# ЛАБ2А

**ASP.NET Core MVC** — это платформа для создания веб-приложений и API-интерфейсов с использованием шаблона Model-View-Controller, которая является частью платформы ASP.NET Core. Упрощает организацию и создание приложений.

**Компоненты:**

**Модель (model):** описывает используемые данные, а также логику этих данных. Как правило, объекты моделей хранятся в базе данных. Модель не должна содержать логику взаимодействия с пользователем и не должна определять механизм обработки запроса. Кроме того, модель не должна содержать логику отображения данных в представлении.

**Два типа моделей:**

*модели представлений* (используются представлениями для отображения и передачи данных),

*модели домена* (логика управления данными).

**Представление (view):** отвечают за визуальную часть или пользовательский интерфейс (html-страница). Может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными.

**Контроллер (controller):** представляет центральный компонент MVC, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей.

**Попроще:**

**Контроллер** — это компонент, который обеспечивает взаимосвязь между пользователем и приложением. Он обрабатывает входящие запросы и, либо, сразу генерирует данные для ответа (отправляет данные в **представление**), либо использует **модель** для манипуляции с данными и, затем, генерирует данные для представления.

**Модель (model)** — это компонент, содержащий всю бизнес-логику приложения. Модель ничего не знает о контроллерах и представлениях в приложении, но знает всё о том, как манипулировать данными приложения — записывать данные в БД, удалять данные, производить сложные расчёты, анализ и т.д.

**Представление (view)** — это та часть приложения, которая отвечает за отображение данных пользователю. Представление получает данные от контроллера и на основании этих данных формирует готовый ответ пользователю, например, генерирует html-страницу. Представление не должно содержать логику обработки запроса пользователя (за это отвечает контроллер) или управления данными (за это отвечает модель).

**Как взаимодействуют**

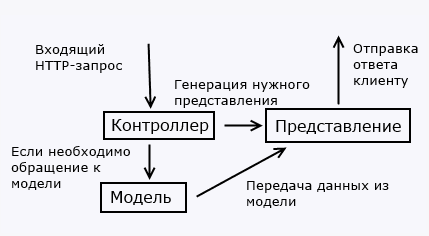
Сначала браузер отправляет запрос Контроллеру. Затем Контроллер взаимодействует с Моделью для отправки и получения данных.

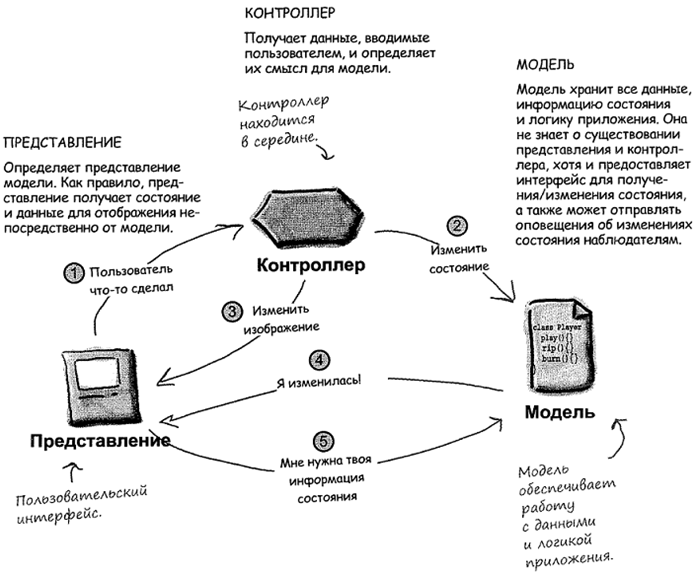
Потом Контроллер взаимодействует с Представлением для визуализации данных. Представление занимается только тем, как представить информацию, а не окончательным отображением. Отображать данные на основе того, что отправляет Контроллер, будет уже динамический HTML-файл.

Наконец, Представление отправит окончательный вид Контроллеру, а Контроллер передаст эти данные на вывод пользователям.

Важно то, что Представление и Модель никогда не взаимодействуют напрямую. Единственное взаимодействие между ними происходит через Контроллер.

Это означает, что логика приложения и интерфейс никогда не пересекаются.





**ЖЦ**

Фактически MVC определяется двумя жизненными циклами: жизненным циклом приложения и жизненным циклом запроса.

**Жизненный цикл приложения**, в котором процесс приложения запускает работающий сервер до момента его остановки. и он пометил два события в файле запуска вашего приложения. т. е. события запуска и завершения приложения.

Это отдельно от **жизненного цикла запроса**, который представляет собой последовательность событий или этапов, выполняемых каждый раз, когда HTTP-запрос обрабатывается приложением.

Отправной точкой для каждого приложения MVC является маршрутизация. После этого полученный запрос вычисляет и определяет, как его следует обрабатывать с помощью модуля маршрутизации URL. Модуль маршрутизации отвечает за сопоставление входящего URL с маршрутами, которые мы определили в нашем приложении.

У каждого маршрута есть связанный с ним обработчик маршрута. Если запрос соответствует маршруту, определенному в нашем приложении MVC, обработчик маршрута MVC извлекает экземпляр MVC HttpHandler.

Обработчик MVC Запускает процесс инициализации и выполнения контроллера. Платформа MVC обрабатывает преобразование данных маршрута в определенный контроллер, который может обрабатывать запросы.

Это достигается с помощью компонентов MVC, таких как фабрика контроллеров и Активаторы, которые отвечают за создание экземпляра класса Controller.

После создания контроллера компонент, называемый средством вызова действия, находит и выбирает соответствующий метод действия для вызова на нашем контроллере.

Привязка модели происходит перед вызовом метода, который сопоставляет данные из нашего HTTP-запроса с параметрами наших методов действий. Также называется фильтром действий до и после генерации результатов действий.

Теперь, после того, как результат нашего действия подготовлен, запускается следующий этап - Выполнение результата.

Если результатом является тип представления, будет вызван механизм представления, который найдет представление и отобразит его.

Если не тип представления, то результат действия будет выполняться сам по себе. Выполнение этого результата есть не что иное, как генерация фактического ответа на этот исходный HTTP-запрос.

**Получение данных в контроллере**

Контроллеры часто нуждаются в доступе к данным из входящего запроса, таким как значения строки запроса, значения формы и параметры, извлеченные из URL системой маршрутизации. Существуют три основных способа доступа к таким данным:

* извлечение данных из набора объектов контекста;
* передача данных в качестве параметров методу действия;
* явное обращение к средству привязки моделей инфраструктуры.

**Передача данных в контроллер**

Данные в контроллер могут передаваться различными способами. Одним из широко используемых в настоящее время способов передачи данных в контроллер от пользователя является использование строки запроса (query string).

Передавать значения для параметров можно различными способами. При отправке GET-запроса значения передаются через строку запроса. Стандартный get-запрос принимает примерно следующую форму: название\_ресурса?параметр1=значение1&параметр2=значение2.

Определение в методах контроллера параметров ничем не отличается от определения параметров в языке C#. Параметры могут представлять примитивные типы, как int или string, а могут представлять и более сложные классы:

1. Передача простых значений
2. Передача объектов
3. Передача массивов
4. Передача массива объектов
5. Передача словарей

**КОНТРОЛЛЕРЫ**

Ключевым элементом контроллера являются его действия. Действия контроллера - это **публичные** методы, которые могут сопоставляться с запросами. Контроллер также может иметь непубличные методы - такие методы не рассматриваются как действия и соответственно не могут соотноситься с запросами.

Атрибут [ActionName] позволяет для метода задать другое имя действия. Например:

[ActionName("Welcome")]

public string Hello()

{

return "Hello ASP.NET";

}

Сопоставление запроса с контроллером и его действием происходит благодаря системе маршрутизации. Метод app.MapControllerRoute добавляет один маршрут с именем default и шаблоном "{controller=Home}/{action=Index}/{id?}". Данный шаблон устанавливает трехсегментную структуру строки запроса: controller/action/id.

**ПРЕДСТАВЛЕНИЯ**

**Рендеринг**

За работу с представлениями отвечает объект ViewResult. Он производит рендеринг представления в веб-страницу и возвращает ее в виде ответа клиенту. Чтобы возвратить объект ViewResult, в методе контроллера вызывается метод View:

Вызов метода View возвращает объект ViewResult. Затем уже ViewResult производит рендеринг определенного представления в ответ. **Получение данных**

**Передача данных**

**ViewData**

ViewData представляет словарь из пар ключ-значение:

ViewData["Message"] = "Hello ASP.NET Core";

**ViewBag**

ViewBag во многом подобен ViewData. Он позволяет определить различные свойства и присвоить им любое значение.

ViewBag.Message = "Hello ASP.NET Core";

**Модель представления**

Модель представления является во многих случаях более предпочтительным способом для передачи данных в представление. Для передачи данных в представление используется одна из версий метода View:

List<string> countries = new List<string> { "Бразилия", "Аргентина", "Уругвай", "Чили" };

return View(countries);