НОВОСТНОЙ ПОРТАЛ ДЛЯ ПРОГРАММИСТОВ «ITPROGER»

Пресман А.Г.

Институт информационных технологий БГУИР,
г. Минск, Республика Беларусь

Образцова О.Н. – и.о. зав. кафедрой ИСиТ, к.т.н., доцент

В работе рассматривается паттерн программирования MVC и приводится пример архитектуры приложения «Новостной портал для программистов itproger», реализующее данный паттерн на языке C# с использованием ASP.NET MVC 5.

Model-View-Controller (MVC). Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

– к одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы;

– не затрагивая реализацию видов, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) – для этого достаточно использовать другой контроллер;

– ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (модели), вообще не будут осведомлены о том, какое представление будет использоваться.

Схема работы паттерна MVC:

1) При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.

2) Выполняется действие index фронт-контроллера, которое генерирует представление главной страницы.

3) Представление отображается пользователю.

4) После того, как приложение получит запрос от пользователя, создается экземпляр запрошенного контроллера и вызывается указанное действие.

5) В этом действии вызываются методы модели, изменяющие ее.

6) Генерируется представление (или же представление оповещается об обновлении модели).

7) Представление запрашивает данные для отображения.
8) Модель возвращает запрошенные данные.
9) Представление отображает результаты пользователю.

Графическая модель паттерна MVC изображена на рисунке 1.

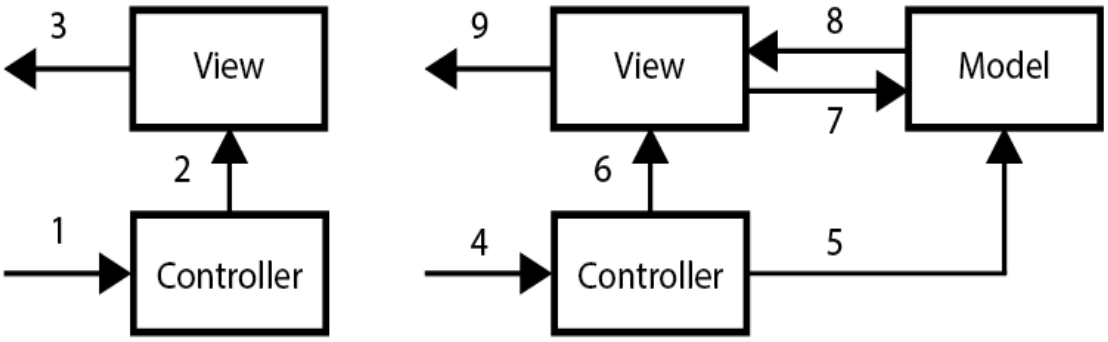

Рисунок 1 – Графическая модель паттерна MVC.

На данный момент существует несколько паттернов производных от MVC:

– MVP (Model-View-Presenter) - шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса. Элемент Presenter в данном шаблоне берёт на себя функциональность посредника (аналогично контроллеру в MVC) и отвечает за управление событиями пользовательского интерфейса (например, использование мыши) так же, как в других шаблонах обычно отвечает представление;

– HMVC (Hierarchical Model View Controller). Приложение представляет иерархию независимых друг от друга MVC триад. При этом, каждая триада может напрямую обратиться к контроллеру другой триады. Такой подход позволяет решить некоторые проблемы масштабируемости приложений, имеющих классическую MVC-архитектуру, уменьшить зависимость между различными частями приложения, облегчить дальнейшую поддержку и повторное использование кода;

– MVVM (Model-View-ViewModel). Первоначально MVVM был описан для Silverlight и имеет преимущества для сложных интерфейсов с определенной логикой, которая отличается от логики приложения. MVVM отличается более «тесной» связью между Моделью и Представлением посредством слоя Представление-Модель, который синхронизирует данные как при событии на стороне Модели, так и на стороне Представления. В MVC логика зашита в Модели, ее можно также помещать в Контроллер, но это справедливо подвергается критике (Stupid Fat Controller). В MVVM, напротив, логика помещается в «промежуточный» слой ViewModel.

Стоит отметить, что MVC часто трактуют просто как разделение трех уровней приложения, и никак не регламентируют связи между ними. Поэтому, довольно часто, встречаются диаграммы (выше была приведена одна из таких), на которых Модель и Представление связаны стрелками, хотя очевидно, что таким образом теряются полезные свойства масштабируемости при использовании разных Представлений и иерархичность Контроллеров.

Для начала нужно создать пустое решение в Visual Studio. Далее добавляем в него три проекта: Domain, WebUI, UnitTests. Domain - содержит сущности и логику предметной области; настраивается на обеспечение постоянства посредством хранилища, которое создано с помощью инфраструктуры Entity Framework. WebUI - Содержит контроллеры и представления; выступает в качестве пользовательского интерфейса для приложения. UnitTests - Содержит модульные тесты для других двух проектов.

Проект Domain состоит из следующих частей:

– Abstract – содержит интерфейсы для работы с сущностями (просмотр, удаление, сохранение);

– Concrete – содержит классы, которые ассоциируют модели с базой данных (БД);

– Entities – содержит сущности (модели).

Проект WebUI состоит из:

– AppStart – содержит настройки переадресации сайта;

– Content – подключаемые стили и скрипты;

– Controllers – содержит контроллеры;

– Fonts – шрифты;

– Infrastructure – содержит классы для работы с формами;

– Views – содержит шаблоны (представления), которые отображаются в браузере пользователя.

Проект UnitTests содержит юнит-тесты.

Юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны. Этот тип тестирования обычно выполняется программистами.

Список использованных источников:

1. Мэйерс С. Эффективное использование C#. 55 верных способов улучшить структуру и код ваших программ- - М.: ДМК Пресс, 2014. С. – 49.
2. Саттер, Герб. Новые сложные задачи на C#. Пер. с англ. – М.: Издательский дом «Вильяме», 2015. С. – 121.
3. Стивен Прата Язык программирования C#. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб: ОО «ДиаСофтЮП», 2015. С. – 71.
4. Стивен Сандерсон - ASP.NET MVC для профессионалов – СПБ: Вильямс, 2010 – С. 557.