю рабочий код, который будет иметь в своей основе MVC подход, то есть будет иметь четкое разделение логики на контролеры, экшены, шаблоны (представления) и модели.

И так начнем, как я уже писал в предыдущей статье, паттерн MVC подразумевает одну точку входа – index.php, через это скрипт будут проходить все запросы, через него будет работать вся логика проекта. Для того чтобы реализовать такой подход необходимо настроить сервер, подразумевается, что сайт работает на сервере apache, поэтому нам достаточно создать файл .htaccess, в котором мы укажем правила маршрутизации URL. Помимо определения точки входа, маршрутизация позволяет создавать ЧПУ(человеко-понятные урлы). То есть после правильной настройки, адреса страниц буду выглядеть вот так site.ru/article/new.

Для начала, давайте составим .htaccess, который перенаправит обработку всех страниц на скрипт index.php. Код выглядит вот так:

[php]

RewriteEngine on

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.\*)$ index.php?route=$1 [L,QSA]

[/php]

Файл .htaccess должен лежать в корневой папке сайта, тут же необходимо создать скрипт index.php, который является точкой входа. Давайте запишем в index.php одну строку, для проверки работы перенаправления:

[php]

Echo “test”;

[/php]

Теперь можно проверять работу работу перенаправления, введите любой адрес и посмотрите, что получиться: [test-mvc.web/sdf/sdf/](http://test-mvc.web/sdf/sdf/) или [test-mvc.web/sdf/sdf/](http://test-mvc.web/sdf/sdf/)2342/не важно, на экране в любом случае, должно появиться «Test». Если Вы увидели эту надпись, значит, у нас все получилось.

Продолжим, давайте для удобства создадим в корне сайта файл config.php, в котором будем задавать различные константы, облегчающие своим существование настройку сайта. Это могут быть различные пути к скриптам, подступы к базе данных и так далее. Сейчас в конфиге давайте зададим следующее:

[php]

// Задаем константы:

define ('DS', DIRECTORY\_SEPARATOR); // разделитель для путей к файлам

$sitePath = realpath(dirname(\_\_FILE\_\_) . DS);

define ('SITE\_PATH', $sitePath); // путь к корневой папке сайта

// для подключения к бд

define('DB\_USER', 'root');

define('DB\_PASS', '');

define('DB\_HOST', 'localhost');

define('DB\_NAME', 'blog\_mvc');

[/php]

Для того, чтобы константы и другие данные конфига мы могли использовать во всем проекте, в файле index.php необходимо подключить скрипт config.php.

Помимо подключения файла с настройками, в index.php нужно создать подключение к базе данных, подключить скрипт с ядром сайта и запустить роутер, в котором будет происходить маршрутизация.

Теперь по порядку, создание **соединения с базой данных** будет находиться в index.php для того, чтобы соединение открывалось только один раз. Единожды открыв соединение, мы сможем использовать его во всех контроллерах и моделях, но об этом чуть позже. Сейчас просто создадим соединение с базой. Для работы с бд я решил использовать PDO. Подробнее почитать про PDO можно тут: <http://www.php.net/manual/ru/intro.pdo.php>

**Ядро** сайта расположим в папке core и назовем скрипт core.php, тут мы напишем функцию, которая будет сама подключать, необходимы для работы классы. Такая функция очень облегчит и упростит нам работу с контролерами, моделями и тд. Поскольку, забегая вперед скажу, что каждый контролер и каждая модель будут представлять собой отдельный класс.

Помимо авто подключения классов, добавим в ядро создания хранилища (реестра), в котором будем хранить все необходимые объекты и переменные, которые могут пригодиться в любом месте проекта.

**Роутер** тоже подключим в индексном файле, он будет анализировать URL и подключать необходимый контроллер и экшен. Что такое контролер я писал в предыдущей статье, а информацию про экшен я пропустил умышленно, не став нагружать лишней информацией. Так что же такое экшен?

Контролер это класс, в котором заключены различные методы, при MVC подходе каждый метод будет являться экшеном. То есть экшен(action) – это метод класса, который будет обрабатывать данные и передавать их в представление (в шаблон). Может быть, пока не совсем понятно, но после примера все станет на свои места.

На данном этапе теории достаточно, давайте перейдем к практике. Приведу код файлов, работу которых, я описывал выше.

Код скрипта index.php:

[php]

// включим отображение всех ошибок

error\_reporting (E\_ALL);

// подключаем конфиг

include ('/config.php');

// Соединяемся с БД

$dbObject = new PDO('mysql:host=' . DB\_HOST . ';dbname=' . DB\_NAME, DB\_USER, DB\_PASS);

// подключаем ядро сайта

include (SITE\_PATH . DS . 'core' . DS . 'core.php');

// Загружаем router

$router = new Router($registry);

// записываем данные в реестр

$registry->set ('router', $router);

// задаем путь до папки контроллеров.

$router->setPath (SITE\_PATH . 'controllers');

// запускаем маршрутизатор

$router->start();

[/php]

Скрипт core.php:

[php]

// Загрузка классов "на лету"

function \_\_autoload($className) {

$filename = strtolower($className) . '.php';

// определяем класс и находим для него путь

$expArr = explode('\_', $className);

if(empty($expArr[1]) OR $expArr[1] == 'Base'){

$folder = 'classes';

}else{

switch(strtolower($expArr[0])){

case 'controller':

$folder = 'controllers';

break;

case 'model':

$folder = 'models';

break;

default:

$folder = 'classes';

break;

}

}

// путь до класса

$file = SITE\_PATH . $folder . DS . $filename;

// проверяем наличие файла

if (file\_exists($file) == false) {

return false;

}

// подключаем файл с классом

include ($file);

}

// запускаем реестр (хранилище)

$registry = new Registry;

[/php]

Класс хранилища Registry.php, будет находиться в папке /classes/

[php]

// Класс хранилища

Class Registry {

private $vars = array();

// запись данных

function set($key, $var) {

if (isset($this->vars[$key]) == true) {

throw new Exception('Unable to set var `' . $key . '`. Already set.');

}

$this->vars[$key] = $var;

return true;

}

// получение данных

function get($key) {

if (isset($this->vars[$key]) == false) {

return null;

}

return $this->vars[$key];

}

// удаление данных

function remove($var) {

unset($this->vars[$key]);

}

}

[/php]

Код файла router.php, который находиться в папке /classes/

[php]

// класс роутера

Class Router {

private $registry;

private $path;

private $args = array();

// получаем хранилище

function \_\_construct($registry) {

$this->registry = $registry;

}

// задаем путь до папки с контроллерами

function setPath($path) {

$path = trim($path, '/\\');

$path .= DS;

// если путь не существует, сигнализируем об этом

if (is\_dir($path) == false) {

throw new Exception ('Invalid controller path: `' . $path . '`');

}

$this->path = $path;

}

// определение контроллера и экшена из урла

private function getController(&$file, &$controller, &$action, &$args) {

$route = (empty($\_GET['route'])) ? '' : $\_GET['route'];

unset($\_GET['route']);

if (empty($route)) {

$route = 'index';

}

// Получаем части урла

$route = trim($route, '/\\');

$parts = explode('/', $route);

// Находим контроллер

$cmd\_path = $this->path;

foreach ($parts as $part) {

$fullpath = $cmd\_path . $part;

// Проверка существования папки

if (is\_dir($fullpath)) {

$cmd\_path .= $part . DS;

array\_shift($parts);

continue;

}

// Находим файл

if (is\_file($fullpath . '.php')) {

$controller = $part;

array\_shift($parts);

break;

}

}

// если урле не указан контролер, то испольлзуем поумолчанию index

if (empty($controller)) {

$controller = 'index';

}

// Получаем экшен

$action = array\_shift($parts);

if (empty($action)) {

$action = 'index';

}

$file = $cmd\_path . $controller . '.php';

$args = $parts;

}

function start() {

// Анализируем путь

$this->getController($file, $controller, $action, $args);

// Проверка существования файла, иначе 404

if (is\_readable($file) == false) {

die ('404 Not Found');

}

// Подключаем файл

include ($file);

// Создаём экземпляр контроллера

$class = 'Controller\_' . $controller;

$controller = new $class($this->registry);

// Если экшен не существует - 404

if (is\_callable(array($controller, $action)) == false) {

die ('404 Not Found');

}

// Выполняем экшен

$controller->$action();

}

}

[/php]

Теперь необходимо создать папки для хранения контроллеров, шаблонов и моделей – в корне создадим три папки controllers, views и models. И создадим несколько тестовых файлов /controllers/index.php, /views/index/index.php и /models/model\_users.php, а теперь заполним файлы:

Для контроллера:

[php]

// контролер

Class Controller\_Index Extends Controller\_Base {

// шаблон

public $layouts = "first\_layouts";

// экшен

function index() {

$model = new Model\_Users();

$userInfo = $model->getUser();

$this->template->vars('userInfo', $userInfo);

$this->template->view('index');

}

}

[/php]

Для отображения(/views/index/index.php)

[php]

Test view <br/>

id: <?=$userInfo['id'];?><br/>

name: <?=$userInfo['name'];?>

[/php]

И модель:

[php]

// модель

Class Model\_Users{

public function getUser(){

return array('id'=>1, 'name'=>'test\_name');

}

}

[/php]

Как вы могли заметить, класс контролера наследуется от родительского класса Controller\_Base. Это сделано, для того, чтобы упростить класс контролера. Поскольку нам еще необходимо подключать класс для работы с шаблонами, его подключение вынесено в Controller\_Base.

Приведу его код, он расположен в папке /classes/ и называется controller\_base.php :

[php]

// абстрактый класс контроллера

Abstract Class Controller\_Base {

protected $registry;

protected $template;

protected $layouts; // шаблон

public $vars = array();

// в конструкторе подключаем шаблоны

function \_\_construct($registry) {

$this->registry = $registry;

// шаблоны

$this->template = new Template($this->layouts, get\_class($this));

}

abstract function index();

}

[/php]

Теперь осталось только разобраться с шаблонами. В абстрактном классе Controller\_Base мы вызываем класс Template и передаем ему имя шаблона и имя контроллера.

Код класса Template, который лежит тут /classes/ и называется template.php

[php]

// класс для подключения шаблонов и передачи данных в отображение

Class Template {

private $template;

private $controller;

private $layouts;

private $vars = array();

function \_\_construct($layouts, $controllerName) {

$this->layouts = $layouts;

$arr = explode('\_', $controllerName);

$this->controller = strtolower($arr[1]);

}

// установка переменных, для отображения

function vars($varname, $value) {

if (isset($this->vars[$varname]) == true) {

trigger\_error ('Unable to set var `' . $varname . '`. Already set, and overwrite not allowed.', E\_USER\_NOTICE);

return false;

}

$this->vars[$varname] = $value;

return true;

}

// отображение

function view($name) {

$pathLayout = SITE\_PATH . 'views' . DS . 'layouts' . DS . $this->layouts . '.php';

$contentPage = SITE\_PATH . 'views' . DS . $this->controller . DS . $name . '.php';

if (file\_exists($pathLayout) == false) {

trigger\_error ('Layout `' . $this->layouts . '` does not exist.', E\_USER\_NOTICE);

return false;

}

if (file\_exists($contentPage) == false) {

trigger\_error ('Template `' . $name . '` does not exist.', E\_USER\_NOTICE);

return false;

}

foreach ($this->vars as $key => $value) {

$$key = $value;

}

include ($pathLayout);

}

}

[/php]

Если вы внимательно прочитали код, то наверняка поняли, что для отображения на страницах у нас используется шаблон first\_layouts и вьюха(отображение) index.php – ее код я приводил чуть выше. Все что нам осталось, это создать файл шаблона first\_layouts. Расположим его в папке /views/layouts/first\_layouts.php

Шаблон будет содержать вот такой код:

[php]

<h1> header </h1>

<?php

include ($contentPage);

?>

<h1> footer </h1>

[/php]

Вот и все, на этом создание «каркаса» закончено. Сейчас у нас получилась самая простая структура, использующая в своей основе паттерн MVC. В этой статье я не стал затрагивать работу с базой данных, только вскользь упомянул ее, поскольку статья и так получилась большая. Непосредственно работу с базой данных я опишу в следующей статье.

На этом статья закончена, скачать исходники можно архивом тут.