Федеральное государственной образовательное учреждение

Высшего профессионального образования

Национальный исследовательский технологический университет «МИСИС»

Институт информационных технологий и компьютерных наук (ИТКН)

Кафедра автоматизированных систем управления (АСУ)

Курсовая работа

по курсу « Объектно-ориентированное программирование »

Тема:

« Разработка программного обеспечения.  
Сбор и обработка данных »

Название:

##### «Сайт с базой данных с использованием C#, HTML, CSS, Visual Studio - ASP.NET Web App (Model-View-Controller)»

Выполнил: студент 1 курса

гр. БИВТ-22-9 Федосеев Сергей

Руководитель: Мурадханов С. В.

МОСКВА 2023

Оглавление

[1. **Введение** 3](#_Toc134651142)

[1.1 **Краткое описание NET.CORE** 4](#_Toc134651143)

[1.2 **Краткое описание SQL** 5](#_Toc134651144)

[1.3 **Внедрение зависимости (Dependency injection, DI)** 7](#_Toc134651145)

[1.4 **Пакеты NuGet, которые были использованы в проекте** 7](#_Toc134651146)

[1.5 **Описание HTML & CSS** 9](#_Toc134651147)

[1.6 **HTML и CSS: использование и особенности** 10](#_Toc134651148)

[1.7 **MVC (Models-Views-Controllers)** 11](#_Toc134651149)

[2. **Практическая часть** 12](#_Toc134651150)

[2.1 **Файл проекта** 12](#_Toc134651151)

[2.2 **Папка свойств** 14](#_Toc134651152)

[2.3 **Корневая папка wwwroot** 15](#_Toc134651153)

[2.4 **Program.cs + Маршрутизация** 15](#_Toc134651154)

[2.5 **Layout и Tag Helpers** 18](#_Toc134651155)

[2.6 **IActionResult, папка Model + база данных в SQL** 20](#_Toc134651156)

[2.7 **Миграции и контроллер (CategoryController) для работ с категориями** 23](#_Toc134651157)

[Заключение 30](#_Toc134651158)

[Список использованных источников и литературы 31](#_Toc134651159)

1. **Введение**

*Для создания данного проекта использовались инструменты такие как:*

* **.NET 6**

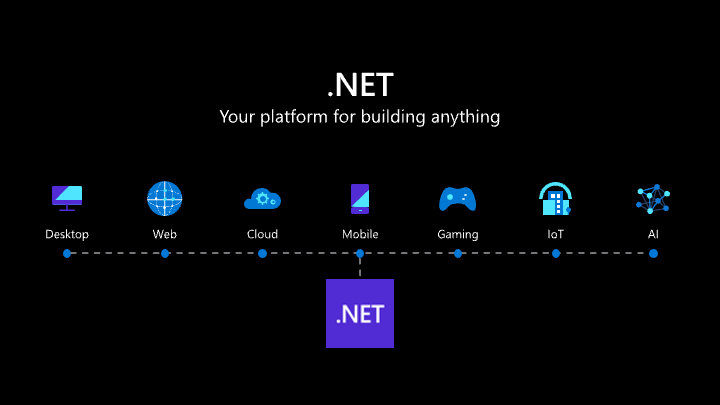


Рис. 1

* **ИСР (интегрированная среда разработки) Visual Studio 2022**



Рис. 2

* **SQL Server Management Studio 19**



Рис. 3

* 1. **Краткое описание NET.CORE**

Изображение выглядит как текст, круг, диаграмма, Шрифт

Автоматически созданное описание

Рис. 4

В 2002 году Microsoft представила веб-формы в то время. У веб-форм есть свои недостатки и их нужно было как-то преодолеть. Именно поэтому команда .NET разработала новую архитектуру, которая назвалась .NET MVC (Model-View-Controller). В июне 2016 года компания Microsoft выпустила ASP .NET CORE. Это была первая версия .NET CORE, созданная поверх нового *framework* .NET CORE.

*Framefork - заготовка, готовая модель в IT для быстрой разработки, на основе которой можно дописать собственный код.*

Изображение выглядит как текст, Шрифт

Автоматически созданное описание

Рис. 5

Первая версия полностью переписана и является кроссплатформенной, то есть не привязана к Windows. Кроме того, .NET CORE было создано с учётом облачных технологий, которые являются надежными.

Затем в 2018 году компания Microsoft выпустила .NET CORE 2. Команда была крайне активна с выпуском новых версий, произошло большое изменение с 2.1 .NET CORE и .NET CORE 2.2, потому что пришлось обновить небольшое количество библиотек, из-за которых возникло немного трудностей. Но уже с тех пор команда .NET CORE выпускает новые версии с тремя 3.1 и .NET 5, которые были выпущены в ноябре 2020-го. После этого появится .NET 6 в ноябре 2021-го.

.NET CORE имеет много преимуществ:

1. ASP .NET CORE – быстрый и с открытым исходным кодом. Если сравнивать его с традиционными приложениями .NET, то у него было много *benchmarks*. Его скорость очень большая по сравнению с веб-формами или даже .NET MVC.

*Benchmark - задача, служащая эталонным тестом производительности компьютерной системы.*

1. .NET CORE - кроссплатформенный, классический .NET был привязан к IIS и Windows. Но с тех пор .NET CORE переписан, оно убрало эту зависимость с .NET CORE.
2. Есть встроенная поддержка внедрения зависимостей, которая экономит много времени.
3. Новые версии – легко обновляемые.
4. Облачная архитектура. .NET CORE – полностью совместим со всеми облачными компонентами.
5. .NET CORE превосходит все предыдущие версии и даже новые версии .NET, которые были выпущены. Они заменяют предыдущую версию. Код на самом деле становится более оптимизированным, что приводит к повышению производительности.
   1. **Краткое описание SQL**

SQL - это язык структурированных запросов (Structured Query Language), на котором базируется управление и доступ к данным в реляционных базах данных. Он создан для работы с данными, которые хранятся в реляционных базах данных, где информация хранится в формате таблиц с различными атрибутами.

Изображение выглядит как текст, снимок экрана, коробка

Автоматически созданное описание

Рис. 6

История создания SQL началась в 1970 году в исследовательском центре IBM. Тогда Дж. Ф. Кодд, Э. Ф. Кодд и Р. Г. Бойс предложили новую концепцию хранения информации в реляционных таблицах и язык запросов, который позволял бы обладателям данной информации быстро извлекать нужные им данные. Этот язык запросов получил название SEQUEL (Structured English Query Language) и использовался на практике в рамках проекта System R - исследовательского проекта IBM по созданию промышленной системы управления базами данных.

PostgreSQL, MySQL, Oracle, MS SQL Server - это наиболее распространенные реляционные базы данных, основанные на SQL.

Использование SQL вместе с .NET Core предоставляет множество преимуществ:

1. Простота создания запросов и получения данных из базы данных в .NET Core.

2. SQL позволяет легко манипулировать данными, в том числе добавлять, обновлять и удалять информацию из базы данных, что делает работу с данными более эффективной.

3. Использование SQL вместе с .NET Core обеспечивает высокую скорость запросов и обработки данных.

4. SQL поддерживает стандарты ACID (атомарность, согласованность, изолированность, долговечность), что обеспечивает безопасную и надежную работу с данными.

5. С помощью SQL можно управлять большим объемом данных и обрабатывать многопользовательские запросы без задержек или ошибок.

Таким образом, SQL является важной частью инструментов, используемых разработчиками .NET Core для работы с данными в реляционных базах данных, что позволяет создавать высокопроизводительные, безопасные и надежные приложения.

* 1. **Внедрение зависимости (Dependency injection, DI)**

Внедрение зависимостей контейнера – неотъемлемая часть архитектуры ASP .NET CORE

.NET CORE внедряет объекты классов зависимостей через конструктора, используя встроенный контейнер IOC.

*Контейнер IOC помогает уменьшить количество рутины, позволяя задать соответствие между интерфейсом и его конкретной реализацией, чтобы потом везде этим пользоваться.*

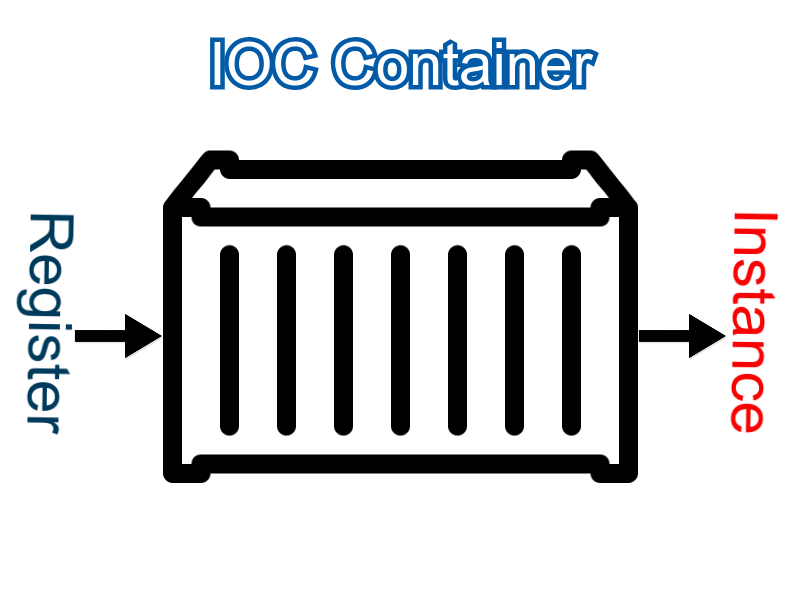


Рис. 7

С .NET CORE у нас есть встроенный контейнер внедрения зависимостей, у которого есть свои особенности.

С помощью DI нам не нужно иметь дело с созданием объекта, размещающего или управляющего этим объектом внутри наших страниц. Наши страницы будут выглядеть очень чистыми только с интерфейсом.

Вся реализация будет выполняться контейнером для внедрения зависимостей. Это позволит нам изменять или заменять класс элемента, какие-либо изменения на страницах вносить не нужно. Всё, что нужно сделать, это изменить реализацию внутри класс элемента.

* 1. **Пакеты NuGet, которые были использованы в проекте**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт

Автоматически созданное описание

Рис. 8

* **Описание пакета (AspNetCore.Mvc.Razor.RuntimeCompilation)**

Пакет предоставляет инструменты для компиляции представлений Razor во время выполнения на фреймворке ASP.NET Core.

Свойства:

- `AddMvcRazorRuntimeCompilation()` - добавляет сервисы компиляции представлений Razor во время выполнения в приложение ASP.NET Core.

- `services.AddControllersWithViews()` - добавляет контроллеры и представления в приложение ASP.NET Core.

* **Описание пакета (EntityFrameworkCore)**

Пакет предоставляет инструменты для работы с базой данных в .NET приложениях.

Свойства:

- `DbContext` - класс для работы с данными в базе данных.

- `DbSet<TEntity>` - свойство, которое позволяет получать доступ к коллекциям сущностей в базе данных.

- `AddDbContext<TContext>` - метод для добавления DbContext в приложение.

* **Описание пакета (EntityFrameworkCore.SqlServer)**

Пакет предоставляет инструменты для работы с базой данных SQL Server в .NET приложениях.

Свойства:

- `UseSqlServer()` - метод для настройки подключения к базе данных SQL Server в приложении.

- `SqlServerDbContextOptionsExtensions` - класс с расширениями для настройки DbContext для работы с базой данных SQL Server.

* **Описание пакета (EntityFrameworkCore.Tools)**

Пакет предоставляет инструменты для разработки с базой данных в .NET приложениях, такие как миграции базы данных.

Свойства:

- `Add-Migration` - утилита командной строки для генерации миграций базы данных.

- `Update-Database` - утилита командной строки для обновления скриптов базы данных на основе миграции.

* **Описание пакета (VisualStudio.Web.CodeGeneration.Design)**

Пакет, который позволяет создавать код для MVC-сервиса приложений в Visual Studio 2019. Он предоставляет набор инструментов и шаблонов, которые упрощают создание базового кода для создания веб-приложений на платформе .NET Core.

Свойства:

- `AspNetCore.Scaffolding.Design` - класс, включающий сборки для создания базового кода веб-приложений AspNetCore.

- `Command Name` - набор команд для выполнения инструкций по генерации кода.

- `Context` - класс, который представляет базу данных и используется для создания контекста базы данных.

- `Model` - класс, который представляет модель данных, используемую в приложении, и который используется для создания базового кода.

* 1. **Описание HTML & CSS**

HTML (HyperText Markup Language) — это язык разметки документов для создания веб-страниц. Он позволяет создавать различные элементы страницы, такие как заголовки, абзацы, списки, таблицы, ссылки, кнопки, изображения и многое другое. HTML используется для описания структуры и содержания документа, который затем может быть отображен веб-браузером.

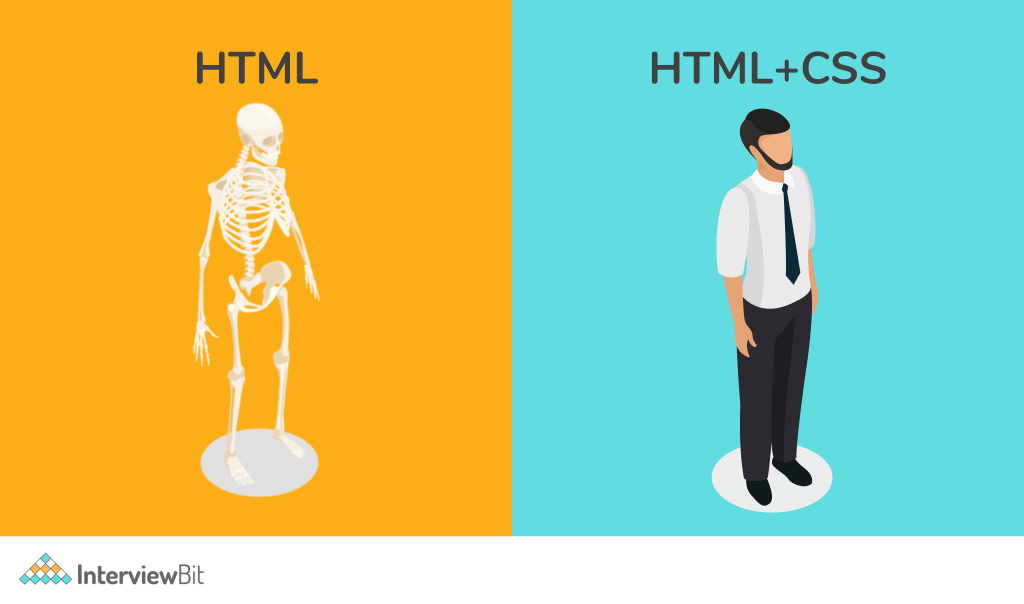


Рис. 9

CSS (Cascading Style Sheets) - это язык таблиц стилей, который определяет внешний вид отдельных элементов HTML-документа и веб-страницы в целом. С помощью CSS можно задавать цвет, шрифт, размер текста, отступы, границы, выравнивание и множество других стилевых свойств элементов страницы. CSS позволяет разделять структуру и содержимое веб-страницы от ее визуального отображения, что делает процесс разработки и поддержки веб-сайтов более эффективным.

* 1. **HTML и CSS: использование и особенности**

**HTML**

Используется для создания структуры веб-страницы. Каждый элемент на странице создается с помощью HTML-тегов.

Например, чтобы создать заголовок, используется тег `<h1>`, для абзаца - `<p>`, для ссылки - `<a>`, для изображения - `<img>`.

HTML-код можно редактировать в любом текстовом редакторе, просматривать и тестировать в веб-браузере.

**CSS**

Используется для стилизации элементов HTML-страницы. Он позволяет задавать размер, цвет, шрифт, расположение, границы и другие свойства элементов на странице. CSS можно добавлять в HTML-документ внутри тегов `<style>` или ссылаться на отдельный файл стилей с помощью атрибута `link`.

Основные особенности HTML и CSS:

1. **Раздельность структуры и визуальных стилей**: HTML-теги используются для создания структуры документа, а CSS-стили - для определения внешнего вида элементов на странице.
2. **Использование селекторов для стилизации элементов**: CSS-селекторы используются для задания стилей только для конкретных элементов на странице, что позволяет точно управлять отображением элементов.
3. **Использование классов и идентификаторов**: CSS-классы и идентификаторы позволяют задавать одинаковые стили для нескольких элементов на странице или же создавать уникальные стили для конкретных элементов.
4. **Использование вложенных стилей**: CSS позволяет создавать вложенные стили, когда определенный стиль применяется только внутри другого элемента.
5. **Создание адаптивной веб-страницы**: CSS позволяет создавать адаптивные веб-страницы, которые подстраиваются под размер экрана устройства, на котором они просматриваются. Например, можно использовать медиазапросы, чтобы изменять видимость элементов на странице или изменять размер текста в зависимости от ширины экрана.
   1. **MVC (Models-Views-Controllers)**

Model = classes (objects)

View = web page (Razor HTML)

Controller = связи моделей, бизнес логика и веб-страницы

MVC (Model-View-Controller) - это паттерн проектирования, который широко используется в веб-разработке для создания приложений на основе модели, представления и контроллера. В MVC-модели приложения разделены на три основных компонента - модель (Model), представление (View) и контроллер (Controller).

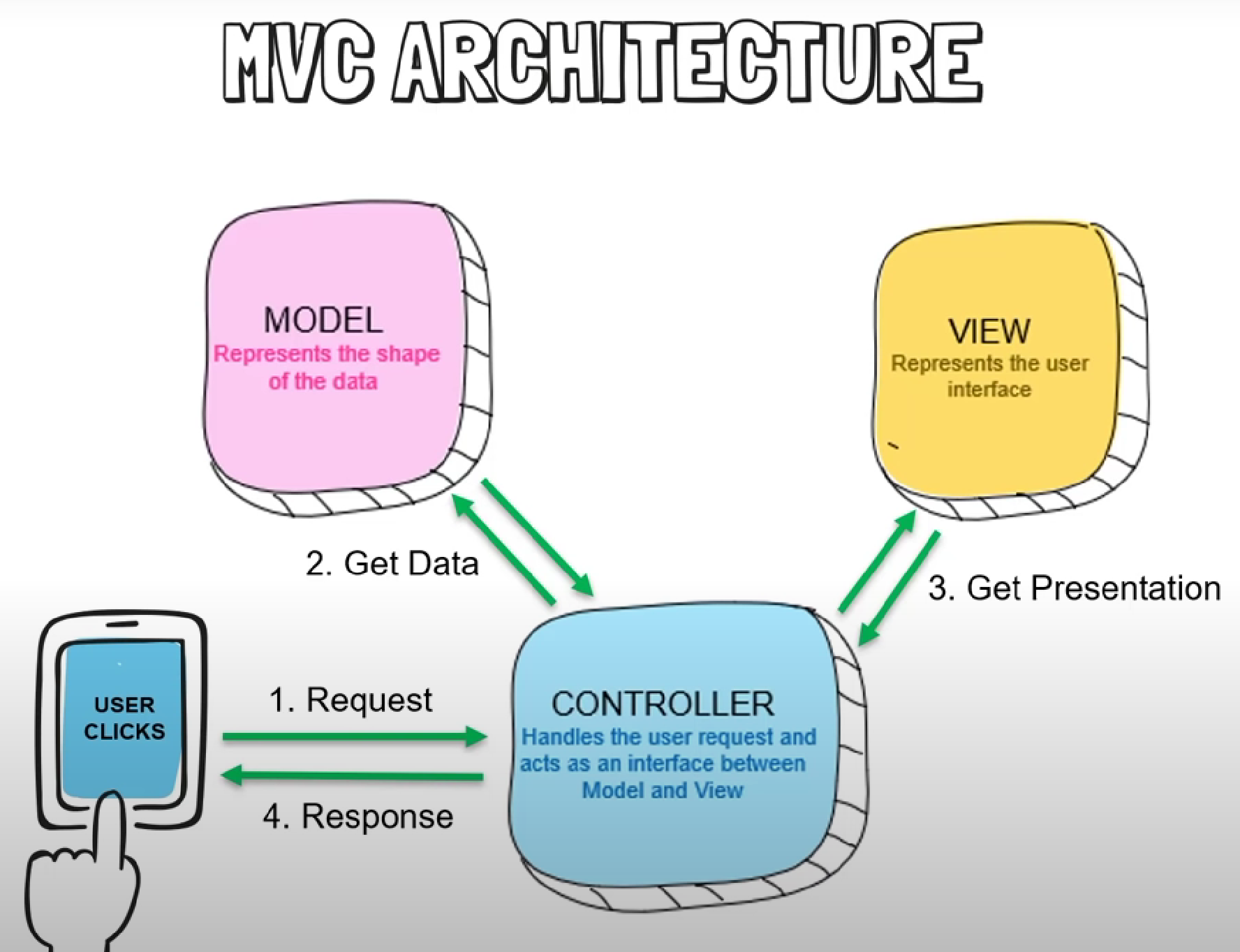


Рис. 10

- Модель (Model) — это слой приложения, который отвечает за хранение данных и логику работы с ними. Модель содержит всю необходимую информацию об объектах и сущностях приложения и их взаимосвязи.

- Представление (View) — это слой, который отображает данные на экране, а также отвечает за интерфейс пользователя. Представление имеет доступ только для чтения к данным модели, а также может использовать внешние ресурсы для оформления страницы.

- Контроллер (Controller) — это слой, который обрабатывает запросы, полученные от пользователя, и управляет логикой приложения. Контроллер получает данные из модели и передает их в представление.

MVC используется для разделения кода и улучшения организации приложения. Это позволяет ускорить разработку и тестирование приложения. Возможности MVS также включают разделение обязанностей в команде разработки

1. **Практическая часть**

*В этой части мы рассмотрим работу и структуру кода изнутри.*

* 1. **Файл проекта**

Первое, на что следует обратить внимание после создания проекта – это файл проекта. Для получения доступа к нему следует найти проект со значком веб-сайта в обозревателе решений в самом верху.

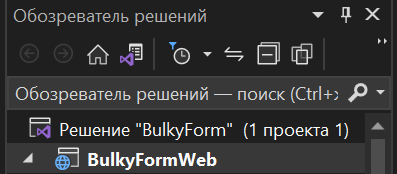


Рис. 10

Далее нажать на него правой кнопкой мыши и выбрать в появившемся окне «Изменить файл проекта»

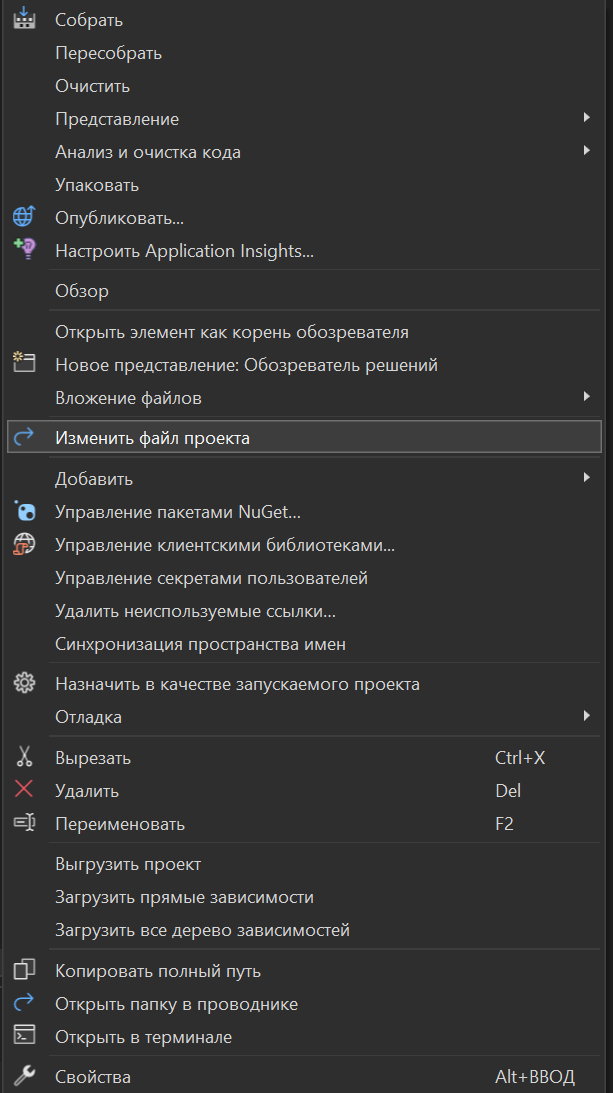


Рис. 11

После его открытия можно увидеть несколько конфигураций.

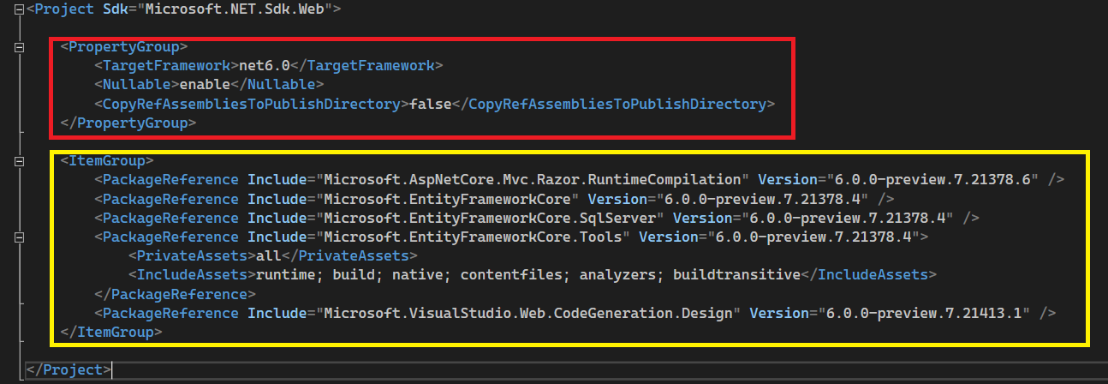


Рис. 12

В красной табличке находится раздел, который определяет, какую целевую платформу разработчик будет использовать. В нашем случае используется .NET 6

В жёлтой табличке находится не менее важная вещь внутри этого файла – это группа элементов. Эта группа будет содержать все новые пакеты получения, которые будут использоваться в проекте.

* 1. **Папка свойств**

Внутри этой папки у нас есть настройки запуска через JSON.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 13

В ней есть разные профили, с помощью которых мы можем запускать приложение.

Профиль IIS Express – настройка. Если работать с IIS Express, для SSL PORT будет использоваться указанный перед ним номер порта.

Изображение выглядит как текст, снимок экрана, Шрифт, Цвет электрик

Автоматически созданное описание

Рис. 14

IIS Express (Internet Information Services Express) - это сервис веб-сервера, который может быть использован для локальной разработки, тестирования и отладки веб-приложений на платформе Microsoft. IIS Express представляет собой легковесную и относительно быструю по сравнению с полноценным IIS версии веб-сервер, который может быть запущен на переносном компьютере или сервере с поддержкой платформы Windows.

* 1. **Корневая папка wwwroot**

Любые статические файлы, такие как CSS, JavaScript, изображения, библиотеки находятся в этой папке.

В этой папке не будет каких-либо файлов C#. Эта папка предназначена только для хранения статических файлов приложения.

Если понадобится добавить какой-либо статический файл, то следует заглянуть в корневую папку wwwroot.



Рис. 15

* 1. **Program.cs + Маршрутизация**

*Этот файл отвечает за запуск приложения.*

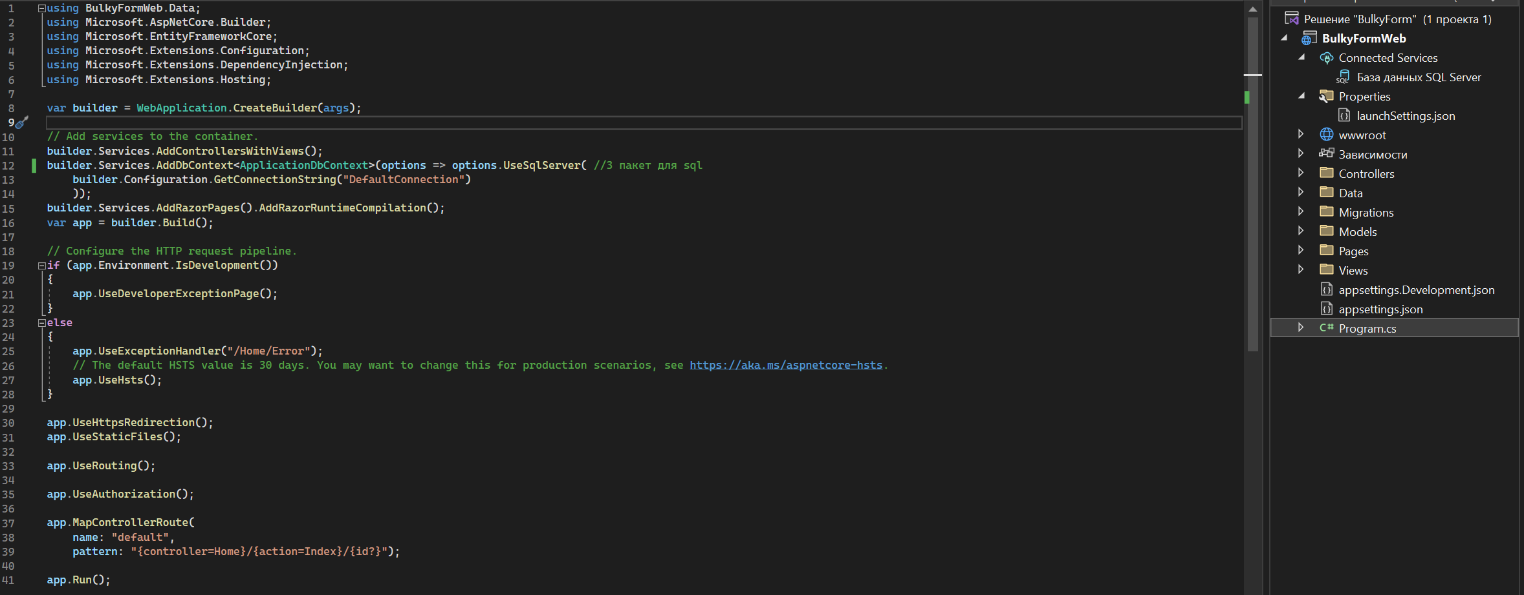


Рис. 16

На самой первой строчке кода используется построитель переменных, где WebApplication.CreateBuilder передается со встроенными аргументами.

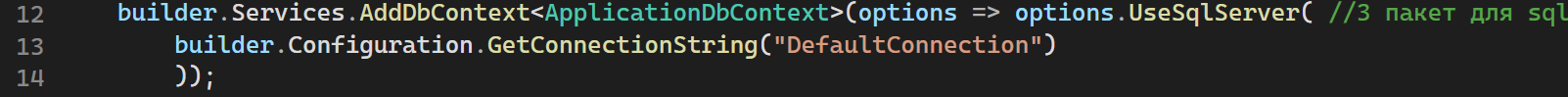
Эта строка представляет собой начало конфигурации приложения ASP.NET Core, которое позволяет создать новое приложение и настроить его согласно требованиям проекта.

// Add services to the container.

После неё идут строки, которые используются для настройки сервисов и добавления их в контейнер зависимостей приложения ASP.NET Core.



Метод добавляет контроллеры и представления в контейнер зависимостей. Это позволяет приложению использовать MVC (Model-View-Controller) шаблон для обработки запросов.



Метод добавляет контекст базы данных с помощью Entity Framework Core. В данном случае мы используем SQL Server в качестве провайдера базы данных, чтобы его можно было использовать в приложении. Этот метод добавляет `ApplicationDbContext` в контейнер зависимостей с настройками, а также конфигурирует его с помощью строки подключения, полученной из конфигурации приложения. (для пункта 2.6)



Метод добавляет поддержку страниц Razor и компиляцию Razor во время выполнения.

С каждым запросом приложение создает новый экземпляр контроллера, сервисов и других зависимостей, которые используются в обработке запроса. Контейнер зависимостей - это механизм, который позволяет управлять этим процессом создания экземпляров и обеспечивать доступ к зависимостям через инъекцию зависимостей (Dependency Injection).

// Configure the HTTP request pipeline.

Конвейер (pipeline) — это цепочка обработчиков (middleware), которые обрабатывают запросы, именуемых по порядку добавления в цепочку. Каждый обработчик может выполнить какую-то обработку запроса, либо передать управление следующему обработчику в цепочке.

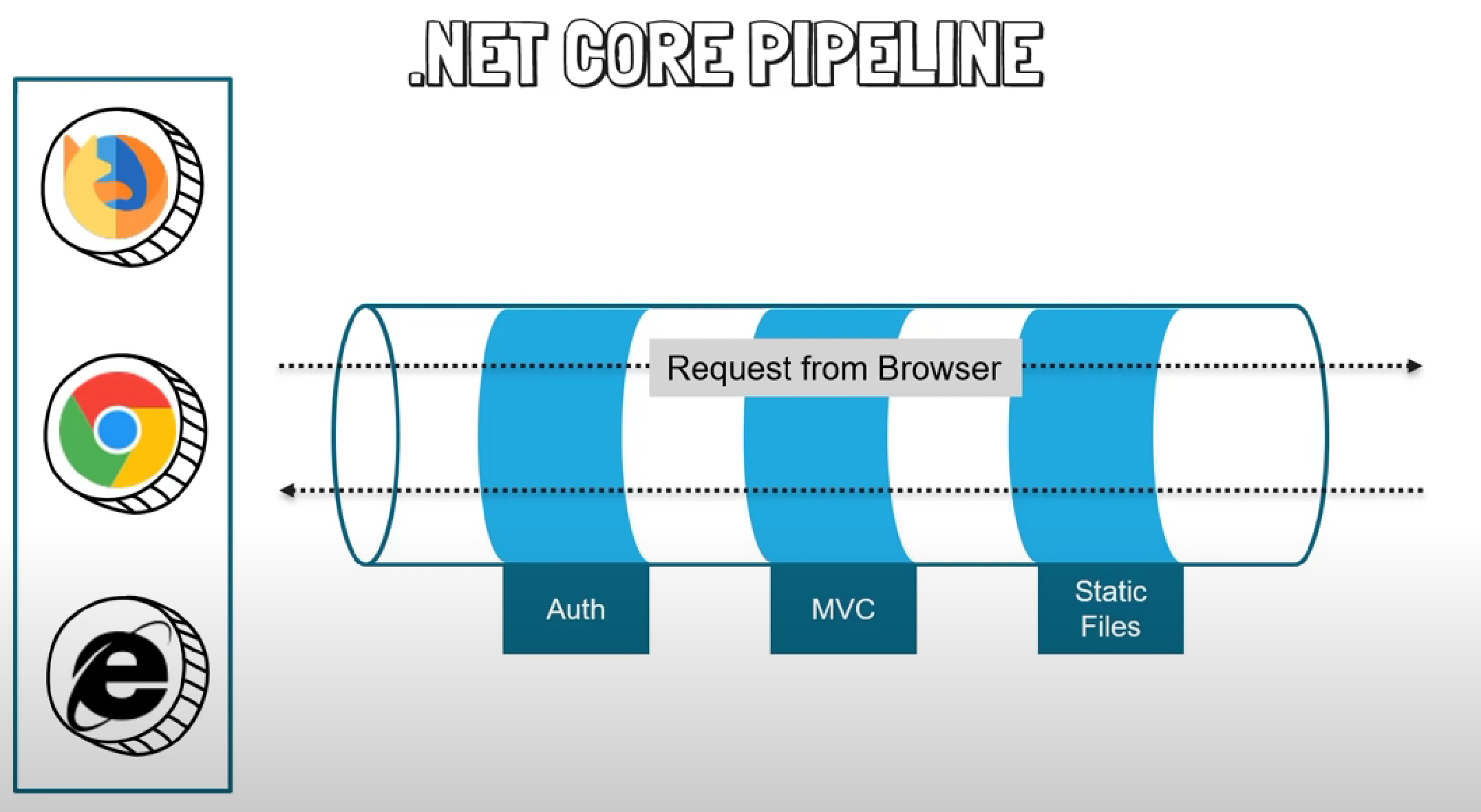


Рис. 17

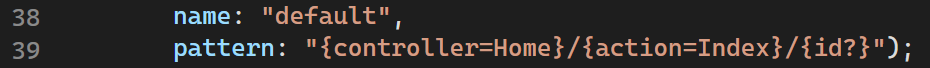
Конвейер (pipeline) указывает, как приложение должно реагировать на полученный запрос. Когда приложение получает запрос от браузера, этот запрос проходит через конвейер. В нём разработчик может добавлять элементы, которые захочет. Конвейер состоит из промежуточных программ (например MVC).

Вот объяснение каждого из компонентов метода:

* `app.Environment.IsDevelopment()` - это свойство объекта `IApplicationBuilder`, определяющее, выполняется ли приложение в режиме разработки или нет. Если приложение находится в режиме разработки, то используется
* `UseDeveloperExceptionPage()`, который отображает детальную информацию об ошибках для разработчика. Если не в режиме разработки, он использует
* `UseExceptionHandler()` для перенаправления на страницу с предупреждением об ошибке.
* `UseHttpsRedirection()` добавляет middleware для перенаправления http запросов на https.

* `UseStaticFiles()` - это middleware, который позволяет обслуживать статические файлы, такие как изображения и статические HTML-страницы.
* `UseRouting()` добавляет цепочку обработчиков middleware, который определяет маршруты, которые браузер должен запросить для отображения нужных страниц.
* `UseAuthorization()` добавляет middleware для авторизации пользователя.
* `app.MapControllerRoute()` добавляет маршруты к контроллерам, что позволяет определять URL-адрес, который пользователь будет запрашивать для каждого действия контроллера.
* `app.Run()` - это middleware, который завершает цепочку обработки запроса. Запуск этого middleware означает, что если он был вызван, то все другие обработчики, добавленные в конвейер обработки запроса, уже были выполнены.

**Маршрутизация:**



Определяет маршрут для конкретного контроллера (в данном случае, контроллера Home). `name: "default"` — это имя маршрута, которое можно использовать в приложении, чтобы ссылаться на этот маршрут.

`pattern: "{controller=Home}/{action=Index}/{id?}"` — это паттерн маршрута, который определяет URL-адрес запрашиваемой страницы. В данном случае, при запросе корневого URL-адреса, приложение будет обращаться к действию `Index` контроллера `Home`.

`{controller=Home} ` - здесь мы указываем, что по умолчанию контроллер должен быть `Home`, если часть URL-адреса `controller` не указана в запросе.

`{action=Index} ` - здесь мы указываем, что по умолчанию метод контроллера должен быть `Index`, если часть URL-адреса `action` не указана в запросе

`{id?} ` - этот маршрут содержит дополнительный параметр `id`, который может быть передан в запросе, но не является обязательным. Если параметр `id` не будет передан в запросе, то он будет равен `null`.

* 1. **Layout и Tag Helpers**

\_Layout.cshtml — это шаблон представления в ASP.NET Core, который позволяет определить общую структуру и элементы пользовательского интерфейса для всех страниц приложения. Он может содержать общие элементы дизайна, такие как заголовок страницы, навигационную панель, подвал, сценарии JavaScript и т. д.

Он состоит из следующих элементов:

<!DOCTYPE html> - объявление типа документа

<html> - начало HTML-документа

<head> - информация о документе, такая как заголовок страницы, ссылки на стили, сценарии JavaScript и т. д.

<body> - основное содержимое документа

@RenderBody() - область, где будет вставлено содержимое конкретной страницы

<header> - область для размещения навигационной панели и других элементов верхней части страницы

<div class="container body-content"> - область для размещения содержимого страницы

<footer> - область для размещения информации о копирайте и других дополнительных элементов подвала страницы

@RenderSection() - области для вставки дополнительных стилей и сценариев JavaScript для конкретной страницы

**Файл \_ValidationScriptsPartial.cshtml**

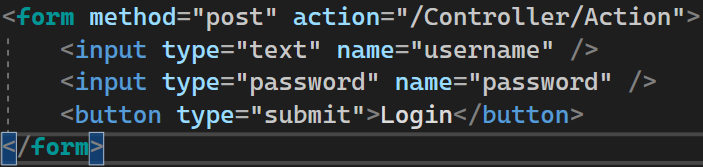
\_ValidationScriptsPartial — это частичное представление (partial view) в ASP.NET Core, которое генерирует HTML-код для включения клиентской валидации на стороне клиента с помощью JavaScript. Этот код обычно включает ссылки на библиотеки jQuery и jQuery Validate, а также скрипты для настройки и запуска валидации на стороне клиента.

**Tag Helpers**

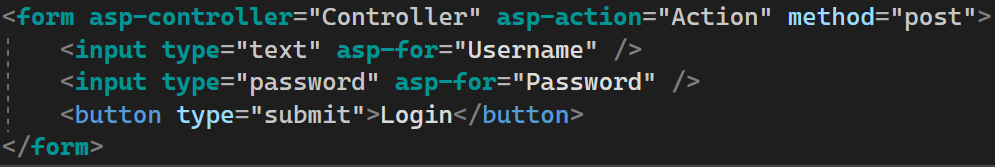
Тег-хелперы (Tag Helpers) — это механизм в ASP.NET Core MVC, который позволяет упростить и улучшить написание HTML-кода в представлениях. Они представляют собой классы C#, которые могут изменять и расширять поведение HTML-элементов в представлении.

Тег-хелперы используются для генерации HTML-кода из C#-кода в представлениях ASP.NET Core MVC. Они позволяют разработчикам использовать знакомый C#-синтаксис для генерации HTML-кода, а также добавлять дополнительную функциональность к HTML-элементам.

Например, вместо написания следующего кода в представлении:



Можно использовать тег-хелпер для генерации аналогичного кода:



Здесь asp-controller и asp-action являются атрибутами тег-хелпера, которые указывают контроллер и действие, к которым будет отправлен запрос. asp-for указывает свойство модели, которое будет использоваться для генерации соответствующего элемента формы.

* 1. **IActionResult, папка Model + база данных в SQL**

ActionResult — это базовый класс для всех результатов действий (actions) в ASP.NET MVC. Он представляет результат выполнения действия, который может быть отправлен обратно клиенту в виде HTTP-ответа.

Существуют различные типы результатов действий (action results) в ASP.NET MVC:

* ViewResult - возвращает HTML-страницу в виде представления Razor.
* PartialViewResult - возвращает частичное представление, которое может быть встроено в другое представление.
* JsonResult - возвращает данные в формате JSON.
* FileResult - возвращает файл в виде HTTP-ответа.
* RedirectResult - выполняет перенаправление на другую страницу.
* ContentResult - возвращает произвольный текст в качестве HTTP-ответа.
* StatusCodeResult - возвращает определенный HTTP-статус код.

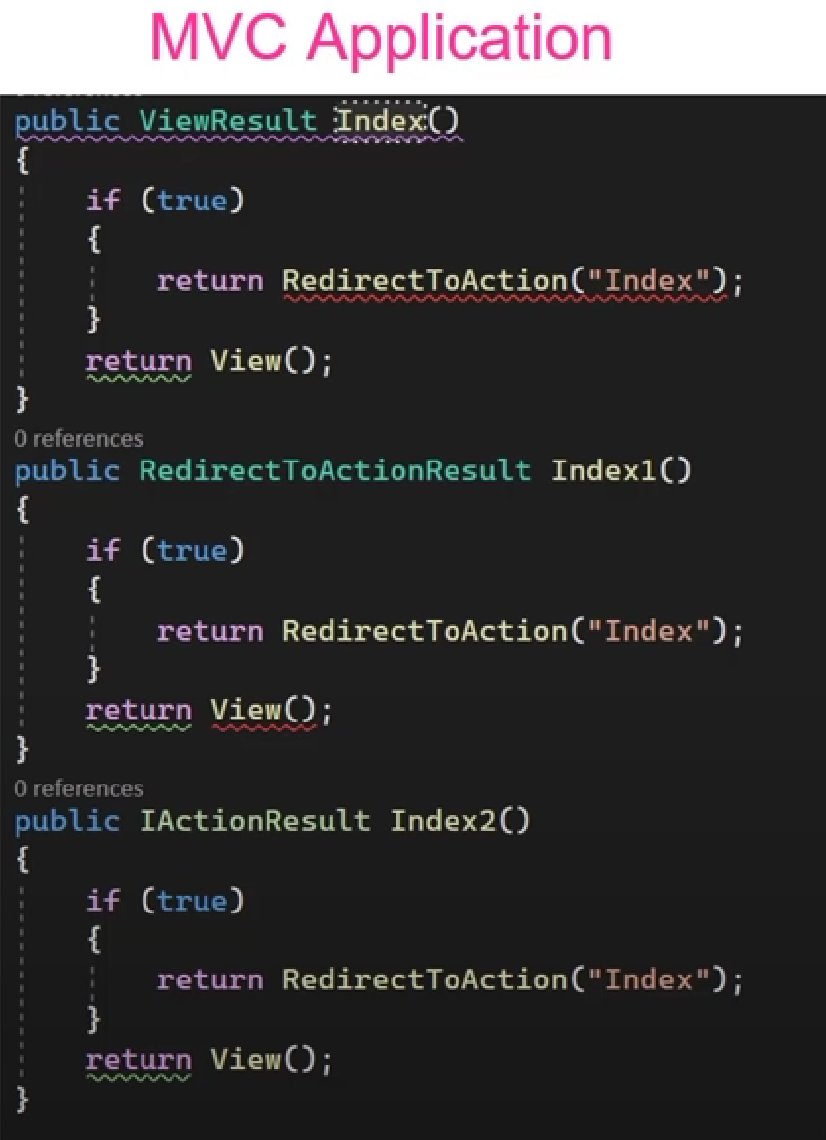


Рис. 18

IActionResult содержит один метод - ExecuteResultAsync, который принимает контекст действия (action context) и выполняет логику, необходимую для создания результатов действий. Результаты действий могут быть возвращены в виде различных типов, таких как HTML-страницы, JSON-данные, файлы и другие.

**Папка Model**

Папка Model в проекте .NET Core — это место, где обычно хранятся классы моделей данных, используемые в приложении.

Классы моделей обычно описывают структуру и свойства объектов данных, которые приложение будет использовать. Например, веб-приложение, которое работает с базой данных пользователей, класс модели может описывать свойства пользователя, такие как имя, адрес электронной почты и пароль.

Папка Model может также содержать классы, которые обрабатывают данные, используемые в вашем приложении, например, классы, которые преобразуют данные из формата, используемого в базе данных, в формат, необходимый для отображения на странице веб-приложения.

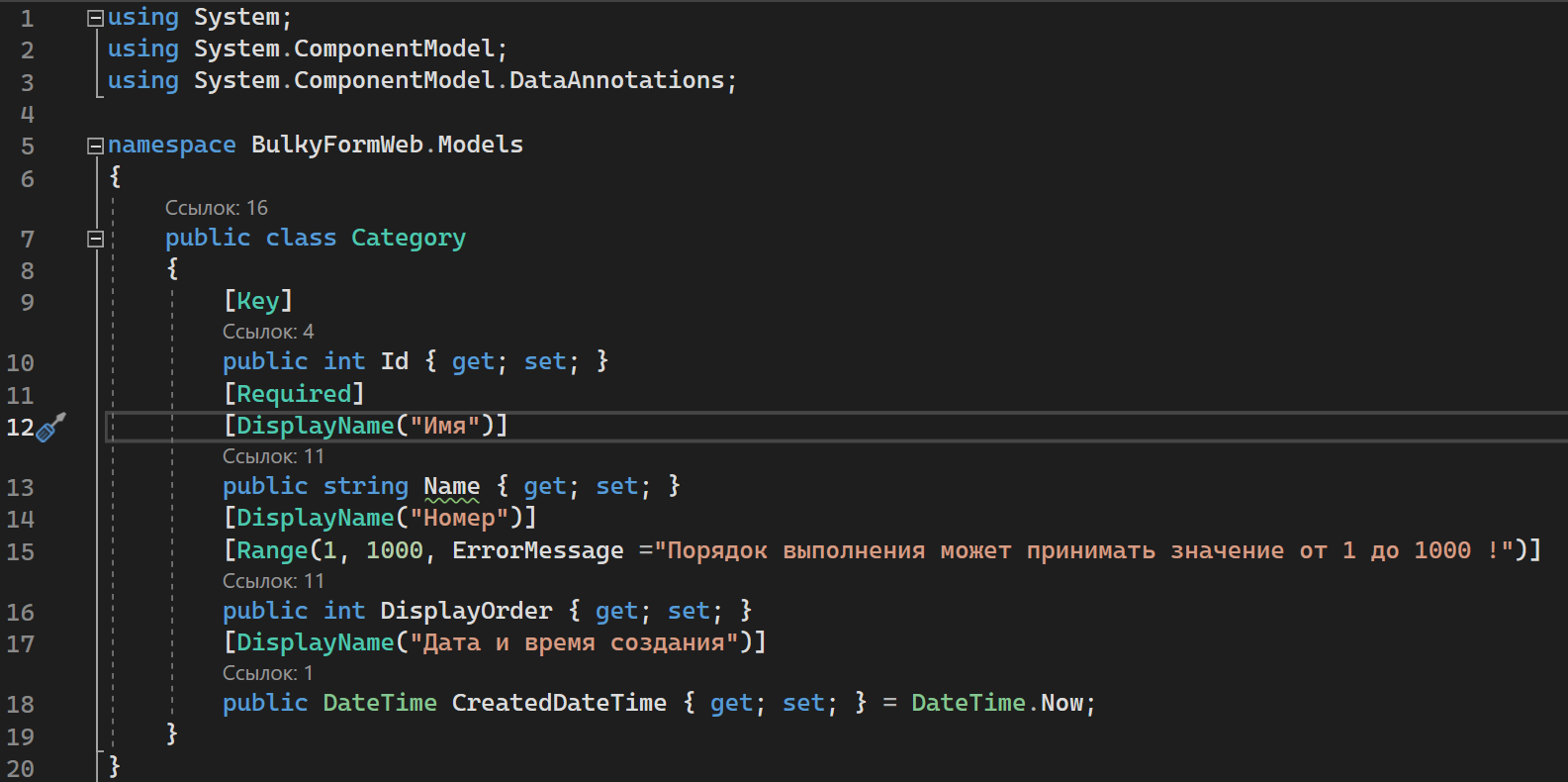


Рис. 19

Атрибут [Key] указывает, что свойство Id в классе Category является первичным ключом (primary key) таблицы, которая будет создана в базе данных при использовании Entity Framework.

*Первичный ключ (primary key) — это уникальный идентификатор каждой записи в таблице, который используется для определения записей в связанных таблицах и для обеспечения целостности данных.*

Атрибут [Required] указывает, что это свойство обязательно должно быть заполнено, иначе будет возникать ошибка валидации модели при создании записи в базе данных.

**Связка сервера SQL с кодом .NET CORE**

"appsettings.json" - это файл конфигурации, который используется в проектах на платформе .NET Core для хранения настроек приложения. В нем можно определить различные параметры, такие как строки подключения к базам данных, настройки сервисов и т.д.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 20

Для связи только что созданного сервера SQL с "appsettings.json" в проекте на платформе .NET Core необходимо выполнить следующие шаги:

1. Открыть файл "appsettings.json" в проекте.
2. Добавить в него новый параметр "ConnectionStrings" (если его еще нет) и задать значение в формате строки подключения к созданной базе данных на сервере SQL.
3. В коде приложения можно использовать строку подключения из "appsettings.json" с помощью объекта IConfiguration, который доступен через внедрение зависимостей.

Также, в .NET Core папка Data используется для размещения классов, связанных с доступом к данным, таких как классы контекста базы данных, классы репозитория и классы модели данных.

*Класс в папке Data*

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис.21

Этот код используется для создания контекста базы данных, который может быть использован для создания, чтения, обновления и удаления объектов Category в базе данных.

* 1. **Миграции и контроллер (CategoryController) для работ с категориями**

*Теперь, когда файл Program.cs настроен полностью, то следующим шагом - создание базы данных и таблицы.*

Это можно легко сделать с помощью Migrations:

Migrations в .NET Core - это механизм управления версиями базы данных, который позволяет легко создавать и обновлять схему базы данных на основе изменений в моделях данных (например, добавление новых таблиц или изменение существующих полей).

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Рис. 22

Файл миграции (migration file) содержит изменения, которые нужно внести в базу данных, и может включать добавление, изменение и удаление таблиц, столбцов, ключей и других элементов схемы базы данных. Каждый файл миграции обычно имеет имя, содержащее временную метку, чтобы обеспечить уникальность имени.

Чтобы фактически создать базу данных или внести изменения в таблицу

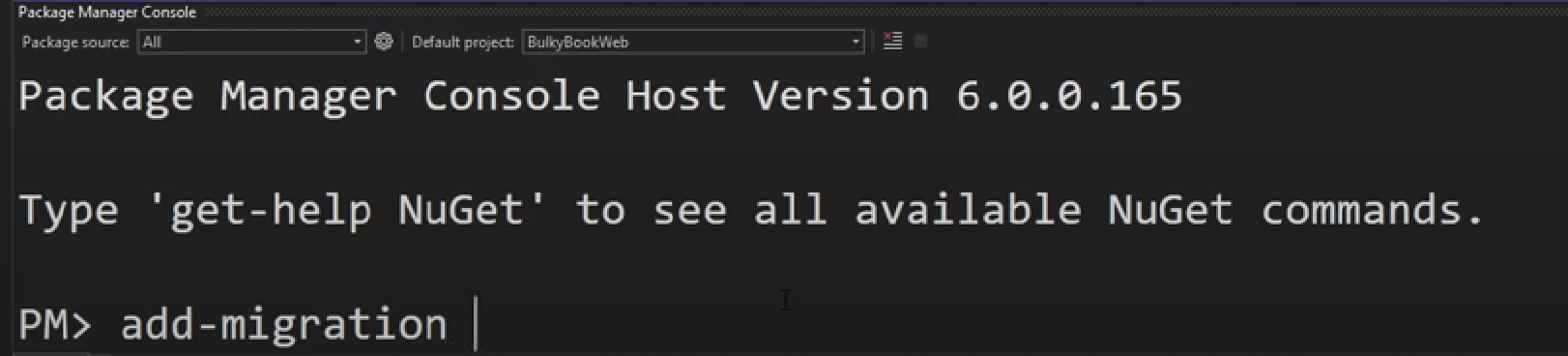


Рис. 23

Кроме того, migrations позволяют автоматически создавать базу данных на основе модели данных, используя команду dotnet ef database update. При этом будет создана новая база данных, если ее еще нет, и применены все миграции в порядке их создания.

Теперь, когда база данных работает корректно, создадим контроллер **(CategoryController)**, чтобы создавать, изменять и удалять категории в (Category).



Рис. 24

На (рис. 24) изображен путь создания **(CategoryController)**. Далее следует выбрать Контроллер MVC – Пустой.

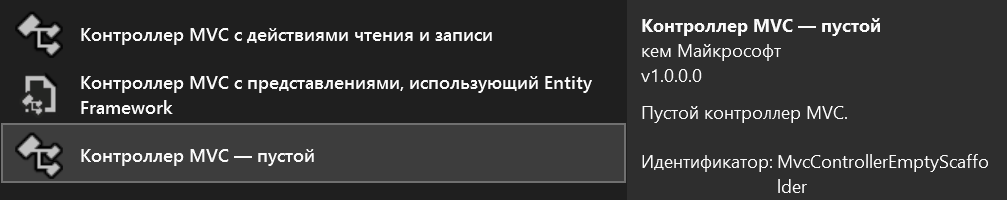


Рис. 25

В этом проекте используется такой тип контроллера, т. к. Контроллер MVC - пустой (Empty) создается без каких-либо методов действий (actions) и представлений (views). Такой контроллер может быть использован в случае, когда разработчик хочет создать свои собственные методы действий и представления для обработки запросов.

Контроллер **(HomeController)** содержит Index action method.

*Index action method - это метод действия (action method) в ASP.NET Core, который используется для обработки запроса на отображение стартовой страницы (главной страницы) приложения. Этот метод действия имеет стандартное имя "Index" и обычно возвращается как действие по умолчанию, когда пользователь заходит на сайт.*

Здесь метод действия Index возвращает объект типа IActionResult, который содержит результат выполнения действия. В данном случае, он возвращает результат отображения (view result), созданный с использованием метода View() без аргументов, что означает, что будет использоваться представление по умолчанию, имеющее то же имя, что и метод действия (Index).

Поэтому следует также создать Index action method для нового контроллера **(HomeController)**, и уже после его создания отобразить все категории, сохраненные в базе данных, на странице Index.

Реализовывается это с помощью приведённого на (рис. 26) кода:

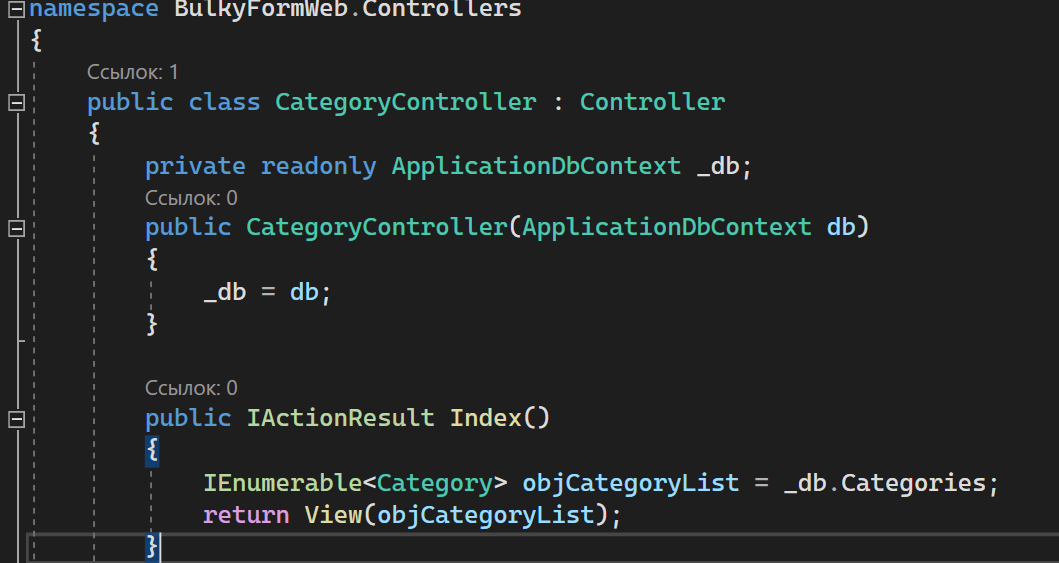


Рис. 26

В данном коде создается контроллер CategoryController, который наследуется от базового контроллера Controller в ASP.NET Core MVC. В конструкторе контроллера происходит инициализация поля \_db типа ApplicationDbContext, который используется для работы с базой данных.

Проще говоря, мы сообщаем нашему приложению, что нам нужна реализация этого контекста БД приложения, где соедение с базой данной, реализованной с помощью внедрения зависимости (см. пункт 1.3)

*ApplicationDbContext — это класс контекста базы данных, который настроен для работы с БД в приложении ASP.NET Core.*

Далее, в методе действия Index() контроллера происходит получение списка объектов категорий (Category) из базы данных с помощью свойства Categories объекта \_db. Categories представляет коллекцию объектов категорий, которые хранятся в базе данных и соответствуют определенной таблице в базе данных.

Для создания, редактирования и удаления пользователя, предварительно следует создать для каждого действия окно, с полями ввода и специальными кнопками, используя HTML.

(см. рис. 27)

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 27

*После этого рассмотрим сами методы (Добавления, Редактирования и Удаления), написанных на C#.*

Стоит также отметить, что работа всех методов схожа между собой, рассмотрим метод создания, но всё же некоторые из них содержат различающиеся элементы.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 28

Сначала следует обратить внимание на атрибуты [HttpPost] и [ValidateAntiForgeryToken]

Они используются в ASP.NET Core для обеспечения безопасности и защиты от атак подделки межсайтовых запросов (CSRF - Cross-Site Request Forgery).

[HttpPost]: Этот атрибут указывает, что метод действия должен обрабатывать только POST-запросы, а не GET-запросы. POST-запросы обычно используются для отправки данных с формы на сервер. Использование атрибута [HttpPost] гарантирует, что метод действия будет вызываться только при получении POST-запроса, а не при обращении по URL-адресу через браузер.

[ValidateAntiForgeryToken]: Этот атрибут помогает предотвратить атаки CSRF. CSRF-атака возникает, когда злоумышленник пытается выполнить операцию от имени пользователя, используя его аутентификационные данные. Атрибут [ValidateAntiForgeryToken] генерирует и проверяет токен запроса (CSRF-токен), который автоматически добавляется к формам в представлениях.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 29

Принцип работы следующий – программа отвечает за обработку запроса на создание новой категории (объекта Category) и выполнение соответствующих операций.отвечает за обработку запроса на создание новой категории (объекта Category) и выполнение соответствующих операций.

В метод действия передается параметр obj типа Category, который представляет объект категории, который будет создан.

Например, в методе создания нового пользователя мы не запрашиваем проверку о существовании пользователя с конкретным индивидуальным номером. А в методах редактирования и удаления мы её используем:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 30

Код выполняет следующие шаги:

Проверяет, что id не равно null и не равно нулю. Если это условие не выполняется, то возвращается результат NotFound(), что означает, что запрошенная категория не найдена.

Если id проходит проверку, то выполняется поиск категории в базе данных по заданному идентификатору. Для этого используется метод \_db.Categories.Find(id), где \_db представляет экземпляр контекста базы данных. Метод Find ищет запись в таблице категорий, основываясь на первичном ключе, и возвращает найденную категорию.

Если categoryFromdDb (категория, найденная в базе данных) равна null, то снова возвращается результат NotFound(), так как категория с указанным идентификатором не найдена.

Если категория найдена, то возвращается представление View с передачей найденной категории (categoryFromdDb). Это представление может быть использовано для отображения информации о категории и подтверждения удаления.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 31

И уже на (рис. 31) можно увидеть метод самого Удаления. Аналогично и поступаем с методом редактирования.

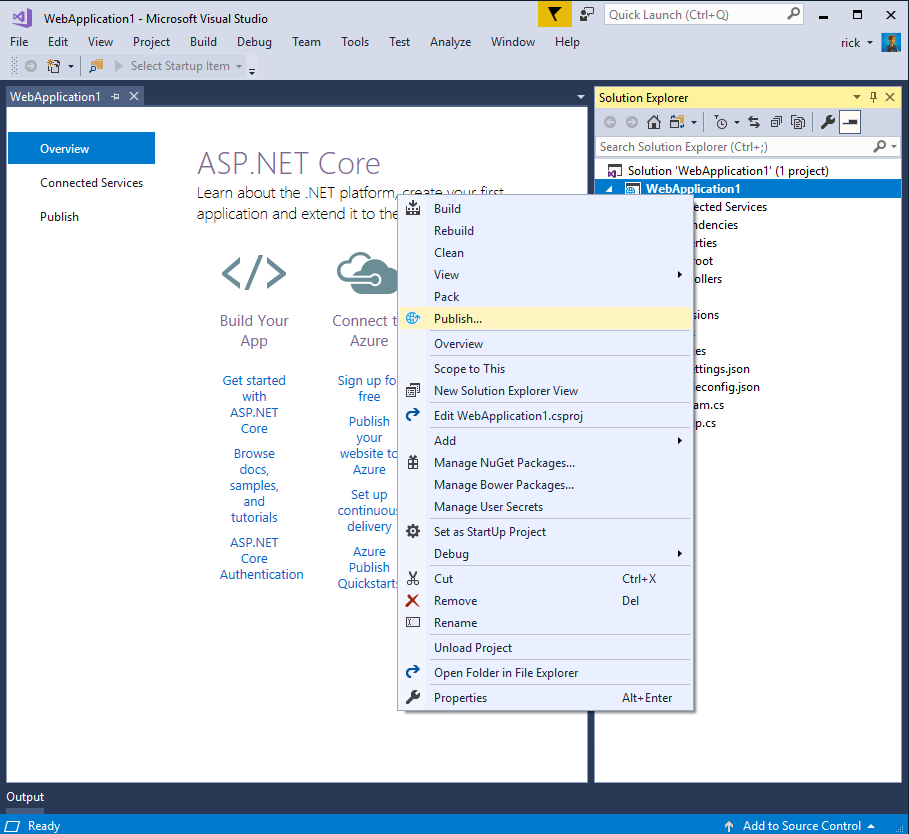
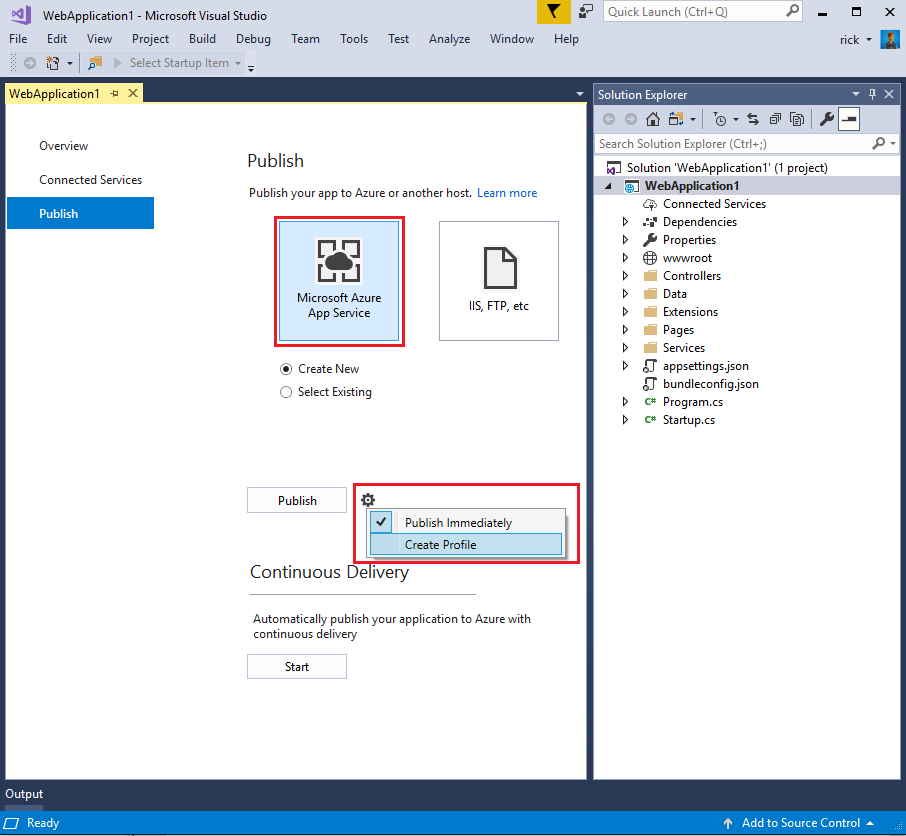
Из-за отсутствия доступа к сервису Microsoft Azure, загрузить сайт на хостинг не удастся

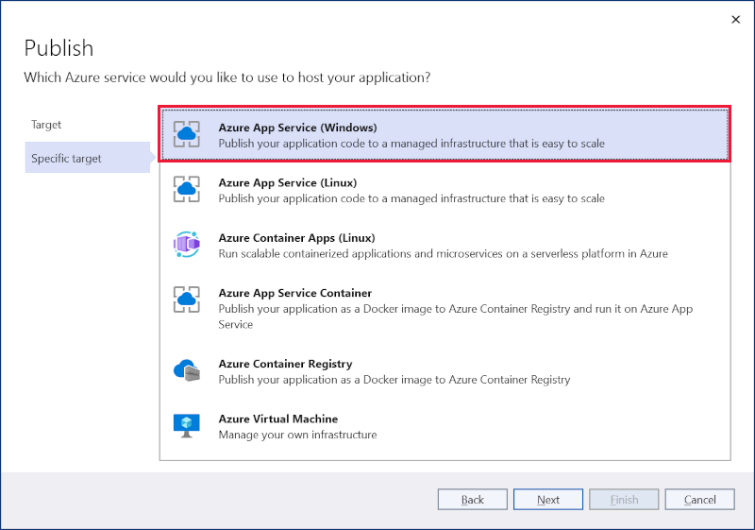
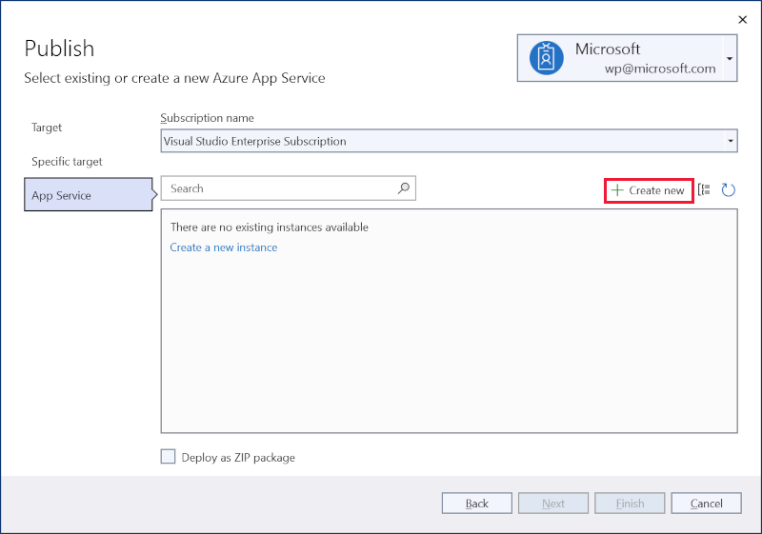
Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

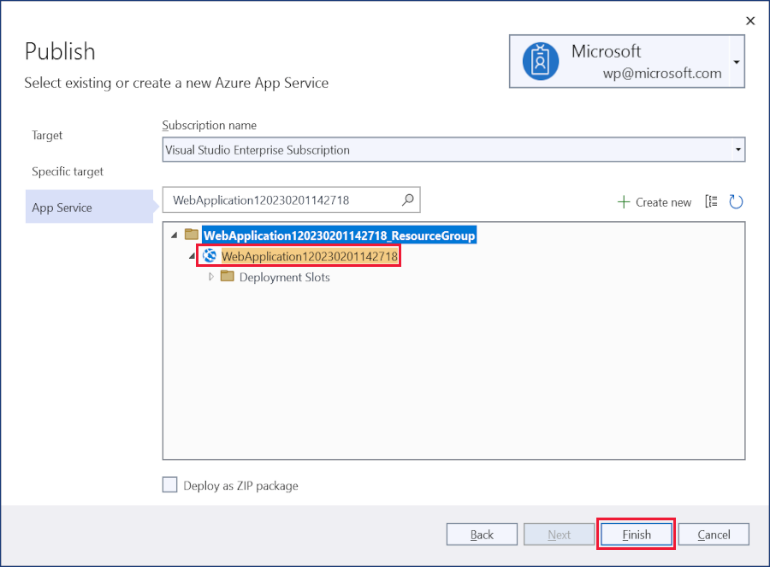
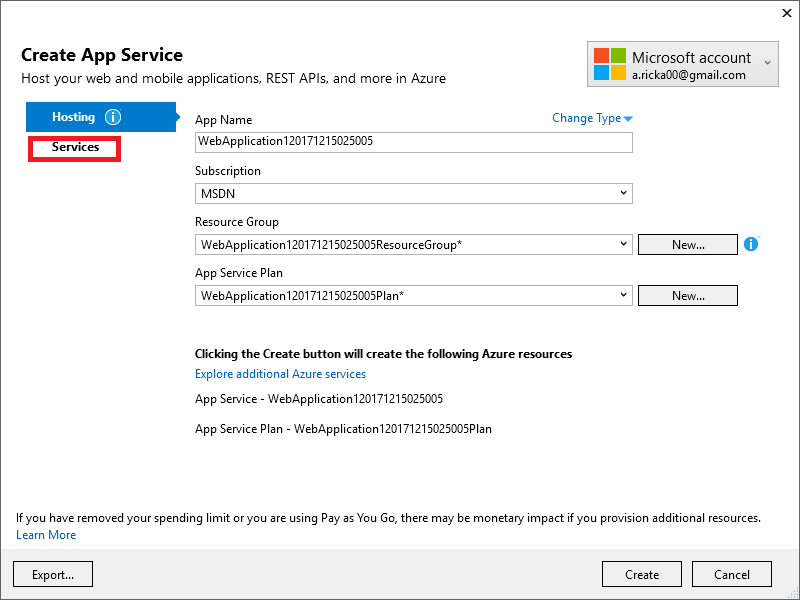
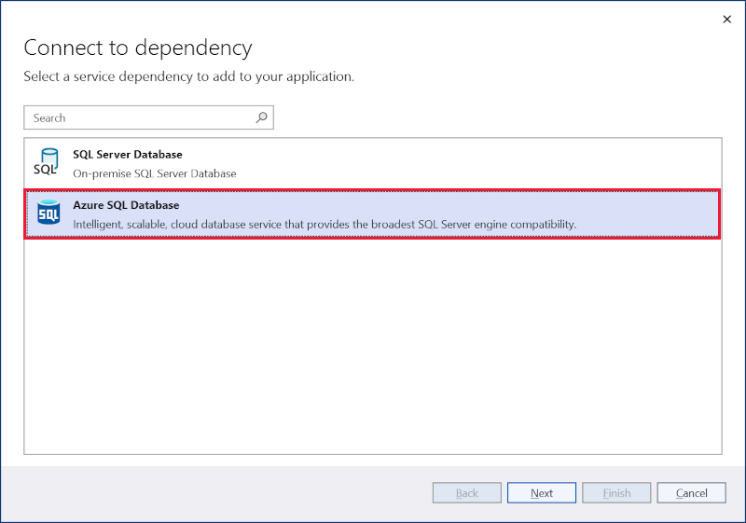
Автоматически созданное описание

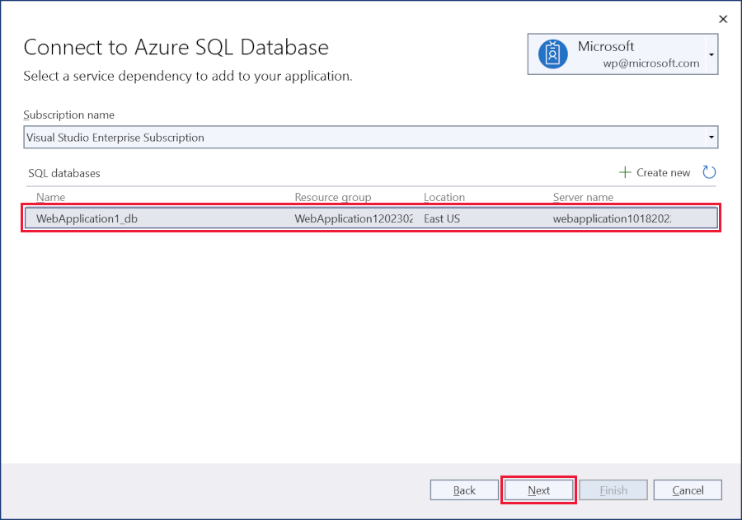
Рис. 32

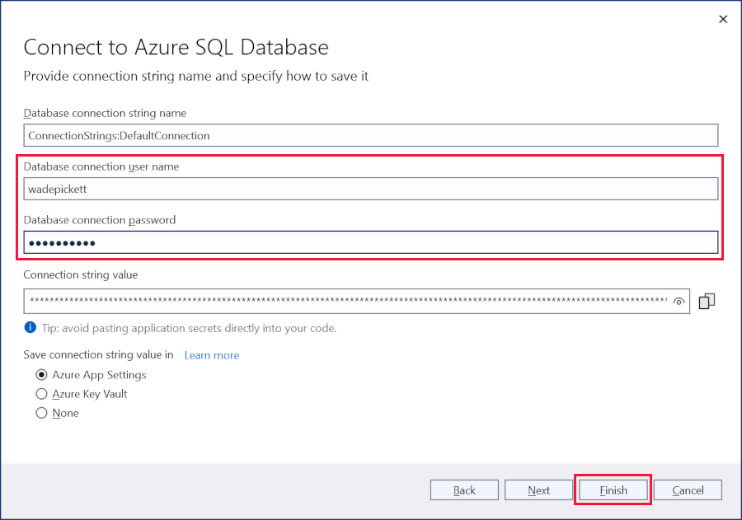
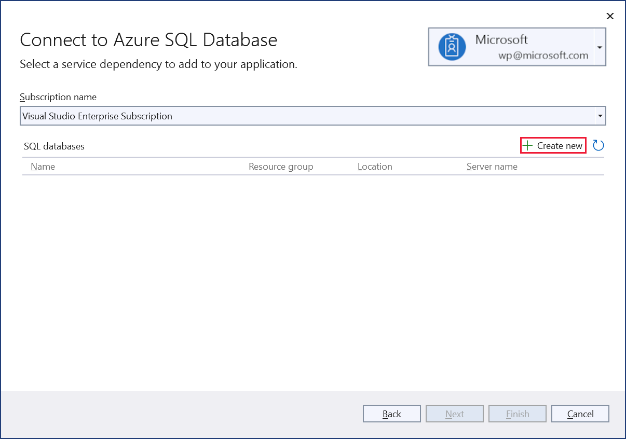
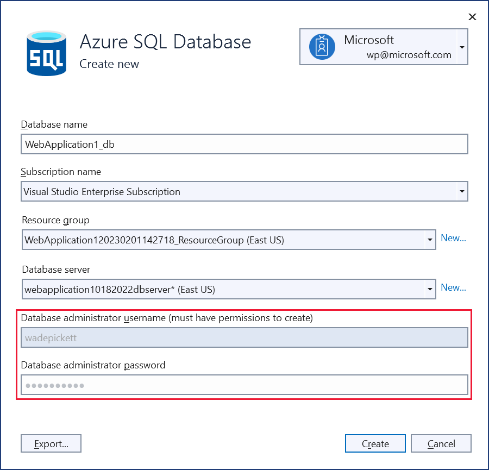
В следующие шаги (слева направо) покажут, как сделать это с имеющейся возможностью работать с Azure.











# 

# Диалоговое окно Заключительный шаг

# Заключение

В процессе выполнения работы был создан веб-сайт на C# .NET CORE, включающий планирование и проектирование функциональной и серверной частей, создание базы данных на SQL, верстку интерфейса с использованием HTML и CSS, а также тестирование и отладку приложения. Для реализации функциональности были использованы библиотеки Entity Framework и LINQ, позволяющие создать базу данных и взаимодействовать с ней, а также встроенный механизм авторизации и аутентификации пользователей в .NET CORE. JavaScript был использован для создания уведомлений об изменении БД, а также для реализации интерактивных элементов на страницах, таких как кнопки, формы и модальные окна.

# Список использованных источников и литературы

1. ".NET Core documentation", <https://docs.microsoft.com/en-us/dotnet/core/>
2. "SQL Server documentation", <https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15>
3. "HTML & CSS: The Definitive Guide" by Eric Meyer and Estelle Weyl
4. "CSS: The Missing Manual" by David McFarland
5. "JavaScript: The Definitive Guide" by David Flanagan
6. "Pro ASP.NET Core MVC" by Adam Freeman
7. "C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development" by Mark J. Price
8. "Pro SQL Server 2019 Administration" by Peter A. Carter
9. "Beginning SQL Server 2019 Administration: From the Data Professional’s Perspective" by Peter A. Carter
10. "Entity Framework Core in Action" by Jon Smith
11. "Learning SQL" by Alan Beaulieu
12. "HTML and CSS: Design and Build Websites" by Jon Duckett
13. "CSS Secrets: Better Solutions to Everyday Web Design Problems" by Lea Verou
14. "Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques, 3rd Edition" by Ben Frain
15. "SQL Queries for Mere Mortals: A Hands-On Guide to Data Manipulation in SQL" by John L. Viescas

Литература из России:

1. "ASP.NET Core 2.0. Разработка веб-приложений" автора Александра Шевчука.
2. "ASP.NET Core. Разработка веб-приложений" автора Михаила Головача.