**MVC паттерны**

[Реализация MVC паттерна на примере создания сайта-визитки на PHP 2](#_Toc21767228)

[Оглавление 2](#_Toc21767229)

[Введение 2](#_Toc21767230)

[1. Теория 3](#_Toc21767231)

[1.1. Front Controller и Page Controller 5](#_Toc21767232)

[1.2. Маршрутизация URL 6](#_Toc21767233)

[2. Практика 7](#_Toc21767234)

[2.1. Реализация маршрутизатора URL 9](#_Toc21767235)

[Содержимое файла route.php 9](#_Toc21767236)

[2.2. Возвращаемся к реализации MVC 10](#_Toc21767237)

[Содержимое файла **model.php** 10](#_Toc21767238)

[Содержимое файла **view.php** 11](#_Toc21767239)

[Содержимое файла **controller.php** 12](#_Toc21767240)

[2.3. Реализация классов потомков Model и Controller, создание View's 13](#_Toc21767241)

[**2.3.1. Созда**ем главную страницу 14](#_Toc21767242)

[2.3.2. Создаем страницу «Портфолио» 16](#_Toc21767243)

[2.3.3. Создаем остальные страницы 19](#_Toc21767244)

[3. Результат 19](#_Toc21767245)

[4. Заключение 20](#_Toc21767246)

[5. Подборка полезных ссылок по сабжу 20](#_Toc21767247)

[Шаблон проектирования MVC 22](#_Toc21767248)

[Что такое MVC 22](#_Toc21767249)

[Вид или представление 22](#_Toc21767250)

[Контроллер 22](#_Toc21767251)

[Модель 22](#_Toc21767252)

[Графическая схема 23](#_Toc21767253)

[Правка точки входа на сайт. Пример простой CMS 24](#_Toc21767254)

[Главный файл данной CMS 24](#_Toc21767255)

[Создание модели 25](#_Toc21767256)

[Редактируем ядро – класс ACore 27](#_Toc21767257)

[Редактирование классов: main, menu и category 28](#_Toc21767258)

[Файл category.php: 29](#_Toc21767259)

[Создание шаблонов 29](#_Toc21767260)

[Шаблон верхнего горизонтального меню (файл menu\_top.php): 29](#_Toc21767261)

[Шаблон футера (файл footer.php): 30](#_Toc21767262)

[И три динамических шаблона. 30](#_Toc21767263)

[Шаблон главной страницы (main.php): 30](#_Toc21767264)

[Шаблон страницы категорий (category.php): 30](#_Toc21767265)

[Шаблон для вывода страниц верхнего меню (menu.php): 31](#_Toc21767266)

[Разработка веб-приложения на PHP 31](#_Toc21767267)

[Похожие статьи: 32](#_Toc21767268)

[MVC для веб: проще некуда 33](#_Toc21767269)

[Определение паттерна MVC 33](#_Toc21767270)

[Классическая схема веб-приложения 33](#_Toc21767271)

[Рисунок 1 33](#_Toc21767272)

[Рисунок 2 34](#_Toc21767273)

[Рисунок 3 35](#_Toc21767274)

[Пример 36](#_Toc21767275)

[Листинг 1. Шаблон product.tpl.php отображает данные о Товаре (которые к моменту его вызова уже содержит объект $product): 36](#_Toc21767276)

[Листинг 2. Шаблон error.tpl.php отображает сообщение об ошибке (которое содержится в переменной $error): 37](#_Toc21767277)

[Листинг 3. Контроллер product.php, служащий для отобоажения Товара, будет выглядеть примерно так: 37](#_Toc21767278)

[Рисунки-иллюстрации 40](#_Toc21767279)

# Реализация MVC паттерна на примере создания сайта-визитки на PHP

<https://habr.com/ru/post/150267/>

* [[](https://habr.com/ru/users/vitalyswipe/)vitalyswipe](https://habr.com/ru/users/vitalyswipe/)27 августа 2012 в 07:03 [Разработка веб-сайтов](https://habr.com/ru/hub/webdev/), [PHP](https://habr.com/ru/hub/php/)

Как вы уже догадались из названия статьи, сегодня речь пойдет о самом популярном, разве что после [Singleton](http://ru.wikipedia.org/wiki/Singleton), шаблоне проектирования [MVC](http://ru.wikipedia.org/wiki/MVC), хотя такое сравнение не совсем уместно. Понимание концепции MVC может помочь вам в рефакторинге и разрешении неприятных ситуаций в которые, возможно попал ваш проект. Дабы восполнить пробел, мы реализуем шаблон MVC на примере простого сайта-визитки.

## Введение

Многие начинают писать проект для работы с единственной задачей, не подразумевая, что это может вырасти в многопользовательскую систему управления, ну допустим, контентом или упаси бог, производством. И всё вроде здорово и классно, всё работает, пока не начинаешь понимать, что тот код, который написан — состоит целиком и полностью из костылей и хардкода. Код перемешанный с версткой, запросами и костылями, неподдающийся иногда даже прочтению. Возникает насущная проблема: при добавлении новых фич, приходится с этим кодом очень долго и долго возиться, вспоминая «а что же там такое написано то было?» и проклинать себя в прошлом.

Вы можеть быть даже слышали о шаблонах проектирования и даже листали эти прекрасные книги:

* Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидесс «Приемы объектно ориентированного проектирования. Паттерны проектирования»;
* М. Фаулер «Архитектура корпоративных программных приложений».

А многие, не испугавшись огромных руководств и документаций, пытались изучить какой-либо из современных фреймворков и столкнувшись со сложностью понимания (в силу наличия множества архитектруных концепций хитро увязанных между собой) отложили изучение и применение современных интсрументов в «долгий ящик».

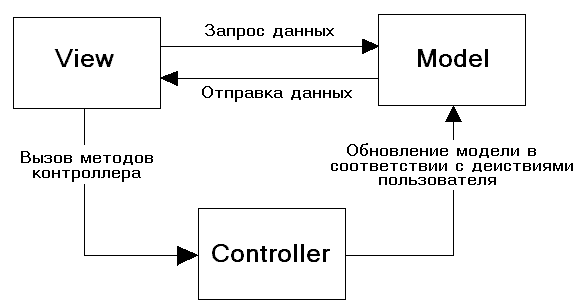
Представленная статья будет полезна в первую очередь новичкам. Во всяком случае, я надеюсь что за пару часов вы сможете получить представление о реализации MVC паттерна, который лежит в основе всех современных веб-фреймворков, а также получить «пищу» для дальнейших размышлений над тем — «как стоит делать». В конце статьи приводится подборка полезных ссылок, которые также помогут разобраться из чего состоят веб-фреймворки (помимо MVC) и как они работают.

Прожженные PHP-программисты вряд ли найдут в данной статье что-то новое для себя, но их замечания и комментарии к основному тексту были бы очень кстати! Т.к. без теории практика невозможна, а без практики теория бесполезна, то сначала будет чуть-чуть теории, а потом перейдем к практике. Если вы уже знакомы с концепцией MVC, можете пропустить раздел с теорией и сразу перейти к практике.

## 1. Теория

Шаблон MVC описывает простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса. В результате, приложение легче масштабируется, тестируется, сопровождается и конечно же реализуется.

Рассмотрим концептуальную схему шаблона MVC (на мой взгляд — это наиболее удачная схема из тех, что я видел):



В архитектуре MVC модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс, а контроллер обеспечивает взаимодействие между моделью и представлением.

Типичную последовательность работы MVC-приложения можно описать следующим образом:

1. При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.
2. При этом отображается вид, скажем главной страницы сайта.
3. Приложение получает запрос от пользователя и определяет запрошенные контроллер и действие. В случае главной страницы, выполняется действие по умолчанию (*index*).
4. Приложение создает экземпляр контроллера и запускает метод действия,
5. в котором, к примеру, содержаться вызовы модели, считывающие информацию из базы данных.
6. После этого, действие формирует представление с данными, полученными из модели и выводит результат пользователю.

**Модель** — содержит бизнес-логику приложения и включает методы выборки (это могут быть методы ORM), обработки (например, правила валидации) и предоставления конкретных данных, что зачастую делает ее очень толстой, что вполне нормально.

Модель не должна напрямую взаимодействовать с пользователем. Все переменные, относящиеся к запросу пользователя должны обрабатываться в контроллере.

Модель не должна генерировать HTML или другой код отображения, который может изменяться в зависимости от нужд пользователя. Такой код должен обрабатываться в видах.

Одна и та же модель, например: модель аутентификации пользователей может использоваться как в пользовательской, так и в административной части приложения. В таком случае можно вынести общий код в отдельный класс и наследоваться от него, определяя в наследниках специфичные для подприложений методы.

**Вид** — используется для задания внешнего отображения данных, полученных из контроллера и модели.

Виды cодержат HTML-разметку и небольшие вставки PHP-кода для обхода, форматирования и отображения данных.

Не должны напрямую обращаться к базе данных. Этим должны заниматься модели.

Не должны работать с данными, полученными из запроса пользователя. Эту задачу должен выполнять контроллер.

Может напрямую обращаться к свойствам и методам контроллера или моделей, для получения готовых к выводу данных.

Виды обычно разделяют на общий шаблон, содержащий разметку, общую для всех страниц (например, шапку и подвал) и части шаблона, которые используют для отображения данных выводимых из модели или отображения форм ввода данных.

**Контроллер** — связующее звено, соединяющее модели, виды и другие компоненты в рабочее приложение. Контроллер отвечает за обработку запросов пользователя. Контроллер не должен содержать SQL-запросов. Их лучше держать в моделях. Контроллер не должен содержать HTML и другой разметки. Её стоит выносить в виды.

В хорошо спроектированном MVC-приложении контроллеры обычно очень тонкие и содержат только несколько десятков строк кода. Чего, не скажешь о Stupid Fat Controllers (SFC) в CMS Joomla. Логика контроллера довольно типична и большая ее часть выносится в базовые классы.

Модели, наоборот, очень толстые и содержат большую часть кода, связанную с обработкой данных, т.к. структура данных и бизнес-логика, содержащаяся в них, обычно довольно специфична для конкретного приложения.

### 1.1. Front Controller и Page Controller

В большинстве случае, взаимодействие пользователя с web-приложением проходит посредством переходов по ссылкам. Посмотрите сейчас на адресную строку браузера — по этой ссылке вы получили данный текст. По другим ссылкам, например, находящимся справа на этой странице, вы получите другое содержимое. Таким образом, ссылка представляет конкретную команду web-приложению.

Надеюсь, вы уже успели заметить, что у разных сайтов могут быть совершенные разные форматы построения адресной строки. Каждый формат может отображать архитектуру web-приложения. Хотя это и не всегда так, но в большинстве случаев это явный факт.

Рассмотрим два варианта адресной строки, по которым показывается какой-то текст и профиль пользователя.

Первый вариант:

1. [www.example.com/article.php?id=3](http://www.example.com/article.php?id=3)
2. [www.example.com/user.php?id=4](http://www.example.com/user.php?id=4)

Здесь каждый сценарий отвечает за выполнение определённой команды.

Второй вариант:

1. [www.example.com/index.php?article=3](http://www.example.com/index.php?article=3)
2. [www.example.com/index.php?user=4](http://www.example.com/index.php?user=4)

А здесь все обращения происходят в одном сценарии **index.php**.

Подход с множеством точек взаимодействия вы можете наблюдать на форумах с движком phpBB. Просмотр форума происходит через сценарий **viewforum.php**, просмотр топика через **viewtopic.php** и т.д. Второй подход, с доступом через один физический файл сценария, можно наблюдать в моей любимой CMS MODX, где все обращения проходят через **index.php**.

Эти два подхода совершенно различны. Первый — характерен для шаблона контроллер страниц (Page Controller), а второй подход реализуется паттерном контроллер запросов (Front Controller). Контроллер страниц хорошо применять для сайтов с достаточно простой логикой. В свою очередь, контроллер запросов объединяет все действия по обработке запросов в одном месте, что даёт ему дополнительные возможности, благодаря которым можно реализовать более трудные задачи, чем обычно решаются контроллером страниц. Я не буду вдаваться в подробности реализации контроллера страниц, а скажу лишь, что в практической части будет разработан именно контроллер запросов (некоторое подобие).

### 1.2. Маршрутизация URL

Маршрутизация URL позволяет настроить приложение на прием запросов с URL, которые не соответствуют реальным файлам приложения, а также использовать [ЧПУ](http://ru.wikipedia.org/wiki/%D0%A7%D0%9F%D0%A3_(%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82)), которые семантически значимы для пользователей и предпочтительны для поисковой оптимизации.

К примеру, для обычной страницы, отображающей форму обратной связи, URL мог бы выглядеть так:

<http://www.example.com/contacts.php?action=feedback>

Приблизительный код обработки в таком случае:

switch($\_GET['action'])

{

case "about" :

require\_once("about.php"); *// страница "О Нас"*

break;

case "contacts" :

require\_once("contacts.php"); *// страница "Контакты"*

break;

case "feedback" :

require\_once("feedback.php"); *// страница "Обратная связь"*

break;

default :

require\_once("page404.php"); *// страница "404"*

break;

}

Думаю, почти все так раньше делали.

С использованием движка маршрутизации URL вы сможете для отображения той же информации настроить приложение на прием таких запросов:

<http://www.example.com/contacts/feedback>

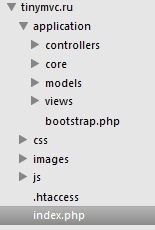
Здесь contacts представляет собой контроллер, а feedback — это метод контроллера contacts, отображающий форму обратной связи и т.д. Мы еще вернемся к этому вопросу в практической части.

Также стоит знать, что маршрутизаторы многих веб-фреймворков позволяют создавать произвольные маршруты URL (указать, что означает каждая часть URL) и правила их обработки.

Теперь мы обладаем достаточными теоретическими знаниями, чтобы перейти к практике.

## 2. Практика

Для начала создадим следующую структуру файлов и папок:



Забегая вперед, скажу, что в папке core будут храниться базовые классы Model, View и Controller.

Их потомки будут храниться в директориях controllers, models и views. Файл **index.php** это точка в хода в приложение. Файл **bootstrap.php** инициирует загрузку приложения, подключая все необходимые модули и пр.

Будем идти последовательно; откроем файл index.php и наполним его следующим кодом:

ini\_set('display\_errors', 1);

require\_once 'application/bootstrap.php';

Тут вопросов возникнуть не должно.

Следом, сразу же перейдем к фалу **bootstrap.php**:

require\_once 'core/model.php';

require\_once 'core/view.php';

require\_once 'core/controller.php';

require\_once 'core/route.php';

Route::start(); *// запускаем маршрутизатор*

Первые три строки будут подключать пока что несуществующие файлы ядра. Последние строки подключают файл с классом маршрутизатора и запускают его на выполнение вызовом статического метода start.

### 2.1. Реализация маршрутизатора URL

Пока что отклонимся от реализации паттерна MVC и займемся мрашрутизацией. Первый ш﻿аг, который нам нужно сделать, записать следующий код в **.htaccess**:

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule .\* index.php [L]

Этот код перенаправит обработку всех страниц на **index.php**, что нам и нужно. Помните в первой части мы говорили о Front Controller?!

Маршрутизацию мы поместим в отдельный файл **route.php** в директорию core. В этом файле опишем класс Route, который будет запускать методы контроллеров, которые в свою очередь будут генерировать вид страниц.

#### Содержимое файла route.php

Замечу, что в классе реализована очень упрощенная логика (несмотря на объемный код) и возможно даже имеет проблемы безопасности. Это было сделано намерено, т.к. написание полноценного класса маршрутизации заслуживает как минимум отдельной статьи. Рассмотрим основные моменты…

В элементе глобального массива $\_SERVER['REQUEST\_URI'] содержится полный адрес по которому обратился пользователь.

Например: [example.ru/contacts/feedback](http://example.ru/contacts/feedback)

С помощью функции *explode* производится разделение адреса на составлющие. В результате мы получаем имя контроллера, для приведенного примера, это контроллер *contacts* и имя действия, в нашем случае — *feedback*.

Далее подключается файл модели (модель может отсутствовать) и файл контроллера, если таковые имеются и наконец, создается экземпляр контроллера и вызывается действие, опять же, если оно было описано в классе контроллера.

Таким образом, при переходе, к примеру, по адресу:

[example.com/portfolio](http://example.com/portfolio)

или

[example.com/portfolio/index](http://example.com/portfolio/index)

роутер выполнит следующие действия:

1. подключит файл model\_portfolio.php из папки models, содержащий класс Model\_Portfolio;
2. подключит файл controller\_portfolio.php из папки controllers, содержащий класс Controller\_Portfolio;
3. создаст экземпляр класса Controller\_Portfolio и вызовет действие по умолчанию — action\_index, описанное в нем.

Если пользователь попытается обратиться по адресу несуществующего контроллера, к примеру:

[example.com/ufo](http://example.com/ufo)

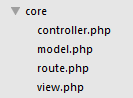
то его перебросит на страницу «404»:

[example.com/404](http://example.com/404)

То же самое произойдет если пользователь обратится к действию, которое не описано в контроллере.

### 2.2. Возвращаемся к реализации MVC

Перейдем в папку core и добавим к файлу route.php еще три файла: **model.php, view.php и controller.php**



Напомню, что они будут содержать базовые классы, к написанию которых мы сейчас и приступим.

#### Содержимое файла **model.php**

class Model

{

public function get\_data()

{

}

}

Класс модели содержит единственный пустой метод выборки данных, который будет перекрываться в классах потомках. Когда мы будем создавать классы потомки все станет понятней.

#### Содержимое файла **view.php**

class View

{

*//public $template\_view; // здесь можно указать общий вид по умолчанию.*

function generate($content\_view, $template\_view, $data = null)

{

*/\**

*if(is\_array($data)) {*

*// преобразуем элементы массива в переменные*

*extract($data);*

*}*

*\*/*

include 'application/views/'.$template\_view;

}

}

Не трудно догадаться, что метод *generate* предназначен для формирования вида. В него передаются следующие параметры:

1. $content\_file — виды отображающие контент страниц;
2. $template\_file — общий для всех страниц шаблон;
3. $data — массив, содержащий элементы контента страницы. Обычно заполняется в модели.

Функцией include динамически подключается общий шаблон (вид), внутри которого будет встраиваться вид

для отображения контента конкретной страницы.

В нашем случае общий шаблон будет содержать header, menu, sidebar и footer, а контент страниц будет содержаться в отдельном виде. Опять же это сделано для упрощения.

#### Содержимое файла **controller.php**

class Controller {

public $model;

public $view;

function \_\_construct()

{

$this->view = new View();

}

function action\_index()

{

}

}

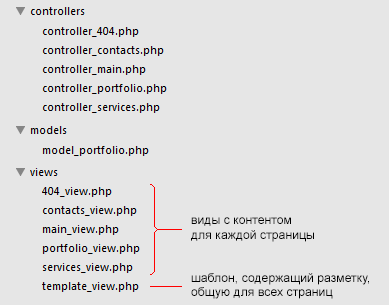
Метод *action\_index* — это действие, вызываемое по умолчанию, его мы перекроем при реализации классов потомков.

### 2.3. Реализация классов потомков Model и Controller, создание View's

Теперь начинается самое интересное! Наш сайт-визитка будет состоять из следущих страниц:

1. Главная
2. Услуги
3. Портфолио
4. Контакты
5. А также — страница «404»

Для каждой из страниц имеется свой контроллер из папки controllers и вид из папки views. Некоторые страницы могут использовать модель или модели из папки models.



На предыдущем рисунке отдельно выделен файл **template\_view.php** — это шаблон, содержащий общую для всех страниц разметку. В простейшем случае он мог бы выглядеть так:

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8">

<title>Главная</title>

</head>

<body>

<?php include 'application/views/'.$content\_view; ?>

</body>

</html>

Для придания сайту презентабельного вида сверстаем CSS шаблон и интегририруем его в наш сайт путем изменения структуры HTML-разметки и подключения CSS и JavaScript файлов:

<link rel="stylesheet" type="text/css" href="/css/style.css" />

<script src="/js/jquery-1.6.2.js" type="text/javascript"></script>

В конце статьи, в разделе «Результат», приводится ссылка на GitHub-репозиторий с проектом, в котором проделаны действия по интеграции простенького шаблона.

#### **2.3.1. Созда**ем главную страницу

Начнем с контроллера **controller\_main.php**, вот его код:

class Controller\_Main extends Controller

{

function action\_index()

{

$this->view->generate('main\_view.php', 'template\_view.php');

}

}

В метод *generate* экземпляра класса View передаются имена файлов общего шаблона и вида c контентом страницы.

Помимо индексного действия в контроллере конечно же могут содержаться и другие действия.

Файл с общим видом мы рассмотрели ранее. Рассмотрим файл контента **main\_view.php**:

<h1>Добро пожаловать!</h1>

<p>

<img src="/images/office-small.jpg" align="left" >

<a href="/">ОЛОЛОША TEAM</a> - команда первоклассных специалистов в области разработки веб-сайтов с многолетним опытом коллекционирования мексиканских масок, бронзовых и каменных статуй из Индии и Цейлона, барельефов и изваяний, созданных мастерами Экваториальной Африки пять-шесть веков назад...

</p>

Здесь содержиться простая разметка без каких либо PHP-вызовов.

Для отображения главной странички можно воспользоваться одним из следующих адресов:

* [example.com](http://example.com/)
* [example.com/main](http://example.com/main)
* [example.com/main/index](http://example.com/main/index)

Пример с использованием вида, отображающего данные полученные из модели мы рассмотрим далее.

#### 2.3.2. Создаем страницу «Портфолио»

В нашем случае, страница «Портфолио» — это единственная страница использующая модель.

Модель обычно включает методы выборки данных, например:

1. методы нативных библиотек pgsql или mysql;
2. методы библиотек, реализующих абстракицю данных. Например, методы библиотеки PEAR MDB2;
3. методы ORM;
4. методы для работы с NoSQL;
5. и др.

Для простоты, здесь мы не будем использовать SQL-запросы или ORM-операторы. Вместо этого мы сэмулируем реальные данные и сразу возвратим массив результатов.

Файл модели **model\_portfolio.php** поместим в папку models. Вот его содержимое:

class Model\_Portfolio extends Model

{

public function get\_data()

{

return array(

array(

'Year' => '2012',

'Site' => 'http://DunkelBeer.ru',

'Description' => 'Промо-сайт темного пива Dunkel от немецкого производителя Löwenbraü выпускаемого в России пивоваренной компанией "CАН ИнБев".'

),

array(

'Year' => '2012',

'Site' => 'http://ZopoMobile.ru',

'Description' => 'Русскоязычный каталог китайских телефонов компании Zopo на базе Android OS и аксессуаров к ним.'

),

*// todo*

);

}

}

Класс контроллера модели содержится в файле **controller\_portfolio.php**, вот его код:

class Controller\_Portfolio extends Controller

{

function \_\_construct()

{

$this->model = new Model\_Portfolio();

$this->view = new View();

}

function action\_index()

{

$data = $this->model->get\_data();

$this->view->generate('portfolio\_view.php', 'template\_view.php', $data);

}

}

В переменную *data* записывается массив, возвращаемый методом *get\_data*, который мы рассматривали ранее.

Далее эта переменная передается в качестве параметра метода *generate*, в который также передаются: имя файла с общим шаблон и имя файла, содержащего вид c контентом страницы.

Вид содержащий контент страницы находится в файле **portfolio\_view.php**.

<h1>Портфолио</h1>

<p>

<table>

Все проекты в следующей таблице являются вымышленными, поэтому даже не пытайтесь перейти по приведенным ссылкам.

<tr><td>Год</td><td>Проект</td><td>Описание</td></tr>

<?php

foreach($data as $row)

{

echo '<tr><td>'.$row['Year'].'</td><td>'.$row['Site'].'</td><td>'.$row['Description'].'</td></tr>';

}

?>

</table>

</p>

Здесь все просто, вид отображает данные полученные из модели.

#### 2.3.3. Создаем остальные страницы

Остальные страницы создаются аналогично. Их код досутпен в репозитории на GitHub, ссылка на который приводится в конце статьи, в разделе «Результат».

## 3. Результат

А вот что получилось в итоге:

Скриншот получившегося сайта-визитки

Ссылка на GitHub: <https://github.com/vitalyswipe/tinymvc/zipball/v0.1>

А вот в [этой версии](https://github.com/vitalyswipe/tinymvc/zipball/v0.2) я набросал следующие классы (и соответствующие им виды):

* Controller\_Login в котором генерируется вид с формой для ввода логина и пароля, после заполнения которой производится процедура аутентификации и в случае успеха пользователь перенаправляется в админку.
* Contorller\_Admin с индексным действием, в котором проверяется был ли пользователь ранее авторизован на сайте как администратор (если был, то отображается вид админки) и действием logout для разлогинивания.

Аутентификация и авторизация — это другая тема, поэтому здесь она не рассматривается, а лишь приводится ссылка указанная выше, чтобы было от чего оттолкнуться.

## 4. Заключение

Шаблон MVC используется в качестве архитектурной основы во многих фреймворках и CMS, которые создавались для того, чтобы иметь возможность разрабатывать качественно более сложные решения за более короткий срок. Это стало возможным благодаря повышению уровня абстракции, поскольку есть предел сложности конструкций, которыми может оперировать человеческий мозг.

Но, использование веб-фреймворков, типа Yii или Kohana, состоящих из нескольких сотен файлов, при разработке простых веб-приложений (например, сайтов-визиткок) не всегда целесообразно. Теперь мы умеем создавать красивую MVC модель, чтобы не перемешивать Php, Html, CSS и JavaScript код в одном файле.

Данная статья является скорее отправной точкой для изучения CMF, чем примером чего-то истинно правильного, что можно взять за основу своего веб-приложения. Возможно она даже вдохновила Вас и вы уже подумываете написать свой микрофреймворк или CMS, основанные на MVC. Но, прежде чем изобретать очередной велосипед с «блекджеком и шлюхами», еще раз подумайте, может ваши усилия разумнее направить на развитие и в помощь сообществу уже существующего проекта?!

P.S.: Статья была переписана с учетом некоторых замечаний, оставленных в комментариях. Критика оказалась очень полезной. Судя по отклику: комментариям, обращениям в личку и количеству юзеров добавивших пост в избранное затея написать этот пост оказалось не такой уж плохой. К сожалению, не возможно учесть все пожелания и написать больше и подробнее по причине нехватки времени… но возможно это сделают те таинственные личности, кто минусовал первоначальный вариант. Удачи в проектах!

## 5. Подборка полезных ссылок по сабжу

В статье очень часто затрагивается тема веб-фреймворков — это очень обширная тема, потому что даже микрофреймворки состоят из многих компонентов хитро увязанных между собой и потребовалась бы не одна статья, чтобы рассказать об этих компонентах. Тем не менее, я решил привести здесь небольшую подборку ссылок (по которым я ходил при написаниие этой статьи), которые так или иначе касаются темы фреймворков.

------------------------------------

# Шаблон проектирования MVC

[Как создать сайт](https://webformyself.com/) » [Web-программирование](https://webformyself.com/category/webprogramming/) » [PHP](https://webformyself.com/category/webprogramming/phpvideo/) » Шаблон проектирования MVC

<https://webformyself.com/shablon-proektirovaniya-mvc/>

**Дата публикации:** 2014-02-25

***От автора:*** в данном уроке я хотел бы поговорить с Вами о шаблоне проектирования MVC. Это очень популярный, а главное полезный шаблон проектирования, который предусматривает полное разделение логики скрипта от его представления (дизайна). Мы разберем его теоретические основы и в качестве практики переделаем ранее созданную CMS из [*мини курса*](https://webformyself.com/minikurs/cmsfree/free.html), который ранее публиковался у нас на сайте, таким образом, чтобы полностью отделить ее логику от дизайна.

[скачать исходники](http://storage.webformyself.com/storagelessons/2014/february/basic/10/wwwminicms.rar)[скачать урок](http://storage.webformyself.com/storagelessons/2014/february/basic/10/mvc.zip)

## Что такое MVC

Это достаточно сложный шаблон проектирования и по моему мнению это не сколько шаблон — сколько метод или идея того как можно организовать наше веб-приложение. И идея эта заключается в том, чтобы отделить логику программирования от представления (то есть от вывода на экран).

Итак, как следует их названия MVC состоит из трех компонентов *модель* (**Model**) – *вид* (**View**) или представление и *контроллер* (**Controller**). Давайте разберем каждый элемент в отдельности.

### **Вид или представление**

**Вид или представление** — эта часть отвечает за вывод информации на экран — это уже дизайнерская часть нашего веб — приложения с минимальным количеством логики. То есть попросту говоря, этот блок отвечает за внешний вид нашего приложения. Задача представления хранить дизайн скрипта.

### **Контроллер**

**Контроллер** — блок, который получает данные от пользователя, обрабатывает, нормализует их, также выполняет проверку правильности ввода и передает эти обработанные данные в нужную модель. Также он принимает данные от модели, затем выбирает нужное представление, наполняет его данными и отображает на экране браузера. Но при этом, еще раз уточню, контроллер не должен содержать в себе никакой информации о внешнем виде веб-приложения. Контроллер можно рассмотреть как связующее звено между представлением и моделью.

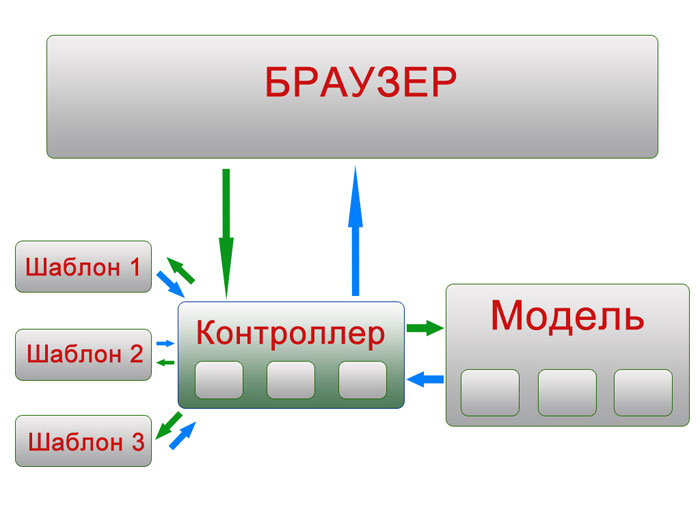
### **Модель**

**Модель** — это основа логики нашего веб-приложения — она отвечает за расчеты, выборку информации из базы данных, изменение информации в БД и т.д. Модель можно представить как библиотеку различных функций, позволяющих реализовывать функционал нашего приложения. То есть — это блок, который получает данные от контроллера, далее на основе этих данных производит необходимые преобразования, либо опять же выбирает данные из БД или изменяет их, а затем передает результат своей работы назад контроллеру.

Если приложение сложное то моделей может быть несколько, например модель для работы со статьями и модель для работы с пользователями.

### Графическая схема

Графически работу шаблона MVC можно представить по такой схеме:



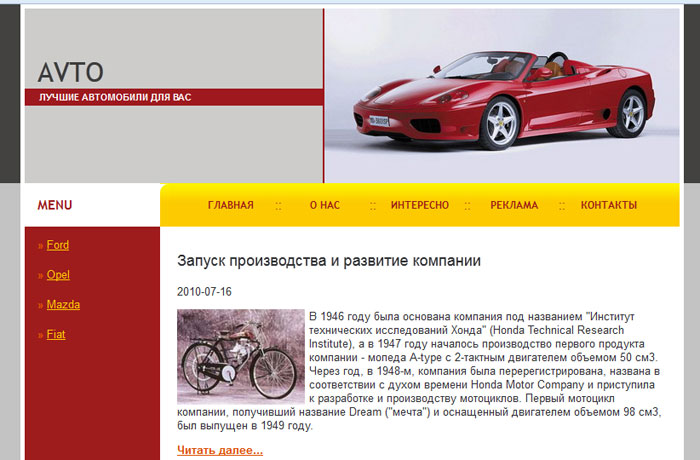
То есть на данной схеме мы с Вами видим взаимодействие между Моделью Контроллером Представлением и браузером пользователя. Зелеными стрелками я обозначил запросы данных или передачу управления элементу, а серыми стрелками обозначено получение результирующих данных.

Смотрите, когда пользователь попадает на страничку, то он совершает какое то действие, к примеру, отправляет форму. То есть, выполняет запрос к контроллеру, контроллер в свою очередь, проверяет эти данные и передает запрос к модели. Модель к примеру сохраняет полученные данные в базе данных и возвращает контроллеру ответ об успешности проведенной операции. Затем контроллер выбирает, какой шаблон отобразить, посылает ему данные и получает ответ, который и направляется в браузер пользователю.

По диаграмме видно, что контроллер может использовать несколько различных шаблонов для отображения страницы скрипта. Далее, как Вы видите, шаблоны напрямую не работают с моделью, они могут получать данные только через контроллер. Аналогично модель не может воздействовать на шаблоны. То есть контроллер выступает связующим звеном между моделью и представлением. Пользователи в свою очередь, также не могут напрямую работать с шаблонами и с моделью, все, что они видят — это страницы предоставляемые контроллером.

### Правка точки входа на сайт. Пример простой CMS

На мой взгляд, наиболее интересно будет перевести уже рабочий готовый проект на структуру MVC, нежели работать с выдуманным примером. Поэтому в данном уроке мы будем работать с простой CMS, которая была создана в мини курсе, что публиковался ранее на нашем сайте. На всякий случай продублирую [ссылку](https://webformyself.com/minikurs/cmsfree/free.html) на данный мини курс. Вот ее вид в браузере:



Думаю, Вы ее прекрасно помните. Данная CMS довольно неплоха и прекрасно выполняет свои задачи. Но у нее есть один существенный недостаток. В ее коде перемешана логика с внешним видом. А это не очень хорошо с точки зрения читаемости кода, да и в дальнейшем, очень трудно будет, что то изменить в коде, особенно если это касается дизайна. Так как не получится изменить дизайн не затрагивая логику работы данной CMS.

Поэтому в данному уроке мы переделаем часть данной CMS на структуру шаблона MVC и если данная идея Вам понравится, то оставшуюся часть Вы доделаете самостоятельно.

### Главный файл данной CMS

Итак, первым делом изменим главный файл данной CMS index.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | <?php  session\_start();  header("Content-Type:text/html;charset=UTF-8");    require\_once("config.php");    function \_\_autoload($c) {  if(file\_exists("controller/".$c.".php")) {  require\_once "controller/".$c.".php";  }  elseif(file\_exists("model/".$c.".php")) {  require\_once "model/".$c.".php";  }    }    if($\_GET['option']) {  $class = trim(strip\_tags($\_GET['option']));  }  else {  $class = 'main';  }      if(class\_exists($class)) {    $obj = new $class;  $obj->get\_body($class);  }  else {  exit("<p>Нет данные для входа</p>");  }      ?> |

Обратите внимание – я убрал ручное подключение необходимых файлов с классами и добавил функцию \_\_autoload() – функция автоматической загрузки классов. Которая будет выполнять подключение недостающих файлов с классами. И убрал проверку наличия файла, перед созданием его объекта. Так как это выполняется в функции \_\_autoload(). Да и папку classes я переименовал в controller (в ней будут содержаться контроллеры). Также создал две пустых папки: model и tpl, для хранения модели и шаблонов соответственно.

### Создание модели

Если мы посмотрим, к примеру, на файл ACore.php мы увидим, что в его методах очень много кода, который отправляет запросы к базе данных. Нам необходимо весь данный код, перенести в модель. При этом в модели необходимо создать отдельные методы, которые будут возвращать массивы данных подученных из базы данных. Поэтому в папке model создаем файл со следующим содержимым:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95 | <?php  class model {    protected $db;    public function \_\_construct() {  $this->db = mysql\_connect(HOST,USER,PASSWORD);  if(!$this->db) {  exit("Ошибка соединения с базой данных".mysql\_error());  }  if(!mysql\_select\_db(DB,$this->db)) {  exit("Нет такой базы данных".mysql\_error());  }  mysql\_query("SET NAMES 'UTF8'");    }    public function get\_left\_bar(){  $query = "SELECT id\_category,name\_category FROM category";    $result = mysql\_query($query);  if(!$result) {  exit(mysql\_error());  }    for($i = 0;$i < mysql\_num\_rows($result); $i++) {  $row[] = mysql\_fetch\_array($result, MYSQL\_ASSOC);  }    return $row;  }    public function menu\_array() {  $query = "SELECT id\_menu,name\_menu FROM menu";    $result = mysql\_query($query);  if(!$result) {  exit(mysql\_error());  }    $row = array();    for($i = 0;$i < mysql\_num\_rows($result); $i++) {  $row[] = mysql\_fetch\_array($result, MYSQL\_ASSOC);  }  return $row;  }    public function get\_main\_content() {    $query = "SELECT id,title,discription,date,img\_src FROM statti ORDER BY date DESC";  $result = mysql\_query($query);  if(!$result) {  exit(mysql\_error());  }    for($i = 0; $i < mysql\_num\_rows($result);$i++) {  $row[] = mysql\_fetch\_array($result,MYSQL\_ASSOC);  }    return $row;  }    public function get\_cat($id\_cat) {    $query = "SELECT id,title,discription,date,img\_src  FROM statti  WHERE cat='$id\_cat'  ORDER BY date DESC";  $result = mysql\_query($query);  if(!$result) {  exit(mysql\_error());  }    $row = array();  for($i = 0; $i < mysql\_num\_rows($result);$i++) {  $row[] = mysql\_fetch\_array($result,MYSQL\_ASSOC);    }  return $row;  }    public function get\_menu($id\_menu) {  $query = "SELECT id\_menu,name\_menu,text\_menu FROM menu WHERE id\_menu='$id\_menu'";  $result = mysql\_query($query);  if(!$result) {  exit(mysql\_error());  }  $row = mysql\_fetch\_array($result,MYSQL\_ASSOC);  return $row;  }      }  ?> |

Думаю, Вы увидите здесь знакомые строки кода. Так как они просто скопированы из классов ACore, main, category и menu.

### Редактируем ядро – класс ACore

Весь лишний код мы из него убрали (перенесли в модель), остались только фрагменты дизайна. Который, нам нужно обязательно исключить, и перенести в отдельный шаблон. Вот исправленный код ACore.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | <?php  abstract class ACore {      protected $m;    public function \_\_construct() {  $this->m = new model();;  }    protected function get\_header() {  return TRUE;  }    protected function get\_left\_bar() {  $result = $this->m->get\_left\_bar();  return $result;  }    protected function get\_menu() {  $row = $this->m->menu\_array();  return $row;  }      protected function get\_footer() {  $row = $this->m->menu\_array();  return $row;  }      public function get\_body($tpl) {  if($\_POST || $\_GET['del']) {  $this->obr();  }  $this->get\_header();  $left\_bar = $this->get\_left\_bar();  $menu\_top = $this->get\_menu();  $content = $this->get\_content();  $footer = $this->get\_footer();  //подключение шаблона  include "tpl/index.php";  }    abstract function get\_content();    }    ?> |

Обратите внимание, какой вид приобрел класс ACore – согласитесь такой код намного удобнее читать и редактировать.

Да хочу обратить внимание на метод get\_body(). Все методы которые вызываются в нем – должны возвращать данные. Которые, в последствии, сохраняются в переменной. Это нужно для того что бы вывести данные в шаблонах. Так как в конце данного метода мы подключаем главный файл шаблона. А значит в подключаемых файлах, будут доступны переменные с данными, что вернут методы get\_header(), get\_left\_bar(), get\_menu() и т.д.

Так же обратите внимание, что метод get\_body(). Теперь должен принимать параметр $tpl – это имя загружаемого класса (текущего контроллера). Данный параметр будет передаваться из файла index.php. И он необходимо для формирования динамических вложенных шаблонов. Все шаблоны сохранены в отдельных файлах с одноименными именами с контроллерами.

### Редактирование классов: main, menu и category

Аналогичным образом изменяем код контроллеров main, menu и category. Файл main.php:



Бесплатный курс по PHP программированию

Освойте курс и узнайте, как создать динамичный сайт на PHP и MySQL с полного нуля, используя модель MVC

**В курсе 39 уроков | 15 часов видео | исходники для каждого урока**

[Получить курс сейчас!](https://webformyself.com/minikurs/secretphp/free.html?utm_medium=systema&utm_source=adsinsingle2&utm_campaign=php_secret)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  class main extends ACore {    public function get\_content() {    $result = $this->m->get\_main\_content();  return $result;    }  }  ?> |

Файл menu.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <?php  class menu extends ACore {    public function get\_content() {      if(!$\_GET['id\_menu']) {  echo 'Не правильные данные для вывода меню';  }  else {  $id\_menu = (int)$\_GET['id\_menu'];  if(!$id\_menu) {  echo 'Не правильные данные для вывода меню';  }  else {  $row = $this->m->get\_menu($id\_menu);  return $row;  }  }  }      }  ?> |

### Файл category.php:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?php  class category extends ACore {    public function get\_content() {      if(!$\_GET['id\_cat']) {  echo 'Не правильные данные для вывода статьи';  }  else {  $id\_cat = (int)$\_GET['id\_cat'];  if(!$id\_cat) {  echo 'Не правильные данные для вывода статьи';  }  else {  $result = $this->m->get\_cat($id\_cat);  return $result;  }  }  }  }  ?> |

Как Вы видите никаких запросов к базе данных и кода внешнего вида.

### Создание шаблонов

Шаблоны – это файлы с кодом, которые отвечают за внешний вид скрипта. Весь код следующих файлов взят из исходников CMS, то есть из трех контроллеров main, menu и category и класса ACore (простое копирование HTML кода). Итак, главный файл шаблона, который собирает в единое целое все части шаблона:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?  include "header.php";  include "left\_bar.php";  include "menu\_top.php";  include $tpl.".php";  include "footer.php";  ?> |

Обратите внимание центральная часть – это динамическая часть и ее шаблон формируется при помощи переменной $tpl – то есть это имя загруженного в данный момент контроллера. Файл шапки остается без изменений – его попросту копируем в папку tpl (header.php). Все фалы шаблона содержатся в папке tpl! Шаблон левой колонки (файл left\_bar.php):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <div class="quick-bg">  <div id="spacer" style="margin-bottom:15px;">  <div id="rc-bg">Menu</div>  </div>  <?php foreach($left\_bar as $row) :?>  <div class='quick-links'>  » <a href='?option=category&id\_cat=<?php echo $row['id\_category']?>'><?php echo $row['name\_category']?></a>  </div>  <?php endforeach;?>  </div> |

### Шаблон верхнего горизонтального меню (файл menu\_top.php):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <div id="mainarea">  <div class="heading">    <div class="toplinks" style="padding-left:30px;">  <a href="?option=main">Главная</a></div>  <div class="sap2">::</div>  <?php $i = 1;?>  <?php foreach($menu\_top as $item) :?>  <div class='toplinks'><a href='?option=menu&id\_menu=<?php echo $item['id\_menu']?>'><?php echo $item['name\_menu']?></a></div>    <?php if($i != count($menu\_top)) :?>  <div class='sap2'>::</div>  <?php endif;?>  <?php $i++;?>  <?php endforeach; ?>  </div> |

### Шаблон футера (файл footer.php):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <div id='bottom'>  <div class="toplinks" style="padding-left:127px;">  <a href="?option=main">Главная</a></div>  <div class="sap2">::</div>  <?php $i = 1;?>  <?php foreach($menu\_top as $item) :?>  <div class='toplinks'><a href='?option=menu&id\_menu=<?php echo $item['id\_menu']?>'><?php echo $item['name\_menu']?></a></div>    <?php if($i != count($menu\_top)) :?>  <div class='sap2'>::</div>  <?php endif;?>  <?php $i++;?>  <?php endforeach; ?>  </div>  <div class="copy"><span class="style1"> Copyright 2010 Название сайта </span>    </div>  </div>  </center></body></html> |

### И три динамических шаблона.

#### Шаблон главной страницы (main.php):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <div id="main">    <?php foreach($content as $row) :?>  <div style='margin:10px;border-bottom:2px solid #c2c2c2'>  <p style='font-size:18px'><?php echo $row['title']?></p>  <p><?php echo $row['date']?></p>  <p><img style='margin-right:5px' width='150px' align='left' src='<?php echo $row['img\_src']?>'><?php echo $row['discription']?></p>  <p style='color:red'><a href='?option=view&id\_text=<?php echo $row['id']?>'>Читать далее...</a></p>    </div>  <?php endforeach;?>  </div>  </div> |

#### Шаблон страницы категорий (category.php):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <div id="main">    <?php foreach($content as $row) :?>  <div style='margin:10px;border-bottom:2px solid #c2c2c2'>  <p style='font-size:18px'><?php echo $row['title'];?></p>  <p><?php echo $row['date'];?></p>  <p><img style='margin-right:5px' width='150px' align='left' src='<?php echo $row['img\_src'];?>'><?php echo $row['discription'];?></p>  <p style='color:red'><a href='?option=view&id\_text=<?php echo $row['id'];?>'>Читать далее...</a></p>    </div>    <?php endforeach;?>  </div>  </div> |

#### Шаблон для вывода страниц верхнего меню (menu.php):

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <div id="main">    <p style='font-size:18px'><?php echo $content['name\_menu'];?></p>  <p><?php echo $content['text\_menu'];?></p>    </div> |

Как Вы видите, шаблоны содержат практически чистый html, за исключением небольшого количества PHP кода, необходимого для работы с данными, переданными в данный шаблон. Вот теперь наша мини CMS построена по очень гибкой и удобной для редактирования структуре.

Согласитесь, что шаблоны настолько просты, что их может редактировать человек, не знающий PHP, а это большой плюс. Если Вам понравилась данная идея, то доработайте CMS до конца и полностью переведите ее логическую структуру на структуру шаблона MVC.

На этом данный урок завершен. Всего Вам доброго и удачного Вам кодирования!!!



Бесплатный курс по PHP программированию

Освойте курс и узнайте, как создать динамичный сайт на PHP и MySQL с полного нуля, используя модель MVC

**В курсе 39 уроков | 15 часов видео | исходники для каждого урока**

[Получить курс сейчас!](https://webformyself.com/minikurs/secretphp/free.html?utm_medium=systema&utm_source=adsinsingle3&utm_campaign=php_secret)



## Разработка веб-приложения на PHP

Создайте веб-приложение на PHP на примере приема платежей на сайте

[Смотреть](https://webformyself.com/minikurs/php/free.html?utm_medium=systema&utm_source=popupbanner&utm_campaign=php_born)

**Метки:** [MVC](https://webformyself.com/tag/mvc/), [шаблоны проектирования](https://webformyself.com/tag/shablony-proektirovaniya/)

* [иконки время](http://webformyself.com/50-svezhix-naborov-ikonok-dlya-dizajna/)
* [jquery stop animate](http://webformyself.com/animaciya-s-jquery-7-shagovaya-programma/)
* [jquery для начинающих в примерах](http://webformyself.com/besplatnaya-kniga-jquery-dlya-nachinayushhix/)

### Похожие статьи:

* [[](https://webformyself.com/minikurs/yii2/free.html?utm_medium=systema&utm_source=similarpost&utm_campaign=yii2_born)Yii2 – наиболее востребованный РНР-фрей...](https://webformyself.com/minikurs/yii2/free.html?utm_medium=systema&utm_source=similarpost&utm_campaign=yii2_born)
* [[](https://webformyself.com/avtomaticheskoe-kopirovanie-bazy-dannyx-mysql-na-php/)Автоматическое копирование базы данны...](https://webformyself.com/avtomaticheskoe-kopirovanie-bazy-dannyx-mysql-na-php/)
* [[](https://webformyself.com/php-avtorizaciya-na-sajte-chtoby-polzovateli-ne-merzli-u-vxoda/)PHP авторизация на сайте - чтобы поль...](https://webformyself.com/php-avtorizaciya-na-sajte-chtoby-polzovateli-ne-merzli-u-vxoda/)
* [[](https://webformyself.com/zagruzka-fajlov-na-server-php-s-pomoshhyu-samogo-prostogo-funkcionala/)Загрузка файлов на сервер PHP с помощ...](https://webformyself.com/zagruzka-fajlov-na-server-php-s-pomoshhyu-samogo-prostogo-funkcionala/)
* [[](https://webformyself.com/metod-post-v-php-kak-glavnoe-sredstvo-peredachi-dannyx-na-server/)Метод POST в PHP как главное средство...](https://webformyself.com/metod-post-v-php-kak-glavnoe-sredstvo-peredachi-dannyx-na-server/)
* [[](https://webformyself.com/carbon-prostoj-i-funkcionalnyj-instrument-php-dlya-raboty-s-datoj-i-vremenem/)Carbon - простой и функциональный инс...](https://webformyself.com/carbon-prostoj-i-funkcionalnyj-instrument-php-dlya-raboty-s-datoj-i-vremenem/)

------------------------

# MVC для веб: проще некуда

<https://habr.com/ru/post/181772/>

[vnaz](https://habr.com/ru/users/vnaz/)1 июня 2013 в 19:08 [Разработка веб-сайтов](https://habr.com/ru/hub/webdev/), [PHP](https://habr.com/ru/hub/php/) [Из песочницы](https://habr.com/ru/sandbox/)

В этой статье мы рассмотрим архитектурный паттерн MVC (Model, View, Controller) в применении к веб-разработке, «в чистом виде», без привлечения каких-то дополнительных, не относящихся к MVC структур и паттернов. Мы будем продвигаться от простого к сложному, поэтому пока не станем рассматривать, например, дальнейшее развитие MVC – паттерн HMVC (Hierarchical MVC). Хотя HMVC, несомненно, намного более интересен для разработки веб-приложений, но его применение не отменяет необходимости понимания «обычного» MVC.

Статья в Википедии (а именно туда, видимо, чаще всего попадают те, кто только начинает изучать MVC), изобилует неточностями и туманными формулировками, само определение, по сути, является неверным, а приведенная [схема](http://upload.wikimedia.org/wikipedia/commons/f/fd/MVC-Process.png) просто напросто не соответствует той, которая применяется в веб вообще и при разработке на PHP – в особенности.

## Определение паттерна MVC

Наиболее корректное определение паттерна MVC я обнаружил [тут](http://www.rsdn.ru/article/patterns/generic-mvc.xml):

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

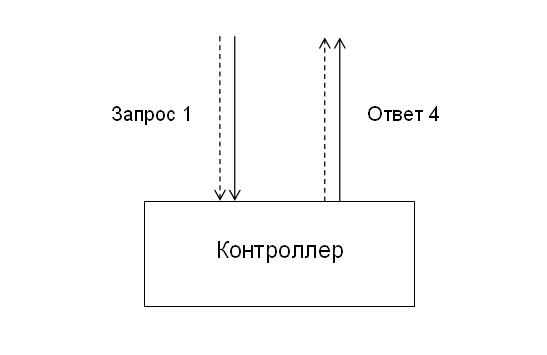
Уточним, что термин «компонент» в данном случае не имеет никакой связи с компонентами некоторых популярных CMS или фреймворков, а компоненты Битрикса, например вообще строятся из всех трёх составляющих MVC.

В приведённом определении под компонентом следует понимать некую отдельную часть кода, каждая из которых играет одну из ролей Контроллера, Модели или Представления, где Модель служит для извлечения и манипуляций данными приложения, Представление отвечает за видимое пользователю отображение этих данных (то есть, в применении к вебу, формирует отдаваемый сервером браузеру пользователя HTML/CSS), а Контроллер управляет всем этим оркестром.

## Классическая схема веб-приложения

Давайте рассмотрим классическую схему веб-приложения:

#### Рисунок 1



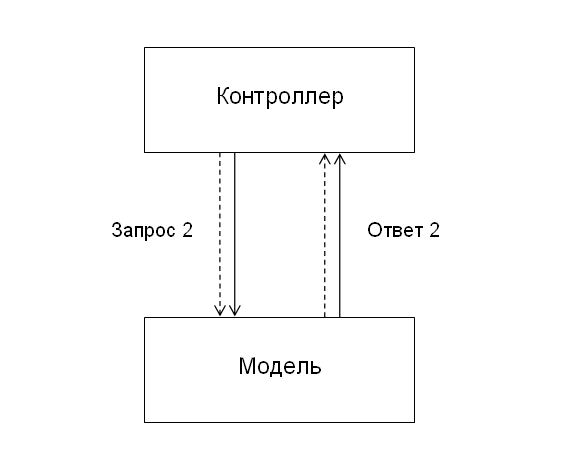
На этом и последующем рисунках пунктирными линиями показана управляющая информация (такая, например, как ID запрашиваемой записи блога или товара в магазине), а сплошными – собственно данные приложения (которые могут храниться в БД, или в виде файлов на диске, или даже, возможно, в оперативной памяти – этот вопрос лежит за пределами паттерна MVC). В применении к вебу запрос и ответ ходят по HTTP, поэтому можно условно считать, что на этом рисунке пунктиром обозначены заголовки HTTP-запроса и ответа, а сплошными линиями – их тела.

Получив Запрос 1, Контроллер его анализирует, и в зависимости от результатов обработки может выдать следующие варианты ответа (почему ответ имеет номер 4, станет понятно из следующих рисунков):

1. Сразу выдать ответ об ошибке (например, при запросе несуществующей страницы отдать только HTTP-заголовок «404 Not found»)

2. Если поступивший Запрос 1 признан корректным, то, в зависимости от того, является он запросом на просмотр или на модификацию данных, Контроллер вызывает соответствующий метод Модели, такой как **Save** или **Load** (Запрос 2 на Рис.2).

#### Рисунок 2



Важное замечание: концепция MVC не только не привязана к какому-то конкретному языку программирования, она также не привязана и к используемой парадигме программирования. То есть, вы вполне можете проектировать своё приложение по MVC, при этом не применяя ООП, и спроектировать Модель Товар для интернет-магазина таким образом:

<?

mixed Product\_Load (int $id) { ... }

*// возвращает ассоциативный массив с данными о Товаре либо FALSE при неудаче*

bool Product\_Save (array $data) { ... }

*// возвращает TRUE при удачном сохранении данных $data, либо FALSE при неудаче*

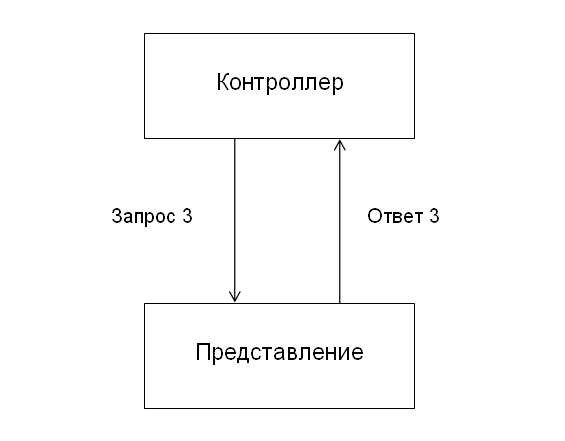
?>

Итак, в зависимости от полученного от Модели Ответа 2 Контроллер решает, какое из Представлений вызвать для формирования итогового ответа на изначальный Запрос 1:

3.1. В случае неудачи – Представление для сообщения об ошибке

3.2. В случае успеха – Представление для отображения запрашиваемых данных либо сообщения об их успешном сохранении (если Запрос 1 был на изменение данных).

#### Рисунок 3



Вопрос о том, кто должен проверять на валидность и права доступа входные данные (Контроллер или Модель), является предметом достаточно многочисленных споров, поскольку паттерн MVC не описывает таких деталей. Это значит, что в этом вопросе выбор за вами (или за вас его сделали авторы вашего любимого фрейvворка или CMS).

Мы в своей практике придерживаемся такого подхода:

Контроллер проверяет входные данные на предмет «общей» (т.е. независящей от конкретного запроса) корректности, соответствие требованиям Модели к валидности сохраняемых данных проверяет соответствующий метод Модели, а права доступа – метод **Access** отдельного класса **User**.

Для вызова Представления в PHP иногда проектируется специальный класс (а то и несколько классов), например, **View** (это часто встречается в описаниях MVC в реализации того или иного фреймворка), однако это не является требованием MVC.

Также файлы Представлений часто называют шаблонами, а при использовании так называемых шаблонизаторов роль Представления играет сам шаблонизатор, а шаблоны (т.е. файлы, содержащие непосредственно HTML-разметку) в некоторых фреймворках называют **layouts**.

Не вполне понятен предыдущий абзац? Забейте, поскольку у нас каждое Представление – это просто отдельный PHP-файл, а сам PHP устроен так, что в случае, если мы используем для выполнения Запроса 3 инструкцию include, Ответ 3 и Ответ 4 (помните, что это ответ на Запрос 1?) отдаются браузеру автоматически, средствами самого PHP.

### Пример

Давайте рассмотрим пример.

У нас есть два варианта Представления (шаблоны), в которых

**<!-- HTML.header -->** будет означать HTML-код, предворяющий в формируемом веб-документе основной контент (т.е. содержит тег doctype, контейнер head, код шапки страницы, и т.п.), а **<!-- HTML.footer -->** – примерно то же, только для подвала страницы.

#### Листинг 1. Шаблон product.tpl.php отображает данные о Товаре (которые к моменту его вызова уже содержит объект $product):

<!-- HTML.header -->

<h1><?=$product->Title;?></h1>

<p>Цена:<b class="price"><?=$product->Price;?></b></p>

<p class="description"><?=$product->Description;?></p>

<!-- HTML.footer -->

#### Листинг 2. Шаблон error.tpl.php отображает сообщение об ошибке (которое содержится в переменной $error):

<!-- HTML.header -->

<h1 class="error">Ошибка: <?=$error;?></h1>

<!-- HTML.footer -->

#### Листинг 3. Контроллер product.php, служащий для отобоажения Товара, будет выглядеть примерно так:

<?

include 'product.class.php'; *// в этом файле декларируются методы Модели Товар*

*// определение этой функции в контроллере, конечно, неправильно*

*// в данном случае она здесь только для наглядности*

function Error ($error) {

*// выводит сообщение об ошибке и завершает работу контроллера, примерно так:*

header('Правильный статус ошибки, например, 400 или 404');

$error = 'Соответствующее ошибке сообщение пользователю, например, Страницы не существует';

include 'error.tpl.php'; *// шаблон для отображения ошибки*

exit;

}

if (!$id = ...) *// проверка "общей" валидности Запроса 1*

error(...);

*// проверка прав доступа*

if (!$user->Access(...))

error(403);

if (!$product = Product::Load($id)) *// Запрос 2 и анализ Ответа 2*

error('Тут скорее всего случилась ошибка БД');

include 'product.tpl.php'; *// Запрос 3 и Ответы 3 и 4*

?>

Те, кто любит красивый и оптимизированный код могут заметить, что блоки **HTML.header** и **HTML.footer** дублируются в обоих шаблонах (они же Представления) **error.tpl.php** и **product.tpl.php**, и наверняка захотят вынести их в Контроллер **product.save.php**:

<!-- HTML.header -->

<?

*// Основной код контролера из Листинга 3*

?>

<!-- HTML.footer -->

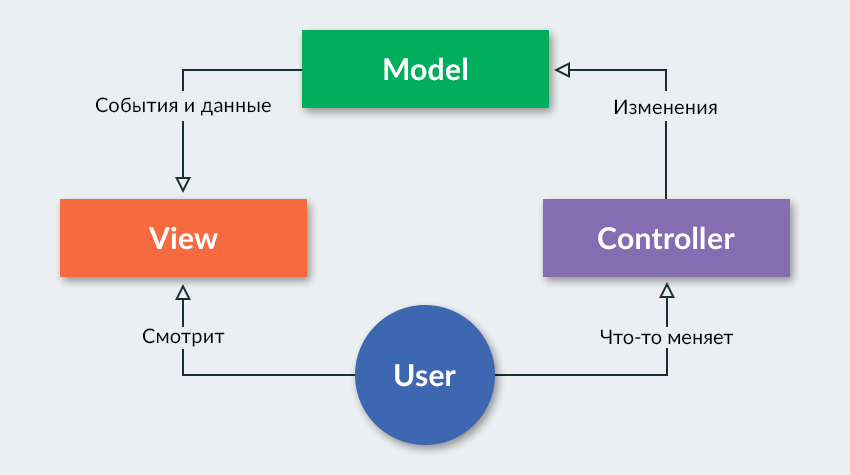
…. и нарушить таким образом **основное правило MVC** – разделяйте Контроллер, Модель и Представление.

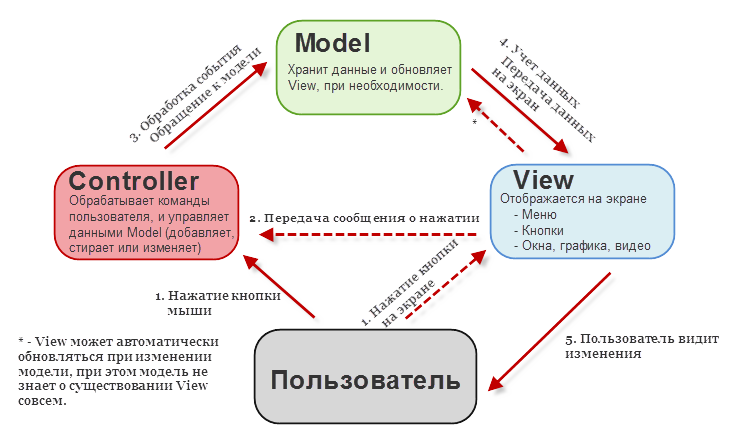
Тем не менее, дублирующийся код – однозначное Зло. Что же делать?

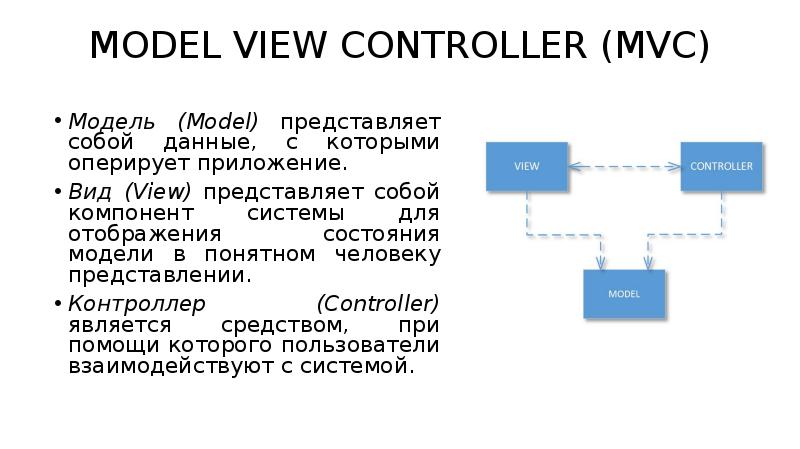
Мы должны перейти от MVC к HMVC!

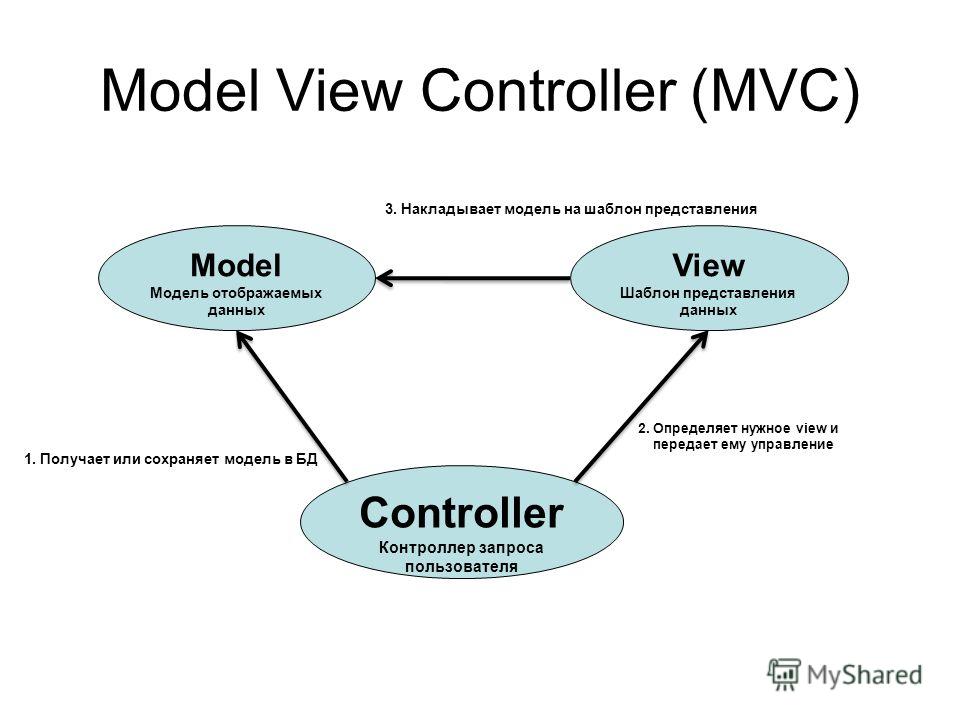
Но это – тема отдельной статьи.

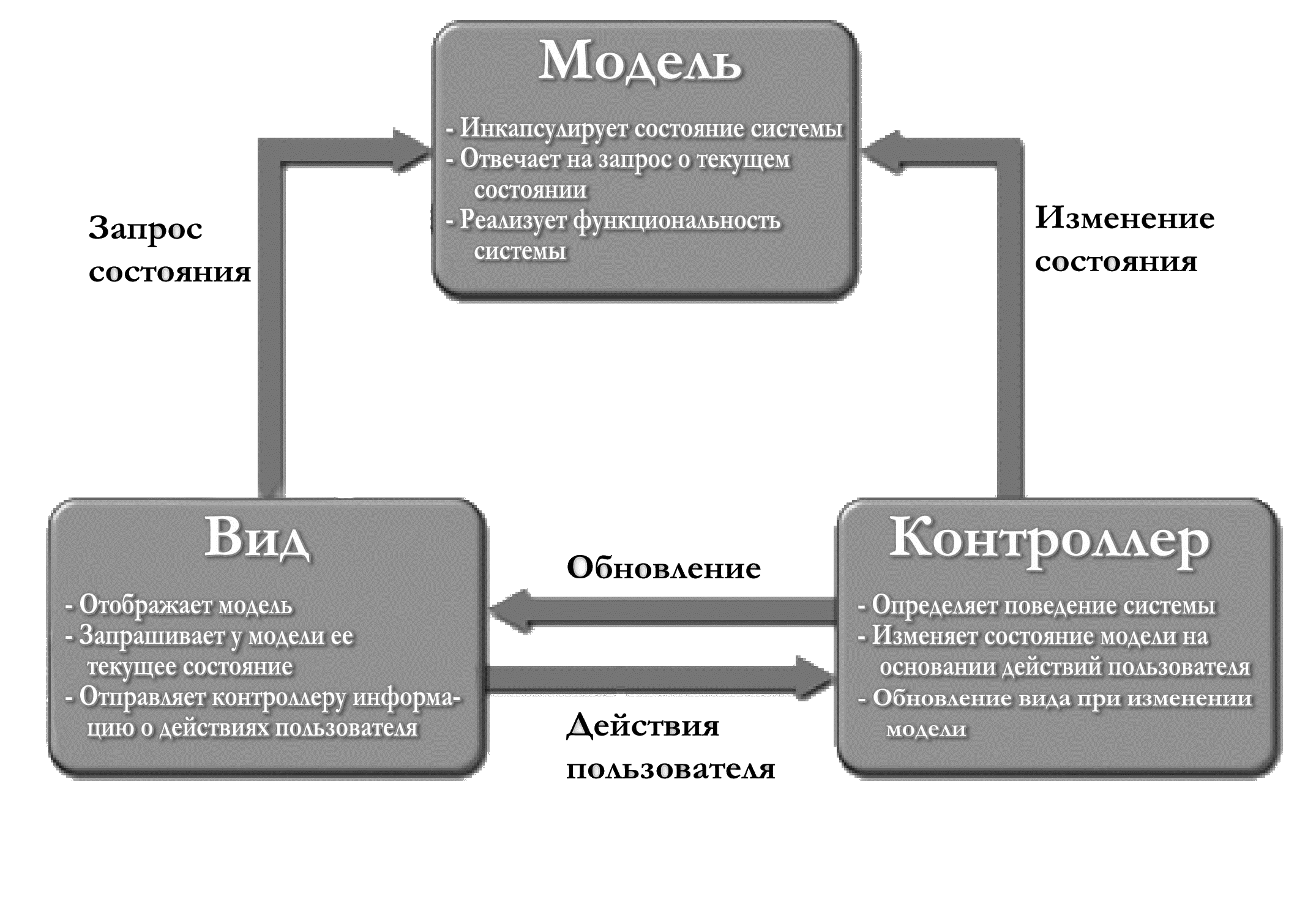
# Рисунки-иллюстрации











--------------------------