Microsoft ASP.NET MVC — это инфраструктура для разработки веб-приложений, построенная поверх популярной платформы.NET Framework. Инфраструктура ASP.NET MVC в основном полагается на проверенные шаблоны и приемы разработки, которые делают акцент на слабо связанной архитектуре приложения и хорошо поддающемся сопровождению коде. В 2008 г. выпустили первую версию ASP.NET MVC. Представляя собой полное отступление от подхода WebForms , ASP.NET MVC отбрасывает архитектуру на основе страниц, а вместо нее полагается на архитектуру модель-представление-контроллер (Model-View-Controller). Шаблон «модель-представление-контроллер» — это архитектурный шаблон, который поддерживает строгую изоляцию между отдельными частями приложения. Такая изоляция более известна как разделение ответственности или, если пользоваться более общими терминами, как слабое связывание. Проектирование архитектуры приложений в слабо связанном стиле привносит ряд как краткосрочных, так и долгосрочных преимуществ:

– Разработка. Отдельные компоненты не зависят напрямую от других компонентов, а это означает, что их более просто разрабатывать в изоляции. Компоненты также легко заменять или замещать, предотвращая возникновение сложностей в ситуациях, когда один компонент влияет на разработку других компонентов, с которыми он может взаимодействовать.

Тестируемость. Слабое связывание компонентов позволяет применять тестовые реализации на месте производственных версий компонентов. Скажем, за счет замены компонента, выполняющего обращения к базе данных, компонентом, который просто возвращает статические данные, можно избежать взаимодействия с физической базой данных на этапе разработки. Способность компонентов легко меняться местами с пробными представлениями существенно упрощает процесс тестирования, который радикально увеличивает надежность системы с течением времени.

– Сопровождение. Изоляция компонентов означает, что изменения в логике обычно изолируются в небольшом числе компонентов — очень часто в одном. С учетом того, что степень влияния изменения обычно зависит от его масштаба, модификация меньшего количества компонентов — это однозначно хорошо. Модель представляет основную бизнес-логику и данные. Модель инкапсулирует свойства и поведение сущности предметной области и открывает свойства, которые описывают эту сущность. Представление отвечает за преобразование модели или моделей в визуальную презентацию. В web-приложениях это чаще всего означает генерацию HTML-разметки для визуализации в браузере пользователя, хотя представления могут проявляться во многих формах. Следуя принципу разделения ответственности, представления должны концентрироваться только на отображении данных и не могут содержать какую-либо бизнес-логику — бизнес-логика остается в модели, которая должна предоставлять представлению все, что необходимо. Контроллер управляет логикой приложения и действует в качестве координатора между представлением и моделью. Контроллеры получают пользовательский ввод через представление и затем взаимодействуют с моделью для выполнения специфичных действий, передавая результаты обратно представлению.

Маршрутизация ASP.NET — это система сопоставления с шаблоном. Во время запуска приложение регистрирует один или большее число шаблонов в таблице маршрутов инфраструктуры, тем самым сообщая системе маршрутизации о том, как поступать с любыми запросами, которые соответствуют этим шаблонам. Когда механизм маршрутизации получает запрос во время выполнения, он сопоставляет URL этого запроса с зарегистрированными шаблонами URL. Если механизм маршрутизации находит совпадающий шаблон в таблице маршрутизации, он перенаправляет запрос соответствующему обработчику для этого запроса. В противном случае, если URL запроса не совпадает ни с одним из зарегистрированных шаблонов маршрутов, механизм маршрутизации указывает на то, что ему не удалось определить, как обрабатывать запрос, возвращая код состояния HTTP 404. Маршруты ASP.NET MVC отвечают за определение того, какой метод контроллера (по-другому известный как действие контроллера) выполнять для данного URL. С маршрутами связаны следующие свойства: уникальное имя (имя может использоваться в качестве специфичной ссылки на конкретный маршрут); шаблон URL (простой синтаксис шаблонов, который разбирает совпадающие URL на значащие сегменты); стандартные значения (необязательный набор стандартных значений для сегментов, определенных в шаблоне URL); ограничения (набор ограничений для применения к шаблону URL с целью более узкого определения URL, для которых он дает совпадение). Маршруты URL могут содержать больше информации, чем механизм маршрутизации способен извлечь. Тем не менее, для обработки запроса ASP.NET MVC механизм маршрутизации должен иметь возможность определять две критические порции информации: контроллер и действие. Затем механизм маршрутизации может передать эти значения исполняющей среде ASP.NET MVC для создания и выполнения указанного действия соответствующего контроллера. В контексте архитектурного шаблона MVC контроллер отвечает на пользовательский ввод (например, когда пользователь щелкает на кнопке «Сохранить») и совместно работает с уровнями моделей, представлений и доступа к данным. В приложении ASP.NET MVC контроллеры — это классы, содержащие методы, которые вызываются инфраструктурой маршрутизации для обработки запроса. Классы контроллеров не являются чем-то особенным, т. е. они не сильно по виду отличаются от других классов.NET.

На самом деле всю обработку запросов выполняют методы классов контроллеров, которые называются действиями контроллеров. Например, класс HomeController может содержать три действия: Index, About, Contact. Таким образом, учитывая стандартный шаблон маршрута {controller}/{action}/{id}, когда производится запрос URL вида /Home/About, инфраструктура маршрутизации определяет, что этот запрос должен быть обработан методом About() класса HomeController. Затем ASP.NET MVC Framework создает новый экземпляр класса HomeController и выполняет его метод About(). В этом случае метод About() передает данные представлению через свойство ViewBag, после чего сообщает ASP.NET MVC Framework о необходимости отображения представления по имени About за счет вызова метода View(), который возвращает ActionResult типа ViewResult.

Важно отметить, что работа контроллера заключается в уведомлении ASP.NET MVC Framework о том, что должно делаться следующим, но не как это должно делаться. Такое взаимодействие происходит с применением ActionResult — возвращаемых значений, который, как ожидается, предоставляет каждое действие контроллера. Например, когда контроллер решает отобразить представление, он сообщает ASP.NET MVC Framework об этом, возвращая ViewResult. Он не визуализирует само представление. Такое слабое связывание является еще одним хорошим примером реализации разделения ответственности (что делать против того, каким образом это делать).

Несмотря на тот факт, что каждое действие контроллера должно возвращать ActionResult, они редко создаются вручную. Действия контроллеров подобны любым другим методам. На самом деле действие контроллера может даже указывать параметры, которые заполняются ASP.NET MVC с использованием информации из запроса, когда он обрабатывается. Эта функциональность называется привязкой модели, и она представляет собой одну из наиболее мощных и полезных возможностей ASP.NET MVC. Привязка модели (modelbinding) — это процесс создания объектов.NET с использованием данных, отправленных браузером в HTTP-запросе. Мы полагаемся на процесс привязки модели всякий раз, когда определяем метод действия, который принимает параметр — объекты параметров создаются привязкой модели [2]. В ASP.NET MVC Framework действия контроллеров, которым необходимо отобразить HTML-разметку пользователю, возвращают экземпляр ViewResult — тип ActionResult, знающий, как визуализировать контент для ответа. Когда наступает время визуализации представления, ASP.NET MVC Framework ищет представление с использованием имени, предоставленного контроллером. Инфраструктура ASP.NET MVC полагается на соглашение, которое заключается в том, что все представления приложения хранятся внутри папки Views в корне web-сайта. Более конкретно, ASP.NET MVC ожидает обнаружить представления в папках, именованных согласно контроллеру, к которому они относятся. Если найти представление с нужным именем в папке Views контроллера не удалось, ASP.NET MVC продолжает поиск в общей папке /Views/Shared. Внутри представления присутствует HTML-разметка и код — это Razor. Razor — это синтаксис, который позволяет комбинировать код и контент в плавной и выразительной манере. Несмотря на введение нескольких символов и ключевых слов, Razor не является новым языком, он позволяет писать код, используя известные языки программирования, такие как C# или VisualBasic.NET. Интеллектуальный анализатор Razor позволяет разработчикам более выразительно представлять логику и упрощает переходы между кодом и разметкой. Хотя синтаксис Razor может отличаться от других синтаксисов разметки, он, в конечном счете, преследует ту же самую цель — визуализацию HTML. Архитектура MVC зависит от модели, представления и контроллера, которые все остаются отдельными, но работают вместе для достижения общей цели. В этом отношении задача контроллера — выступать в качестве «регулятора движения», координируя разнообразные части системы для выполнения логики приложения. Результатом такой обработки обычно является определенный вид данных, которые должны быть переданы пользователю. Тем не менее, отображение их пользователю не входит в ответственность контроллера — для этого предназначены представления. В таком случае возникает вопрос, каким образом контроллер передает эту информацию представлению? Инфраструктура ASP.NET MVC предлагает несколько способов передачи данных в рамках границ «модель-представление-контроллер»: ViewData и TempData — эти объекты являются словарями, доступными в виде свойств в контроллерах и представлениях, которые чаще всего используются. Таким образом, передача данных из контроллера в представление сводится к установке значения в контроллере. Контроллеры и представления ASP.NET MVC, открывающие свойство ViewData, также предоставляют доступ к похожему свойству по имени ViewBag. Свойство ViewBag — это просто оболочка вокруг ViewData, которая открывает словарь ViewData как объект dynamic. В дополнение к базовому поведению словаря, объект ViewData, также предлагает свойство Model, которое представляет главный объект, являющийся целью запроса. Model концептуально ничем не отличается от ViewData [“Model”], оно превращает модель в объект первого класса и считает ее более важной, чем другие данные, которые могут быть в запросе. По умолчанию свойство Model, доступное внутри представлений Razor, является динамическим (dynamic), а это означает, что получать доступ к его значениям можно, не зная точный тип. Модели обычно считаются наиболее важной частью архитектуры MVC. С технической точки зрения модель обычно состоит из нормальных классов, которые открывают доступ к данным в виде свойств и к логике в виде методов. Эти классы бывают всех видов и размеров, но наиболее распространенным примером является «модель данных» или «модель предметной области», основная работа которой заключается в управлении данными.