Шаблон проектирования MVC и PHP, Часть 1

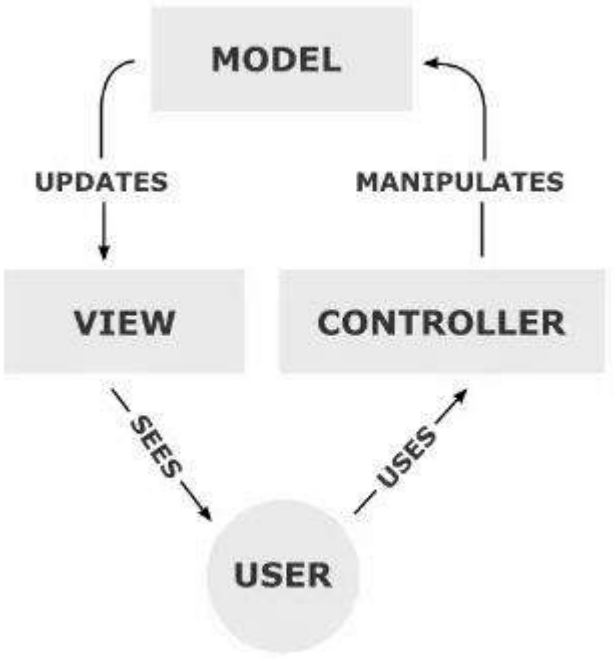
Шаблон проектирования Модель-Представление-Контроллер (MVC) — это шаблон программной архитектуры, построенный на основе сохранения представления данных отдельно от методов, которые взаимодействуют с данными.

Не смотря на то, что схема MVC была первоначально разработана для персональных компьютеров, она была адаптирована и широко используется веб-разработчиками из-за точного разграничения задач и возможности повторного использования кода. Схема стимулирует развитие модульных систем, что позволяет разработчикам быстро обновлять, добавлять или удалять функционал.

В этой статье я опишу основные принципы, а также рассмотрю определение схемы построения и простой MVC PHP пример.

Что такое MVC

Название шаблона проектирования определяется тремя его основными составляющими частями: Модель, Представление и Контроллер. Визуальное представление шаблона **MVC** выглядит, как показано на приведенной ниже диаграмме:



На рисунке показана структура одностороннего потока данных и пути его следования между различными компонентами, а также их взаимодействие.

Модель

Моделью называют постоянное хранилище данных, используемых во всей структуре. Она должна обеспечивать доступ к данным для их просмотра, отбора или записи. В общей структуре «Модель» является мостом между компонентами «Представление» и «Контроллер».

При этом «Модель» не имеет никакой связи или информации о том, что происходит с данными, когда они передаются компонентам «Представление» или «Контроллер». Единственная задача «Модели» — обработка данных в постоянном хранилище, поиск и подготовка данных, передаваемых другим составляющим MVC.

«Модель» должна выступать в качестве «привратника», стоящего возле хранилища данных и не задающего вопросов, но принимающего все поступающие запросы. Зачастую это наиболее сложная часть системы MVC. Компонент «Модель» — это вершина всей структуры, так как без нее невозможна связь между «Контроллером» и «Представлением».

Представление

Представление — это часть системы, в которой данным, запрашиваемым у «Модели», задается окончательный вид их вывода. В веб-приложениях, созданных на основе MVC, «Представление» — это компонент, в котором генерируется и отображается HTML-код.

Представление также перехватывает действие пользователя, которое затем передается «Контроллеру». Характерным примером этого является кнопка, генерируемая «Представлением». Когда пользователь нажимает ее, запускается действие в «Контроллере».

Существует несколько распространенных заблуждений относительно компонента «Представление». Например, многие ошибочно полагают, что «Представление» не имеет никакой связи с «Моделью», а все отображаемые данные передаются от «Контроллера». В действительности такая схема потока данных не учитывает теорию, лежащую в основе MVC архитектуры. В своей статье Фабио Чеваско описывает этот некорректный подход на примере одного из нетрадиционных MVC PHP фреймворков:

«Чтобы правильно применять архитектуру MVC, между «Моделью» и «Представлением» не должно быть никакого взаимодействия: вся логика обрабатывается «Контроллером».

Кроме этого определение «Представления» как файла шаблона также является неточным. Но это не вина одного человека, а результат множества ошибок различных разработчиков, которые приводят общему заблуждению. После чего они неправильно объясняют это другим. На самом деле «Представление» это намного больше, чем просто шаблон. Но современные MVC-ориентированные фреймворки до такой степени впитали этот подход, что никто уже не заботится о том, поддерживается ли верная структура MVC или нет.

Компоненту «Представление» никогда не передаются данные непосредственно «Контроллером». Между «Представлением» и «Контроллером» нет прямой связи — они соединяются с помощью «Модели».

Контроллер

Его задача заключается в обработке данных, которые пользователь вводит и обновлении «Модели». Это единственная часть схемы, для которой необходимо взаимодействие пользователя.

«Контроллер» можно определить, как сборщик информации, которая затем передается в «Модель» с последующей организацией для хранения. Он не содержит никакой другой логики, кроме

необходимости собрать входящие данные. «Контроллер» также подключается только к одному «Представлению» и одной «Модели». Это создает систему с односторонним потоком данных с одним входом и одним выходом в точках обмена данными.

«Контроллер» получает задачи на выполнение только когда пользователь взаимодействует с «Представлением», и каждая функция зависит от взаимодействия пользователя с «Представлением». Наиболее распространенная ошибка разработчиков заключается в том, что они путают «Контроллер» со шлюзом, поэтому присваивают ему функции и задачи, которые относятся к «Представлению».

Также распространенной ошибкой является наделение «Контроллера» функциями, которые отвечают только за обработку и передачу данных из «Модели» в «Представление». Но согласно структуре MVC паттерна это взаимодействие должно осуществляться между «Моделью» и «Представлением».

MVC в PHP

Напишем на PHP веб-приложение, архитектура которого основана MVC. Давайте начнем с примера каркаса:

```php
<?php
class Model
{
    public $string;
    public function __construct(){
        $this->string = "MVC + PHP = Awesome!";
    }
}
<?php
class View
{
    private $model;
    private $controller;
    public function __construct($controller,$model) {
        $this->controller = $controller;
        $this->model = $model;
    }

    public function output(){
        return "<p>" . $this->model->string . "</p>";
    }
}
<?php
class Controller
{
    private $model;
    public function __construct($model) {
        $this->model = $model;
    }
}
```

У нас есть проект с несколькими основными классами для каждой части шаблона. Теперь нужно настроить взаимосвязь между ними:

```php
<?php
$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);
echo $view->output();
```

В приведенном выше примере PHP MVC нет никакого специфического функционала для контроллера, потому что в приложении не определены взаимодействия пользователя. Представление содержит весь функционал, так как наш пример предназначен лишь для демонстрации.

Давайте расширим пример, чтобы показать, как мы будем добавлять функционал контроллера, тем самым добавив в приложение взаимодействия

```php
<?php
class Model
{
    public $string;
    public function __construct(){
        $this->string = “MVC + PHP = Awesome, click here!”;
    }
}
<?php
class View
{
    private $model;
    private $controller;
    public function __construct($controller,$model) {
        $this->controller = $controller;
        $this->model = $model;
    }
    public function output() {
        return '<p><a href="mvc.php?action=clicked"' . $this->model->string . "</a></p>";
    }
}
<?php
class Controller
{
    private $model;
    public function __construct($model){
        $this->model = $model;
    }
    public function clicked() {
        $this->model->string = “Updated Data, thanks to MVC and PHP!”;
    }
}
```

Мы расширили программу базовым функционалом. Настройка взаимодействий между компонентами теперь выглядит следующим образом:

Запустите код и при нажатии на ссылку вы увидите строку для изменения данных.

```php
<?php
$model = new Model();
$controller = new Controller($model);
$view = new View($controller, $model);
if (isset($_GET['action']) && !empty($_GET['action'])) {
    $controller->{$_GET['action']}();
}
echo $view->output();
```

Заключение

Мы рассмотрели базовую теорию модели MVC, и реализовали на его основе простое приложение. В следующей статье этой серии мы рассмотрим несколько конкретных случаев, с которыми вы можете столкнуться при создании веб-приложений на PHP.