



ASP.NET MVC

«Чудесно праздновать свой успех, но более важным является умение выносить уроки из своих провалов.»

© Билл Гейтс




Архитектурный шаблон MVC

С помощью модели MVC можно создавать приложения, которые удобнее тестировать и обновлять по сравнению с традиционными монолитными приложениями.

Model (Модели): Классы модели используют логику проверки, которая позволяет применять бизнес-правила к этим данным. Как правило, объекты модели извлекают и сохраняют состояние модели в базе данных.

View (Представления): компоненты, которые формируют пользовательский интерфейс приложения. Как правило, в пользовательском интерфейсе отображаются данные модели.

Controller (Контроллеры): обрабатывают запросы браузера, получают данные модели, вызывают шаблоны представления вызовов, которые возвращают ответ.



Шаблон MVC далеко не нов, но в наши дни он завоевал огромную популярность в качестве шаблона для веб-приложений по перечисленным ниже причинам:

- Взаимодействие пользователя с приложением MVC осуществляется в соответствии с естественным циклом: пользователь предпринимает действие, в ответ на которое приложение изменяет свою модель данных и доставляет обновленное представление пользователю. Затем цикл повторяется. Это хорошо укладывается в схему веб-приложений, предоставляемых в виде последовательностей запросов и ответов HTTP.
- Веб-приложения, нуждающиеся в комбинировании нескольких технологий (например, баз данных, HTML-разметки и исполняемого кода), обычно разделяются на ряд слоев или уровней. Полученные в результате шаблоны естественным образом вписываются в концепции MVC.



Преимущества

- Расширяемость
- Жесткий контроль над HTML и HTTP
- Тестируемость
- Мощная система маршрутизации
- Открытый код

Реализация MVC в ASP.NET

В инфраструктуре MVC контроллеры - это классы C#, обычно производные от класса **System.Web.Mvc.Controller**. Каждый открытый метод в классе, производном от Controller, является методом действия который посредством системы маршрутизации ASP.NET ассоциируется с конфигурируемым URL. Когда запрос отправляется по URL, связанному с методом действия, операторы в классе контроллера выполняются, чтобы провести некоторую операцию над моделью предметной области и затем выбрать представление для отображения клиенту.





Контроллеры

Каждый запрос, поступающий в приложение, обрабатывается контроллером. ***Контроллер*** может обрабатывать запрос произвольным образом до тех пор, пока он не пересекает границу ответственности модели и представления. Это означает, что контроллеры не должны содержать или сохранять данные, равно как не генерировать пользовательские интерфейсы.



Razor

Механизм визуализации обрабатывает код ASP.NET и ищет инструкции, которые обычно вставляют динамическое содержимое в вывод, отправляемый браузеру, а Razor - это название механизма визуализации MVC Framework.

Razor Pages делает создание кодов сценариев для страниц проще и эффективнее по сравнению с использованием контроллеров и представлений.

Все Страницы Razor попадают в папку *Pages* в корне проекта ASP.NET Core.




Маршрутизация URL

До появления инфраструктуры MVC Framework платформа ASP.NET предполагала наличие прямых отношений между запрашиваемыми URL и файлами на жестком диске сервера. Работа сервера заключалась в получении запроса от браузера и доставке вывода из соответствующего файла.

Для обработки URL в MVC платформа ASP.NET использует *систему маршрутизации*. Она обеспечивает выполнение двух функций:


- Исследование входящих URL и выяснение, для каких контроллеров и действий они предназначены. Как и следовало ожидать, это то, что система маршрутизации должна делать при получении клиентского запроса.
- Генерация исходящих URL. Это URL, которые появляются в HTML-разметке, визуализированной из представлений, так что специфическое действие может быть инициировано, когда пользователь производит щелчок на ссылке (в этот момент ссылка снова становится входящим URL).



URL могут быть разбиты на сегменты. **Сегменты** - это части URL за исключением имени хоста и строки запроса, которые отделяются друг от друга символом /.

<http://mysite.com/Admin/Index>

Первый сегмент содержит слово Admin, а второй - слово Index. В глазах человека совершенно очевидно, что первый сегмент имеет отношение к контроллеру, а второй - к действию. Однако, естественно, это отношение должно быть выражено понятным для системы маршрутизации образом.



При обработке входящего запроса задача системы маршрутизации заключается в сопоставлении запрошенного URL с шаблоном и в последующем извлечении из URL значений для переменных сегментов, определенных в шаблоне.

По умолчанию шаблон URL будет соответствовать любому URL, который имеет подходящее количество сегментов. Например, шаблон "{controller}/{action}" будет соответствовать любому URL с двумя сегментами, как показано ниже:

| Переменные сегментов | URL запроса |
|---|--|
| http://localhost:64399/Admin/Index | controller = Admin, action = Index |
| http://localhost:64399/Index/Admin | controller = Index, action = Admin |
| http://localhost:64399/Apples/Oranges | controller = Apples, action = Oranges |
| http://localhost:64399/Admin | Соответствия нет - сегментов слишком мало |
| http://localhost:64399/Admin/Index/Football | Соответствия нет - сегментов слишком много |

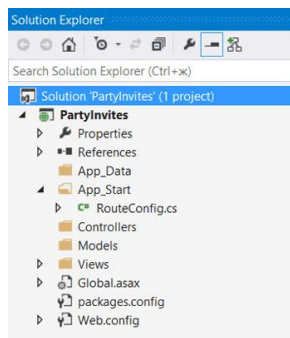


Создание проекта

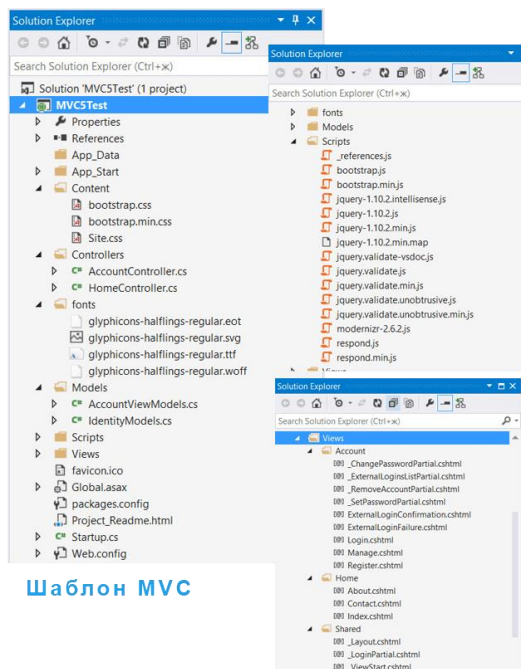
Когда вы впервые создаете новый проект MVC Framework, у вас есть на выбор две базовых отправных точки: **шаблон Empty (Пустой)** и **шаблон MVC**.

Вариант Empty проекта содержит только базовые ссылки, требуемые для MVC Framework, и базовую структуру папок. Шаблон MVC добавляет довольно много содержимого.

Содержимое проекта



Шаблон Empty



Шаблон MVC



Структура

| Каталог | Назначение |
|---------------------------|---|
| <code>/Controllers</code> | Расположение классов контроллера, обрабатывающих запросы URL-адресов |
| <code>/Models</code> | Где вы помещаете классы, представляющие данные и управляющие ими |
| <code>/Views</code> | Расположение файлов шаблонов пользовательского интерфейса, которые отвечают за отрисовку выходных данных |
| <code>/Scripts</code> | Расположение файлов и скриптов библиотеки JavaScript (.js) |
| <code>/Content</code> | Расположение CSS и файлов изображений, а также другое содержимое, отличное от динамического или не javascript |
| <code>/App_Data</code> | Там, где хранятся файлы данных, которые требуется читать и записывать. |



Спасибо за внимание!