ASP.NET C# Programming – Building Web Applications with .NET Core

Abdulkarim M. Jamal Kanaan

IBM Data Science Professional
DeepLearning. AI TensorFlow Developer
Microsoft Certified Professional

Module 1: Understanding ASP.NET & .NET Core

What Is ASP.NET Core?

ASP.NET Core is a **modern, open-source, cross-platform** framework for building web apps and services.

Enables development of:

- ✓ *Dynamic, server-rendered web apps*
- ✓ *RESTful HTTP APIs*
- ✓ *Backend services consumed by mobile or desktop clients*

Diagram showing ASP.NET Core in the center connecting to Web App, API, Mobile App, Microservices, IoT.

Why ASP.NET Core?

- **Cross-platform:** Works on Windows, Linux, macOS.
- **High performance:** Built for scalability and speed.
- **Open source:** Backed by Microsoft and community contributors.
- **Unified programming model:** Web, API, and backend share the same runtime and libraries.
- **Cloud-ready:** Ideal for Azure and container deployments.

🌐 Icon-based chart comparing "Old ASP.NET Framework → ASP.NET Core" (cross-platform, performance, modularity).

The ASP.NET Core Architecture Overview

- ASP.NET Core applications are **layered** for separation of concerns.
- Framework handles **HTTP requests/responses** and routing.
- Developers implement **handlers** (controllers, Razor Pages) to process specific requests.
- Handlers call **business/domain logic**, which contains the application's true behavior.
- Domain logic interacts with **databases or external APIs** but remains framework-independent.

Understanding the Layers

Layer	Description	Example
Framework Layer	Manages HTTP pipeline, routing, dependency injection, configuration.	ASP.NET Core Middleware
Handler Layer	Responds to incoming requests; maps routes to logic.	MVC Controller / Razor Page
Business Logic Layer	Core rules and operations of your app; framework-agnostic.	C# Classes, Domain Models
Data/External Layer	Interfaces with databases, APIs, or storage.	EF Core, SQL Server, REST API

Relationship Between ASP.NET Core and .NET

- ASP.NET Core is **built on top of the .NET runtime**.
- Uses **.NET 7** (or later) for base libraries, performance improvements, and cross-platform execution.
- Developers use **C#** for coding business logic and frameworks (MVC, Razor Pages, Web API).
- This modular setup allows re-use of non-web code in other .NET apps (e.g., desktop, services).

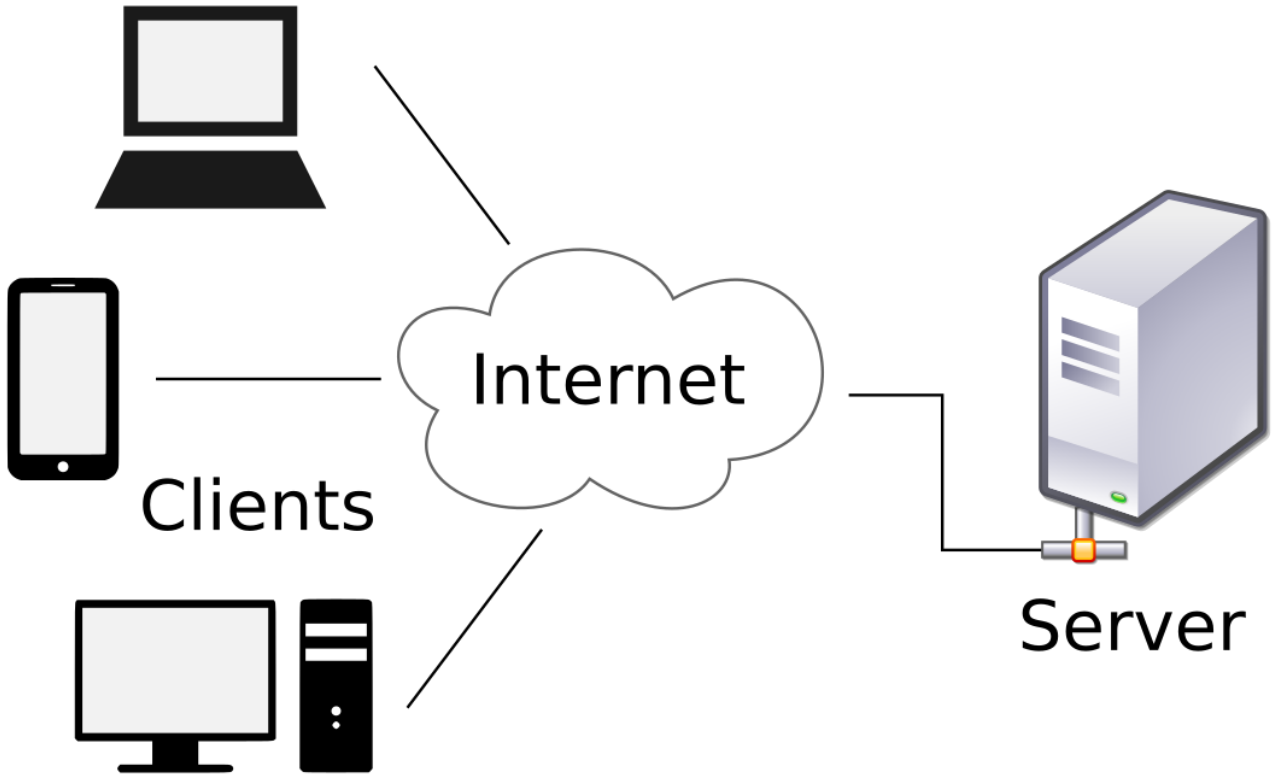
Understanding the Basics

Client-Side vs. Server-Side

Client-side (Frontend): Executes in the user's browser

Server-side (Backend): Executes on the web server

Both cooperate to deliver a seamless user experience



C. (2001, September 25). distributed application structure in computing. Retrieved September 16, 2025, from Wikipedia.org website: https://en.wikipedia.org/wiki/Client%E2%80%93server_model

Understanding the Web Client

Typically a **web browser** (Chrome, Firefox, etc.)

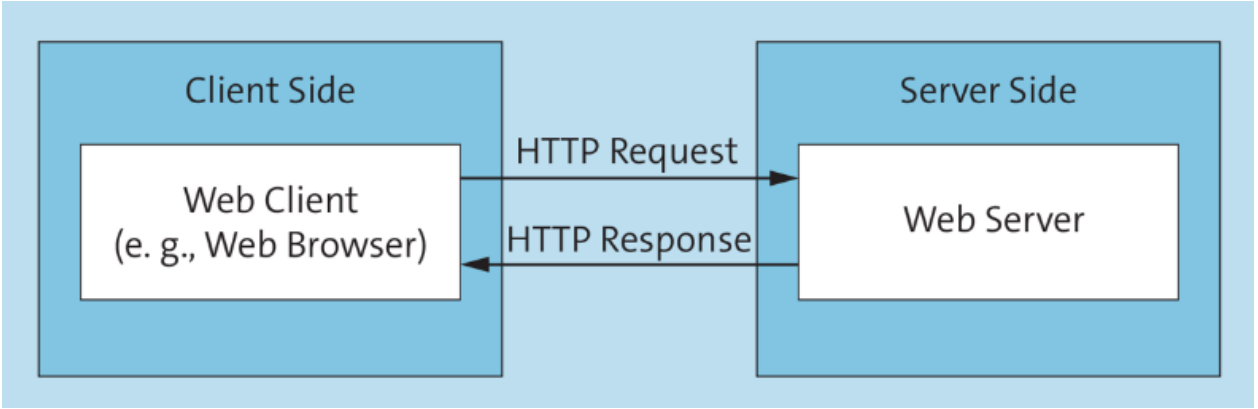
May also be:

- Screen readers
- Command-line HTML clients (e.g., curl, wget)
- **Headless browsers** (e.g., Puppeteer, Selenium)

Luiz Gustavo. (2022, March 15). Configuring Multiple Browsers with Selenium and Python - LuizDeAguiar. Retrieved September 16, 2025, from LuizDeAguiar website: https://www.luizdeaguiar.com.br/2022/03/configuring-multiple-browsers-with-selenium-and-python/

The Web Request Lifecycle

1. User enters a URL in the browser
2. Browser creates an **HTTP request**
3. Request sent to the **web server**
4. Server processes it and sends back an **HTTP response**
5. Browser renders the web page

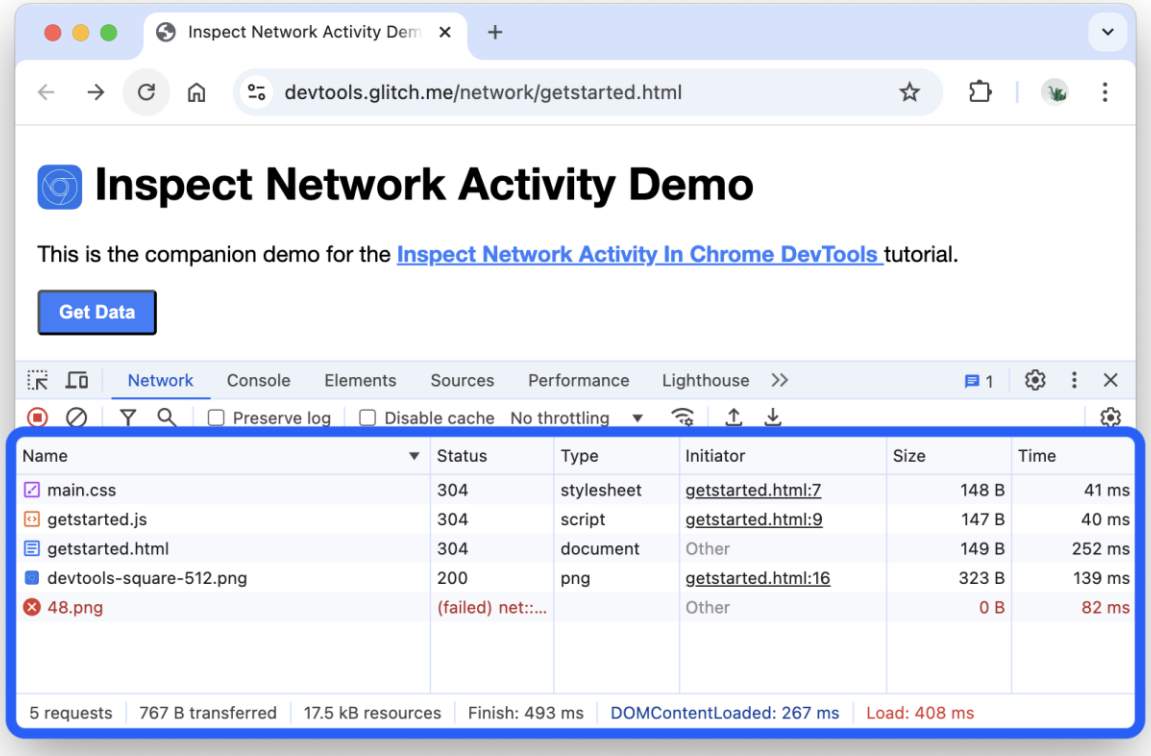


Automatic Resource Loading

Browser downloads:

- CSS files for styling
- JavaScript files for interactivity
- Images and media for presentation

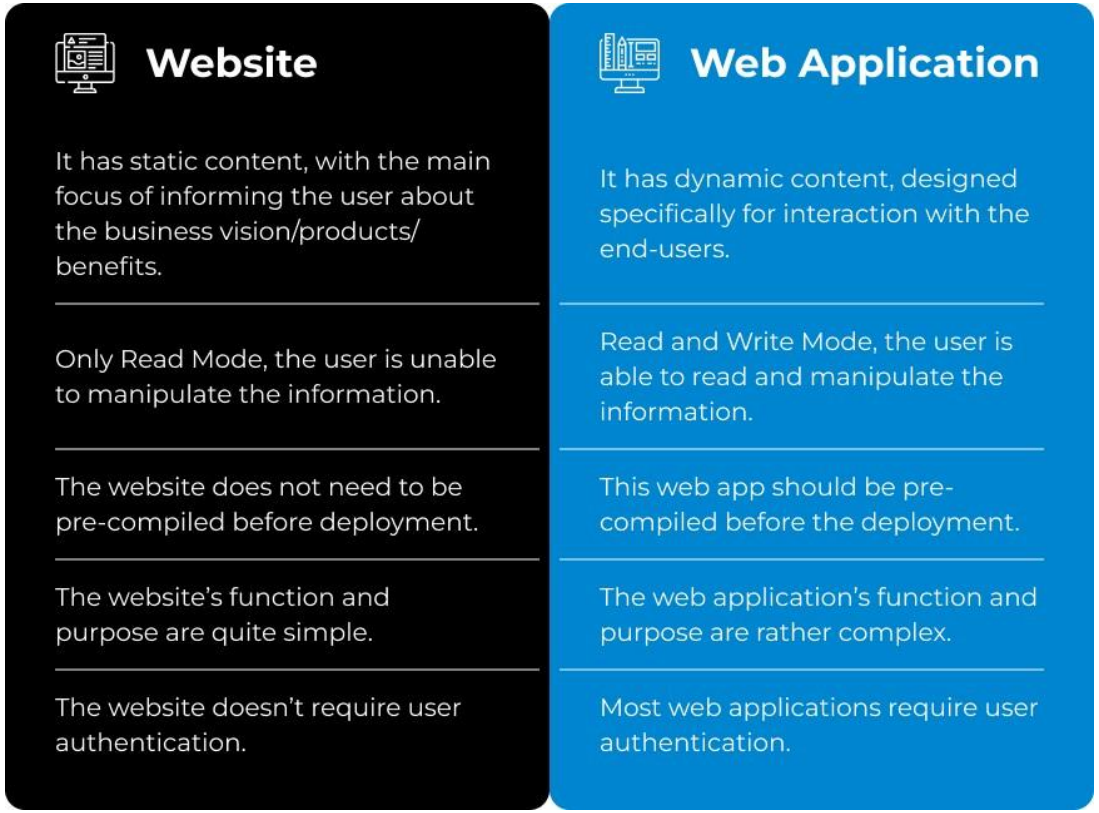
Rendering engine paints the final visual



Inspect network activity. (2024). Retrieved September 16, 2025, from Chrome for Developers website: https://developer.chrome.com/docs/devtools/network?hl=en

Terminology Clarification

Term	Description
Web Page	A single HTML document at a unique URL
Website	A collection of related web pages
Web Application	A dynamic, interactive site behaving like a desktop application (e.g., Google Docs)



Online WebP to JPG converter. (2025). Retrieved September 16, 2025, from Ezgif.com website: https://ezgif.com/webp-to-jpg/ezgif-89d953f36efcb.webp.html

From Request to Response

- ASP.NET Core apps revolve around **handling HTTP requests** and producing **HTTP responses**.
- Every time a user visits a web page, their browser communicates with the web server over the **HTTP protocol**.
- ASP.NET Core extends this standard process to generate **dynamic content** rather than just serving static files.

Relationship between URLs, Domains, and IP Addresses

📍 Understanding how web addresses work

Introduction

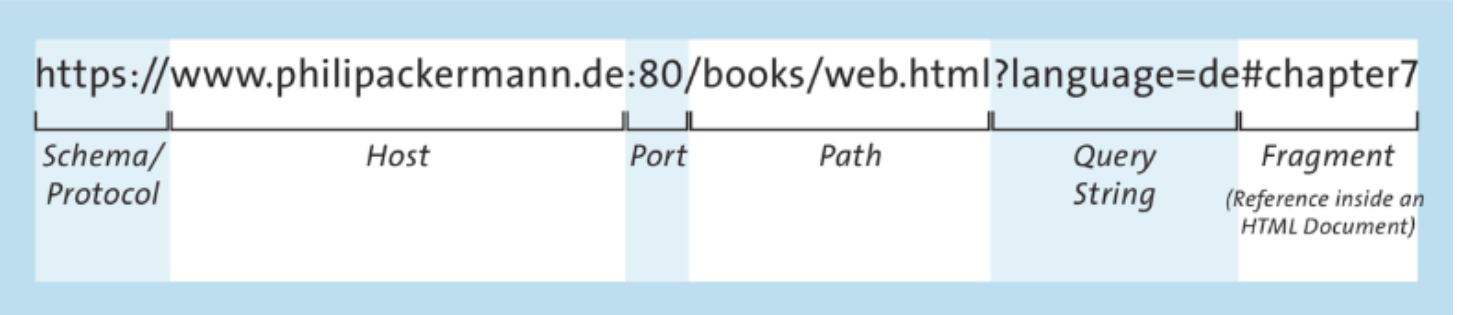
- The address entered in the browser is called a URL
- Examples:

  https://www.philipackermann.de/

  https://www.webdevhandbuch.de/service/api/users?search=max
- URLs connect users → web servers → resources

URL Structure

- A URL is made up of multiple parts:
  - ✓ *Protocol / Schema*
  - ✓ *Host name (Domain + Subdomain + TLD)*
  - ✓ *Port*
  - ✓ *Path*
  - ✓ *Query string*
  - ✓ *Fragment*



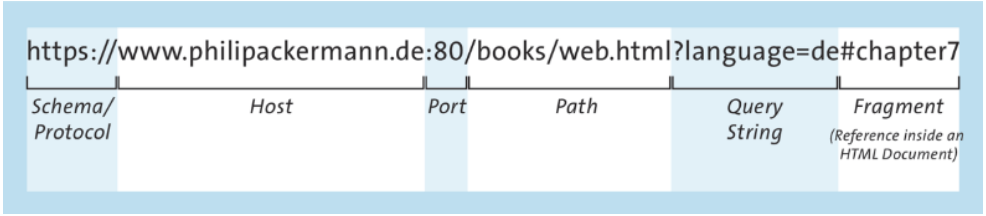
Communication Protocols

HTTP/HTTPS: Used for communication between web browsers and web servers.

FTP (File Transfer Protocol): For transferring files between client and server.

SMTP/IMAP: Protocols used for sending and receiving emails.

TCP/IP: The underlying transport protocol that ensures reliable delivery of packets across networks

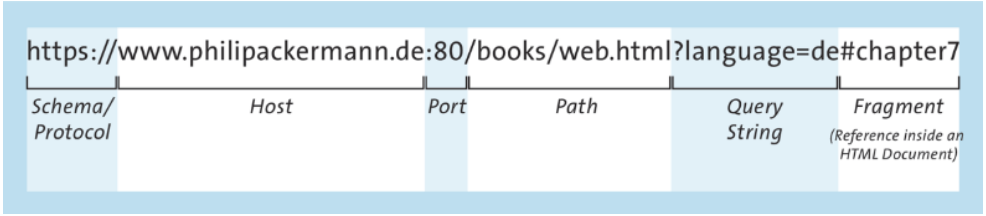


Host Name

Identifies the web server:

Example:
www.cleancoderocker.com

- *Subdomain: www*
- *Domain: cleancoderocker*
- *Top-level domain (TLD): .com*



Port

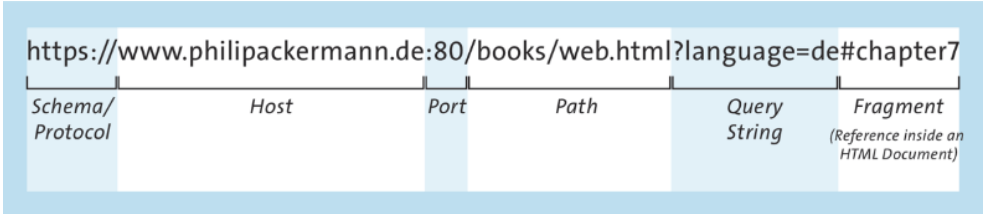
Specifies the communication channel:

Default ports:

- ✓ *80 → HTTP*
- ✓ *443 → HTTPS*

Custom ports common in development

- ✓ *Example: http://localhost:8080/myservice/api/users*



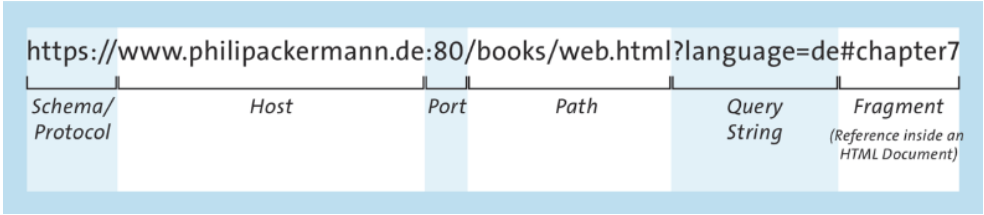
Path

Points to the resource on the server:

Example:URL:

- ✓ https://www.philipackermann.de/static/img/profile.jpg
- ✓ *Path: /static/img/profile.jpg*

Always uses forward slashes /

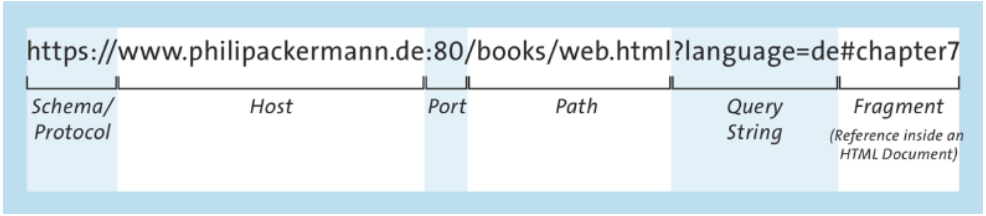


Query String

Passes extra information to the server:

- Introduced by ?
- Format: key=value pairs separated by &
- Example:

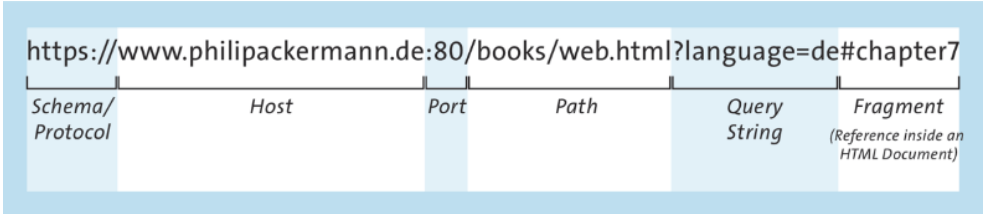
  https://www.philipackermann.de/example?search=javascript&display=list



Fragment

Targets a specific location in a webpage:

- Introduced by #
- Example:
  - ✓ *https://www.philipackermann.de/example#chapter5*
- Browser jumps directly to that section



Connecting to Domains & IP Addresses

- Domain Name System (DNS): translates domain names → IP addresses
- User enters: www.webdevhandbuch.de
- DNS resolves to: 192.168.10.15 (example IP)
- Browser requests the resource from that IP

Staff, E. (2016, June 28). Beginners Guide to Domain Name – WebNots. Retrieved September 16, 2025, from WebNots website: https://www.webnots.com/all-you-should-know-about-domain-name/

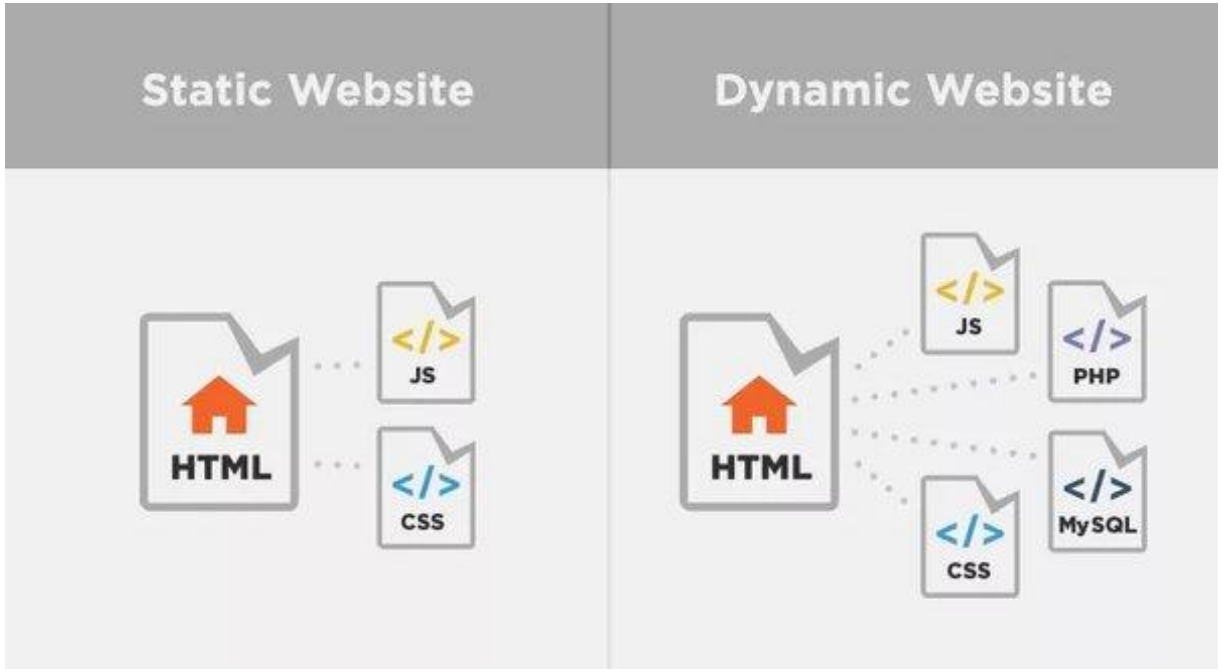
Static vs. Dynamic Web Pages

Static Web Pages:

- Served directly as files (HTML, CSS, JS, images)
- No processing on server

Dynamic Web Pages:

- Server runs scripts (e.g., PHP, ASP.NET, Node.js)
- Content is **generated** or **personalized** before being sent



What are static and dynamic Web pages? (2019). Retrieved September 16, 2025, from Quora website: https://www.quora.com/What-are-static-and-dynamic-Web-pages

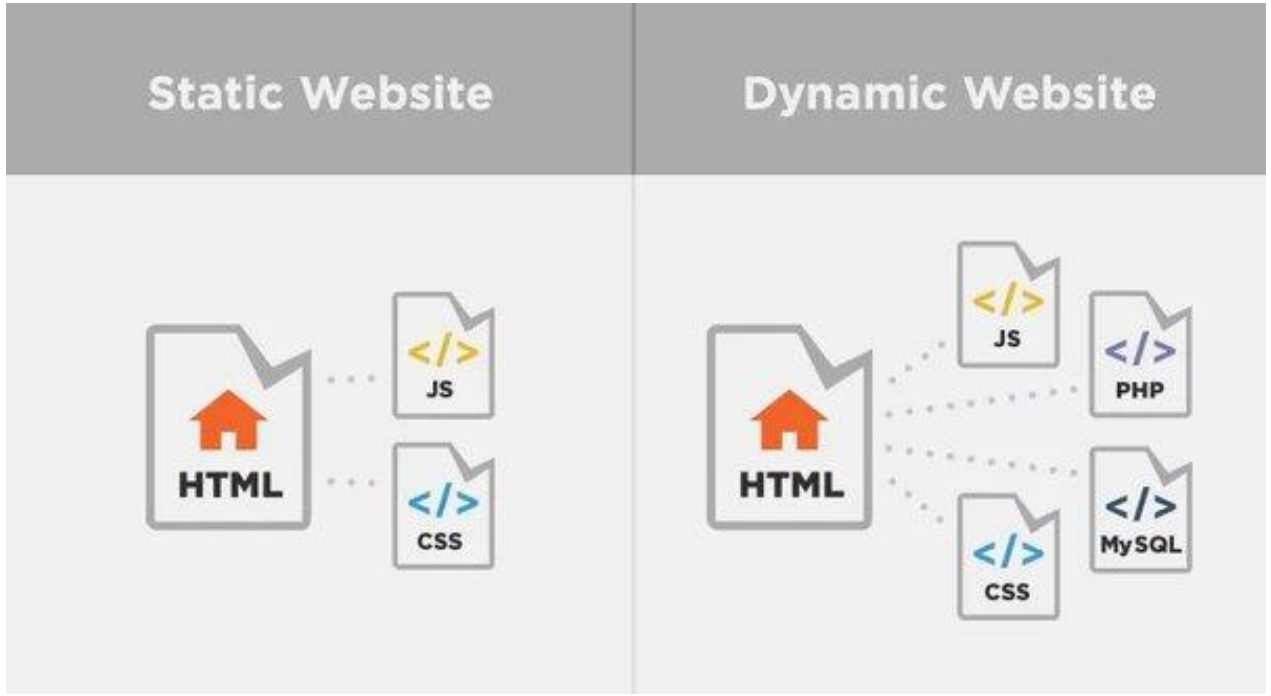
Static vs Dynamic Content

Traditional web servers serve **static files** (fixed HTML, images).

ASP.NET Core can generate **dynamic content** based on:

- User input
- Database queries
- API results

The framework runs your **server-side logic** to produce unique HTML or JSON for each request.



What are static and dynamic Web pages? (2019). Retrieved September 16, 2025, from Quora website: https://www.quora.com/What-are-static-and-dynamic-Web-pages

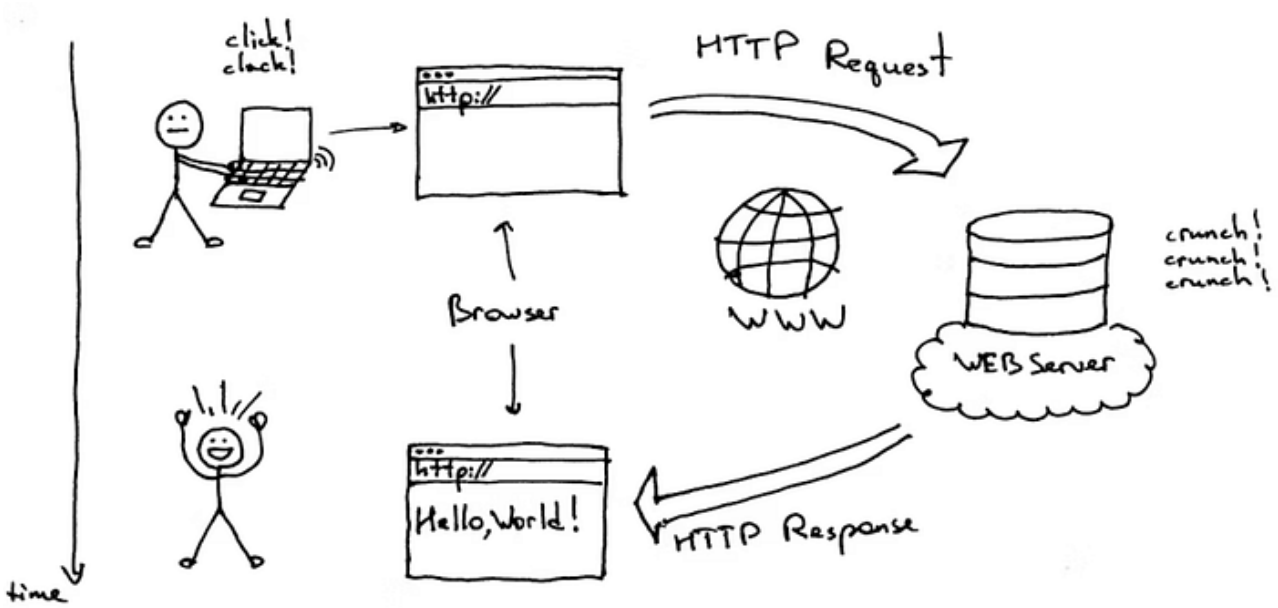
From Browser to Server

ASP.NET Core still communicates using the **standard HTTP protocol**.

What changes is **how the server handles** the incoming request.

ASP.NET Core takes care of everything server-side:

- Validating requests
- Managing authentication and sessions
- Generating HTML or API responses



Khan, R. R. (2024, February 2). Demystifying the Journey of a Web Request: From Browser to Server and Beyond. Retrieved October 24, 2025, from Medium website: https://medium.com/@rukhsarkhan4198/demystifying-the-journey-of-a-web-request-from-browser-to-server-and-beyond-f8a706a847c5

The Request Journey in ASP.NET Core

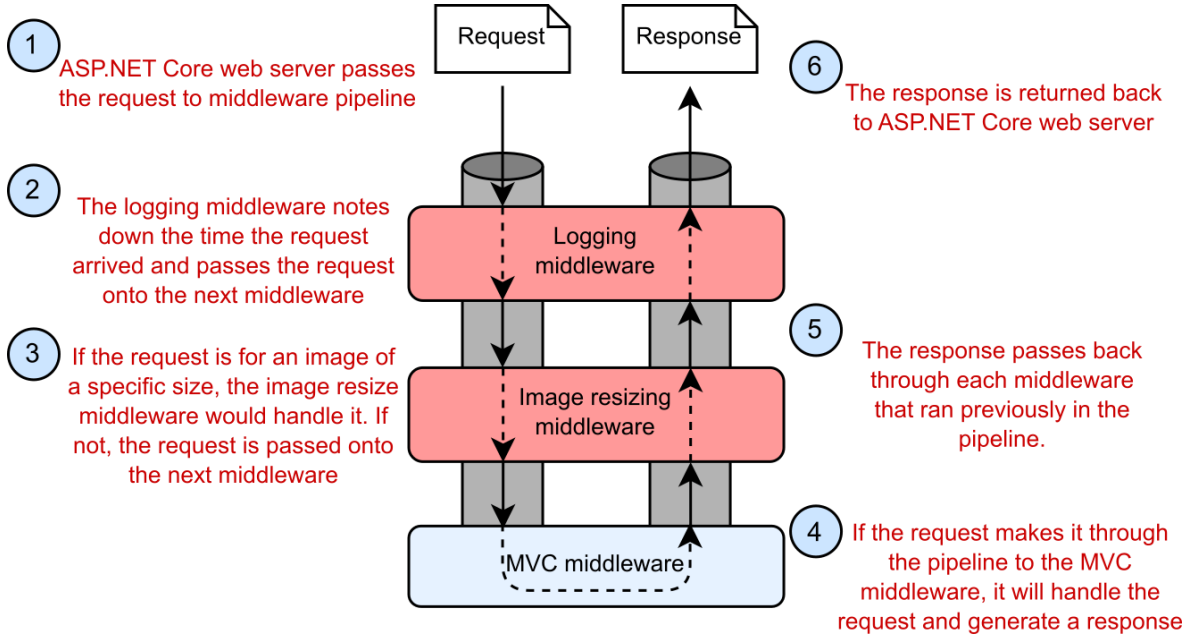
The **browser** sends an HTTP request to the server.

The **ASP.NET Core web server (Kestrel)** receives it.

Kestrel **forwards the request** to the ASP.NET Core pipeline for processing.

The **application logic** runs and creates a response.

The response is sent **back through Kestrel** to the browser.



Lock, A. (2017, December 28). ASP.NET Core in Action - What is middleware? Retrieved October 24, 2025, from Andrew Lock | .NET Escapades website: https://andrewlock.net/asp-net-core-in-action-what-is-middleware/

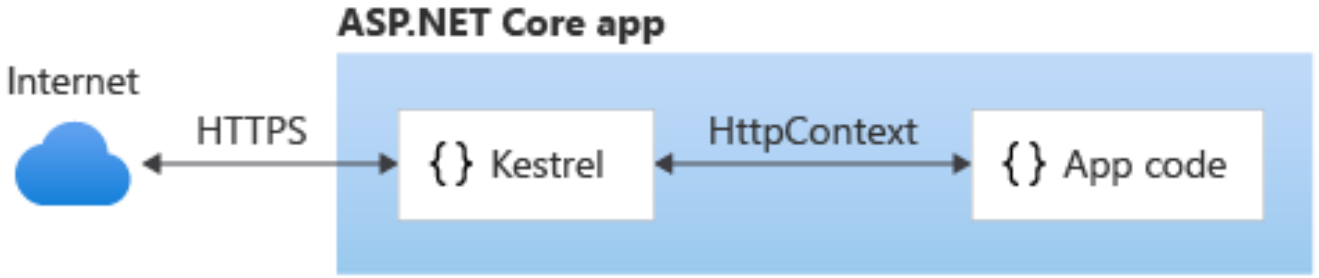
Meet Kestrel: ASP.NET Core's Built-In Web Server

Kestrel is a fast, lightweight, cross-platform web server.

It comes **built into** every ASP.NET Core application—no need for IIS during development.

Responsibilities include:

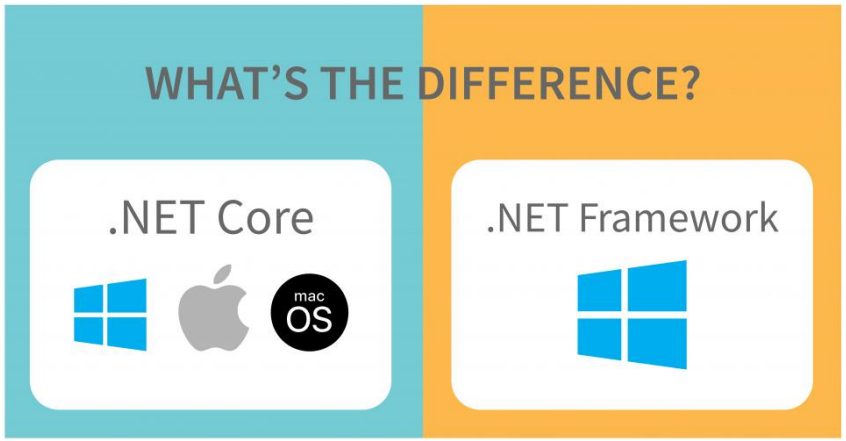
- Listening for incoming network requests
- Reading raw HTTP data
- Building an **HttpContext** object to represent the request
- Passing the context to your app for processing

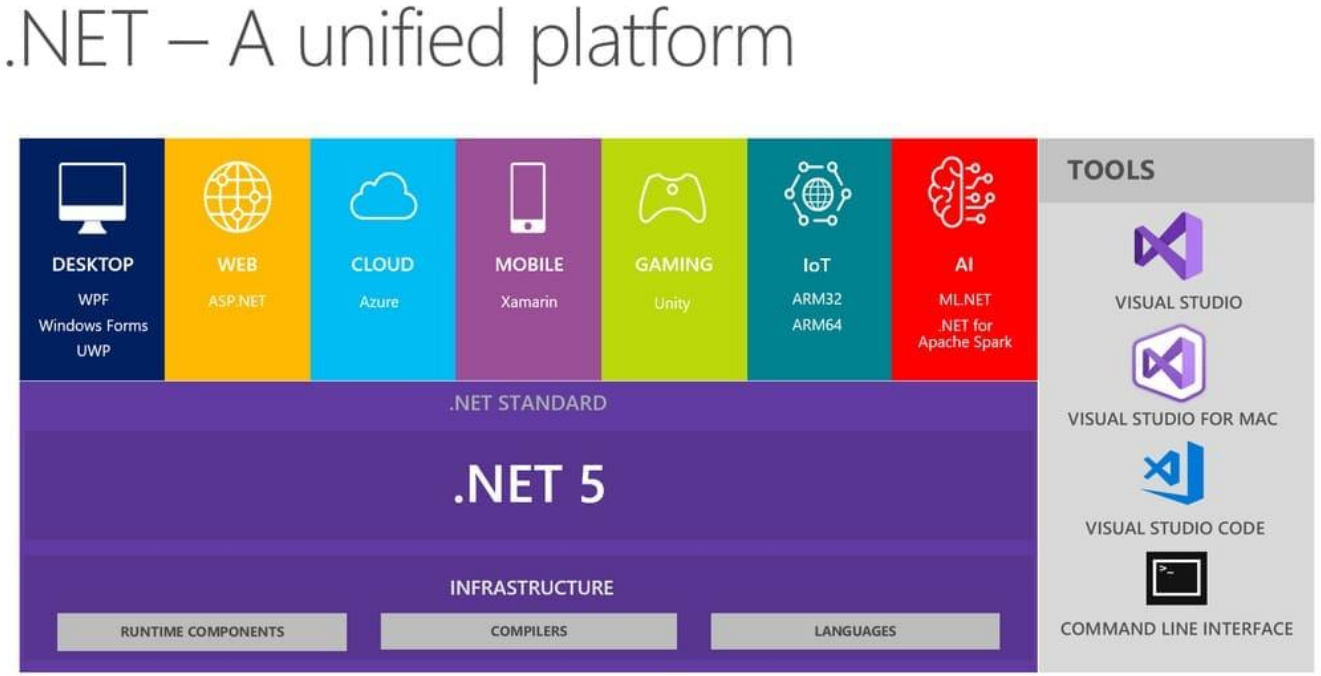


Understanding .NET, .NET Framework, and .NET Core

.NET Framework: Original, Windows-only platform.

.NET Core: Rewritten, cross-platform version—faster and modular.





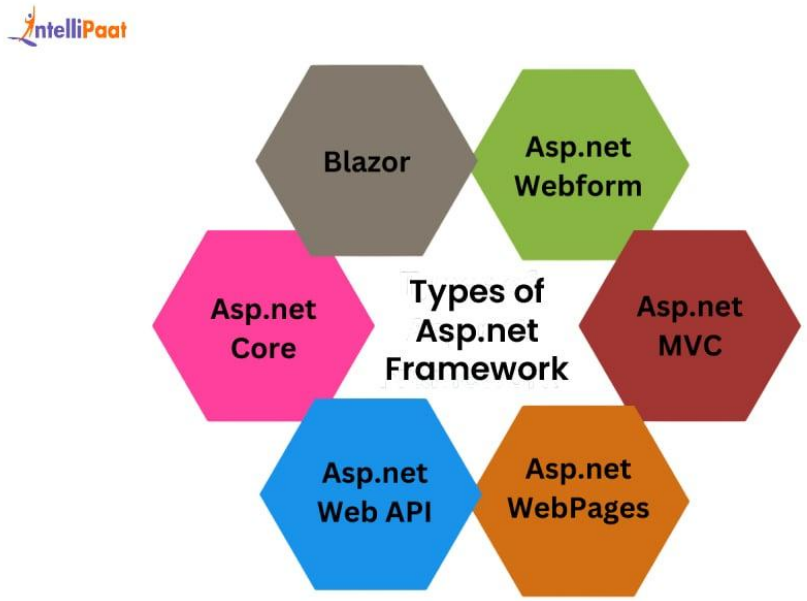
Team, A. E. (2021, December 24). The Good and the Bad of .NET Framework Programming. Retrieved October 24, 2025, from AltexSoft website: https://www.altexsoft.com/blog/the-good-and-the-bad-of-net-framework-programming/

The ASP.NET Core Application Frameworks

ASP.NET Core supports multiple **application frameworks**, each optimized for different needs:

1. MVC Framework
2. Razor Pages
3. Blazor
4. Utility Frameworks (EF Core, Identity)

Each framework plays a specific role while sharing the same runtime and platform features.



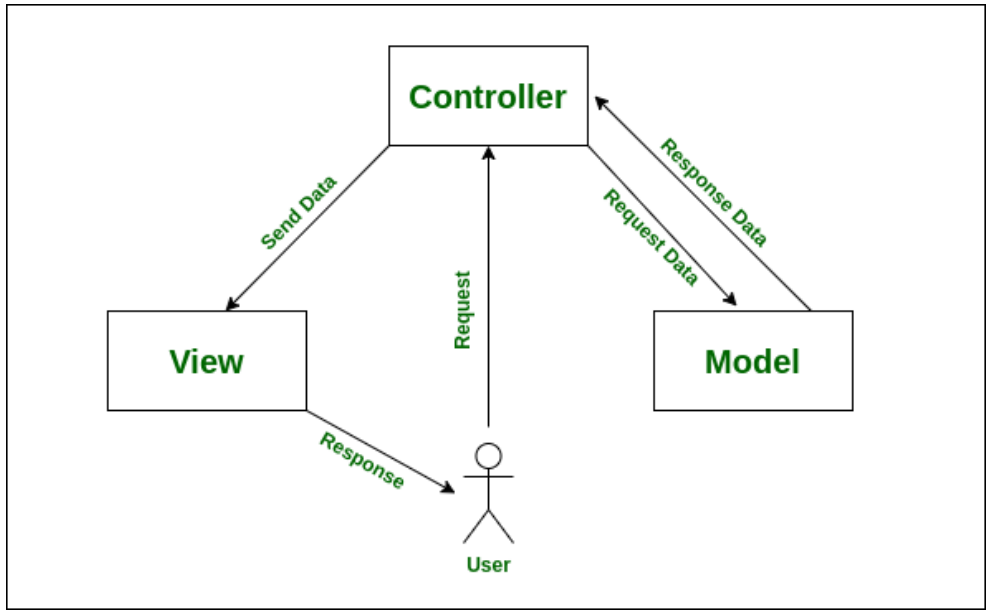
Alam, A. (2025, May 26). What is ASP.NET. Retrieved October 24, 2025, from Intellipaat website: https://intellipaat.com/blog/what-is-aspnet/

MVC Framework

MVC (Model–View–Controller) pattern promotes **separation of concerns**:

- **Model:** Data and business logic
- **View:** User interface (HTML)
- **Controller:** Manages requests/responses

Originally built on **ASP.NET Web Forms**, later redesigned for **.NET Core**.



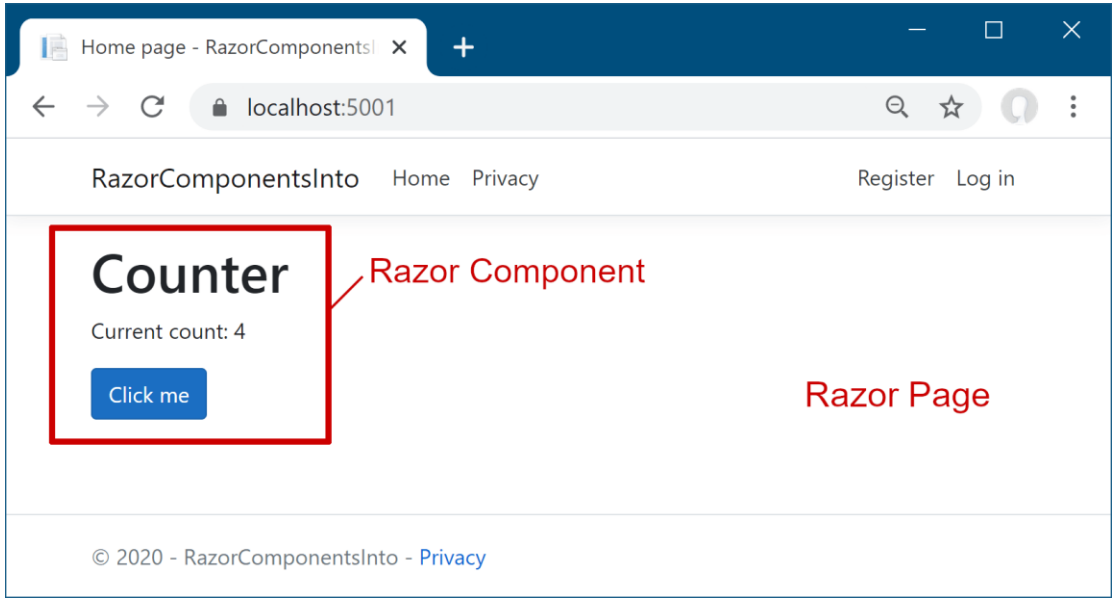
https://www.geeksforgeeks.org/software-engineering/benefit-of-using-mvc/

Razor Pages

- Designed for **simplicity and quick development**.
- Each page combines markup (HTML) and C# code in one file.
- Great for **lightweight apps**, **admin tools**, or **dashboards**.
- Can coexist with MVC within the same project.

Example Use:

- MVC for main website logic
- Razor Pages for settings, reports, or utility pages



Lock, A. (2020, March 17). Don't replace your View Components with Razor Components. Retrieved October 24, 2025, from Andrew Lock | .NET Escapades website: https://andrewlock.net/dont-replace-your-view-components-with-razor-components/

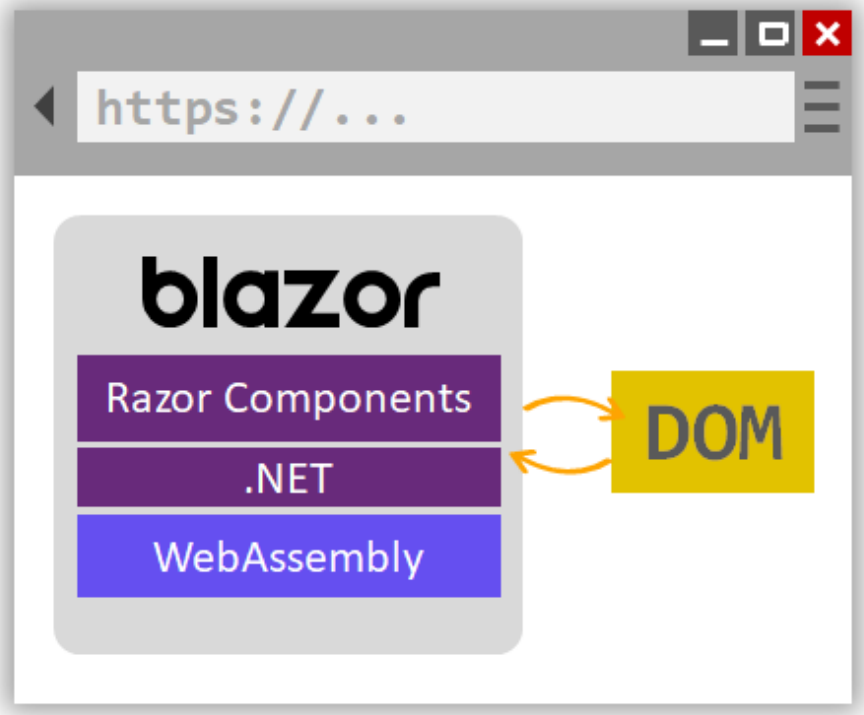
Blazor Framework

Blazor allows building **interactive web apps using C#** instead of JavaScript.

Two hosting models:

1. **Blazor Server** – Executes C# on the server via a live connection (SignalR).
2. **Blazor WebAssembly (WASM)** – Runs C# directly in the browser.

Ideal for developers who prefer staying within the **.NET ecosystem**.



guardrex. (2024, November 12). ASP.NET Core Blazor. Retrieved October 24, 2025, from Microsoft.com website: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-9.0

Utility Frameworks

Entity Framework Core (EF Core):

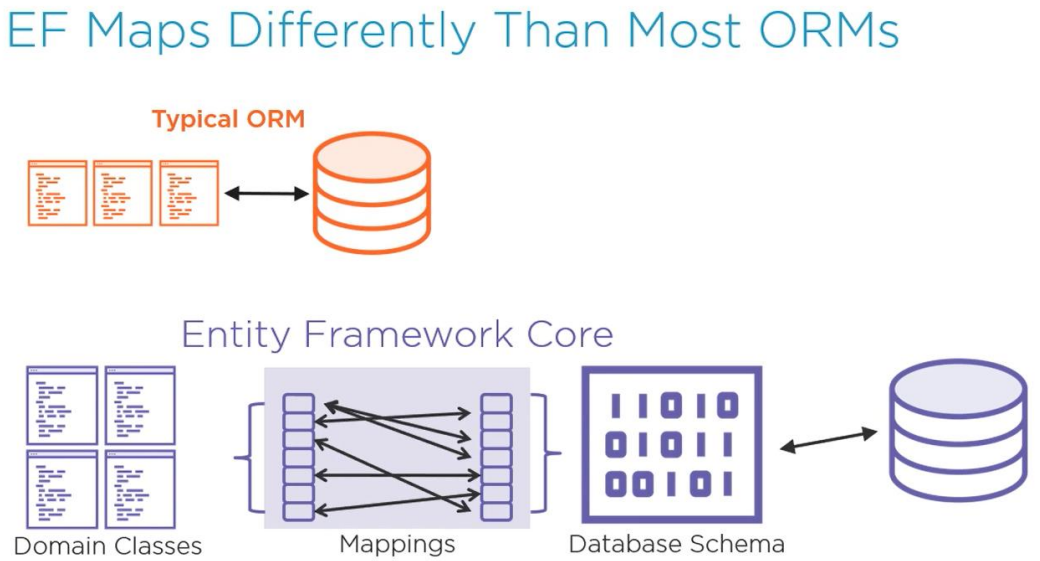
Object-relational mapper (ORM) for database access.

Translates C# models into SQL queries.

ASP.NET Core Identity:

Handles **user authentication** and **authorization**.

Integrates with external providers (Google, Microsoft, etc.).

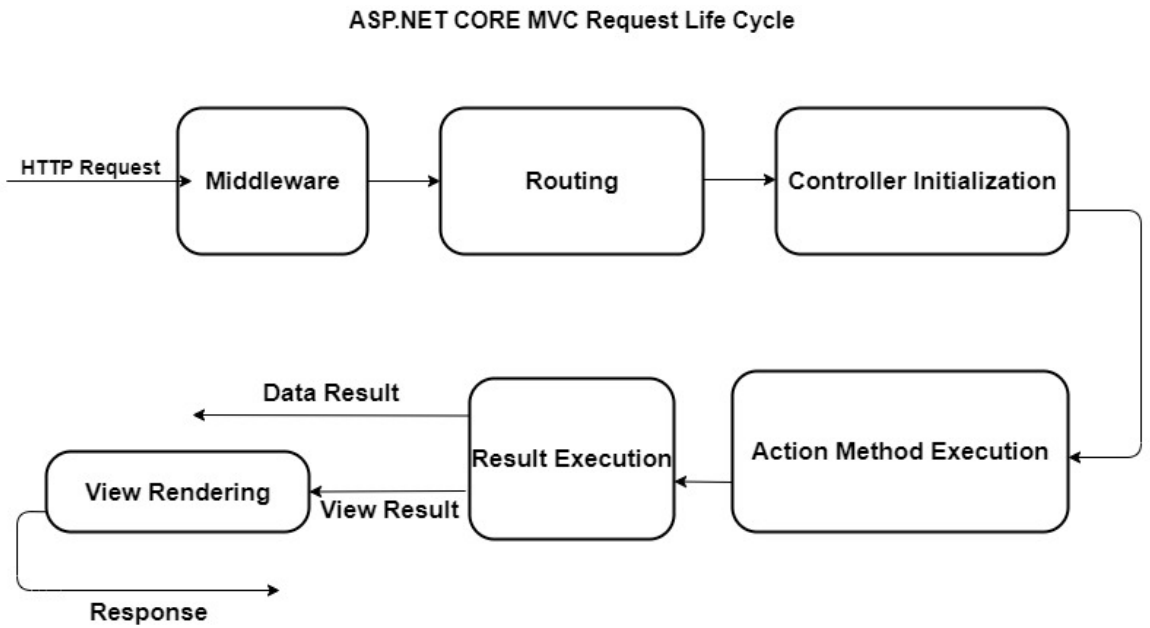


https://www.facebook.com/mason.tope. (2022, January 16). Learning Entity Framework Core 3.1 for C# | Active Programmer. Retrieved October 24, 2025, from Active Programmer website: https://activeprogrammerhq.com/entity-framework-core-getting-started-with-3-1/

Life Cycle Recap

Request Flow Summary

1. Browser → Reverse Proxy (optional)
2. **Kestrel** builds HttpContext
3. **Middleware pipeline** processes the request
4. **Endpoint** generates the response
5. Response flows back through middleware → Kestrel → Browser



Yadav, A. (2020, January 16). ASP.NET Core MVC Request Life Cycle. Retrieved October 24, 2025, from C-sharpcorner.com website: https://www.c-sharpcorner.com/article/asp-net-core-mvc-request-life-cycle/

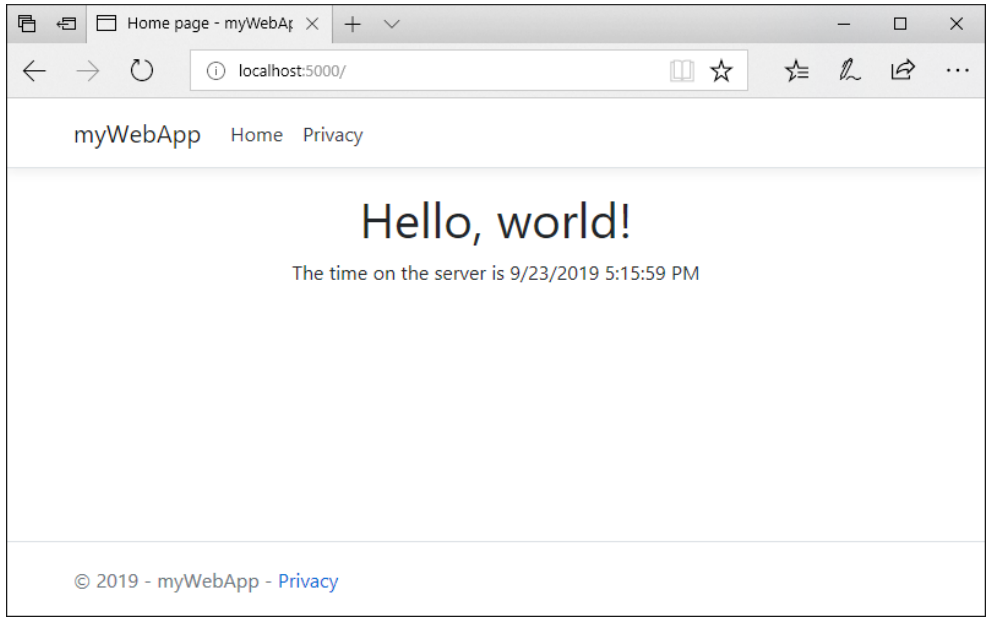
Creating Your First ASP.NET Core Application

Build your first ASP.NET Core app

A simple **Minimal API** returning "Hello World!"

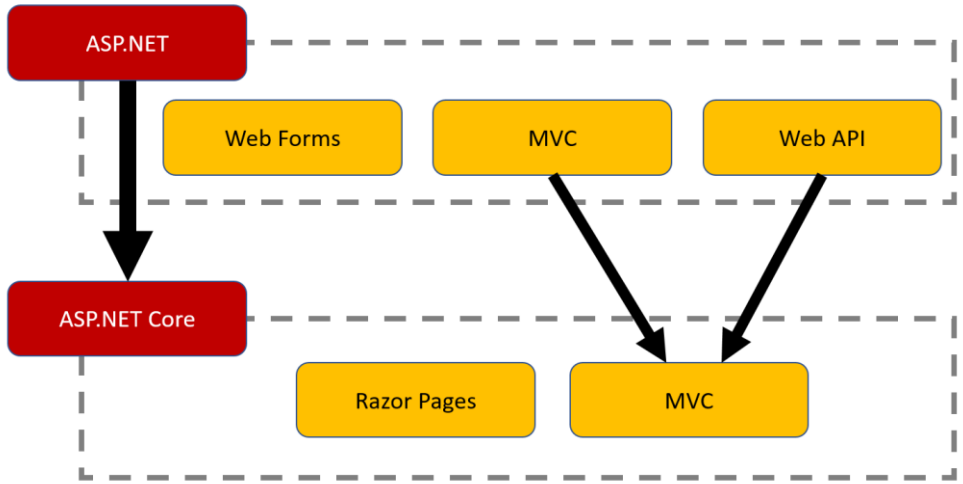
Demonstrates the **core steps** of building and running an ASP.NET Core project.

Works across platforms: **Windows, macOS, Linux**.



Available ASP.NET Core Templates

Template	Description	Output
Minimal API	Lightweight HTTP API (JSON)	For SPAs & mobile apps
Razor Pages	Server-rendered HTML pages	Websites & dashboards
MVC	Full Model–View–Controller pattern	Enterprise apps
Web API	Advanced REST APIs	Data services



ASP.NET Core - A Complete Rewrite of ASP.NET. (2017). Retrieved October 24, 2025, from Danielcrabtree.com website: https://www.danielcrabtree.com/blog/333/asp-net-core-a-complete-rewrite-of-asp-net

C#

Understanding Variables and Constants

Variables store data values in memory.

Declaration syntax: dataType variableName = value;

Constants hold fixed values with const keyword.

Naming conventions: camelCase, meaningful names.

```
int age = 25;

const double Pi = 3.14159;
```

Exploring Data Types in C#

Value types: int, double, char, bool, struct.

Reference types: string, class, array, object.

Nullable types: int?, bool?.

Memory difference: Stack vs Heap.

```
string name = "Ali";
bool isActive = true;
```

Working with Type Conversion and Casting

Implicit conversion: safe, automatic widening (e.g., int → double).

Explicit casting: manual narrowing using (type).

Parsing and Convert class:

```
int num = int.Parse("100");
double d = Convert.ToDouble(num);
```

```
long big = 3_000_000_000;
int small = (int)big;

try
{
    small = Convert.ToInt32(big);
}
catch (OverflowException ex)
{
    Console.WriteLine(ex.Message);
}
```

Using Operators and Expressions

Arithmetic: + - * / %

Comparison: == != > < >= <=

Logical: && || !

Assignment: = += -=

Ternary: condition ? trueValue : falseValue

```
int result = (x > y) ? x : y;
```

Control Structures Overview

Used to control program flow.

Selection: if, else, switch

Iteration: for, while, do-while, foreach

Jump statements: break, continue, return.

Flowchart diagrams recommended.

```
int[] numbers = { 1, 2, 3 };
for (int i = 0; i < numbers.Length; i++)
{
    Console.WriteLine(numbers[i]);
}
```

Conditional Statements (if, switch)

```
if (score >= 80)
Console.WriteLine("A");

else if (score >= 60)
Console.WriteLine("B");

else Console.WriteLine("C");
```

```
DayOfWeek day = DateTime.Now.DayOfWeek;
switch (day)
{
    case DayOfWeek.Monday:
        Console.WriteLine("Start");
        break;
    default:
        Console.WriteLine("Other");
        break;
}
```

Looping Constructs (for, while, foreach, do-while)

for: known count

while: unknown count

do-while: executes at least once

foreach: iterate over collections

```
int[] numbers = { 1, 2, 3 };
foreach (int n in numbers)
{
    Console.WriteLine(n);
}
```

```
foreach (var item in items)
    Console.WriteLine(item);
```

Practical Exercises: Control Flow and Logic Building

Practice Tasks:

- Find even/odd numbers.
- Sum numbers 1–100.
- Grade evaluator with switch.
- Multiplication table using nested loops.

Introduction to Object-Oriented Programming (OOP)

Core principles: **Encapsulation**, **Inheritance**, **Polymorphism**, **Abstraction**.

Benefits: Reusability, maintainability, modularity.

Analogy: Car = object, Engine = class member.

Diagram: Real-world vs program object.

```
public class Car
{
    public string Model;
    public void Drive() => Console.WriteLine("Driving");
}
```

```
Car myCar = new Car();
myCar.Drive();
```

Understanding Fields, Properties, and Methods

Fields: store data (private).

Properties: control access via get/set.

Methods: define behavior.

```
private int _speed;
public int Speed { get; set; }
public void Accelerate() => Speed += 10;
```

Constructors and Destructors

Constructor: initializes objects.

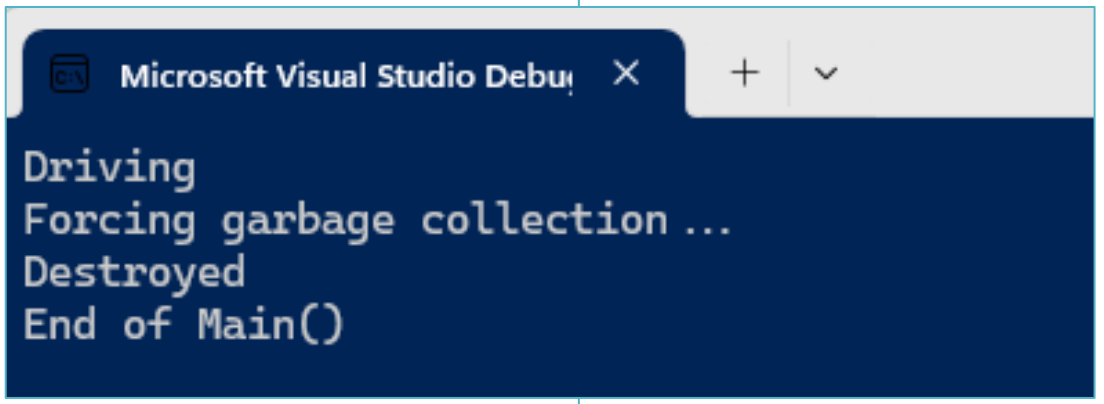
public Car(string model) => Model = model;

Overloading constructors for flexibility.

Destructor: cleanup (rarely needed in managed code).

~Car() {
Console.WriteLine("Destroyed"); }

```
public static void Main()
{
    CreateCar();
    Console.WriteLine("Forcing garbage collection...");
    GC.Collect();
    GC.WaitForPendingFinalizers();
    Console.WriteLine("End of Main()");
}

static void CreateCar()
{
    Car myCar = new Car("Honda");
    myCar.Drive();
    // myCar goes out of scope here
}
```



Encapsulation and Access Modifiers

Encapsulation: hiding internal details.

Access modifiers: public, private, protected, internal.

```
public class BankAccount
{
    private decimal balance;
    public void Deposit(decimal amount) => balance += amount;
}
```

Inheritance and Method Overriding

Inheritance: reuse parent behavior.

```
public class Vehicle
{
    public void Start()
    {
        Console.WriteLine("Start...");
    }
}
```

Overriding: customize base method.

```
public class Vehicle
{
    public virtual void Start()
    {
        Console.WriteLine("Start...");
    }
}
```

Polymorphism

Polymorphism: one interface, many forms.

```
private static void Main(string[] args)
{
    Dog dog = new Dog();
    dog.Speak();

    Animal dog2 = dog;
    dog2.Speak();
}
```

```
public class Animal
{
    public string Name { get; set; }
    public virtual void Speak()
    {
        Console.WriteLine("Animal Speak");
    }
}

public class Dog : Animal
{
    public override void Speak()
    {
        Console.WriteLine("Dog Speak");
    }
}
```

Interfaces

Interfaces: define contracts.

```
private static void Main(string[] args)
{
    IDriveable car = new Car();
    car.Drive();
}
```

```
public interface IDriveable
{
    void Drive();
}

public class Car : IDriveable
{
    public void Drive()
    {
        Console.WriteLine("Drive");
    }

    public void Stop()
    {
        Console.WriteLine("Stop");
    }
}
```

Exception Handling: try, catch, finally, throw

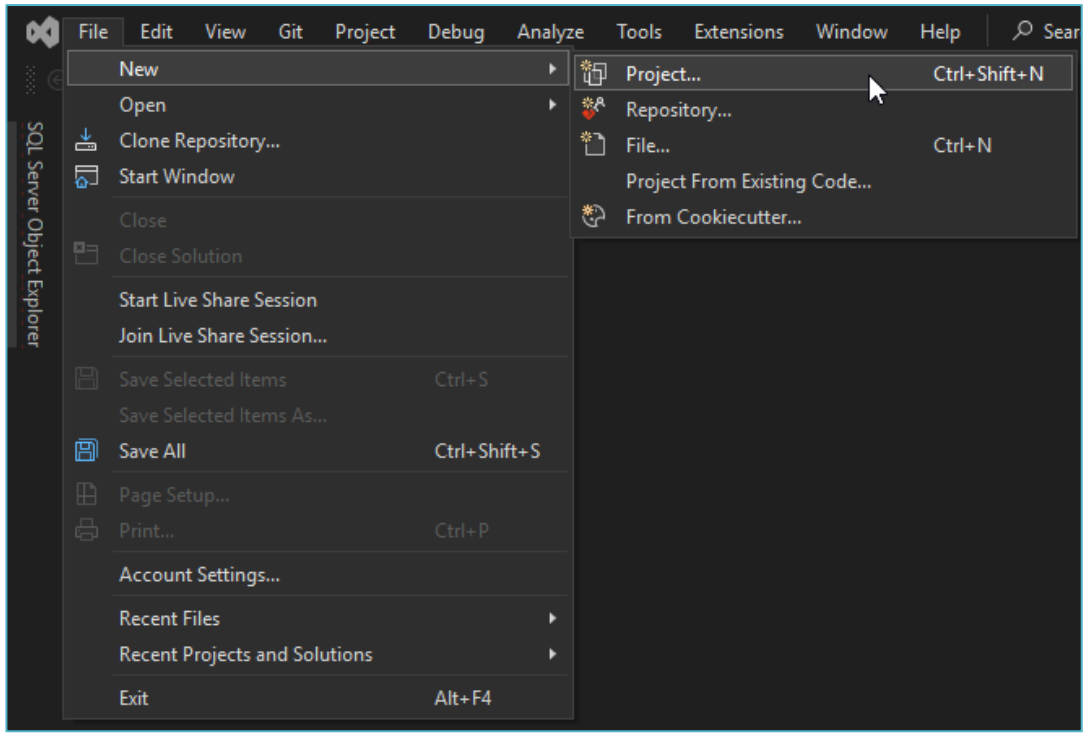
Handle runtime errors gracefully.

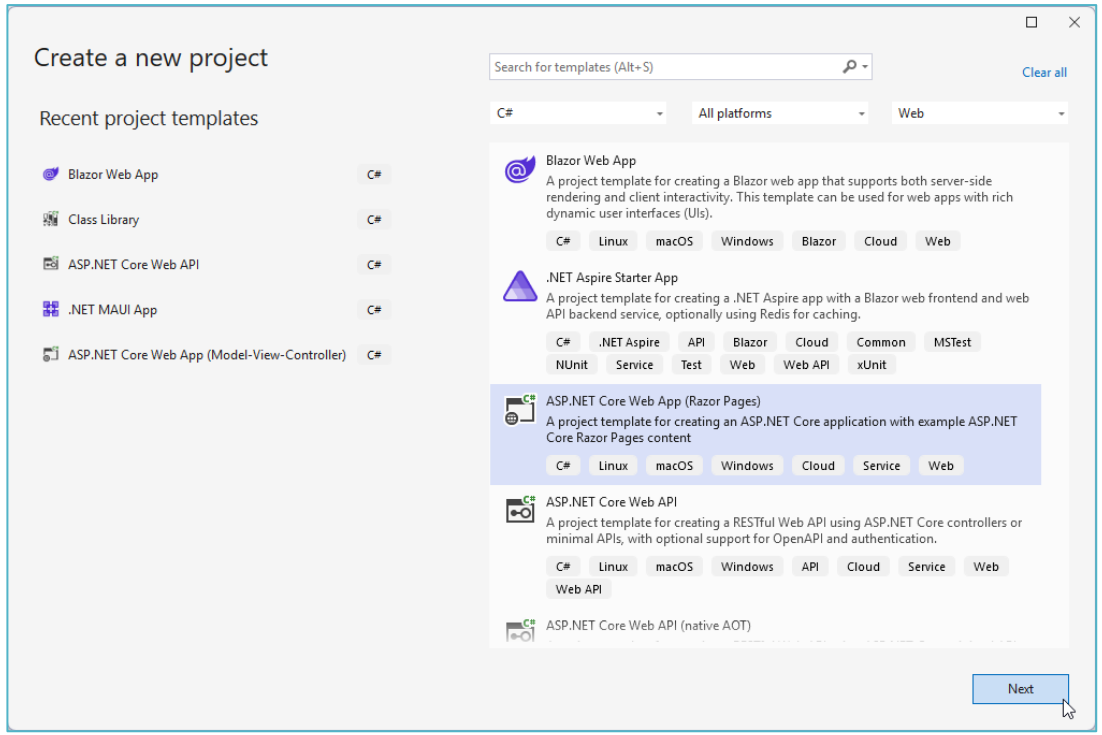
```
try
{
    int x = int.Parse("abc");
}
catch (FormatException ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
finally
{
    Console.WriteLine("Cleanup done.");
}
```

Use throw to re-raise exceptions.

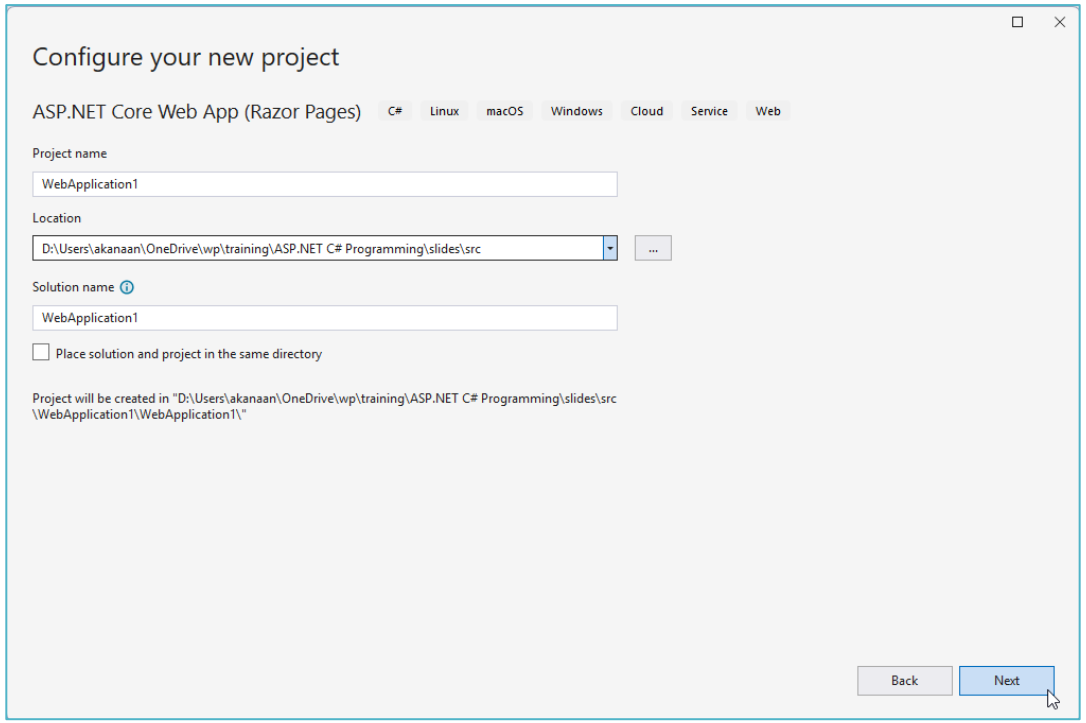
Module 3

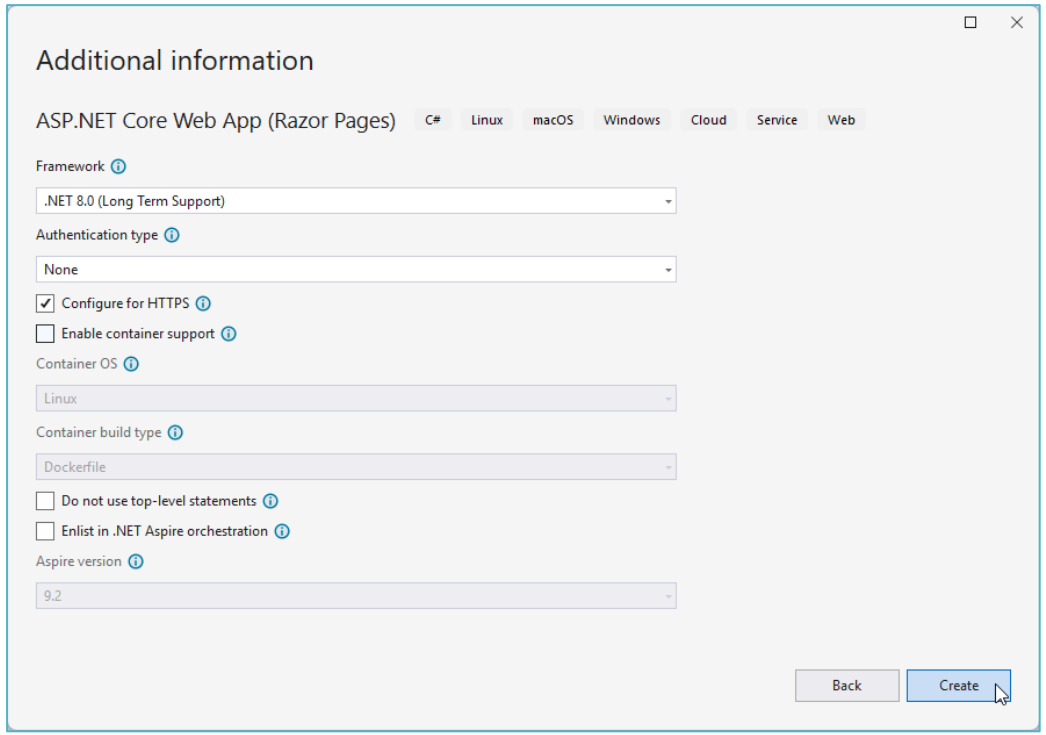
Visual Studio Start Screen



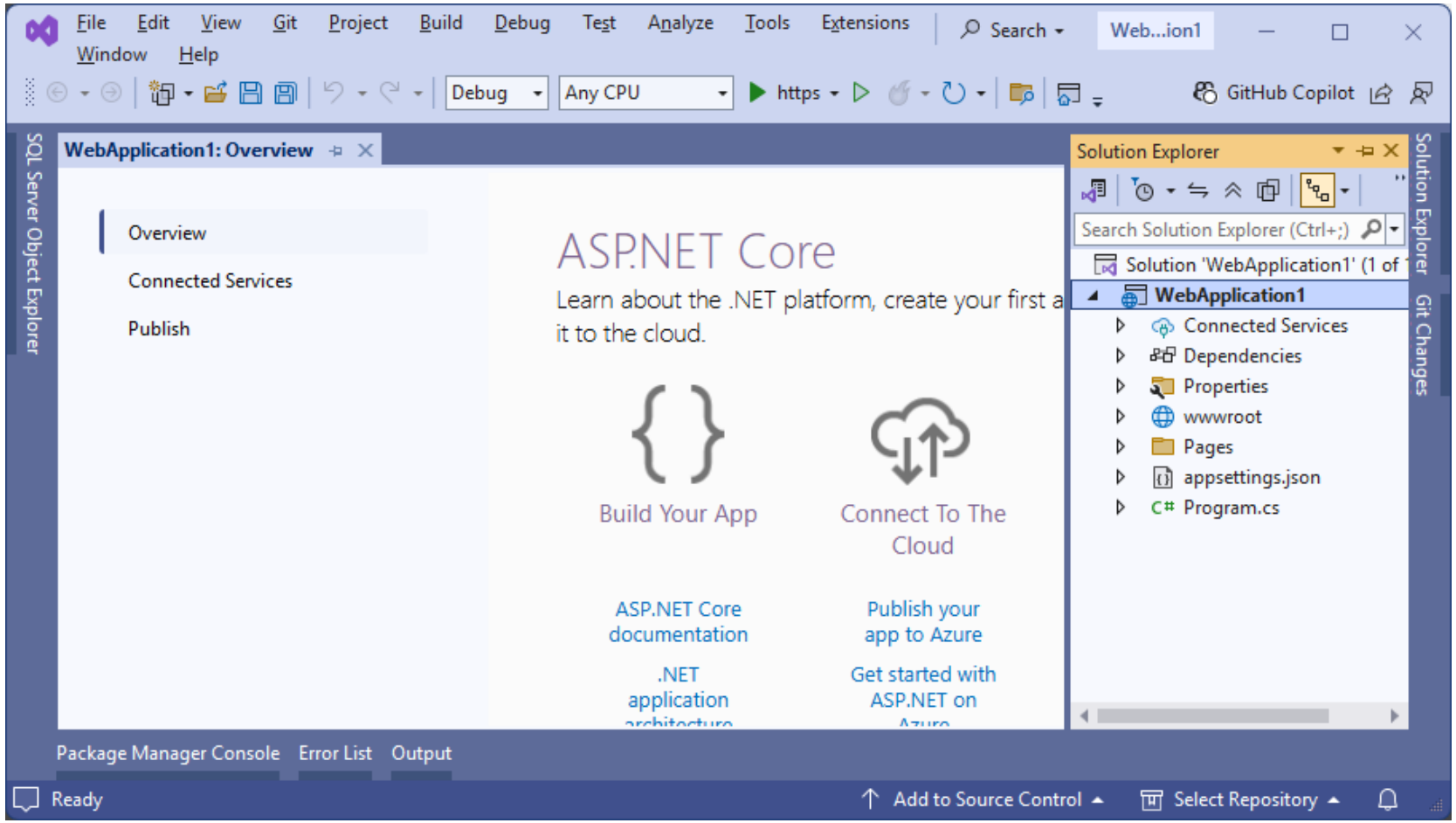


Configure Your New Project

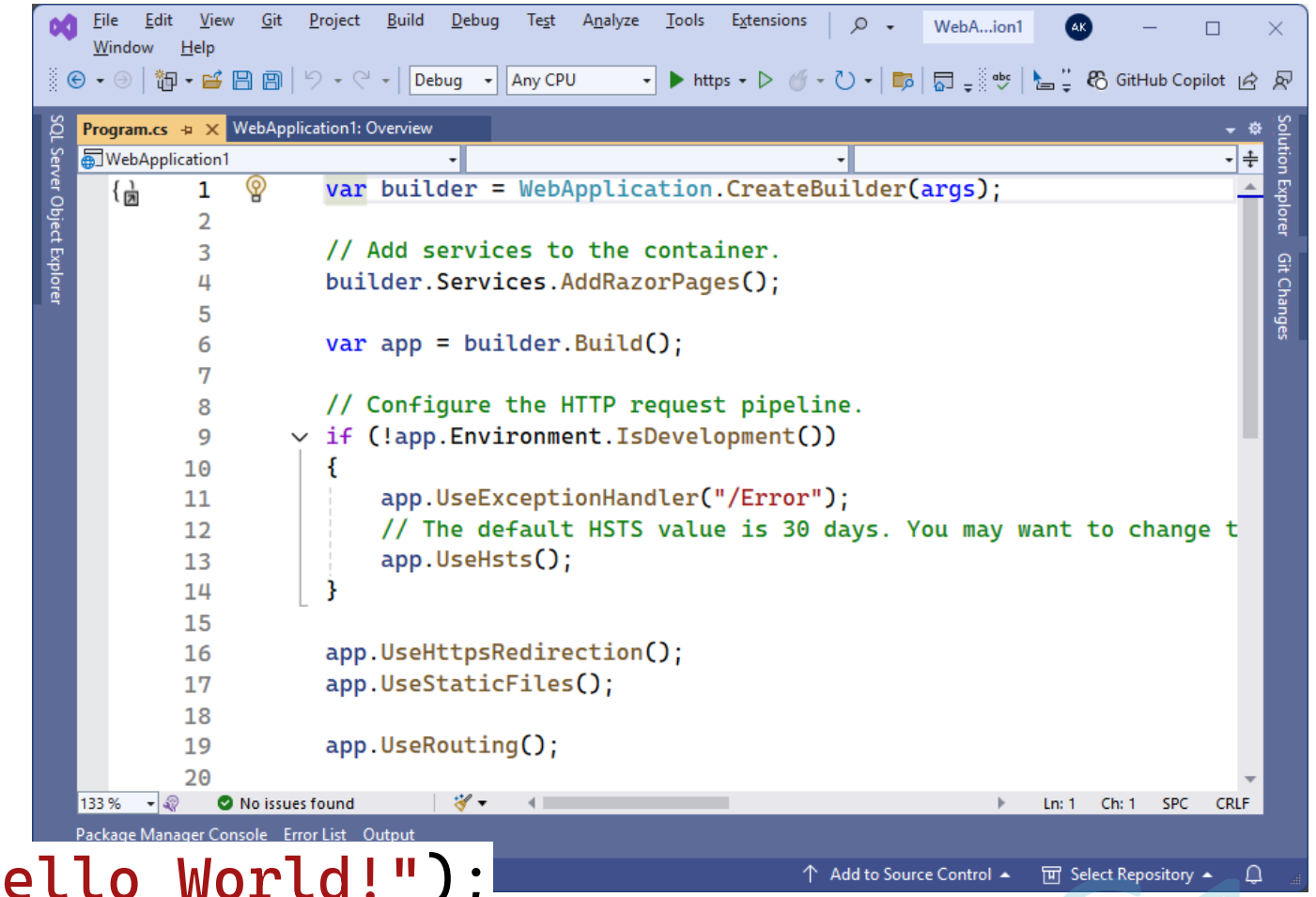




Solution Explorer View

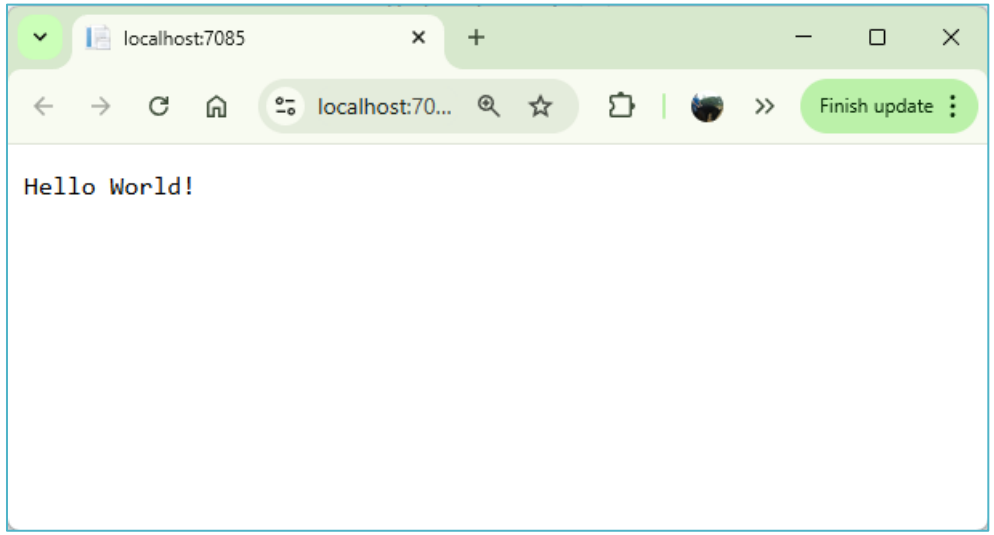


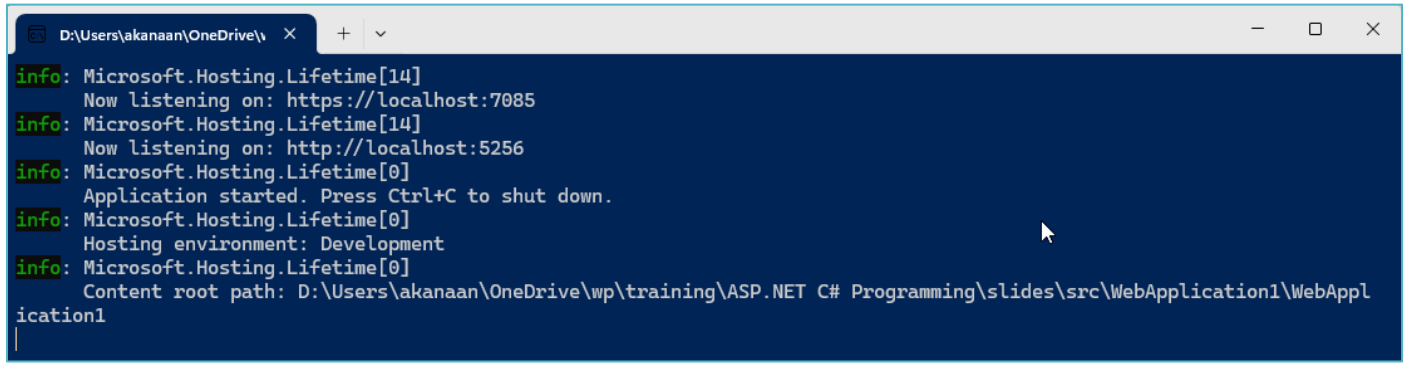
Program.cs Content

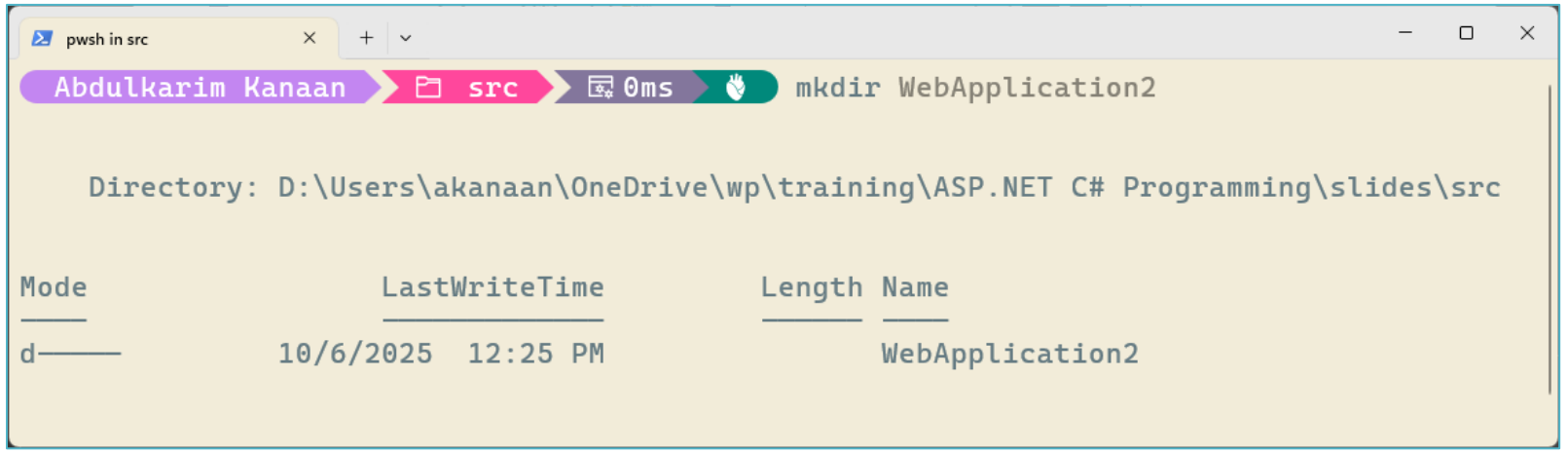


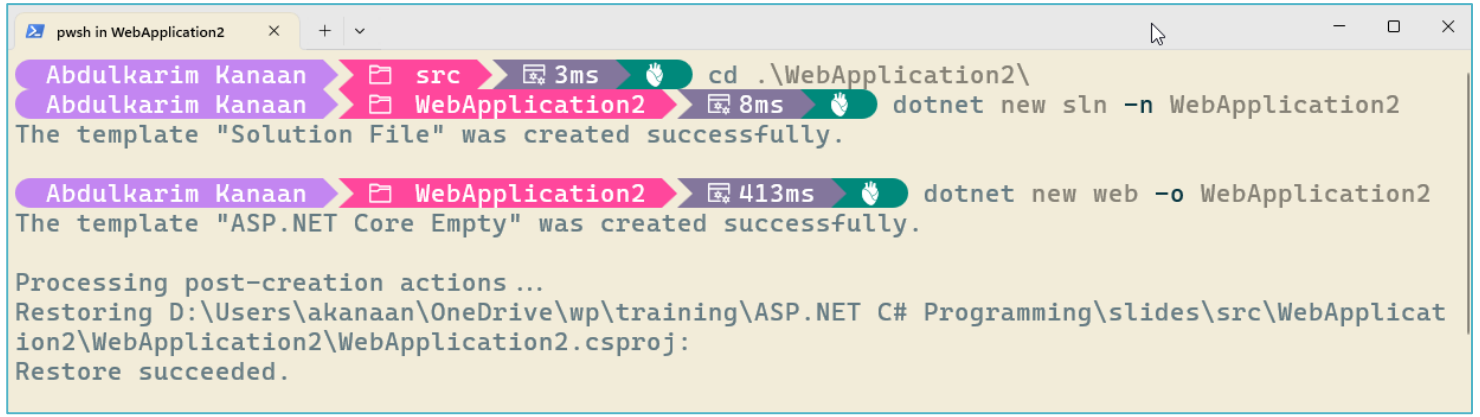

```
app.MapGet("/", () => "Hello World!");
```

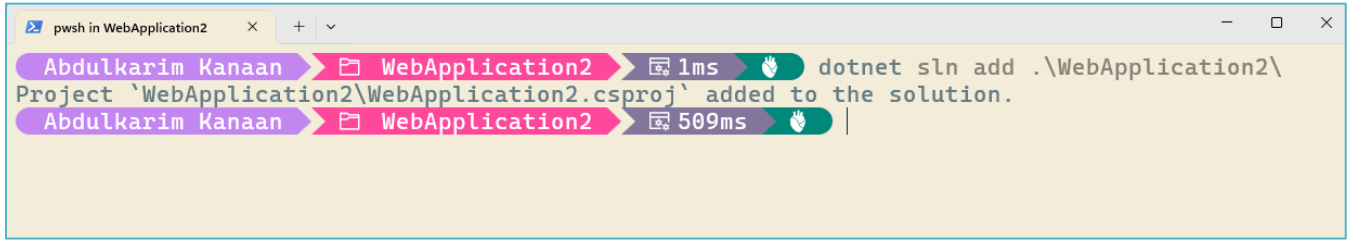
Browser Output






pwsh in src
Abdulkarim Kanaan src 0ms mkdir WebApplication2
Directory: D:\Users\akanaan\OneDrive\wp\training\ASP.NET C# Programming\slides\src
Mode LastWriteTime Length Name
d----- 10/6/2025 12:25 PM WebApplication2

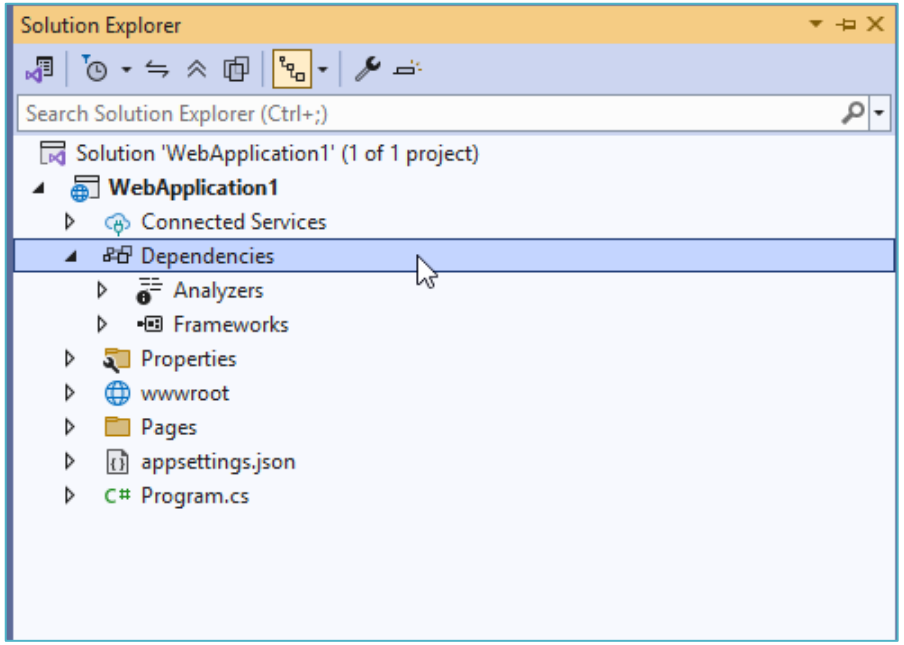

pwsh in WebApplication2
Abdulkarim Kanaan src 3ms cd .\WebApplication2\
Abdulkarim Kanaan WebApplication2 8ms dotnet new sln -n WebApplication2
The template "Solution File" was created successfully.
Abdulkarim Kanaan WebApplication2 413ms dotnet new web -o WebApplication2
The template "ASP.NET Core Empty" was created successfully.
Processing post-creation actions...
Restoring D:\Users\akanaan\OneDrive\wp\training\ASP.NET C# Programming\slides\src\WebApplication2\WebApplication2\WebApplication2.csproj:
Restore succeeded.


pwsh in WebApplication2
Abdulkarim Kanaan WebApplication2 1ms dotnet sln add .\WebApplication2\
Project `WebApplication2\WebApplication2.csproj` added to the solution.
Abdulkarim Kanaan WebApplication2 509ms

Building the ASP.NET Core Application

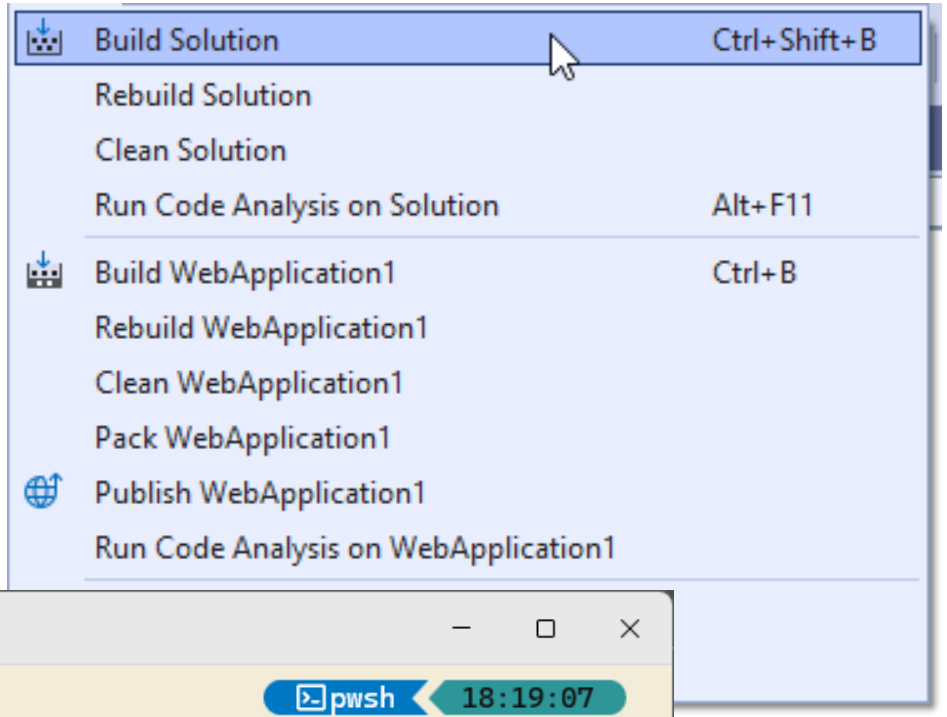
Checking Dependencies (NuGet Restore)

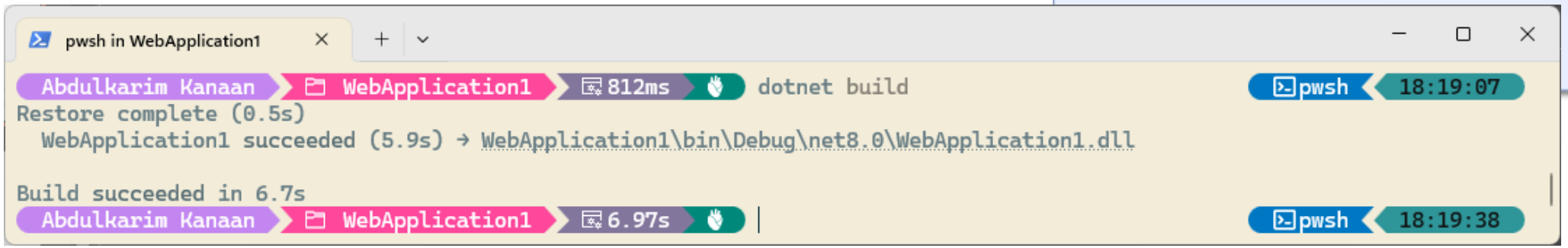
Restore Packages

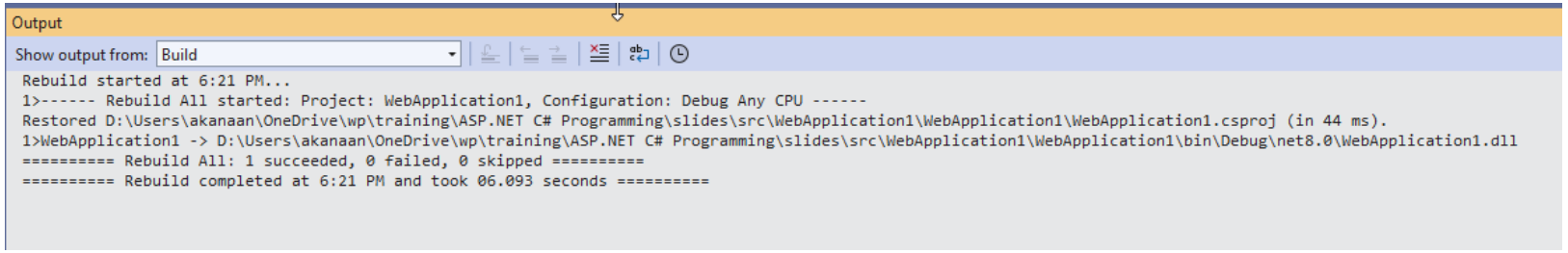


Build Solution

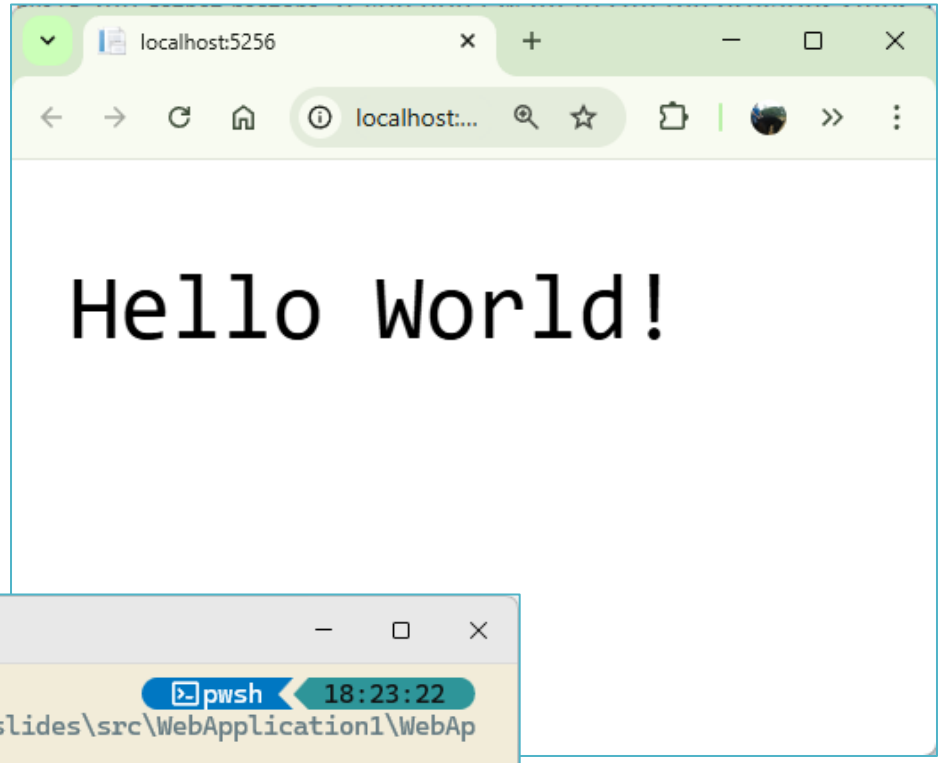
Build > Build Solution or Ctrl + Shift + B.

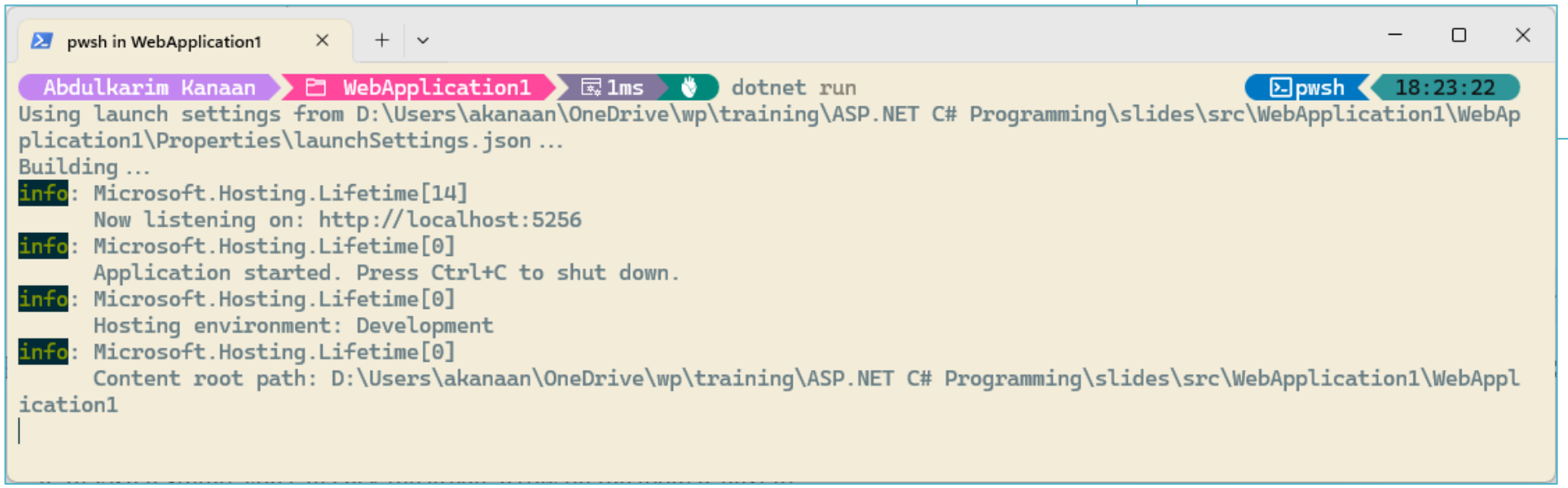




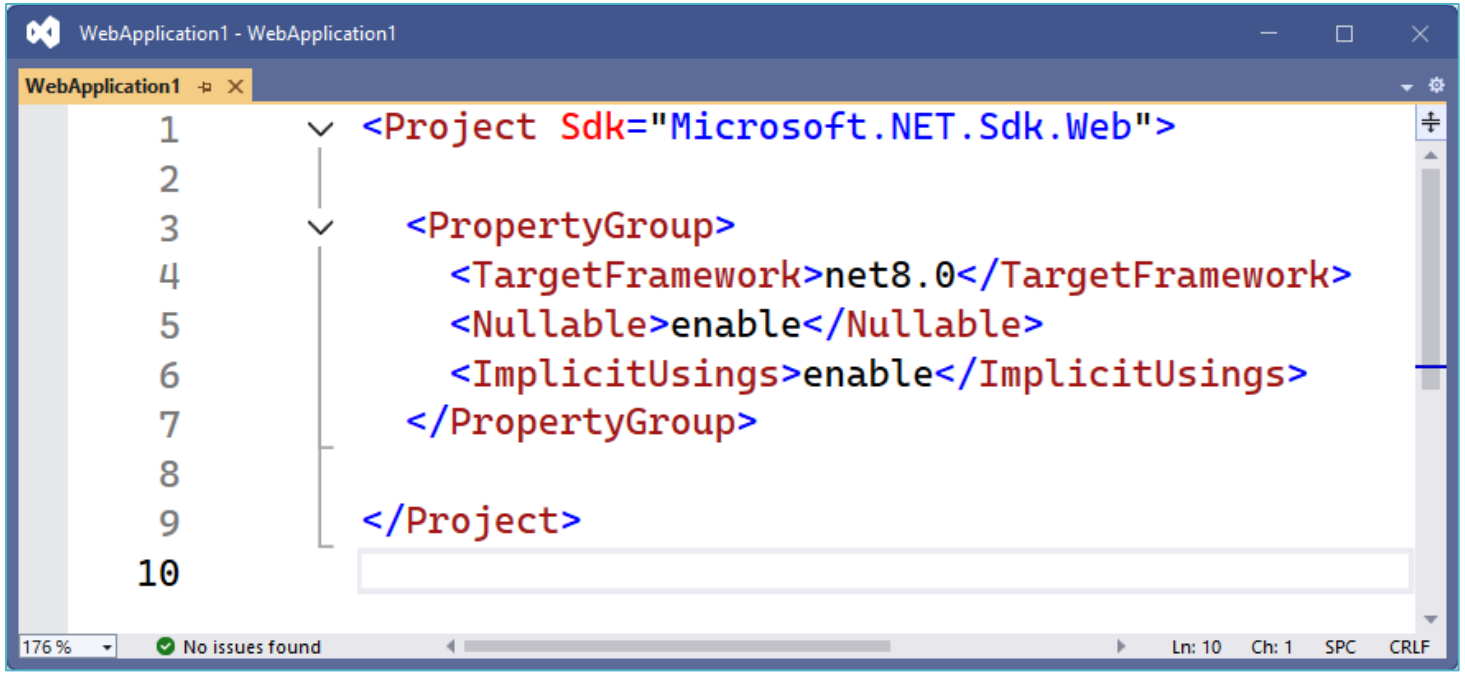


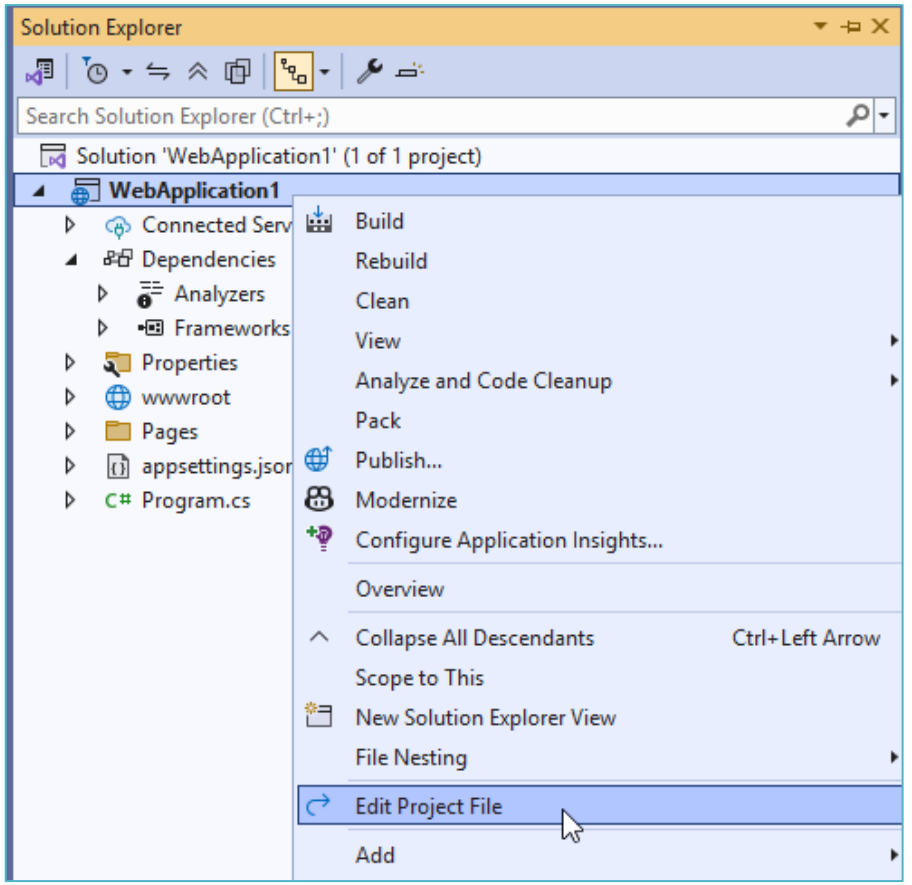
Run App


localhost:5256
localhost:...
Hello World!


pwsh in WebApplication1
Abdulkarim Kanaan
WebApplication1
1ms
dotnet run
pwsh
18:23:22
Using launch settings from D:\Users\akanaan\OneDrive\wp\training\ASP.NET C# Programming\slides\src\WebApplication1\WebApplication1\Properties\launchSettings.json ...
Building ...
info: Microsoft.Hosting.Lifetime[14]
Now listening on: http://localhost:5256
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: D:\Users\akanaan\OneDrive\wp\training\ASP.NET C# Programming\slides\src\WebApplication1\WebApplication1

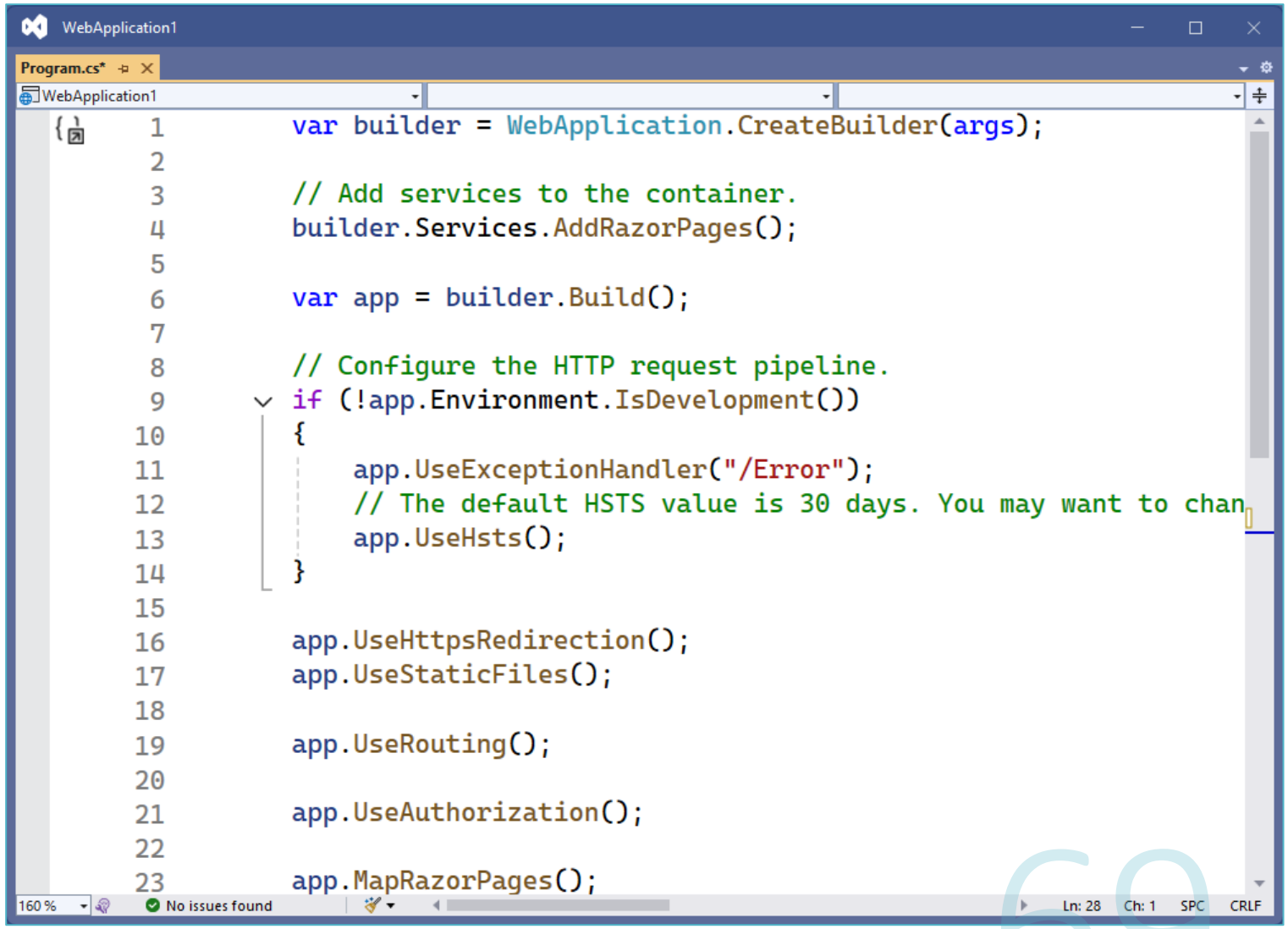
.csproj project file



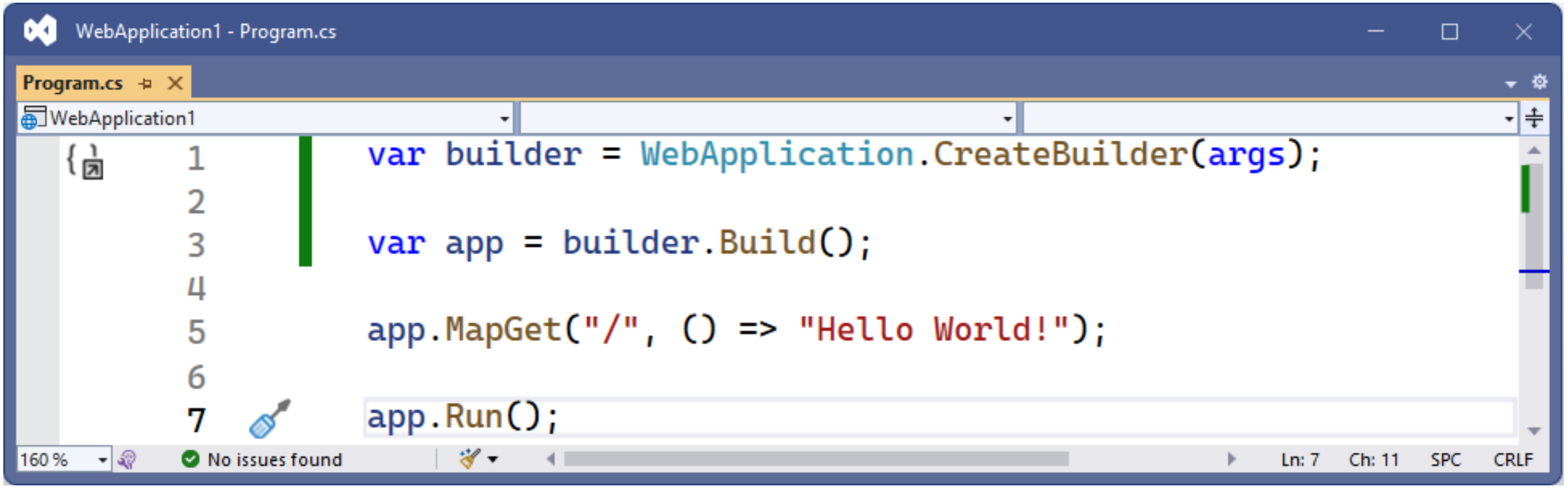


Program.cs File

Top-level statements



The default Program.cs file



Default WebApplication Configuration

🧩 WebApplicationBuilder
- ⚙️ Configuration → Loads settings (JSON, env vars)
- 📊 Logging → Provides observability/debugging
- 🧰 Services → Registers dependencies (DI)
- 🖥️ Hosting → Uses Kestrel web server

🏗️ builder.Build()

↓

🌐 WebApplication
- 🪜 Middleware → Processes each request in sequence
- 🎯 Endpoints → Generate responses (pages, APIs)

Expand Program.cs


```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

builder.Services.AddHttpLogging(opts =>
    opts.LoggingFields = Microsoft.AspNetCore.HttpLogging.HttpLoggingFields.RequestProperties);

builder.Logging.AddFilter(
    "Microsoft.AspNetCore.HttpLogging", LogLevel.None);

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseHttpLogging();
}

app.MapGet("/", () => "Hello World!");
app.MapGet("/person", () => new Person("Karim", "Kanaan"));

app.Run();

1 reference
public record Person(string FirstName, string LastName);
```

Introduction to Logging

Purpose:
Logging helps track what happens inside an application during execution.

Key Points:

Used to record runtime information for diagnostics and troubleshooting.

ASP.NET Core has built-in logging infrastructure.

Works with multiple providers (Console, Debug, File, Azure, etc.).

Useful for **monitoring**, **debugging**, and **auditing**.

Why Configure Logging Levels?

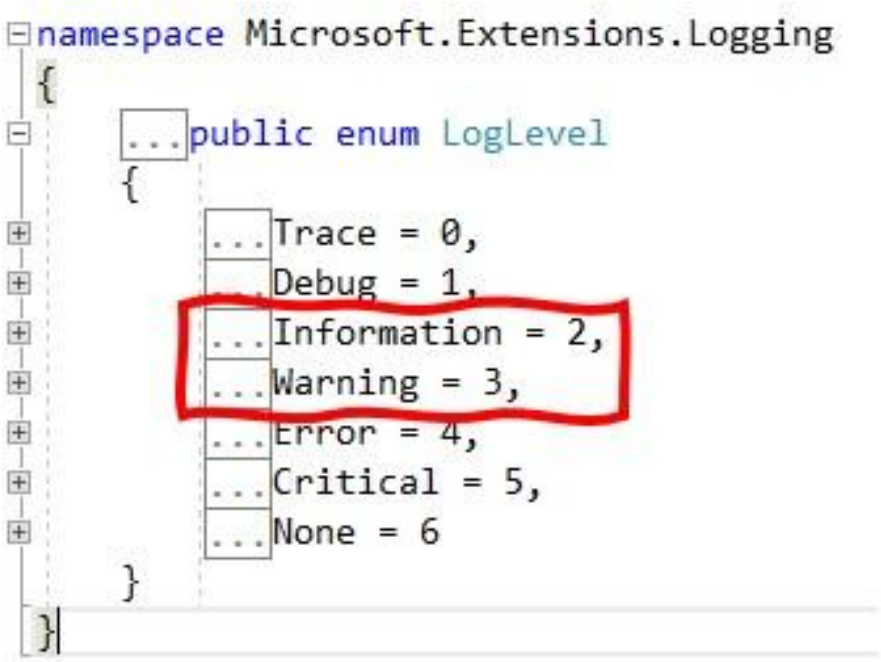
Concept:
Every part of your application (and the libraries it depends on) produces log messages — but not all are shown.

Key Points:

Logging levels control how much information is displayed.

Prevents console overload with unnecessary logs.

Lets you focus on messages relevant to your debugging context.



Filtering Logs by Severity

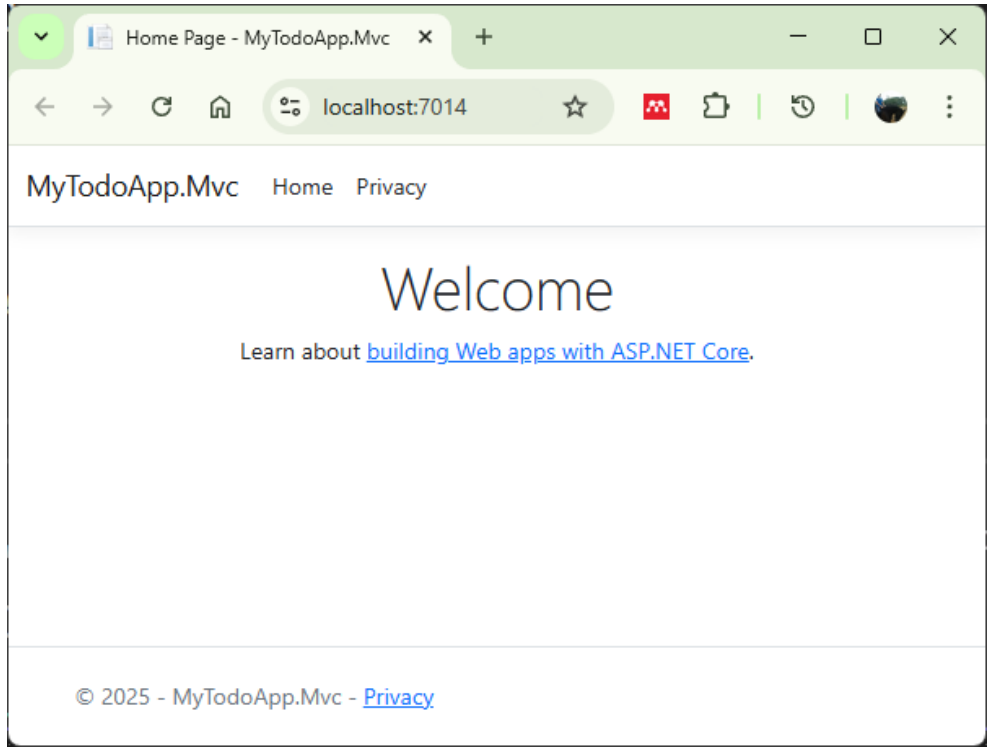
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "Microsoft.Hosting.Lifetime": "Information",
      "Microsoft.EntityFrameworkCore": "None"
  },
}
```

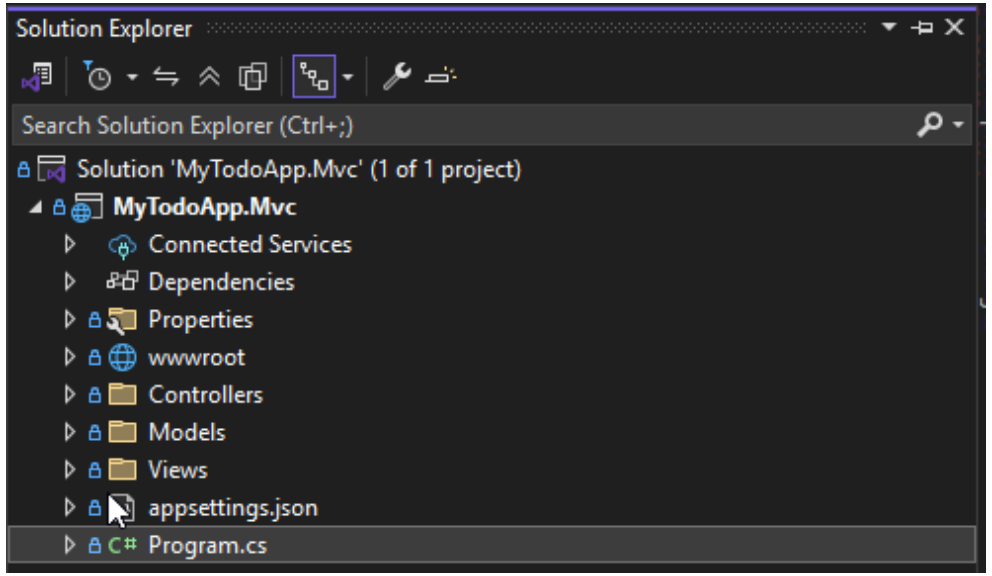
Explanation:

Default: Debug → Show all your app logs.

Microsoft.Hosting.Lifetime: Information → Only show warnings/errors from Microsoft libraries.

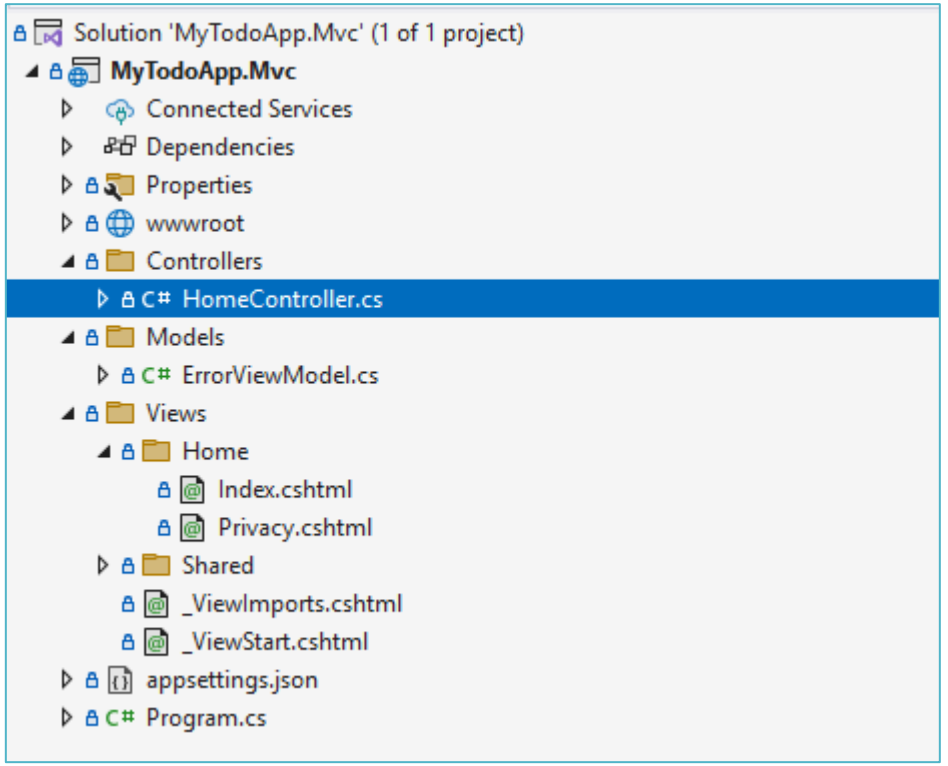
Module 3: Building a Simple ASP.NET Web Ap


Home Page - MyTodoApp.Mvc
localhost:7014
MyTodoApp.Mvc
Home
Privacy
Welcome
Learn about building Web apps with ASP.NET Core.
© 2025 - MyTodoApp.Mvc - Privacy

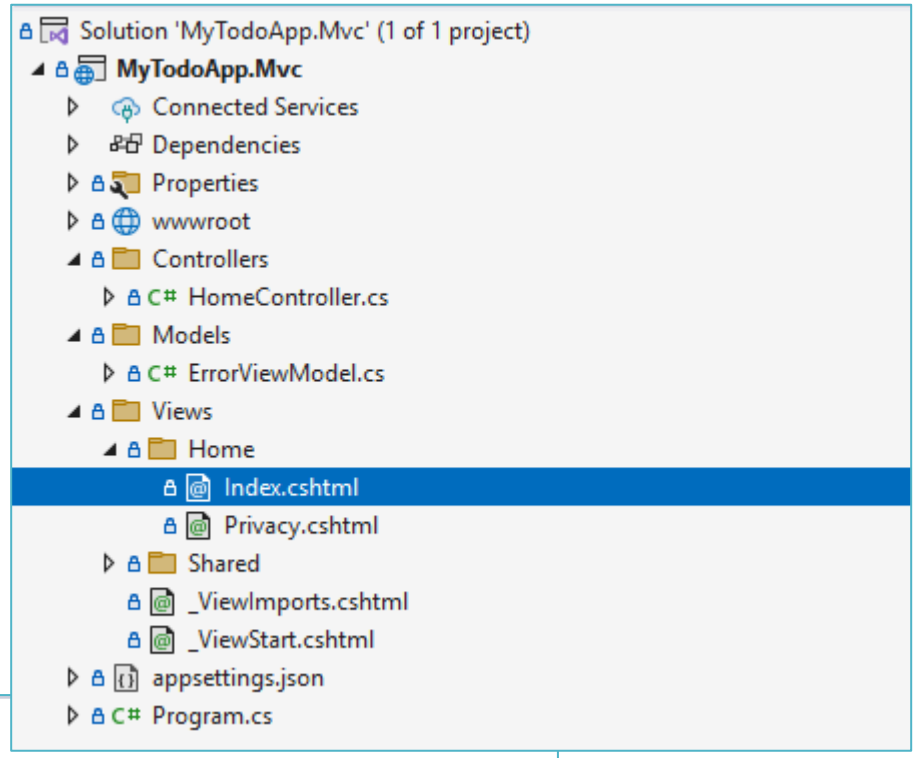

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MyTodoApp.Mvc' (1 of 1 project)
MyTodoApp.Mvc
Connected Services
Dependencies
Properties
wwwroot
Controllers
Models
Views
appsettings.json
Program.cs

HomeController

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    0 references
    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    0 references
    public IActionResult Index()
    {
        return View();
    }

    0 references
    public IActionResult Privacy()
    {
        return View();
    }

    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    0 references
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext
    }
}
```



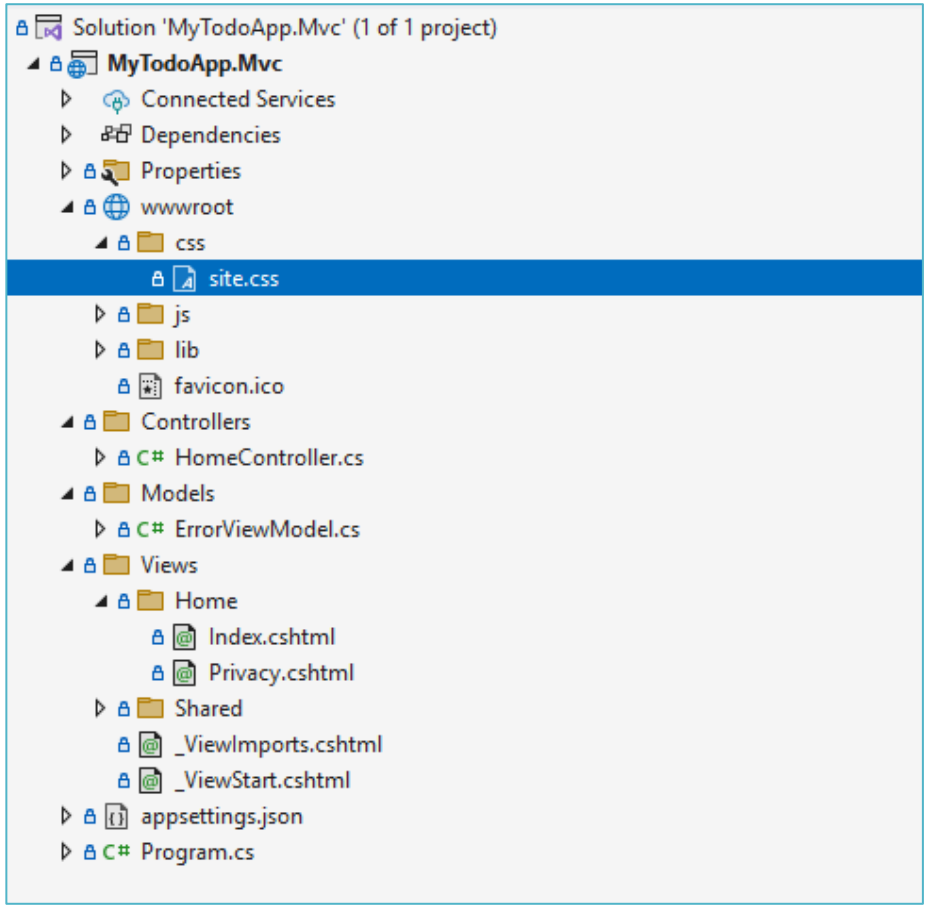
Index.cshtml



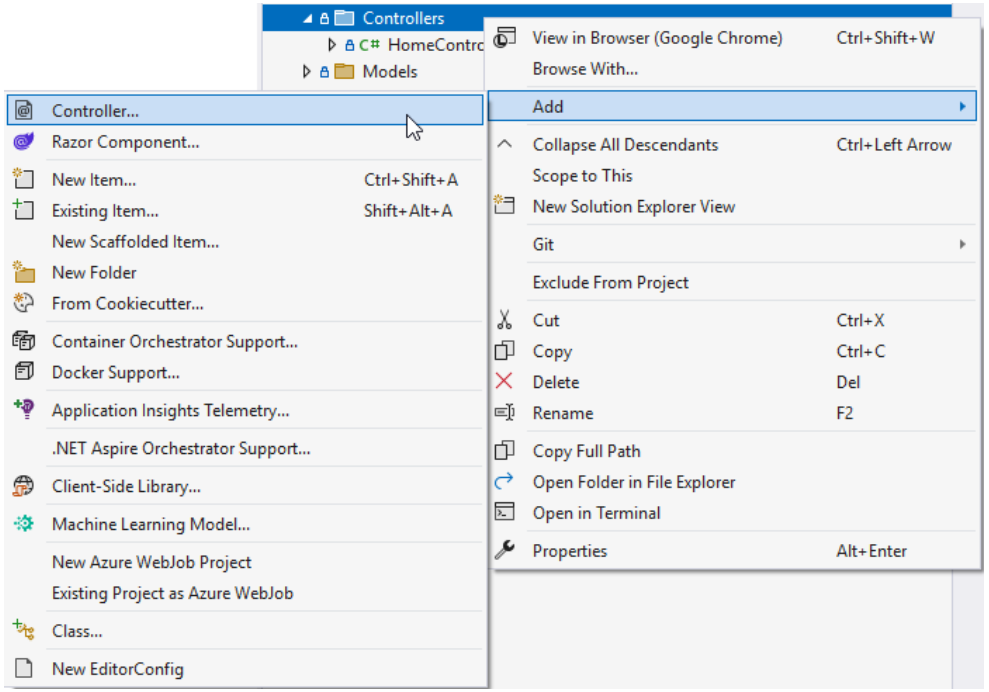

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```

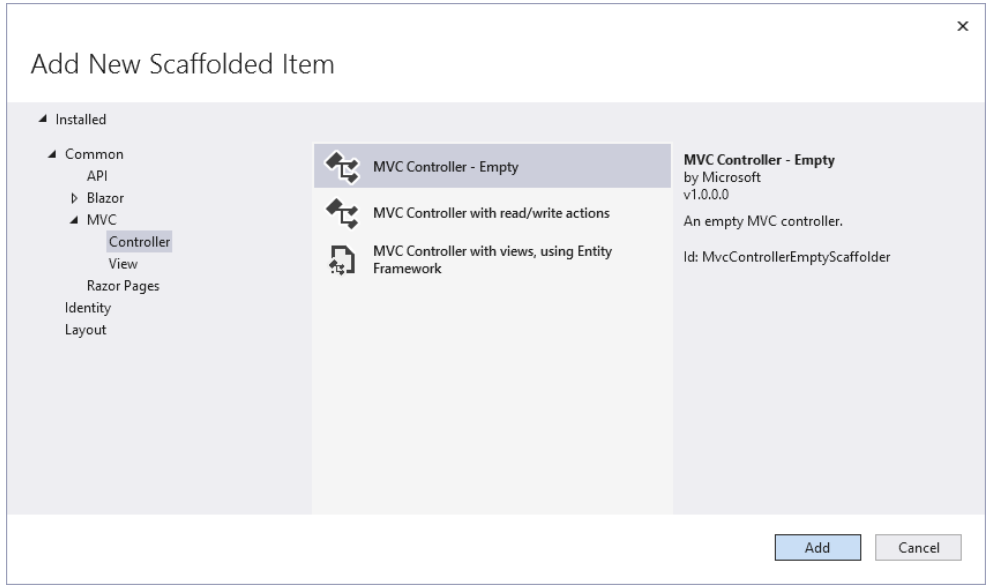
wwwroot

```css
html {
  font-size: 14px;
}

@media (min-width: 768px) {
  html {
    font-size: 16px;
  }
}

.btn:focus, .btn:active:focus, .btn-link.nav-link:focus, .form-control:focus, .form-check-input:focus {
  box-shadow: 0 0 0 0.1rem white, 0 0 0 0.25rem #258cfb;
}

html {
  position: relative;
  min-height: 100%;
}

body {
  margin-bottom: 60px;
}
```

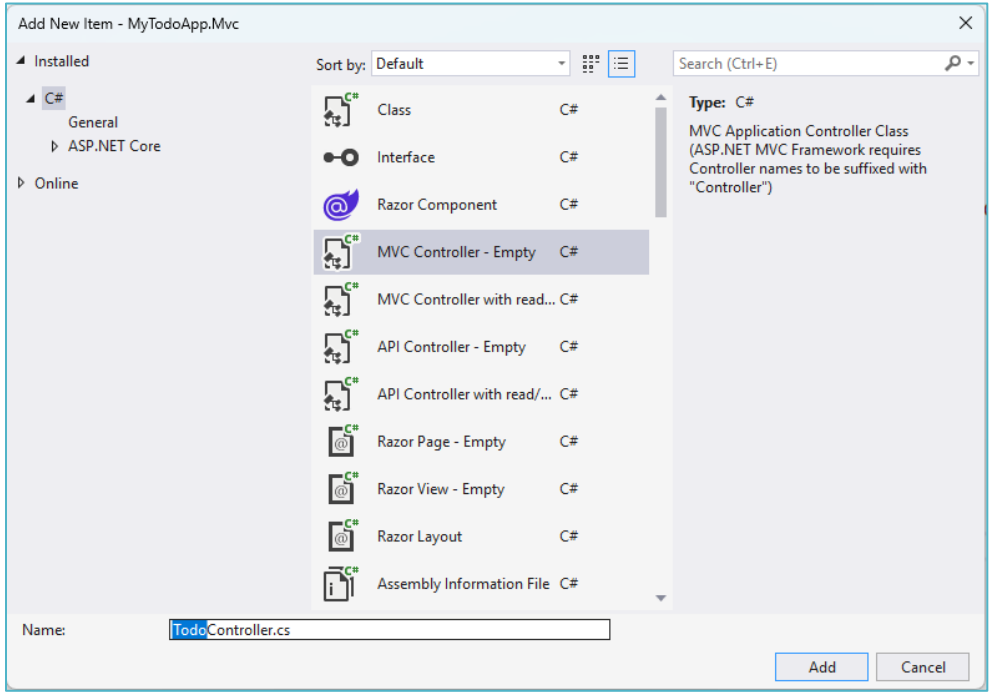


Adding Controller



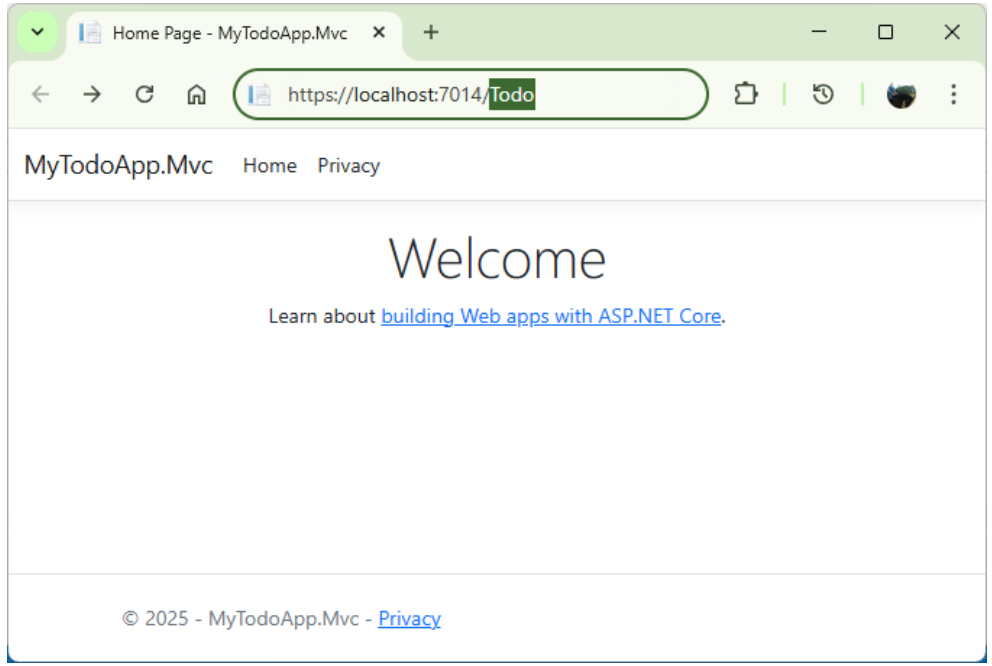


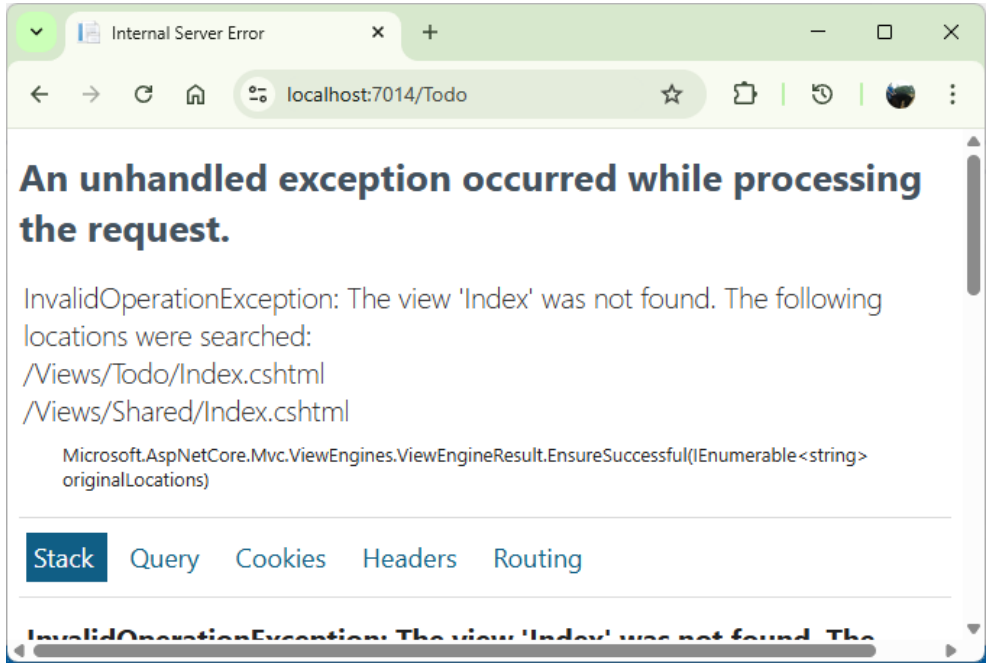
Adding Controller



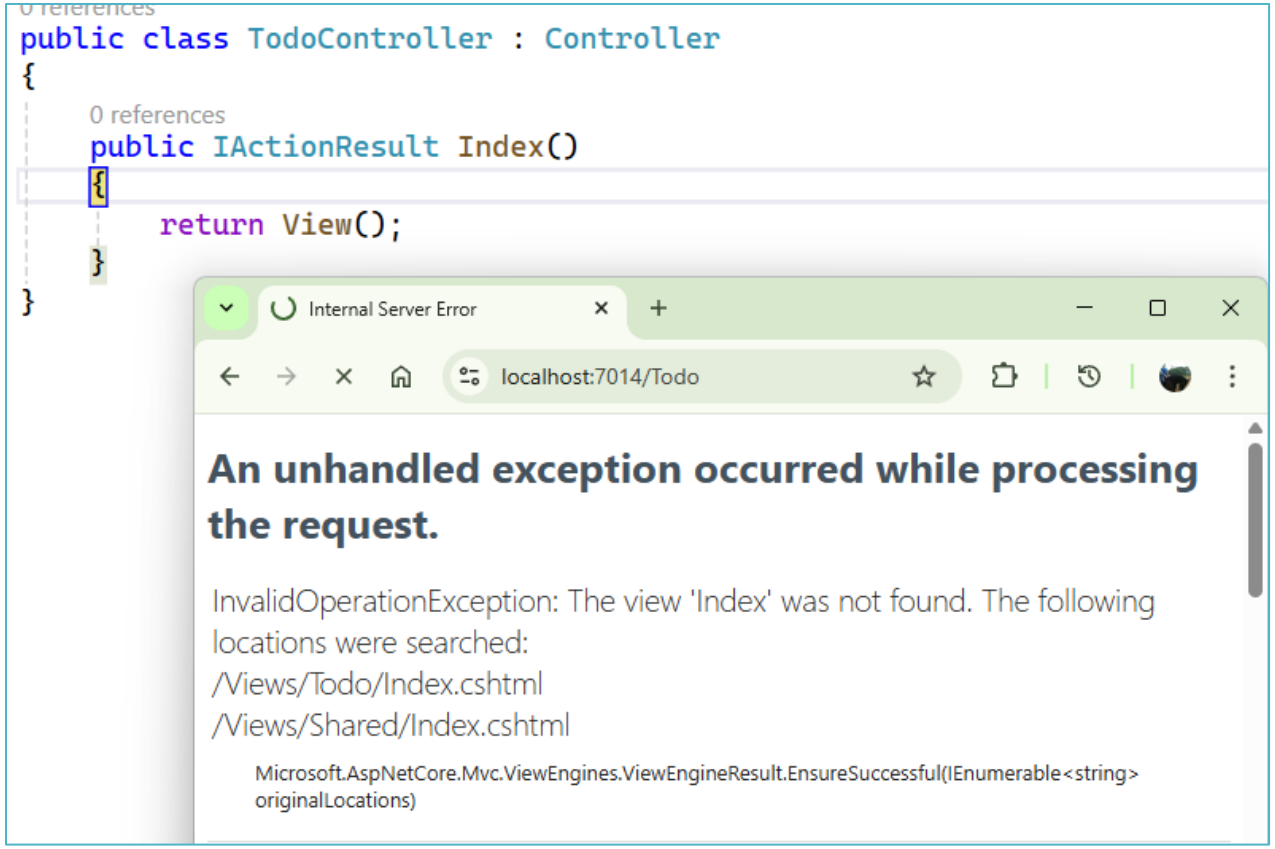

```
namespace MyTodoApp.Mvc.Controllers;

0 references
public class TodoController : Controller
{
    0 references
    public IActionResult Index()
    {
        return View();
    }
}
```

Run & Navigate to Todo

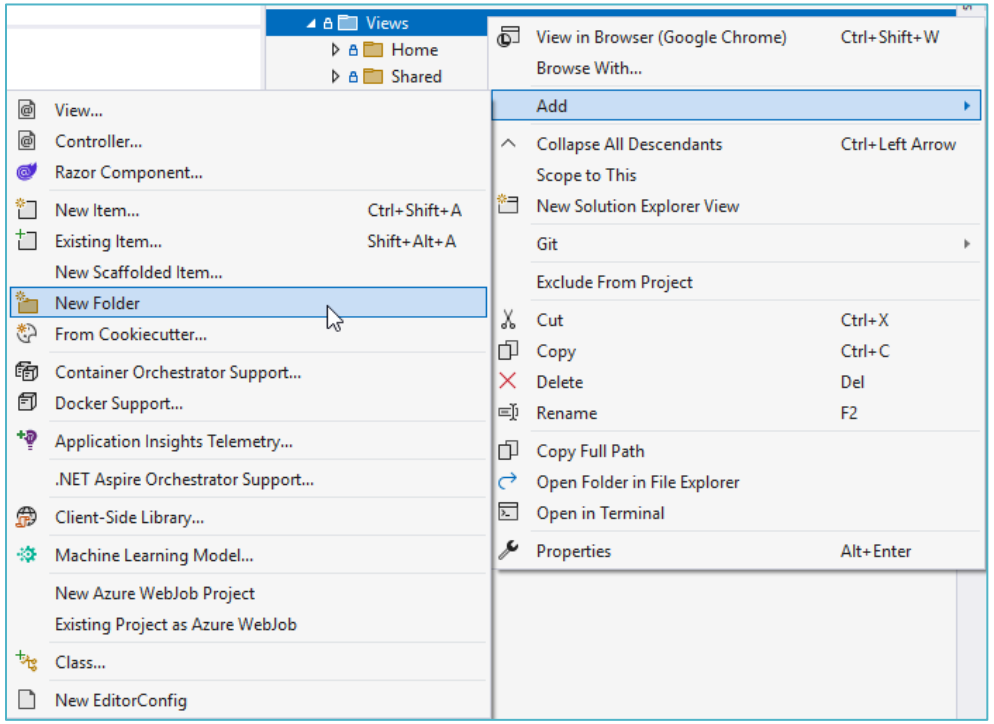


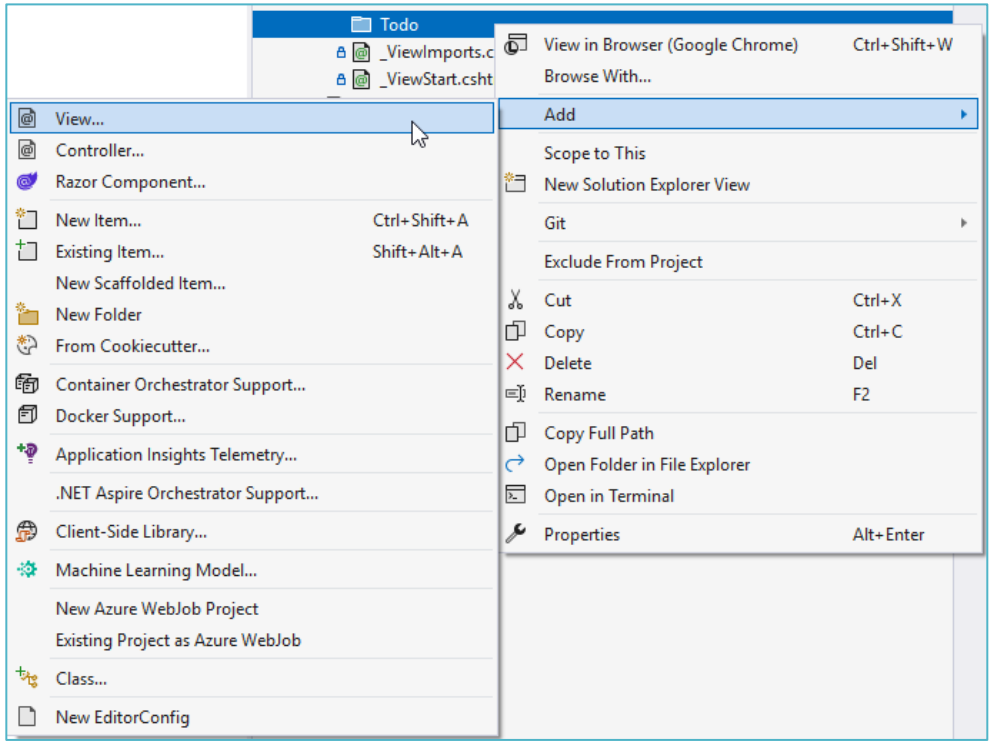


Debugging

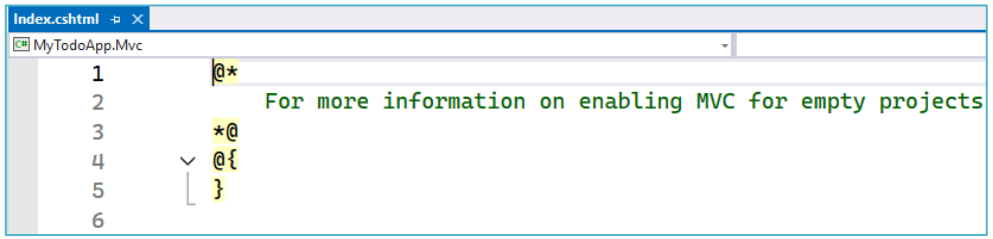

0 references
public class TodoController : Controller
{
0 references
public IActionResult Index()
{
return View();
}
}
Internal Server Error
localhost:7014/Todo
An unhandled exception occurred while processing the request.
InvalidOperationException: The view 'Index' was not found. The following locations were searched:
/Views/Todo/Index.cshtml
/Views/Shared/Index.cshtml
Microsoft.AspNetCore.Mvc.ViewEngines.ViewEngineResult.EnsureSuccessful(IEnumerable<string> originalLocations)

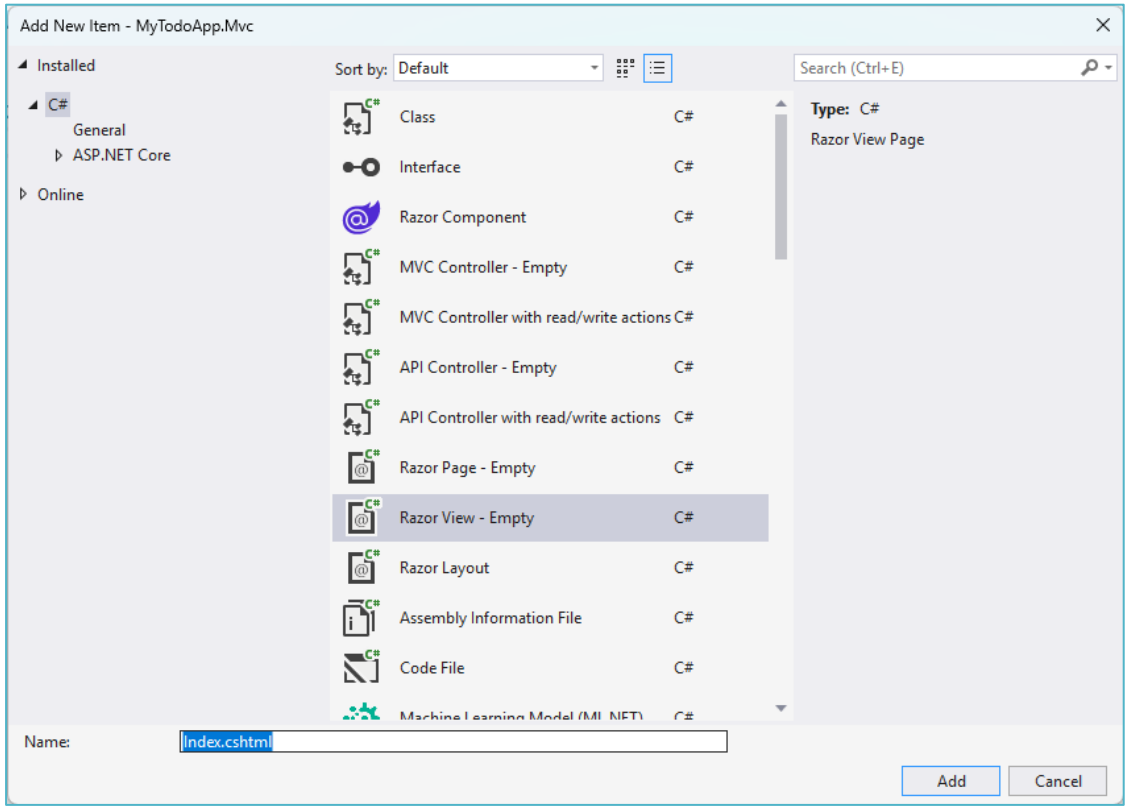
Adding View

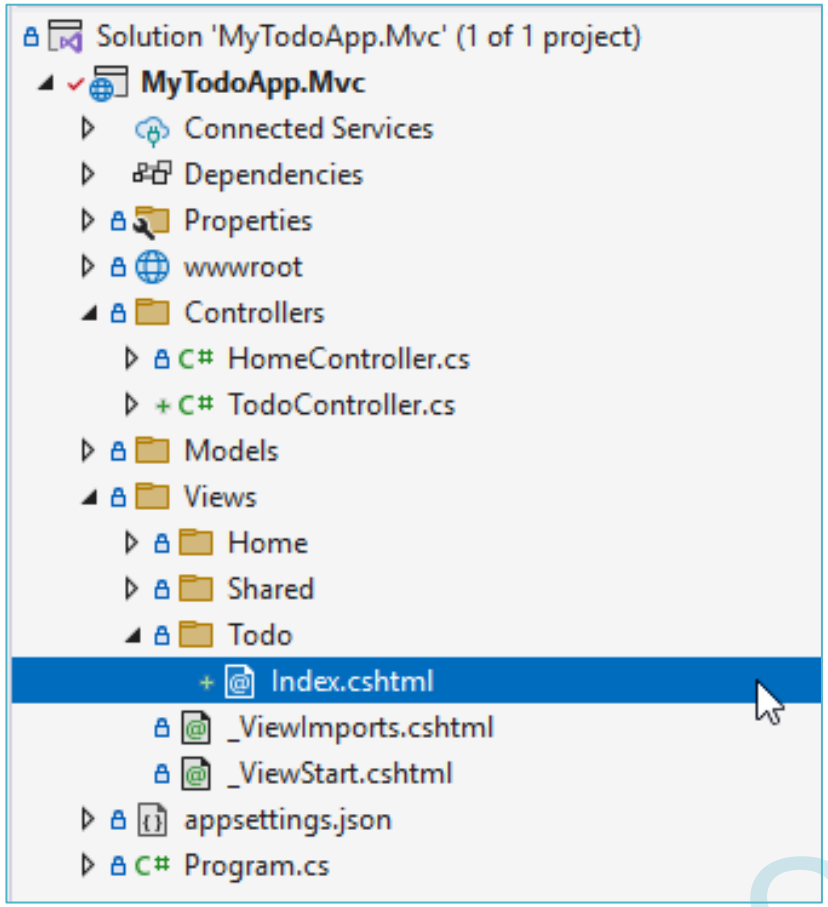

Views
Home
Shared
View...
Controller...
Razor Component...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
From Cookiecutter...
Container Orchestrator Support...
Docker Support...
Application Insights Telemetry...
.NET Aspire Orchestrator Support...
Client-Side Library...
Machine Learning Model...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
New EditorConfig
View in Browser (Google Chrome) Ctrl+Shift+W
Browse With...
Add
Collapse All Descendants Ctrl+Left Arrow
Scope to This
New Solution Explorer View
Git
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Delete Del
Rename F2
Copy Full Path
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter


Todo
_ViewImports.c
_ViewStart.csht
View...
Controller...
Razor Component...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
From Cookiecutter...
Container Orchestrator Support...
Docker Support...
Application Insights Telemetry...
.NET Aspire Orchestrator Support...
Client-Side Library...
Machine Learning Model...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
New EditorConfig
View in Browser (Google Chrome) Ctrl+Shift+W
Browse With...
Add
Scope to This
New Solution Explorer View
Git
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Delete Del
Rename F2
Copy Full Path
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter

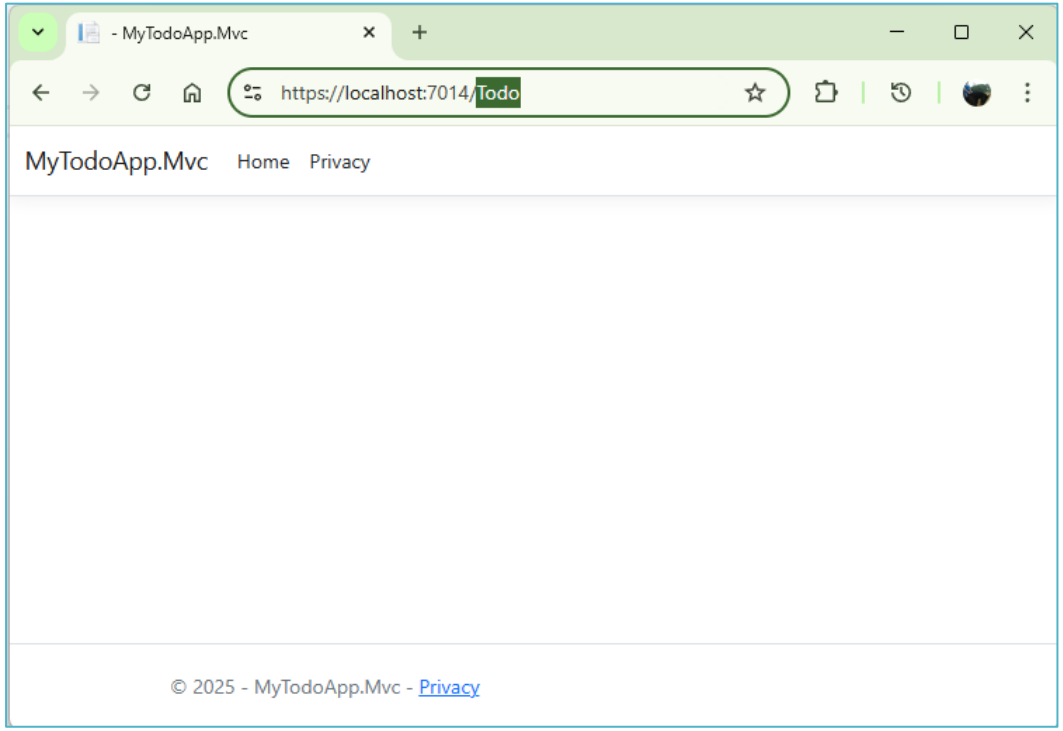
Adding View


Index.cshtml
MyTodoApp.Mvc
1 @*
2 For more information on enabling MVC for empty projects
3 *@
4 @{
5 }
6


Add New Item - MyTodoApp.Mvc
Installed
C#
General
ASP.NET Core
Online
Sort by: Default
Search (Ctrl+E)
Class C#
Interface C#
Razor Component C#
MVC Controller - Empty C#
MVC Controller with read/write actions C#
API Controller - Empty C#
API Controller with read/write actions C#
Razor Page - Empty C#
Razor View - Empty C#
Razor Layout C#
Assembly Information File C#
Code File C#
Type: C#
Razor View Page
Name: Index.cshtml
Add
Cancel


Solution 'MyTodoApp.Mvc' (1 of 1 project)
MyTodoApp.Mvc
Connected Services
Dependencies
Properties
wwwroot
Controllers
HomeController.cs
TodoController.cs
Models
Views
Home
Shared
Todo
Index.cshtml
_ViewImports.cshtml
_ViewStart.cshtml
appsettings.json
Program.cs

Routing

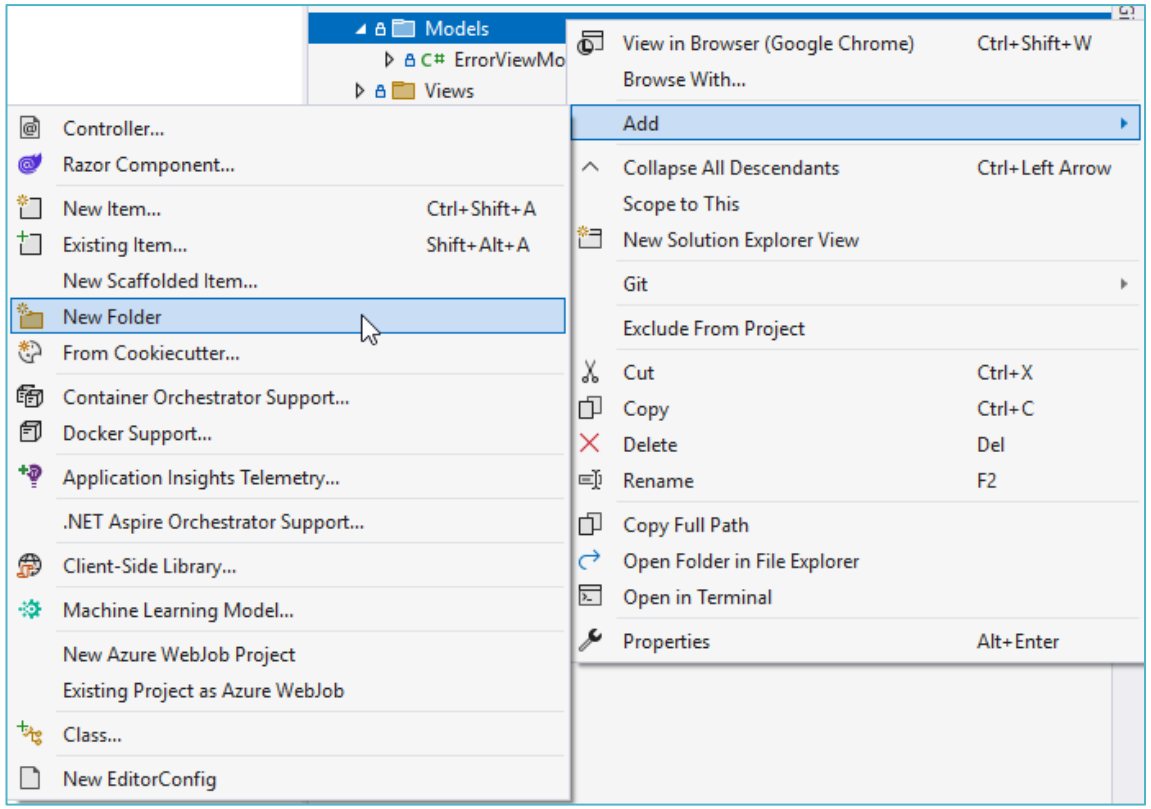


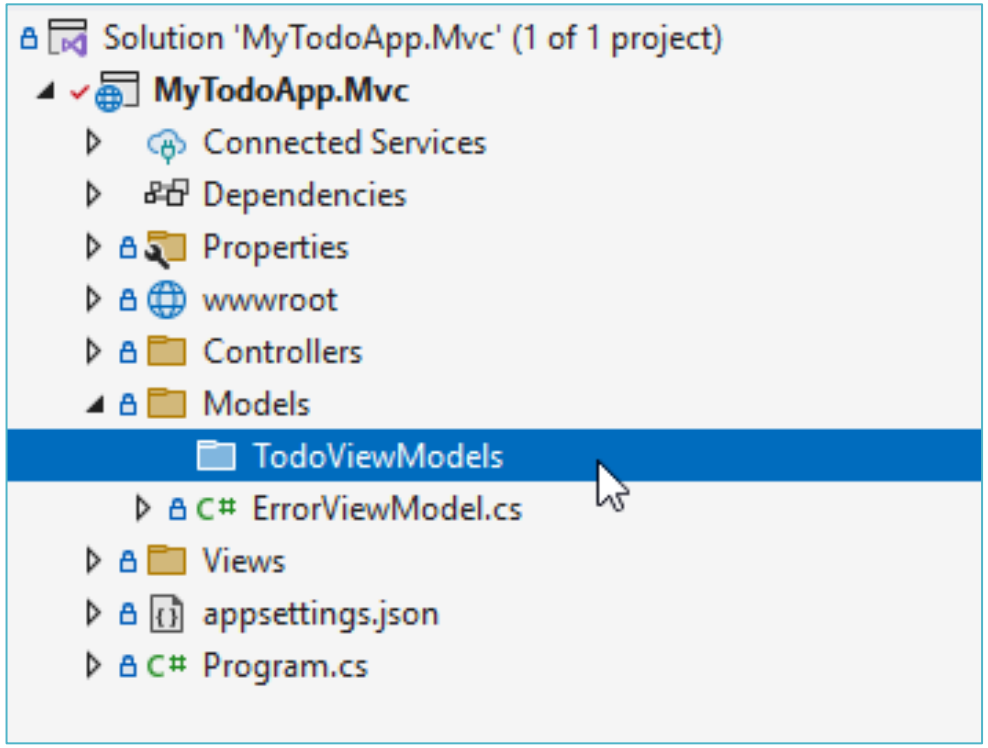
Program.cs

MyTodoApp.Mvc

```
22
23      app.MapControllerRoute(
24          name: "default",
25          pattern: "{controller=Home}/{action=Index}/{id?}");
26
```

Models


Models
ErrorViewMo
Views
Controller...
Razor Component...
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Scaffolded Item...
New Folder
From Cookiecutter...
Container Orchestrator Support...
Docker Support...
Application Insights Telemetry...
.NET Aspire Orchestrator Support...
Client-Side Library...
Machine Learning Model...
New Azure WebJob Project
Existing Project as Azure WebJob
Class...
New EditorConfig
View in Browser (Google Chrome) Ctrl+Shift+W
Browse With...
Add
Collapse All Descendants Ctrl+Left Arrow
Scope to This
New Solution Explorer View
Git
Exclude From Project
Cut Ctrl+X
Copy Ctrl+C
Delete Del
Rename F2
Copy Full Path
Open Folder in File Explorer
Open in Terminal
Properties Alt+Enter


Solution 'MyTodoApp.Mvc' (1 of 1 project)
MyTodoApp.Mvc
Connected Services
Dependencies
Properties
wwwroot
Controllers
Models
TodoViewModels
ErrorViewModel.cs
Views
appsettings.json
Program.cs

Model

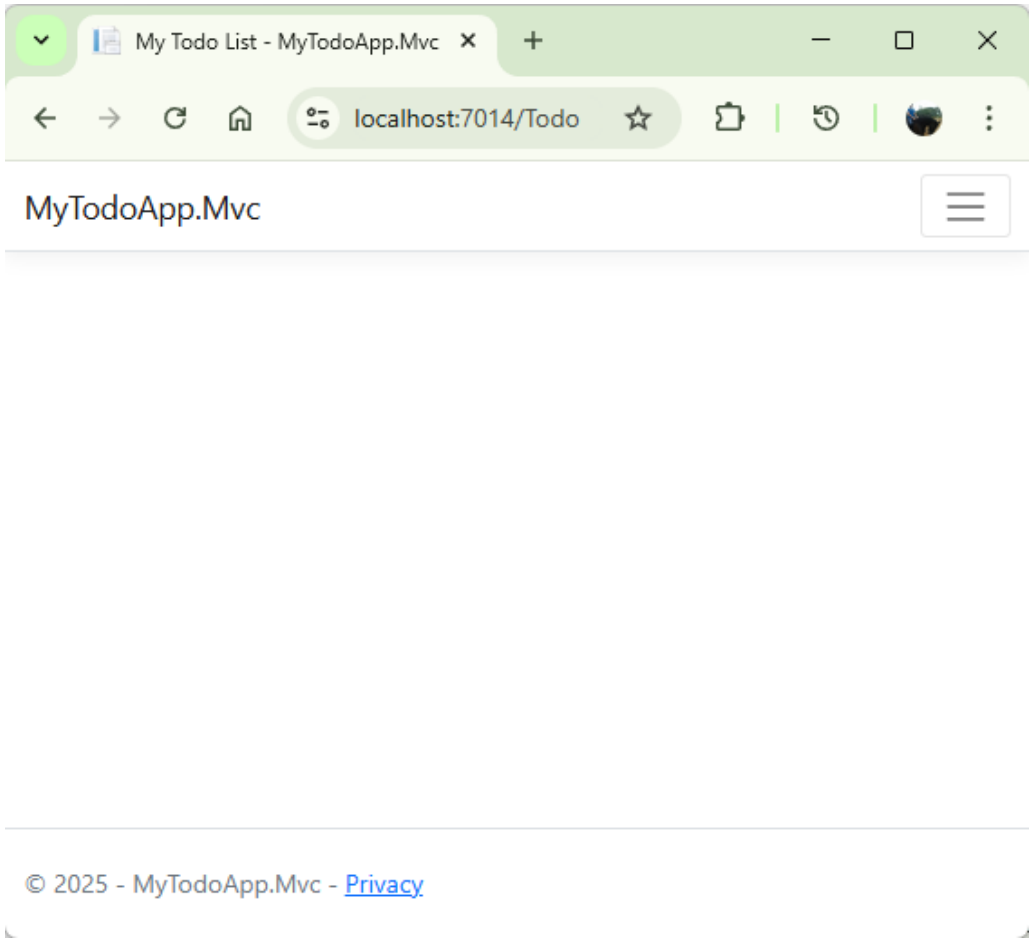
```
public class TodoItem
{
    10 references
    public Guid Id { get; set; }

    9 references
    public string Title { get; set; }

    10 references
    public bool IsDone { get; set; }

    4 references
    public string? OwnerId { get; set; }

    10 references
    public DateTime? DueAt { get; set; }
}
```

Implement Index

```
public IActionResult Index()
{
    var myTodos = new List<TodoItem>
    {
            new TodoItem
            {
                Id = Guid.NewGuid(),
                OwnerId = "user1",
                IsDone = false,
                Title = "Finish ASP.NET MVC assignment",
                DueAt = DateTimeOffset.UtcNow.AddDays(1)
            },
            new TodoItem
            {
                Id = Guid.NewGuid(),
                OwnerId = "user1",
                IsDone = true,
                Title = "Review Entity Framework notes",
                DueAt = DateTimeOffset.UtcNow.AddDays(-2)
            },
            new TodoItem
            {
                Id = Guid.NewGuid(),
                OwnerId = "user2",
                IsDone = false,
                Title = "Prepare presentation slides",
                DueAt = DateTimeOffset.UtcNow.AddDays(3)
            },
    };

    return View(myTodos);
}
```

ViewData

```
@using MyTodoApp.Mvc.Models.TodoViewModels

@model List<TodoItem>
@{
    ViewData["Title"] = "My Todo List";
}
```

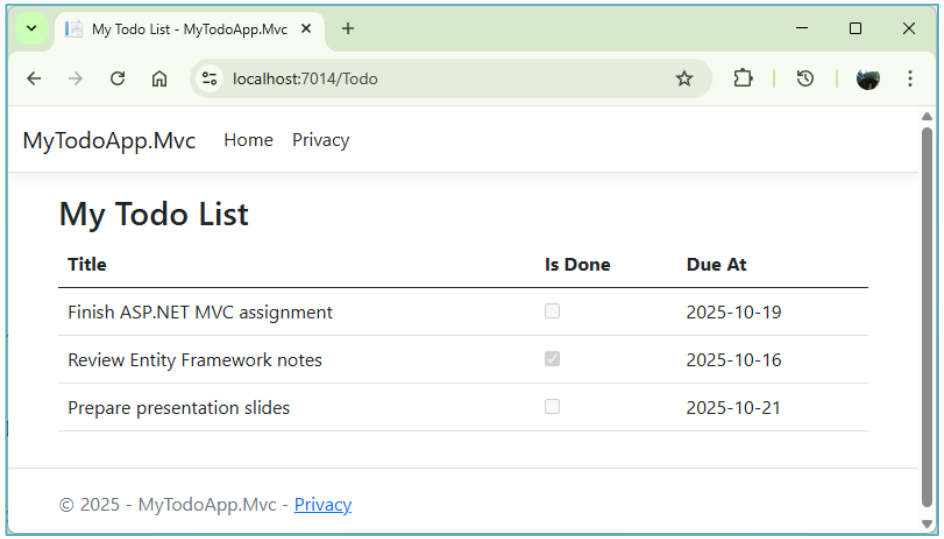


Lifecycle summary

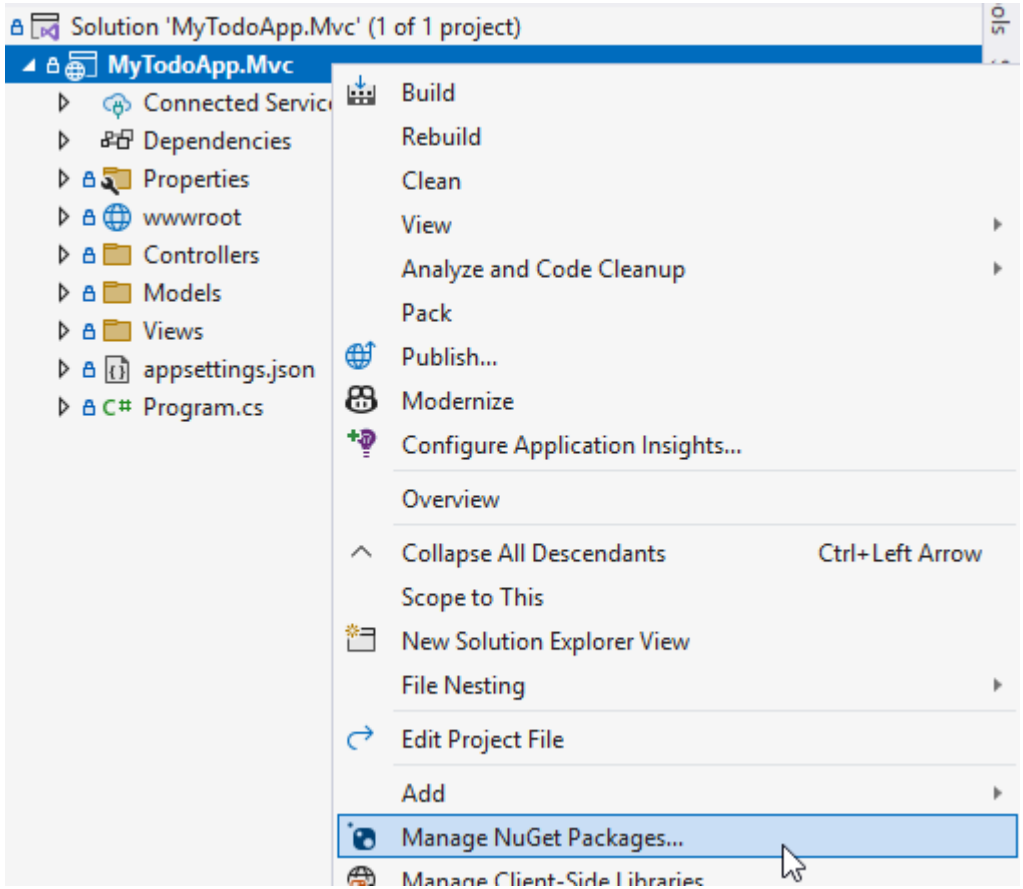
Step	Component	Action
1	Controller	Returns a ViewResult → View engine starts rendering.
2	View (Index.cshtml)	Executes your code block: ViewData["Title"] = "My Todo List";
3	Layout (_Layout.cshtml)	Uses that value in <title>@ViewData["Title"]</title>
4	Razor Engine	Combines both into final HTML sent to browser.

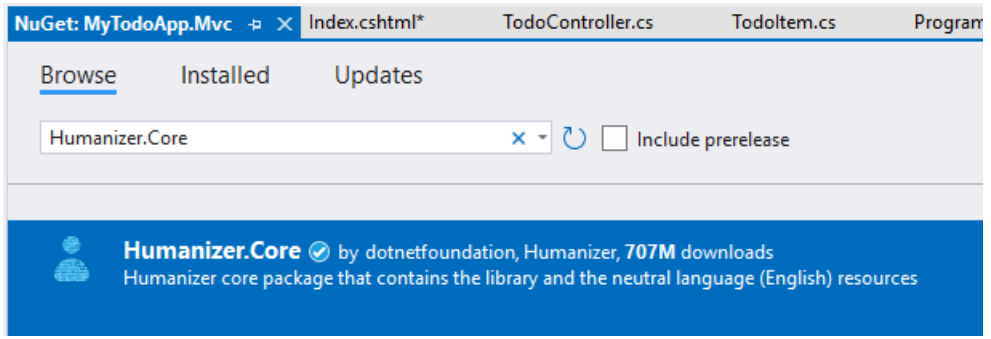
View

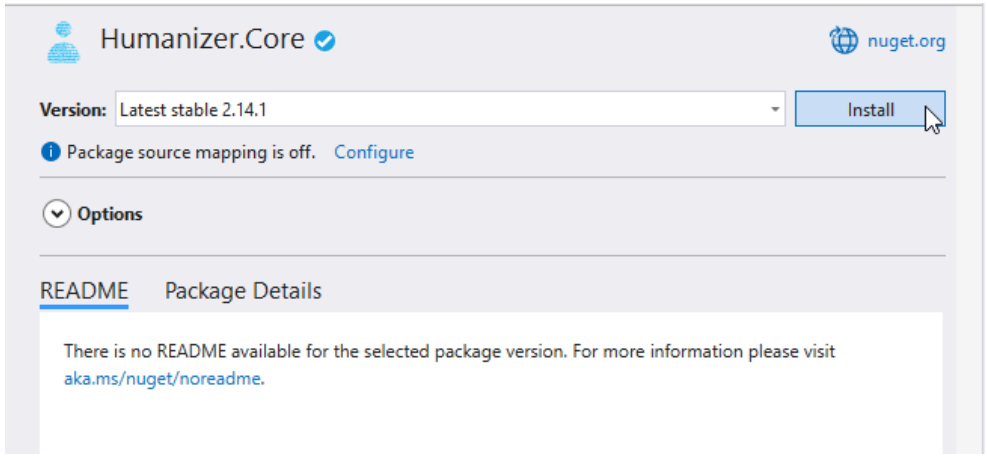
```
<table class="table table-striped">
    <thead>
        <tr>
            <th>Title</th>
            <th>Is Done</th>
            <th>Due At</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @item.Title
                </td>
                <td>
                    <input asp-for="@item.IsDone" disabled />
                </td>
                <td>@item.DueAt?.ToString("yyyy-MM-dd")</td>
            </tr>
        }
    </tbody>
</table>
```



NuGet

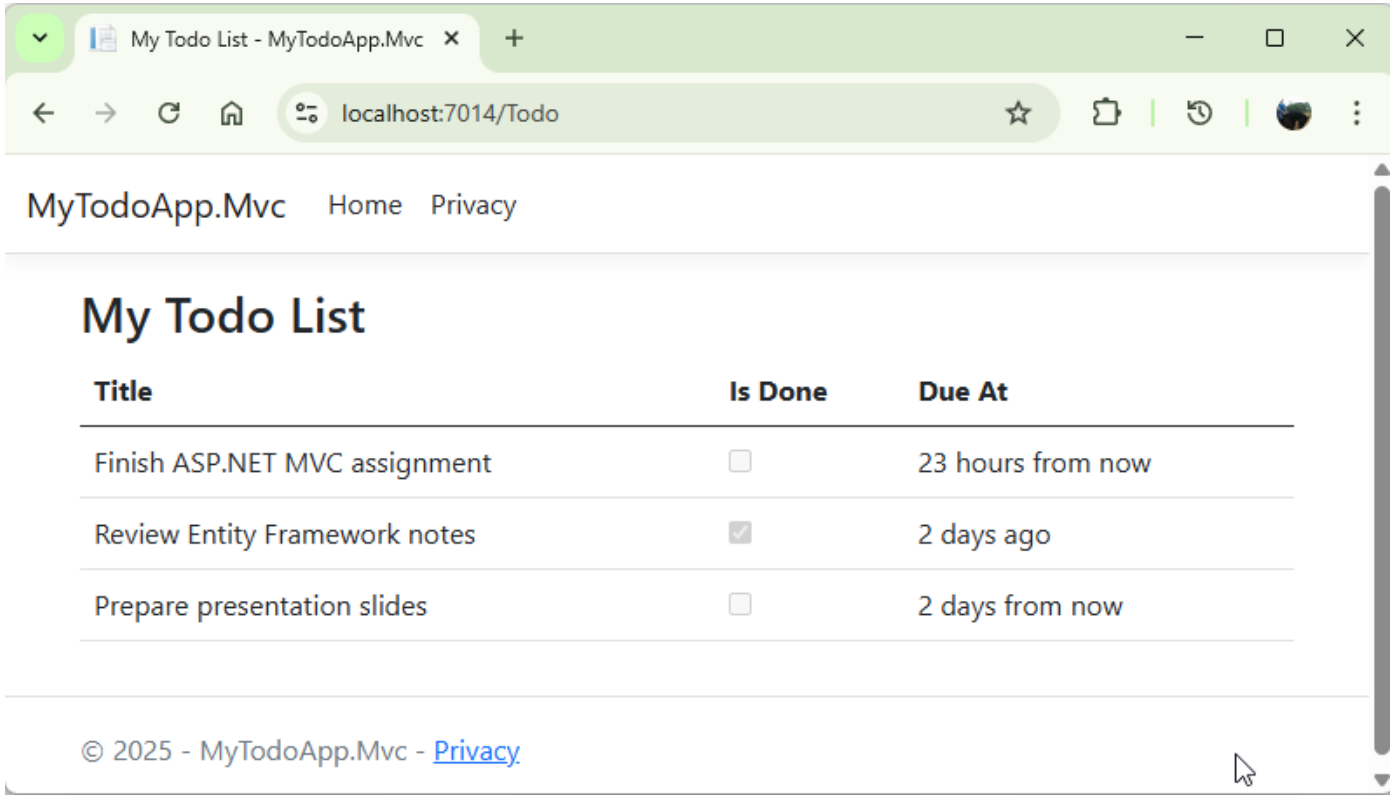

Solution 'MyTodoApp.Mvc' (1 of 1 project)
MyTodoApp.Mvc
Connected Servic
Dependencies
Properties
wwwroot
Controllers
Models
Views
appsettings.json
Program.cs
Build
Rebuild
Clean
View
Analyze and Code Cleanup
Pack
Publish...
Modernize
Configure Application Insights...
Overview
Collapse All Descendants
Ctrl+Left Arrow
Scope to This
New Solution Explorer View
File Nesting
Edit Project File
Add
Manage NuGet Packages...


NuGet: MyTodoApp.Mvc
Index.cshtml*
TodoController.cs
TodoItem.cs
Program
Browse
Installed
Updates
Humanizer.Core
Include prerelease
Humanizer.Core by dotnetfoundation, Humanizer, 707M downloads
Humanizer core package that contains the library and the neutral language (English) resources


Humanizer.Core
nuget.org
Version:
Latest stable 2.14.1
Install
Package source mapping is off.
Configure
Options
README
Package Details
There is no README available for the selected package version. For more information please visit
aka.ms/nuget/noreadme.

Humanize

```razor
@foreach (var item in Model)
{
    <tr>
        <td>
            @item.Title
        </td>
        <td>
            <input asp-for="@item.IsDone" disabled />
        </td>
        <td>@item.DueAt?.Humanize()</td>
    </tr>
}
```



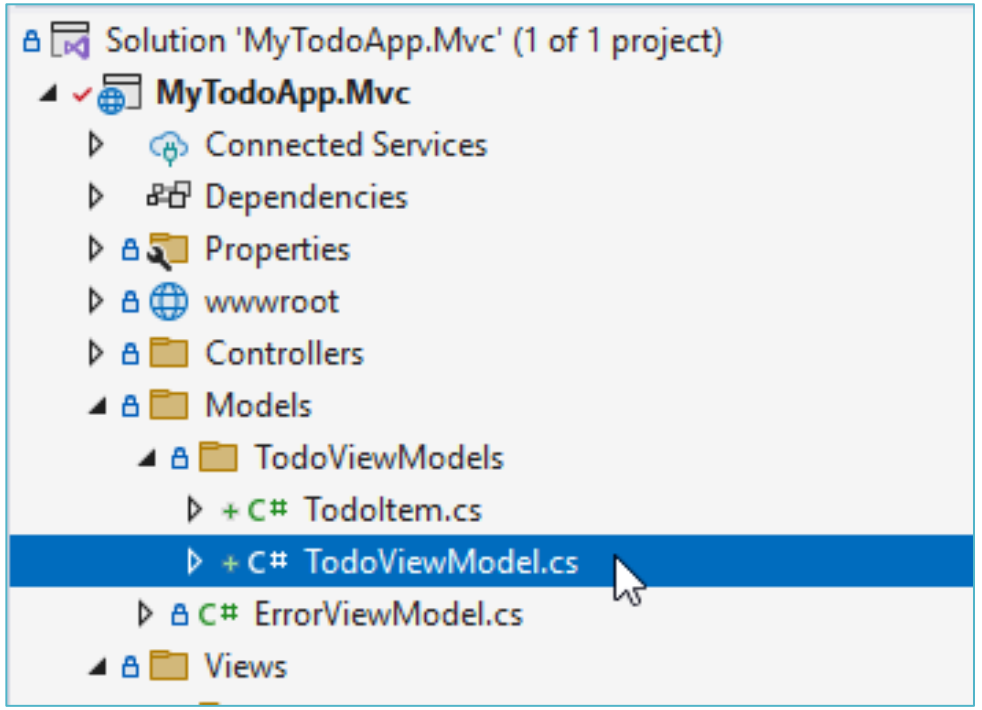
Using ViewModel

TodoViewModel.cs

MyTodoApp.Mvc

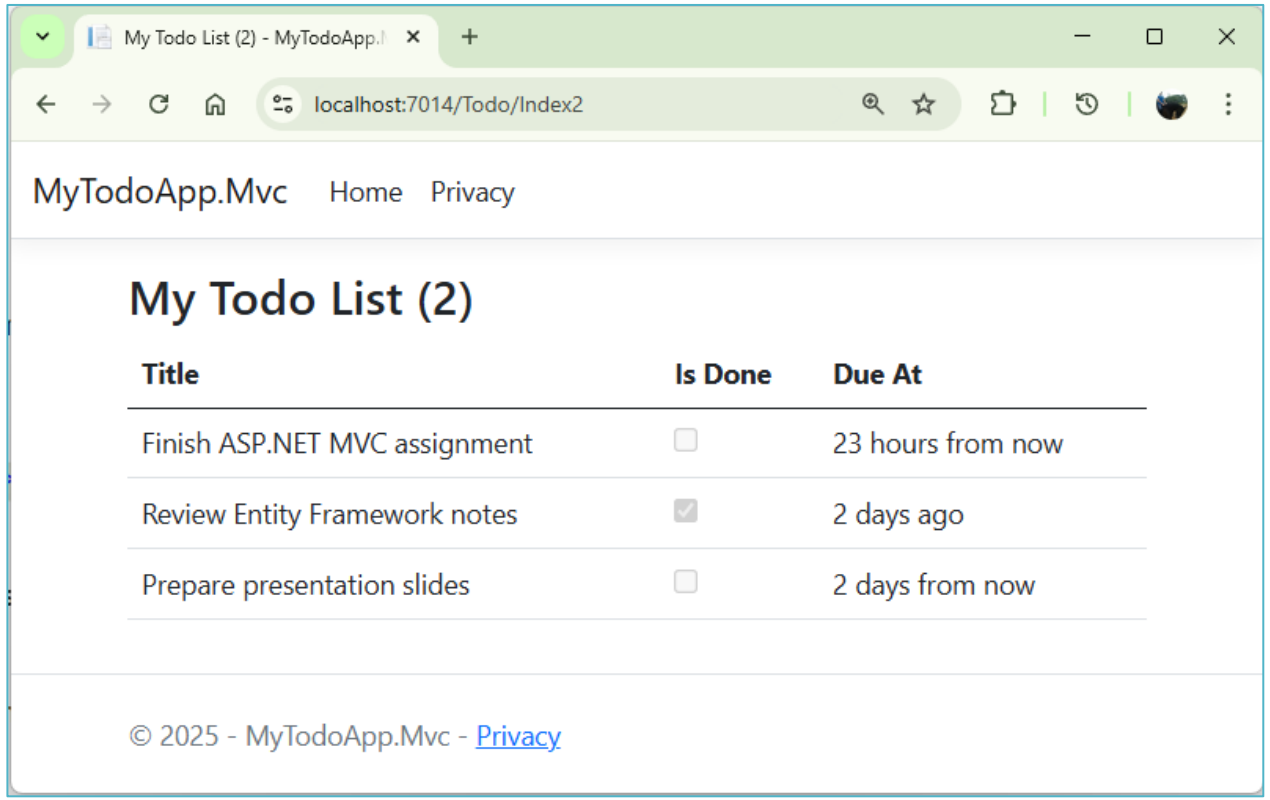
MyTodoApp.Mvc.Mo

```
1  namespace MyTodoApp.Mvc.Models.TodoViewModels;
2
   3 references
3  public class TodoViewModel
4  {
       4 references
5      public List<TodoItem> TodoItems { get; set; }
6  }
```



Using ViewModel

```
@if (Model.TodoItems == null || Model.TodoItems.Count == 0)
{
    <p>No todo items found.</p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Title</th>
                <th>Is Done</th>
                <th>Due At</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model.TodoItems)
            {
                <tr>
                    <td>
                        <span>@item.Title</span>
                    </td>
                    <td>
                        <input asp-for="@item.IsDone" disabled />
                    </td>
                    <td>
                        <span>@item.DueAt?.Humanize()</span>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
```



asp-for

It’s primarily used in form elements (<input>, <textarea>, <select>) to automatically:

set the name and id attributes correctly for model binding

preserve validation attributes

maintain the model value after form submission

Example:

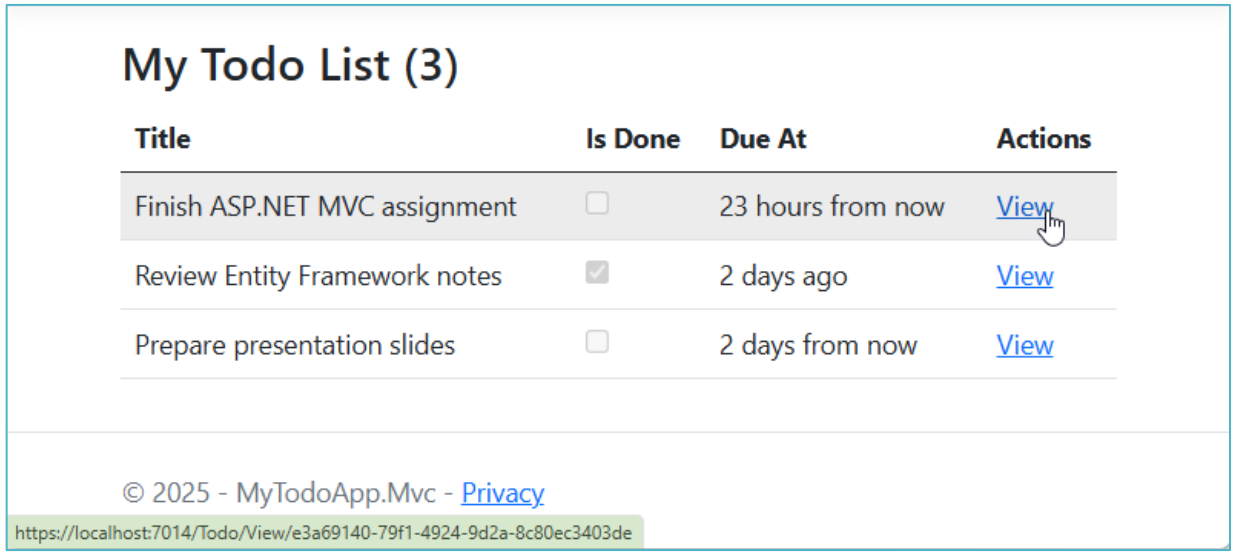
<input asp-for="Title" />

is equivalent to:

<input type="text" id="Title" name="Title" value="@Model.Title" />

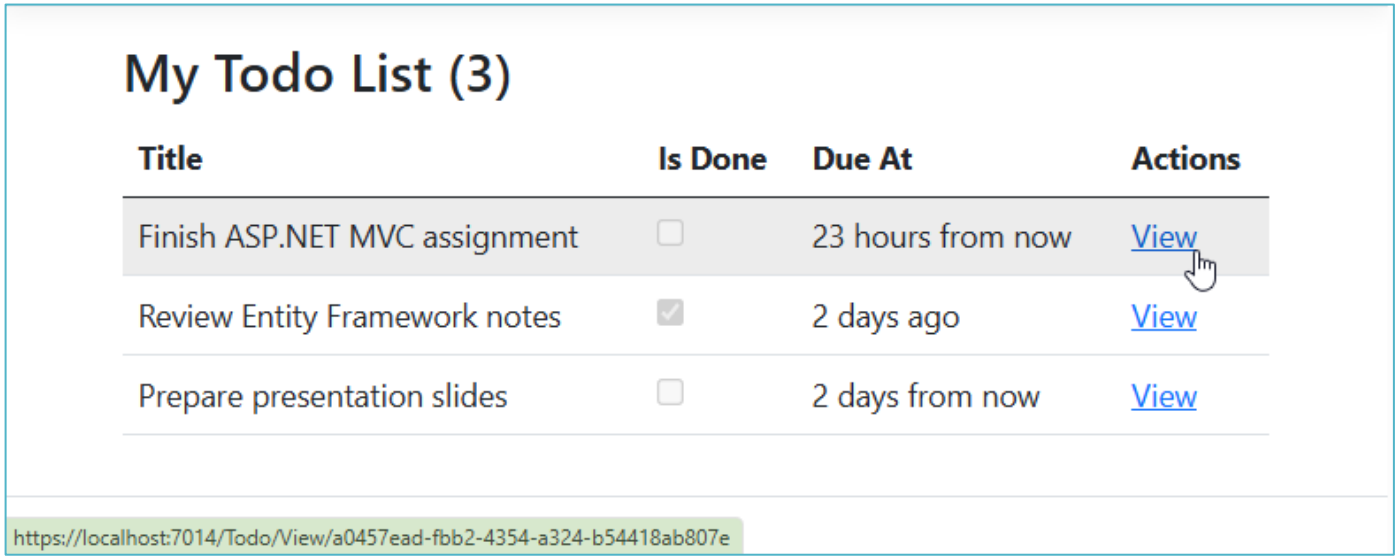
View Action

```
else
{
    <table class="table table-hover">
        <thead>
            <tr>
                <th>Title</th>
                <th>Is Done</th>
                <th>Due At</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model.TodoItems)
            {
                <tr>
                    <td>
                        <span>@item.Title</span>
                    </td>
                    <td>
                        <input asp-for="@item.IsDone" disabled />
                    </td>
                    <td>
                        <span>@item.DueAt?.Humanize()</span>
                    </td>
                    <td>
                        <a asp-controller="Todo"
                           asp-action="View"
                           asp-route-id="@item.Id">View</a>
                    </td>
                </tr>
            }
        </tbody>
    </table>
}
```



Reserve Data

```
private static readonly List<TodoItem> _todoItems = new()
{
    new TodoItem
    {
        Id = Guid.NewGuid(),
        OwnerId = "user1",
        IsDone = false,
        Title = "Finish ASP.NET MVC assignment",
        DueAt = DateTimeOffset.UtcNow.AddDays(1)
    },
    new TodoItem
    {
        Id = Guid.NewGuid(),
        OwnerId = "user1",
        IsDone = true,
        Title = "Review Entity Framework notes",
        DueAt = DateTimeOffset.UtcNow.AddDays(-2)
    },
    new TodoItem
    {
        Id = Guid.NewGuid(),
        OwnerId = "user2",
        IsDone = false,
        Title = "Prepare presentation slides",
        DueAt = DateTimeOffset.UtcNow.AddDays(3)
    }
};
```

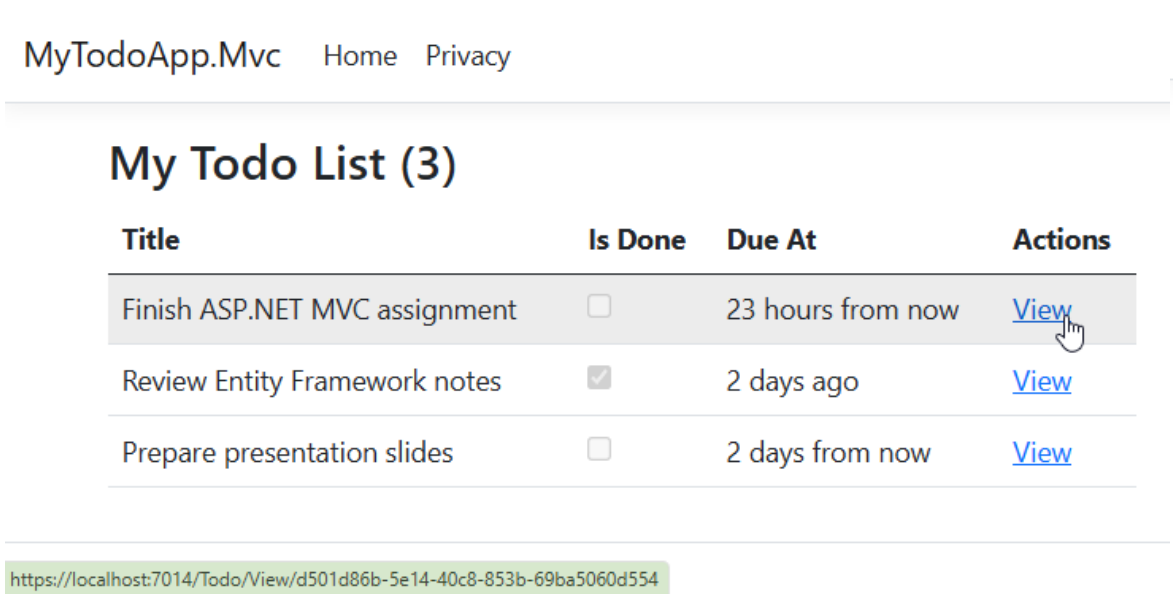


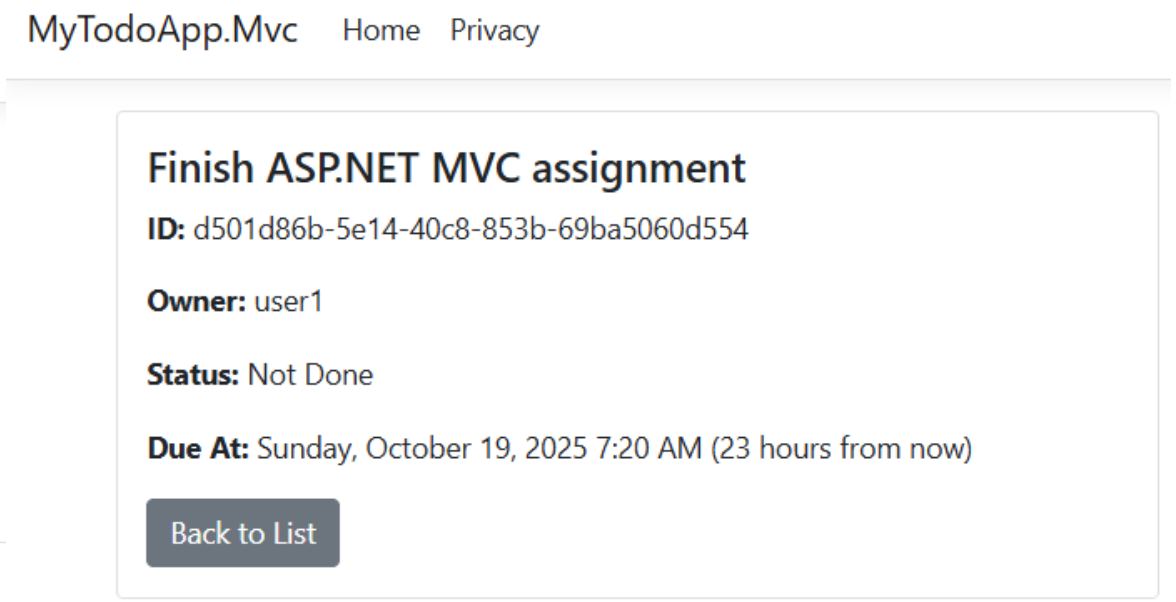
Add View Item

```razor
@model TodoItem
@{
    ViewData["Title"] = "View Item";
}

<div class="card" style="max-width: 500px;">
    <div class="card-body">
        <h4 class="card-title">@Model.Title</h4>

        <p><strong>ID:</strong> @Model.Id</p>
        <p><strong>Owner:</strong> @Model.OwnerId</p>
        <p>
            <strong>Status:</strong>
            @(Model.IsDone ? "Completed" : "Not Done")
        </p>

        @if (Model.DueAt.HasValue)
        {
            <p><strong>Due At:</strong> @Model.DueAt?.ToString("f") (@Model.DueAt?.Humanize())</p>
        }

        <a asp-controller="Todo" asp-action="Index3" class="btn btn-secondary">Back to List</a>
    </div>
</div>
```

```
0 references
public IActionResult View(Guid id)
{
    var item = _todoItems.FirstOrDefault(item => item.Id == id);
    if (item == null) return NotFound();
    return View(item);
}
```

Add View Item





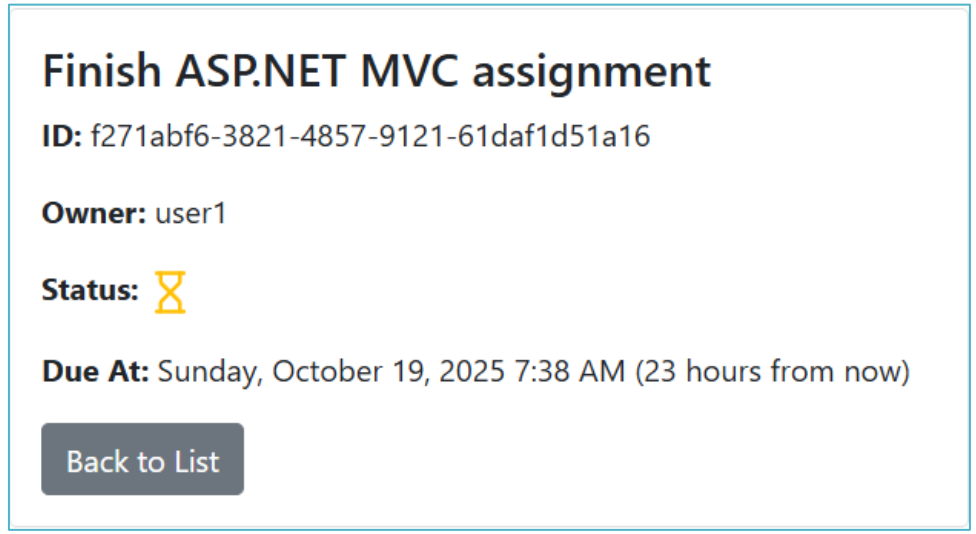
Using lucide.dev

```
<!-- Lucide CDN -->
<script src="https://unpkg.com/lucide@latest"></script>
```

1

```
<div class="card" style="max-width: 500px;">
    <div class="card-body">
        <h4 class="card-title">@Model.Title</h4>

        <p><strong>ID:</strong> @Model.Id</p>
        <p><strong>Owner:</strong> @Model.OwnerId</p>
        <p>
            <strong>Status:</strong>
            @if (Model.IsDone)
            {
                <i data-lucide="check-circle" class="text-success">Completed</i>
            }
            else
            {
                <i data-lucide="hourglass" class="text-warning">Not Done</i>
            }
        </p>

        @if (Model.DueAt.HasValue)
        {
            <p><strong>Due At:</strong> @Model.DueAt?.ToString("f") (@Model.DueAt?.Humanize())</p>
        }

        <a asp-controller="Todo" asp-action="Index3" class="btn btn-secondary">Back to List</a>
    </div>
</div>
```

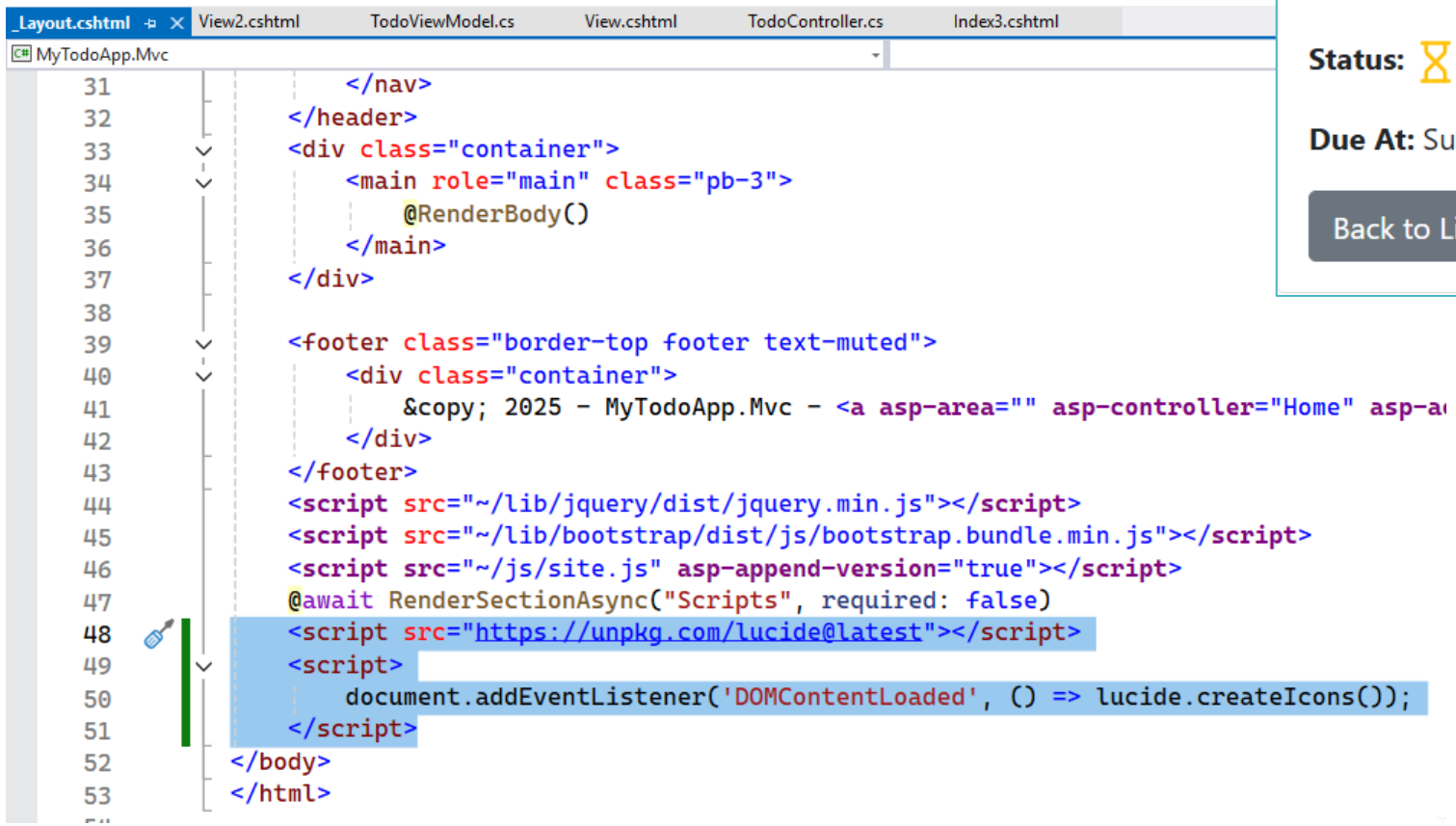
2

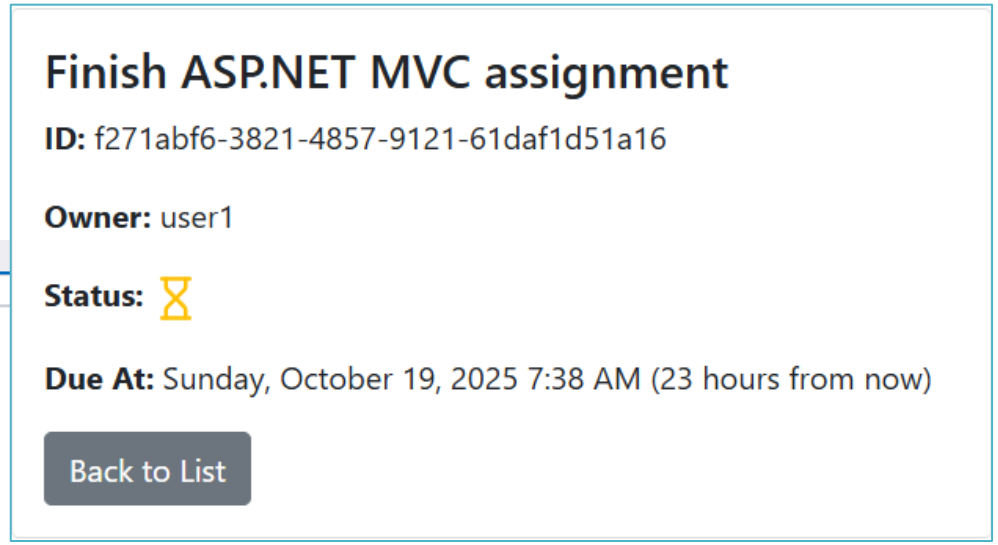



```
<!-- ☑ Activate Lucide icons -->
<script>
    lucide.createIcons();
</script>
```

3

_Layout.cshtml

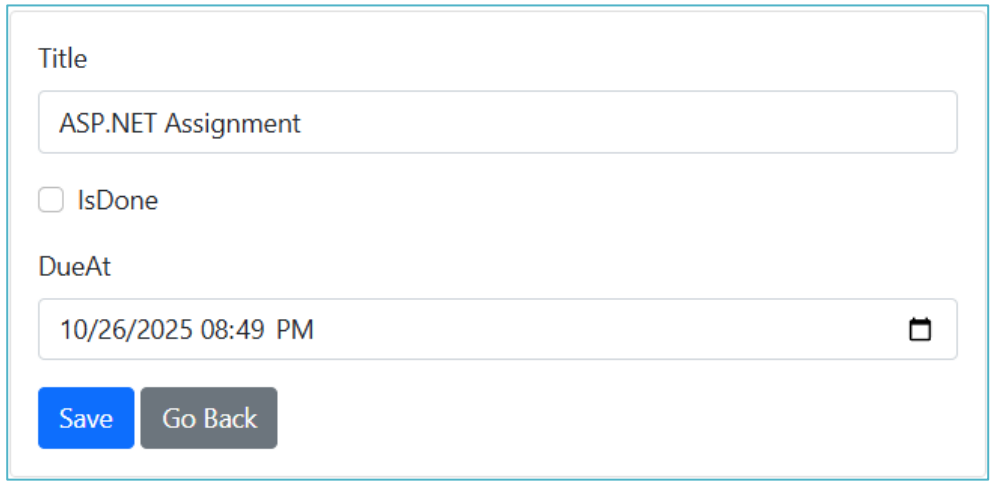




Module 4

Forms

```
<form asp-action="Edit" method="post">

    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="OwnerId" />

    <div class="mb-3">
        <label asp-for="Title" class="form-label"></label>
        <input asp-for="Title" class="form-control" />
    </div>

    <div class="mb-3 form-check">
        <input asp-for="IsDone" class="form-check-input" />
        <label asp-for="IsDone" class="form-check-label"></label>
    </div>

    <div class="mb-3">
        <label asp-for="DueAt" class="form-label"></label>
        <input type="datetime-local" asp-for="DueAt"
               value="@Model.DueAt?.ToString("yyyy-MM-ddTHH:mm")" class="form-control" />
    </div>

    <button type="submit" class="btn btn-primary">Save</button>
    <a asp-action="Index" class="btn btn-secondary">Cancel</a>
</form>
```

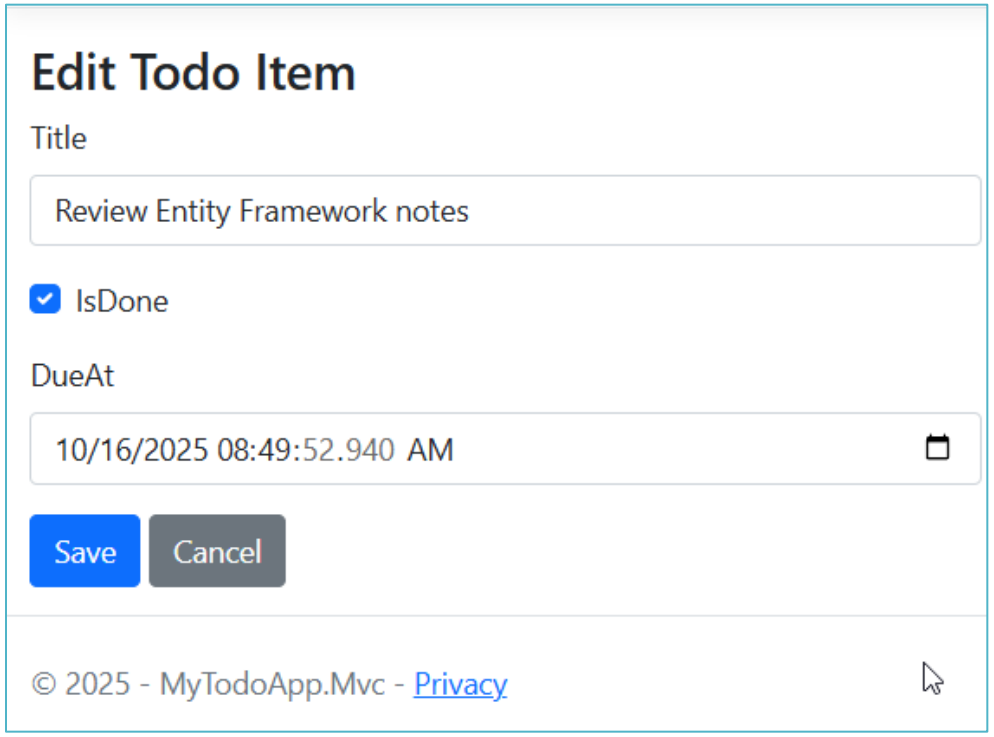


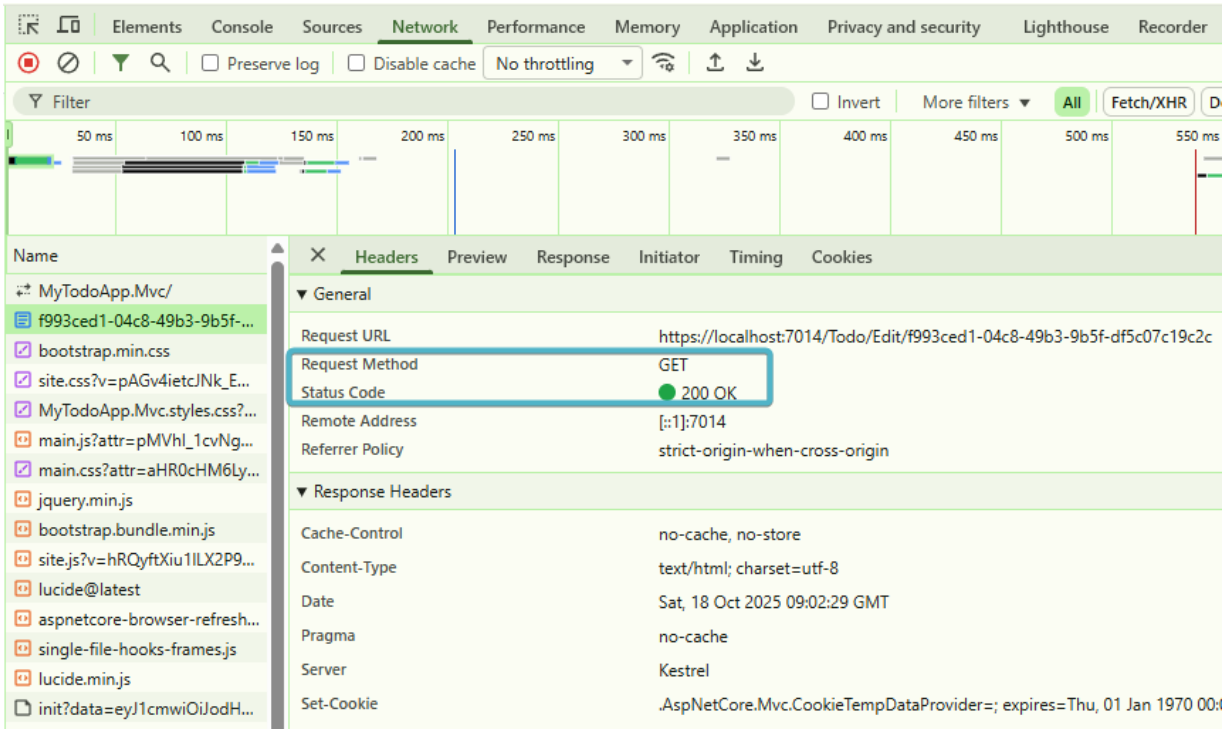
Edit Action

```
[HttpGet]
public IActionResult EditItem(Guid
id)
{
    var item =
_todoItems.FirstOrDefault(x => x.Id
== id);
    if (item == null)
        return NotFound();

    return View(item);
}
```

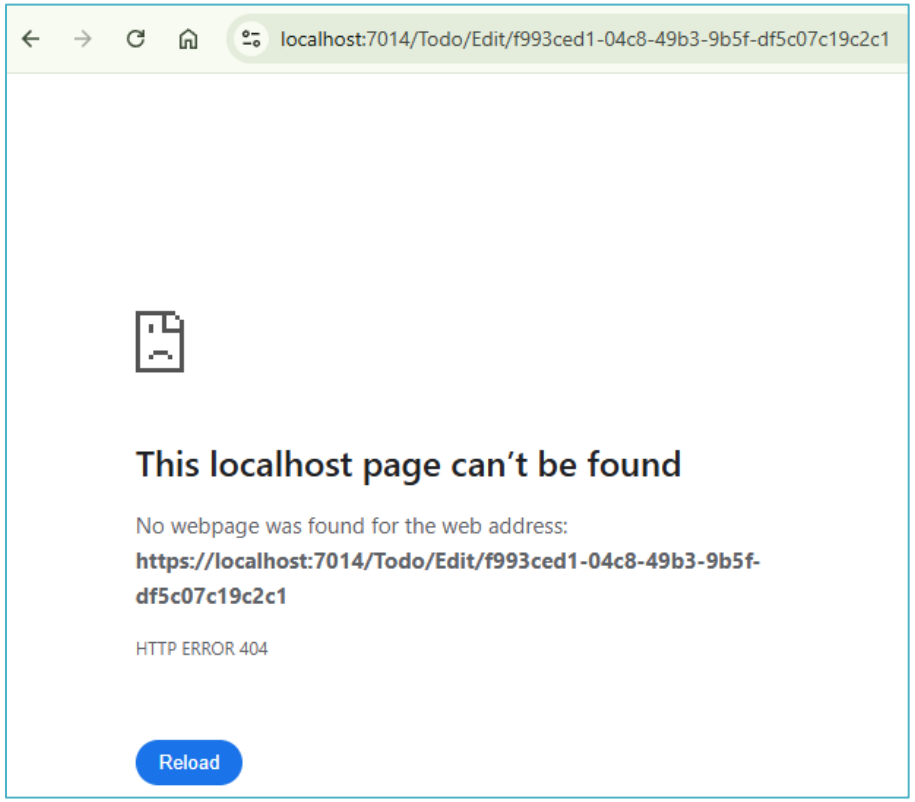
```
[HttpPost]
public IActionResult EditItem(TodoItem updated)
{
    var existing = _todoItems.FirstOrDefault(x =>
x.Id == updated.Id);
    if (existing == null)
        return NotFound();

    // Update only editable fields
    existing.Title = updated.Title;
    existing.IsDone = updated.IsDone;
    existing.DueAt = updated.DueAt;

    return RedirectToAction(nameof(EditItem), new
{ id = existing.Id });
}
```

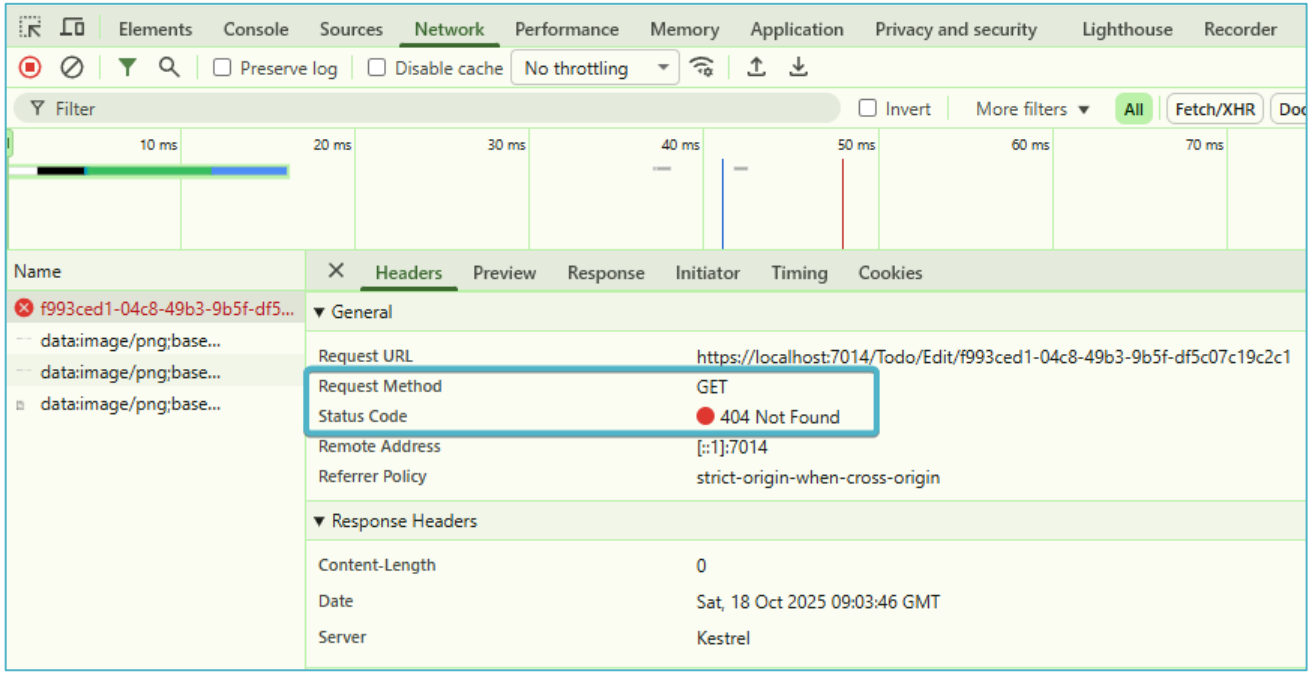
NotFound


Edit Todo Item
Title
Review Entity Framework notes
IsDone
DueAt
10/16/2025 08:49:52.940 AM
Save
Cancel
© 2025 - MyTodoApp.Mvc - Privacy


Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
50 ms
100 ms
150 ms
200 ms
250 ms
300 ms
350 ms
400 ms
450 ms
500 ms
550 ms
Name
MyTodoApp.Mvc/
f993ced1-04c8-49b3-9b5f-...
bootstrap.min.css
site.css?v=pAGv4ietcJNk_E...
MyTodoApp.Mvc.styles.css?...
main.js?attr=pMVhl_1cvNg...
main.css?attr=aHR0cHM6Ly...
jquery.min.js
bootstrap.bundle.min.js
site.js?v=hRQyftXiu1ILX2P9...
lucide@latest
aspnetcore-browser-refresh...
single-file-hooks-frames.js
lucide.min.js
init?data=eyJ1cmwiOiJodH...
Headers
Preview
Response
Initiator
Timing
Cookies
General
Request URL
https://localhost:7014/Todo/Edit/f993ced1-04c8-49b3-9b5f-df5c07c19c2c
Request Method
GET
Status Code
200 OK
Remote Address
[::1]:7014
Referrer Policy
strict-origin-when-cross-origin
Response Headers
Cache-Control
no-cache, no-store
Content-Type
text/html; charset=utf-8
Date
Sat, 18 Oct 2025 09:02:29 GMT
Pragma
no-cache
Server
Kestrel
Set-Cookie
.AspNetCore.Mvc.CookieTempDataProvider=; expires=Thu, 01 Jan 1970 00:

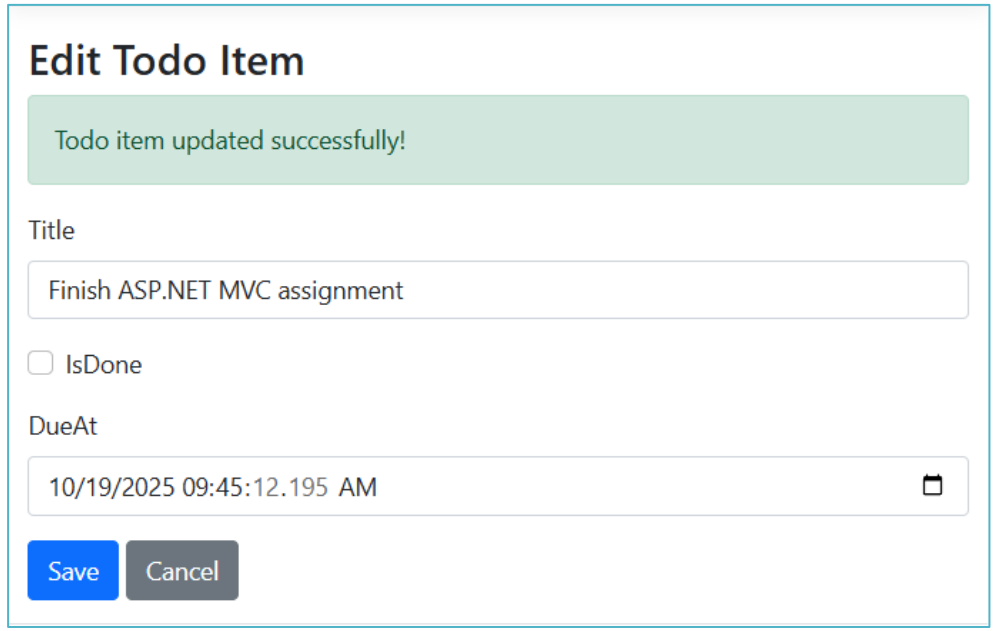
NotFound


localhost:7014/Todo/Edit/f993ced1-04c8-49b3-9b5f-df5c07c19c2c1
This localhost page can't be found
No webpage was found for the web address:
https://localhost:7014/Todo/Edit/f993ced1-04c8-49b3-9b5f-df5c07c19c2c1
HTTP ERROR 404
Reload


Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Preserve log
Disable cache
No throttling
Filter
Invert
More filters
All
Fetch/XHR
10 ms
20 ms
30 ms
40 ms
50 ms
60 ms
70 ms
Name
f993ced1-04c8-49b3-9b5f-df5...
data:image/png;base...
data:image/png;base...
data:image/png;base...
Headers
Preview
Response
Initiator
Timing
Cookies
General
Request URL
https://localhost:7014/Todo/Edit/f993ced1-04c8-49b3-9b5f-df5c07c19c2c1
Request Method
GET
Status Code
404 Not Found
Remote Address
[::1]:7014
Referrer Policy
strict-origin-when-cross-origin
Response Headers
Content-Length
0
Date
Sat, 18 Oct 2025 09:03:46 GMT
Server
Kestrel

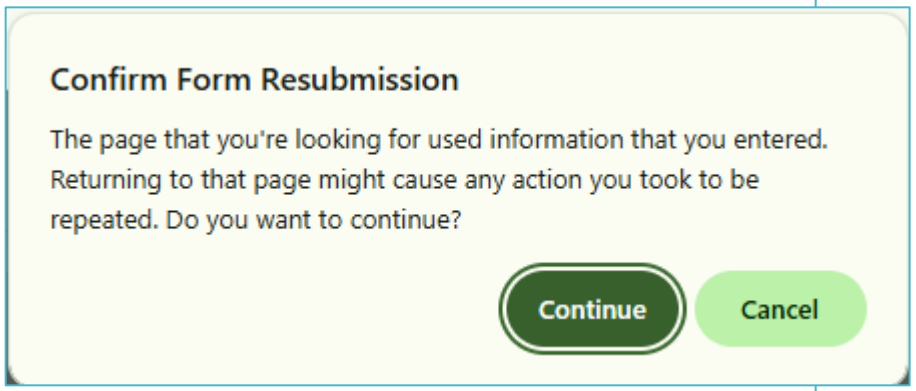
Show Successful Message

```
@if (TempData["Message"] != null)
{
    <div class="alert alert-
success">@TempData["Message"]</div>
}
```



Edit Action

```
[HttpPost]
public IActionResult EditItem(TodoItem updated)
{
    var existing = _todoItems.FirstOrDefault(x => x.Id ==
updated.Id);
    if (existing == null)
        return NotFound();

    // Update only editable fields
    existing.Title = updated.Title;
    existing.IsDone = updated.IsDone;
    existing.DueAt = updated.DueAt;

    TempData["Message"] = "Todo item updated successfully!";
    return View(item);
}
```



RedirectToAction

```
[HttpPost]
public IActionResult EditItem(TodoItem updated)
{
    var existing = _todoItems.FirstOrDefault(x => x.Id == updated.Id);
    if (existing == null)
        return NotFound();

    // Update only editable fields
    existing.Title = updated.Title;
    existing.IsDone = updated.IsDone;
    existing.DueAt = updated.DueAt;

    TempData["Message"] = "Todo item updated successfully!";
    return RedirectToAction(nameof(EditItem), new { id = existing.Id });
}
```

Rendering Simple Data in ASP.NET Core MVC

Key Points:

ASP.NET Core MVC views use the **Razor engine**.

Razor lets you mix **C# code** and **HTML markup**.

Code executed on the **server** produces dynamic content for the **browser**.

```
@{
    ViewData["Title"] = "Home Page";
}
```

Explanation:

Begins with @ → tells Razor we're writing server-side C# code.

Enclosed in { } → forms a *Razor code block*.

Executes before the page is rendered to the browser.

Equivalent to setting a property in the controller.

Setting ViewData in Controller

Controller Example:

```
public IActionResult Index()
{
    ViewData["Message"] = "Razor is Awesome!";
    return View();
}
```

Explanation:

ViewData is a **dictionary** for passing data from the controller to the view.

Keys are strings, values are objects.

It exists for a single request — data does not persist between requests.

Accessing ViewData in the View

View Example:

```
<h1>@ViewData["Title"]</h1>
<h2>@ViewData["Message"]</h2>
```

Concept:

These are **Razor expressions** (not code blocks).

Start with @ followed by C# code inline with HTML.

The value is **evaluated and rendered** directly in the browser.

Rendered Output Example:

```
<h1>Home Page</h1>
<h2>Razor is Awesome!</h2>
```

ViewData, ViewBag, TempData

Feature	Access Syntax	Lifetime	Redirect-Safe
ViewData	ViewData["Key"]	Current request	❌
ViewBag	ViewBag.Key	Current request	❌
TempData	TempData["Key"]	Next request only	✅

Why is OwnerId null??

```
[HttpPost]
1 reference
public IActionResult EditItem(TodoItem item)
{
    var it = _todoItems.Where(i => i.Id == item.Id).SingleOrDefault();

    if (it == null) return NotFound();

    it.Title = item.Title;
    it.DueAt = item.DueAt;
    it.IsDone = item.IsDone;
    it.OwnerId = item.OwnerId;   ≤ 1ms elapsed
                                 item.OwnerId | null

    TempData["Message"] = "Todo item updated successfully!";

    return RedirectToAction(nameof(EditItem), new { it.Id });
}
```

```
<input asp-for="Id" type="hidden" />
<input asp-for="OwnerId" type="hidden"/>
```

Model Validation in ASP.NET Core

Ensure that every input your application receives is **accurate, secure, and meaningful**.

Key Points:

- ✓ Protects against invalid or malicious data.
- ✓ Enforces business rules before saving to the database.
- ✓ Prevents runtime errors and improves data quality.

Approach	Type	Description
Data Annotations	Built-in	Attribute-based validation directly on model properties.
FluentValidation	External Library	Uses a fluent, rule-chaining syntax in a separate validator class.

How Data Annotations Work

- User submits form → Model Binder fills object.
- MVC automatically validates attributes.
- Results stored in ModelState.
- Controller checks:

```
if (!ModelState.IsValid)
    return View(model);
```

- Invalid inputs redisplayed with error messages.

What Is Client-Side Validation?

- Validation can happen **both on the server and in the browser**.
- **Client-side validation** prevents unnecessary postbacks by checking inputs instantly.
- Works through **JavaScript libraries** that interpret your model's Data Annotation rules.
- Enhances **user experience** while reducing server load.

Required Libraries

jQuery Libraries for Validation

You need to include:

- ✓ jQuery
- ✓ jQuery-Validation
- ✓ jQuery-Validation-Unobtrusive

Adding Scripts Manually

Example (placed near end of the page):

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```

- **ASP.NET Core default templates already include** a partial view named _ValidationScriptsPartial.cshtml.
- To enable client-side validation in your view:

```
@section Scripts {
    @Html.Partial("_ValidationScriptsPartial")
}
```

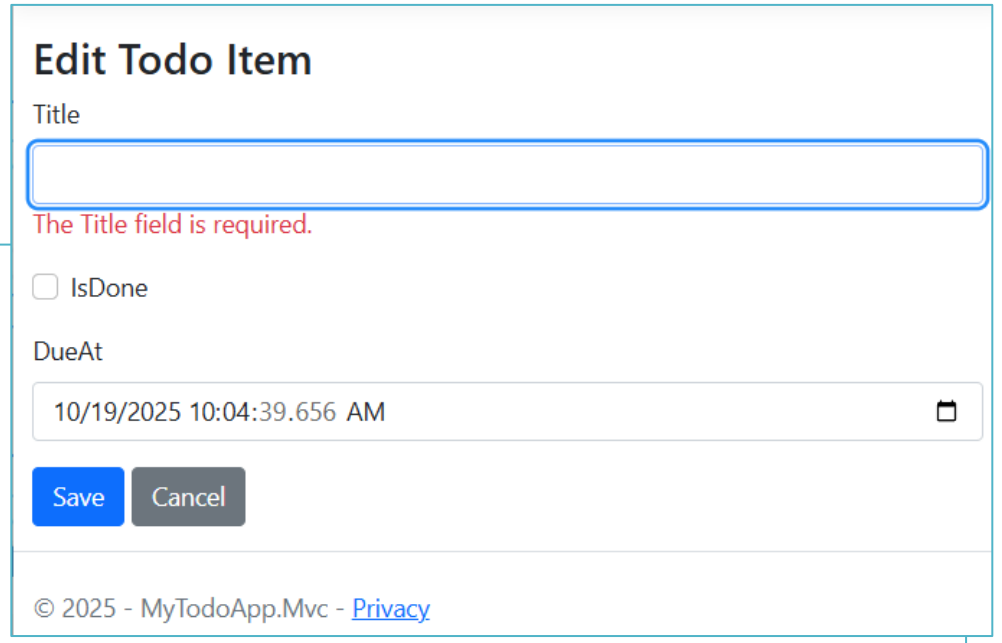
Validation

```razor
@if (TempData["Message"] != null)
{
    <div class="alert alert-success">@TempData["Message"]</div>
}

<form asp-action="Edit_V2" method="post">
    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="OwnerId" />

    <div class="mb-3">
        <label asp-for="Title" class="form-label"></label>
        <input asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger"></span>
    </div>

    <div class="mb-3 form-check">
        <input asp-for="IsDone" class="form-check-input" />
        <label asp-for="IsDone" class="form-check-label"></label>
    </div>

    <div class="mb-3">
        <label asp-for="DueAt" class="form-label"></label>
        <input asp-for="DueAt" class="form-control" type="datetime-local" />
        <span asp-validation-for="DueAt" class="text-danger"></span>
    </div>

    <button type="submit" class="btn btn-primary">Save</button>
    <a asp-action="Index3"  class="btn btn-secondary">Cancel</a>
</form>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

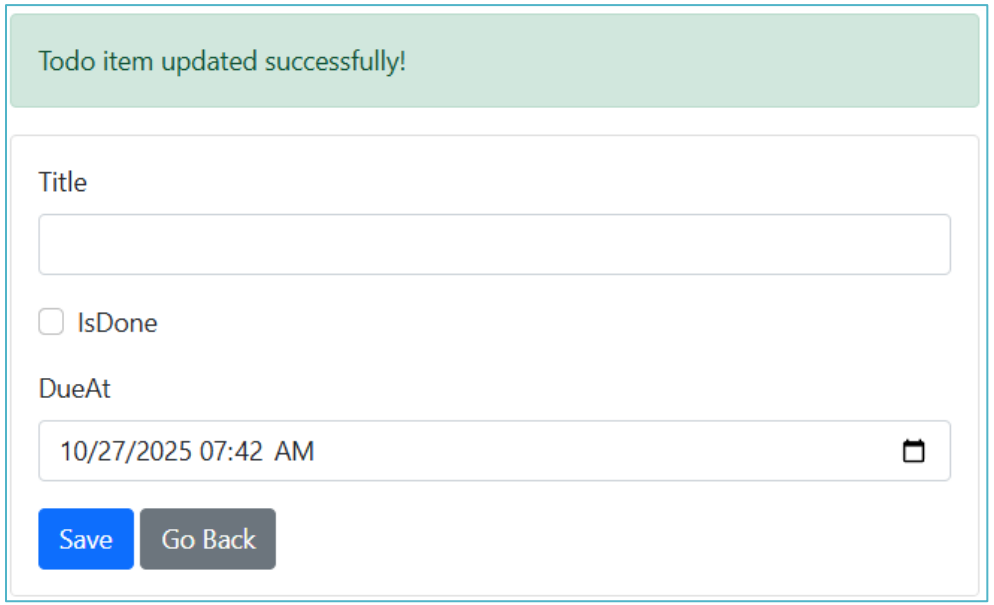
```
23 references
public class TodoItem
{
    22 references
    public Guid Id { get; set; }

    10 references
    public string OwnerId { get; set; }

    19 references
    public bool IsDone { get; set; }

    [Required(ErrorMessage = "Title is required")]
    21 references
    public string Title { get; set; }

    [Required(ErrorMessage = "Due date is required.")]
    24 references
    public DateTimeOffset? DueAt { get; set; }
}
```

Title Validation

```
public class TodoItem
{
    public Guid Id { get; set; }

    public string? OwnerId { get; set; }

    public bool IsDone { get; set; }

    [Required(ErrorMessage = "Title is required")]
    public string Title { get; set; }


    //[Required(ErrorMessage = "Due date is required.")]
    //[DisplayFormat(DataFormatString = "{0:yyyy-MM-ddTHH:mm}",
ApplyFormatInEditMode = true)]
    public DateTimeOffset? DueAt { get; set; }
}
```



JavaScript is Disabled



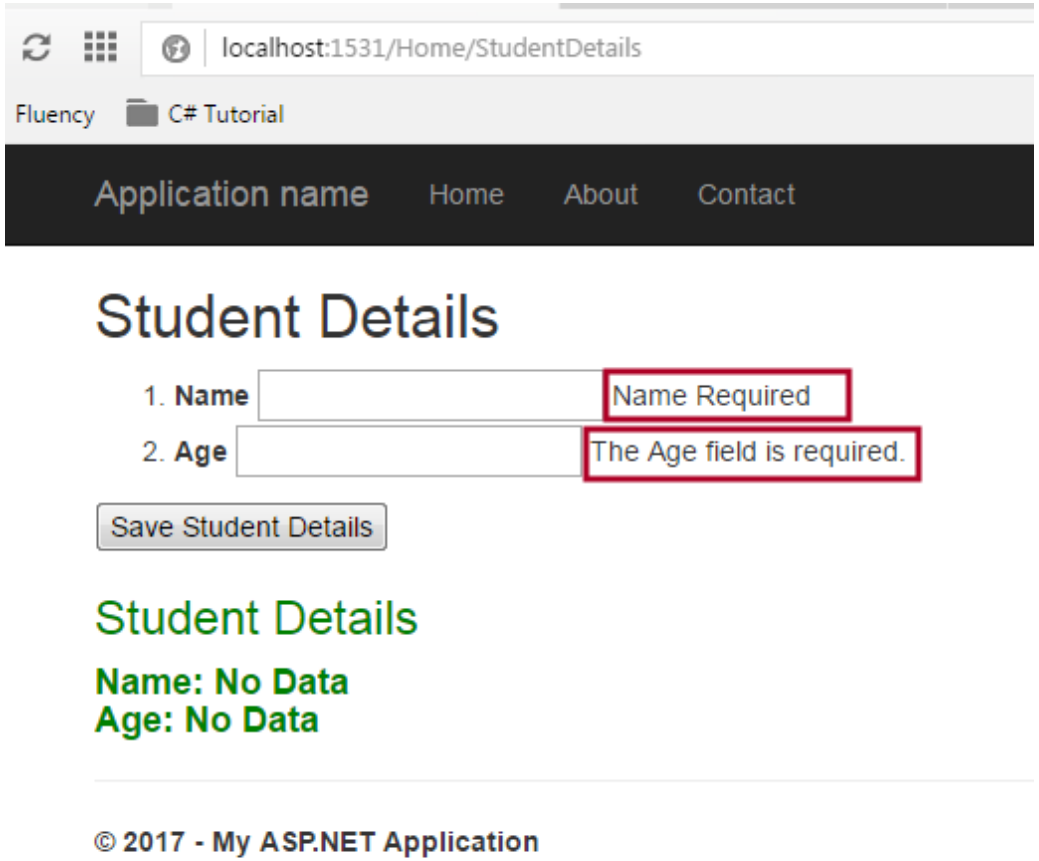

```
if (!ModelState.IsValid)
{
    return View(item);
}
```

ModelState

```
[HttpPost]
public ActionResult RegisterUser(Register userDetails)
{


    if (!ModelState.IsValid)
    {
        return View
    }

    return Redirect
}
```

ModelState {System.Web.Mvc.ModelStateDictionary}

Count	0
IsReadOnly	false
IsValid	true
Keys	Count = 0
Values	Count = 0
Non-Public members	



Understanding Model Binding

Controllers and Razor Pages **work with data from HTTP requests**.

Data can come from:

- Route data
- Form fields
- Query strings
- Headers or body (for APIs)

Model binding automates the process of:

- Retrieving incoming values
- Converting them from strings to .NET types
- Assigning them to parameters or model properties

Model Binding in Action

```
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult EditItem(TodoItem updated)
{
    if (!ModelState.IsValid)
    {
        return View(updated);
    }

    var existing = _todoItems.FirstOrDefault(x => x.Id == updated.Id);
    if (existing == null)
        return NotFound();

    // Update only editable fields
    existing.Title = updated.Title;
    existing.IsDone = updated.IsDone;
    existing.DueAt = updated.DueAt;

    TempData["Message"] = "Todo item updated successfully!";
    return RedirectToAction(nameof(EditItem), new { id = existing.Id
});
}
```

- Framework inspects the request (route, query, form, etc.).
- It tries to match names to parameters or model properties.
- Converts values (e.g.
- Adds any conversion or validation errors to ModelState.
- After binding, validation runs automatically.
- You check the result using:

Returning the View on Invalid Model

When you call return View(model);, MVC re-renders the same view with the current ModelState.

The Razor helpers like:

<input asp-for="Title" />

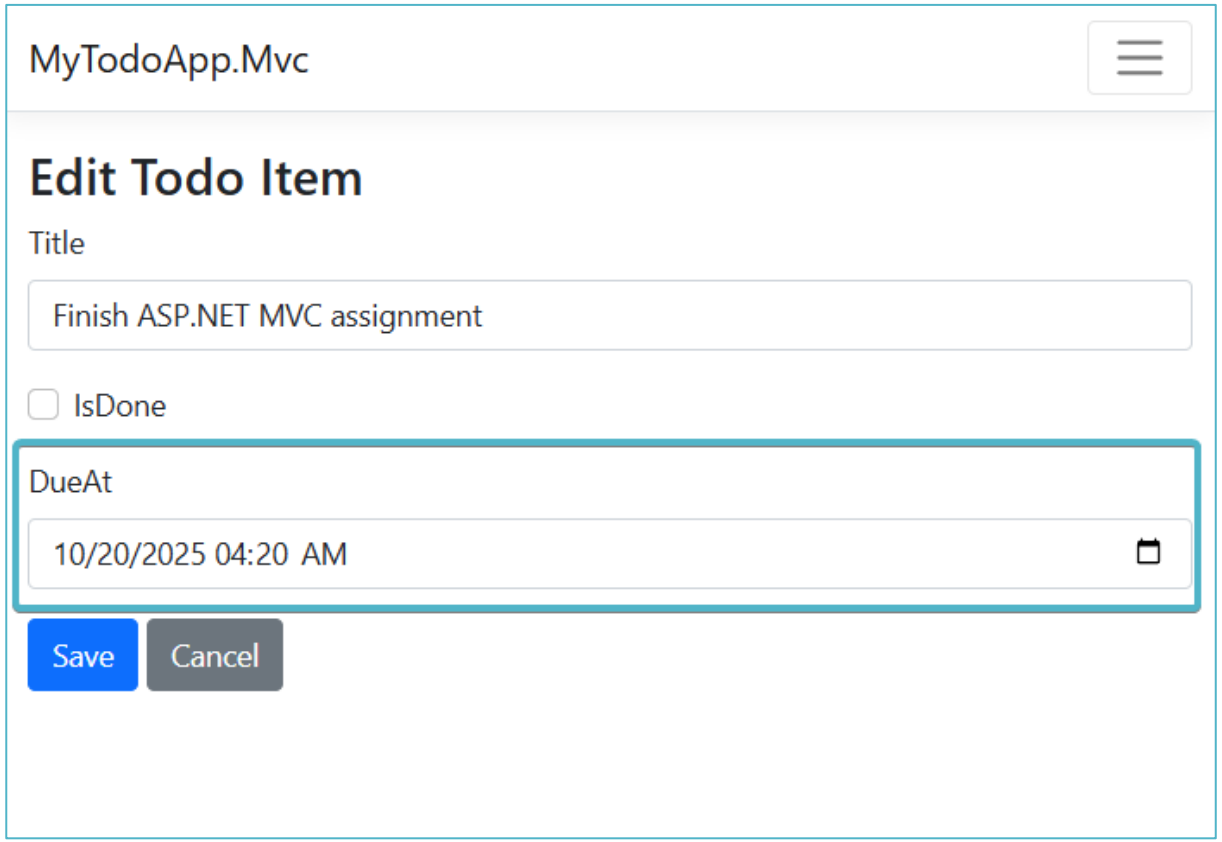
read the ModelState to:

Refill inputs with user-typed values.

Show error messages for invalid fields.

DispalyFormat Attribute

```
[Required(ErrorMessage = "Due date is required.")]
[DisplayFormat(DataFormatString = "{0:yyyy-MM-ddTHH:mm}", ApplyFormatInEditMode = true)]
24 references
public DateTimeOffset? DueAt { get; set; }
```



Introduction to HTML Helpers and Tag Helpers

- **Before Tag Helpers**

Before Tag Helpers were introduced, developers used **HTML Helpers** to render dynamic HTML content in Razor views.
They were written as **C# method calls** inside Razor code blocks.

```
<th>@Html.DisplayNameFor(model => model.Id)</th>
<th>@Html.DisplayNameFor(model => model.Title)</th>
<th>@Html.DisplayNameFor(model => model.DueAt)</th>
```

Moving to Tag Helpers

ASP.NET Core introduced **Tag Helpers** to simplify HTML generation.

✅ **Advantages:**

Cleaner, more readable Razor syntax

Familiar for frontend developers

Full HTML control with IntelliSense support

💡 **Rewriting with Tag Helpers**

```
@{ var first = Model.FirstOrDefault(); }
```

```
<h1>My first Todo item:</h1>

<label asp-for="@first.Title" class="fw-bold"></label>

: @first.Title
```

🧩 What changed:

- <label> is now a **normal HTML element**.
- asp-for Tag Helper binds it to the Title property.
- The tag helper automatically renders the label text.
- ⚠️ The closing tag is **required** — <label asp-for="@first.Title" /> will **not** render the value.

Common Tag Helpers for Forms

Tag Helper	Purpose
asp-for	Binds an HTML element to a model property
asp-action / asp-controller	Generates URLs to controller actions
asp-validation-for	Displays validation errors for a field
asp-validation-summary	Shows all model validation errors
asp-items	Populates <select> options from a data source

Tag Helpers vs HTML Helpers

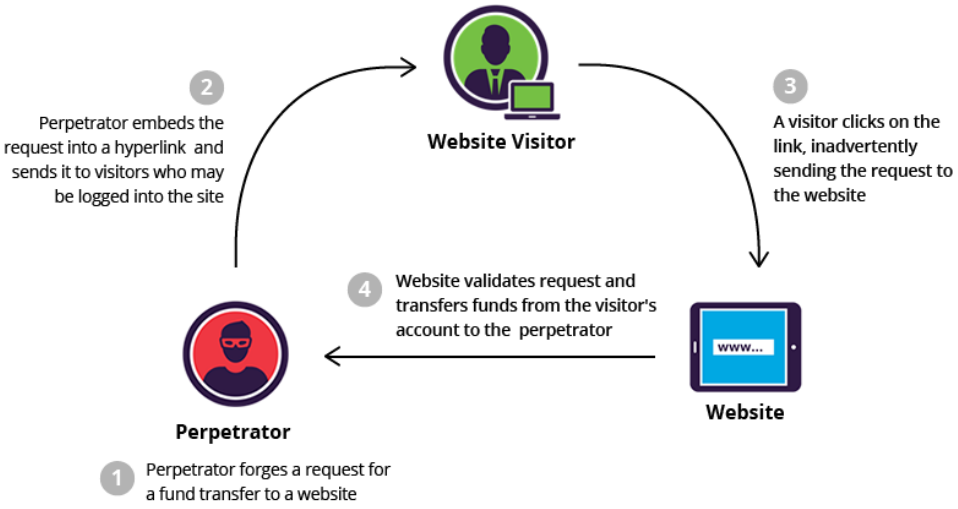
Feature	Tag Helpers	HTML Helpers
Syntax	HTML-like (<input asp-for="Name" />)	C# method (@Html.TextBoxFor(m => m.Name))
Readability	High	Less readable for designers
Modern ASP.NET Core	✅ Recommended	Legacy support

Understanding Cross-Site Request Forgery (CSRF)

- CSRF = *Cross-Site Request Forgery*
- Attacker tricks a user's browser into sending a **malicious request** to a trusted site.
- The request uses the user's **existing authentication (cookies)**.
- The server **cannot distinguish** between a real user action and a forged one.

Talking Points:

"In CSRF, the attacker doesn't steal your credentials. Instead, they use your own browser to perform actions on your behalf while you're logged in."



https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

How ASP.NET Core Protects You

```
[HttpPost]
[ValidateAntiForgeryToken]
1 reference
public IActionResult Edit_V2(TodoItem updated)
{
    var existing = _todoItems.FirstOrDefault(x => x.Id == updated.Id);
    if (existing == null)
        return NotFound();

    // Update only editable fields
    existing.Title = updated.Title;
    existing.IsDone = updated.IsDone;
    existing.DueAt = updated.DueAt;

    TempData["Message"] = "Todo item updated successfully!";
    return RedirectToAction(nameof(Edit_V2), new { id = existing.Id });
}
```

Talking Points:

"The antiforgery token proves the request originated from a valid form served by your site — not from an external attacker."

How ASP.NET Core Protects You

```
<div class="card">
    <div class="card-body">
        <form asp-action="EditItem" method="post" >
            @Html.AntiForgeryToken()
            <input asp-for="Id" type="hidden" />
            <input asp-for="OwnerId" type="hidden"/>

            <div class="mb-3">
                <label asp-for="Title" class="form-label">Title</label>
                <input asp-for="Title" class="form-control" />
            </div>

            <div class="mb-3 form-check">
                <input asp-for="IsDone" type="checkbox" class="form-check-input" />
                <label asp-for="IsDone" class="form-check-label"></label>
            </div>

            <div class="mb-3">
                <label asp-for="DueAt" class="form-label"></label>
                <input type="datetime-local" asp-for="DueAt"
                value="@Model.DueAt?.ToString("yyyy-MM-ddTHH:mm")" class="form-control" />
            </div>

            <button type="submit" class="btn btn-primary">Save</button>
            <a asp-action="Index" class="btn btn-secondary">Go Back</a>
        </form>
    </div>
</div>
```

How Antiforgery Works

Steps:

Server renders a form with a **unique hidden token**.

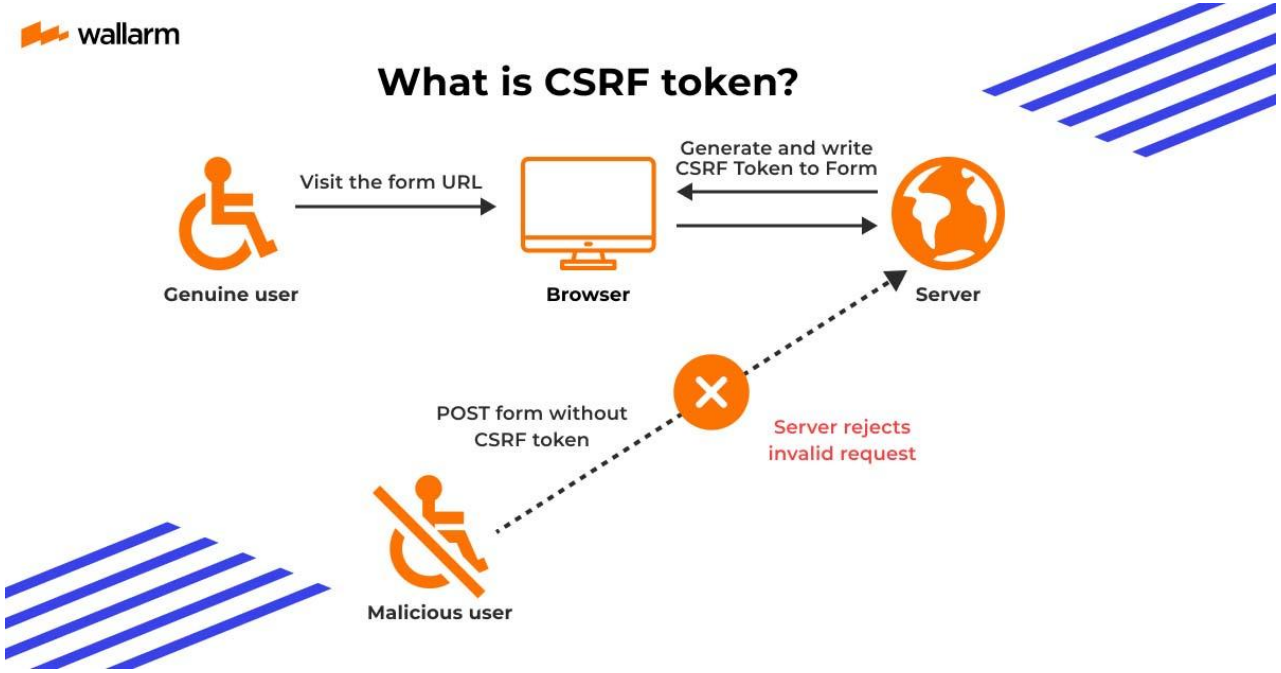
Browser submits the form (token included).

Server compares:

- Token from form body
- Token stored in cookie/session

If both match → request trusted.

If mismatch → **400 Bad Request (CSRF prevented)**.



https://www.wallarm.com/what/csrf-token

Partial Views in ASP.NET Core

A **partial view** is a Razor markup file (.cshtml)

It renders **a piece of HTML** inside another view's output.

Used in:

- ✓ **MVC apps:** markup files are called *views*
- ✓ **Razor Pages apps:** markup files are called *pages*

In both cases, partial views help you **reuse and organize UI sections**.

Example

You can create a partial to show success or error alerts used in many pages.

_AlertPartial.cshtml

```
@if (TempData["Success"] != null)
{
    <div class="alert alert-success">@TempData["Success"]</div>
}
@if (TempData["Error"] != null)
{
    <div class="alert alert-danger">@TempData["Error"]</div>
}
```

Usage inside Edit.cshtml

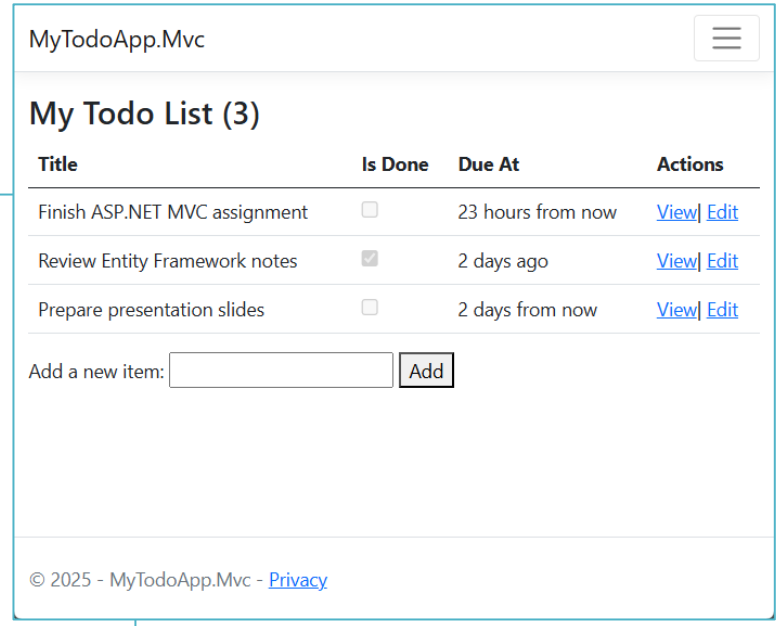
```
<h2>Edit Todo Item</h2>

<partial name="_AlertPartial" />
```

✅ Renders alerts wherever you include it — no code duplication.

Partial View

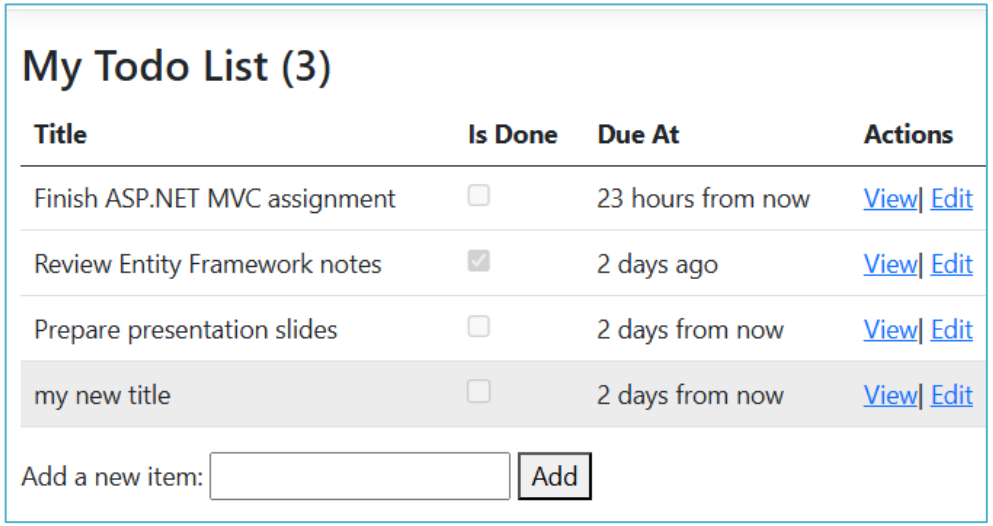
```
@using MyTodoApp.Mvc.Models.TodoViewModels
@model TodoItem

<form asp-action="AddItem" method="POST">
    @Html.AntiForgeryToken()
    <label asp-for="Title">Add a new item:</label>
    <input asp-for="Title">
    <span asp-validation-for="Title" class="text-
danger"></span>
    <button type="submit">Add</button>
</form>
```

```
<div class="panel-footer add-item-form">
    @await Html.PartialAsync("AddItemPartial", new TodoItem())
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```



Partial View

```
[HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> AddItem(TodoItem item)
    {
        if (!ModelState.IsValid)
            return RedirectToAction("Index");

        var currentUser = await CurrentUser();

        item.DueAt = DateTime.Now.AddDays(2);
        item.OwnerId = currentUser?.Id;
        item.IsDone = false;
        _todoItems.AddItem(item);

        return RedirectToAction("Index");
    }
```



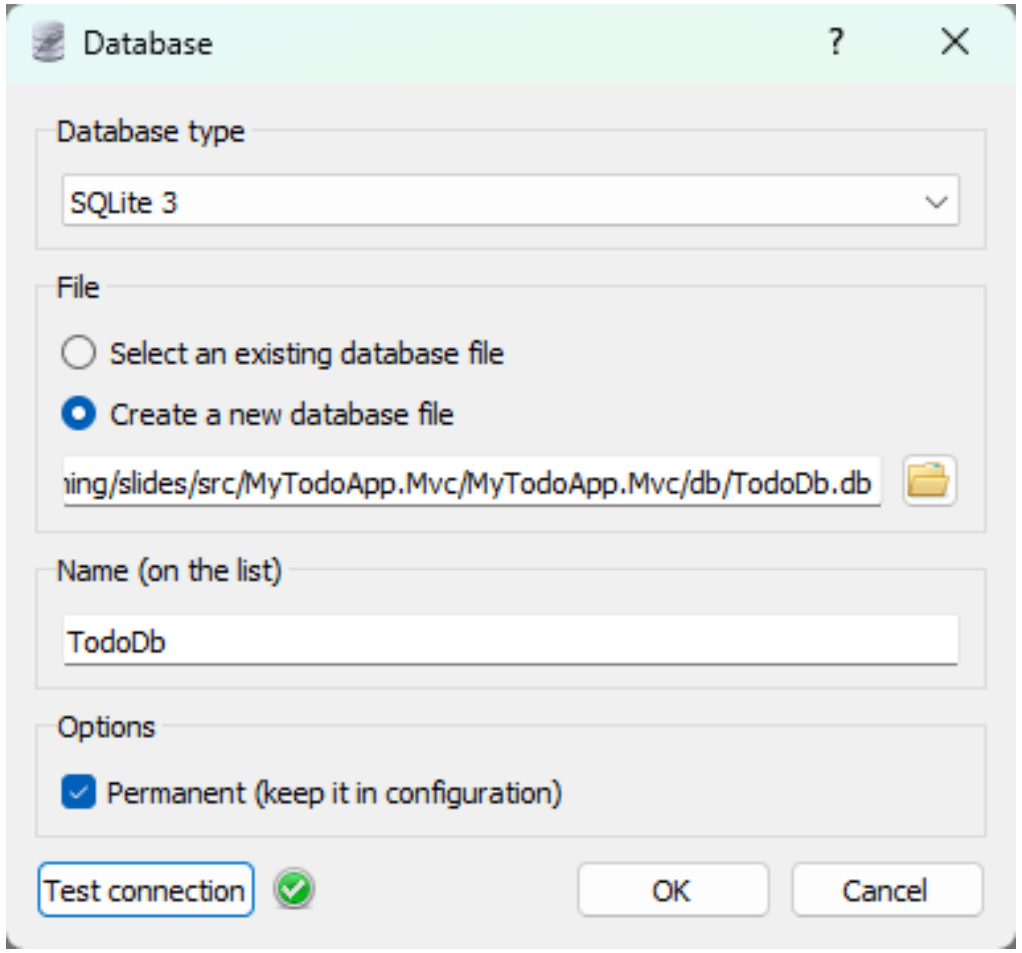
Add Delete Action

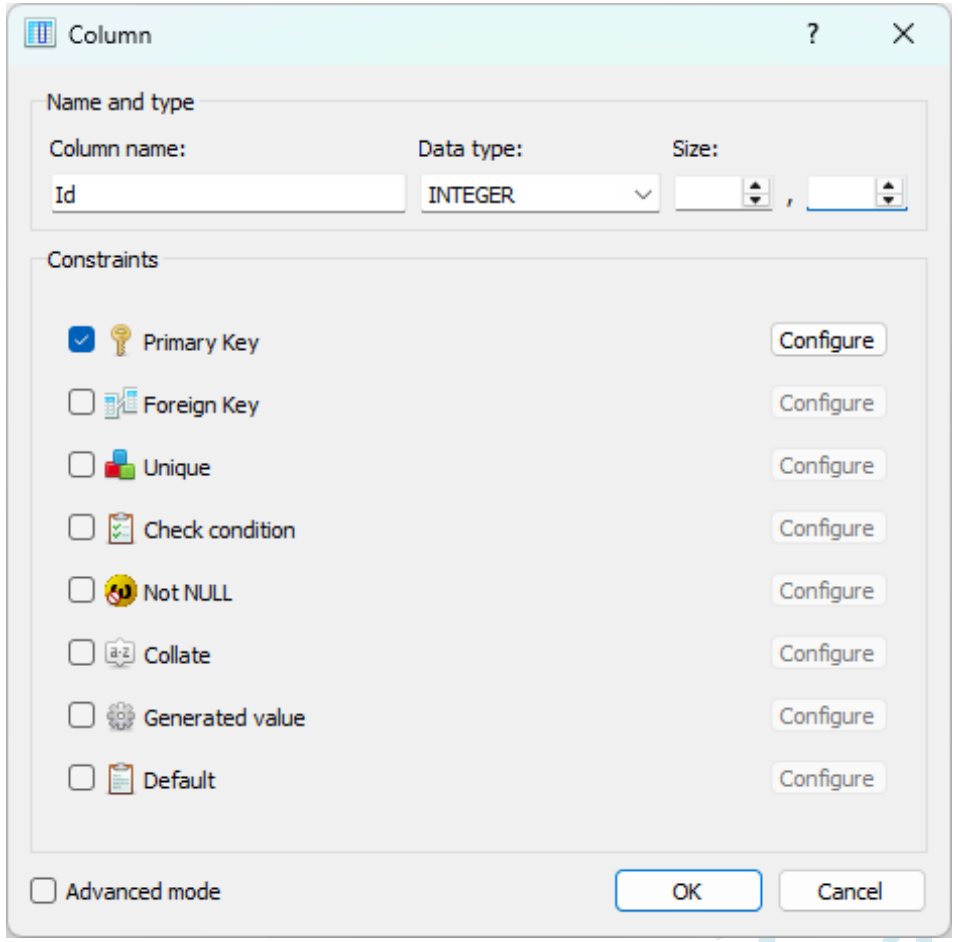
```
<form asp-controller="Todo" asp-action="DeleteItem"
method="post" asp-route-id="@item.Id">
    @Html.AntiForgeryToken()
    <button type="submit"
            onclick="return confirm('Are you sure you want
to delete this item?');"
            class="btn btn-link">
        Delete
    </button>
</form>
```

```
    [HttpPost]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteItem(Guid id)
        {
            if (id == Guid.Empty) return
RedirectToAction(nameof(Index));

…
```

Module 5

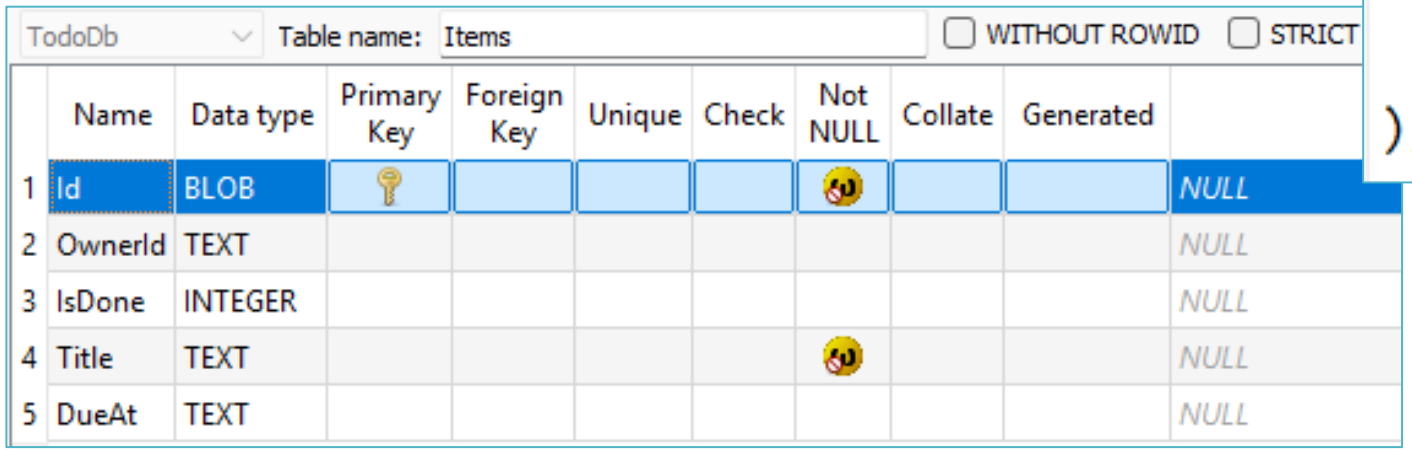
Create Sqlite Database





Create Sqlite Database

```
CREATE TABLE Items (
    Id       BLOB    PRIMARY KEY
                     NOT NULL,
    OwnerId  TEXT,
    IsDone   INTEGER,
    Title    TEXT    NOT NULL,
    DueAt    TEXT
);
```



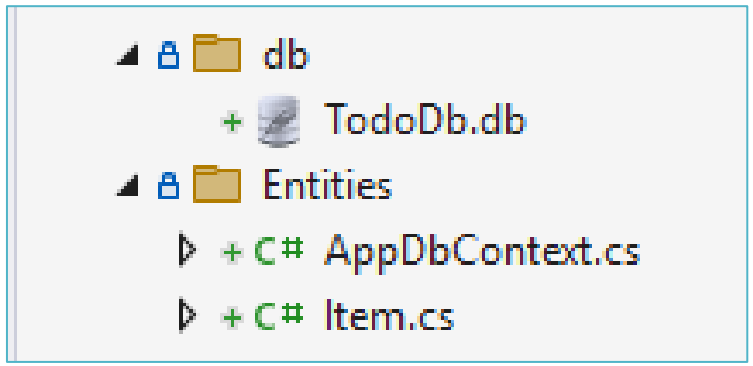
EF Core

```
pwsh in MyTodoApp.Mvc
Abdulkarim Kanaan  MyTodoApp.Mvc  master ≠ ?5 ~4  13ms  dotnet ef dbcontext scaffold
"Data Source=db/TodoDb.db" Microsoft.EntityFrameworkCore.Sqlite --output-dir Entities --context AppDbCont
ext --context-dir Entities
Build started ...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source
code. You can avoid scaffolding the connection strin
tion - see https://go.microsoft.com/fwlink/?linkid=2
gs, see https://go.microsoft.com/fwlink/?LinkId=7232
Abdulkarim Kanaan  MyTodoApp.Mvc  master ≠
```

```
Package Manager Console
Package source: All   Default project: MyTodoApp.Mvc2
PM> Scaffold-DbContext "Data Source=db/TodoAppDb.db"
Microsoft.EntityFrameworkCore.Sqlite -OutputDir Entities -Context
AppDbContext -ContextDir Entities
Build started...
Build succeeded.
To protect potentially sensitive information in your connection
string, you should move it out of source code. You can avoid
scaffolding the connection string by using the Name= syntax to read
it from configuration - see https://go.microsoft.com/fwlink/?
158 %
```

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <UserSecretsId>d216c5f1-37ff-4cbe-b1a9-274670900f31</UserSecretsId>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Humanizer.Core" Version="2.14.1" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.21" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="8.0.21">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="8.0.21" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.21">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
  </ItemGroup>

</Project>
```

```
Package Manager Console
Package source: All   Default project: MyTodoApp.Mvc2
PM> Scaffold-DbContext
"Server=.;Database=TodoDb;Trusted_Connection=True;TrustServ
erCertificate=True" Microsoft.EntityFrameworkCore.
SqlServer -OutputDir SqlEntities -ContextDir SqlEntities
Build started...
Build succeeded.
To protect potentially sensitive information in your
174 %
```



DbContext

```
public partial class AppDbContext : DbContext
{
    0 references
    public AppDbContext()
    {
    }

    0 references
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {
    }

    0 references
    public virtual DbSet<Item> Items { get; set; }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
#warning To protect potentially sensitive information in your connection string, you
        => optionsBuilder.UseSqlite("Data Source=db/TodoDb.db");

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Item>(entity =>
        {
            entity.Property(e => e.Id).ValueGeneratedNever();
        });

        OnModelCreatingPartial(modelBuilder);
    }

    1 reference
    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

```
namespace MyTodoApp.Mvc.Entities;

2 references
public partial class Item
{
    2 references
    public byte[] Id { get; set; } = null!;

    2 references
    public string? OwnerId { get; set; }

    2 references
    public int? IsDone { get; set; }

    2 references
    public string Title { get; set; } = null!;

    2 references
    public string? DueAt { get; set; }
}
```

```
{
  "ConnectionStrings": {
    "DefaultConnection": "DataSource=db/TodoDb.db"
},
```

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite(
        builder
        .Configuration
        .GetConnectionString("DefaultConnection")));
```

Interface Service

```
namespace MyTodoApp.Mvc.Services;

4 references
public interface ITodoService
{
    2 references
    Task<List<TodoItem>> GetItemsAsync();

    2 references
    Task<bool> AddItemAsync(TodoItem item);
}
```

Concrete Service

```
namespace MyTodoApp.Mvc.Services;

2 references
public class TodoService : ITodoService
{
    private readonly AppDbContext _context;

    0 references
    public TodoService(AppDbContext context)
    {
        _context = context;
    }

    2 references
    public async Task<bool> AddItemAsync(TodoItem item)
    {
        item.Id = Guid.NewGuid();

        _context.Items.Add(new Item
        {
            Id = item.Id.ToByteArray(),
            Title = item.Title,
            OwnerId = item.OwnerId,
            DueAt = item.DueAt.ToString(),
            IsDone = Convert.ToInt32(item.IsDone)
        });

        var saveResult = await _context.SaveChangesAsync();
        return saveResult == 1;
    }
```

```
    2 references
    public async Task<List<TodoItem>> GetItemsAsync()
    {
        return await _context.Items
            .Select(item => new TodoItem
            {
                Id = new Guid(item.Id),
                Title = item.Title,
                DueAt = Convert.ToDateTime(item.DueAt),
                IsDone = item.IsDone == 1,
                OwnerId = item.OwnerId
            })
            .ToListAsync();
    }
}
```

Dependency Injection

Dependency Injection is a design pattern that allows classes to depend on abstractions rather than concrete implementations.

It helps you build:

Loosely coupled systems

Easier-to-test components (via mock dependencies)

More flexible architectures that can swap dependencies without rewriting logic

```
3 references
public class TodoController : Controller
{
    private readonly ITodoService _todoService;
    private readonly ILogger<TodoController> _logger;

    0 references
    public TodoController(ITodoService todoService, ILogger<TodoController> logger)
    {
        _todoService = todoService;
        _logger = logger;
    }
```

Program.cs (Dependency Injection)

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped<ITodoService, TodoService>();
```

Built-in Logging Is Automatically Wired

That single line sets up a **default host**, which automatically configures:

- Dependency Injection (DI) container
- Configuration (appsettings.json, environment variables, etc.)
- Logging

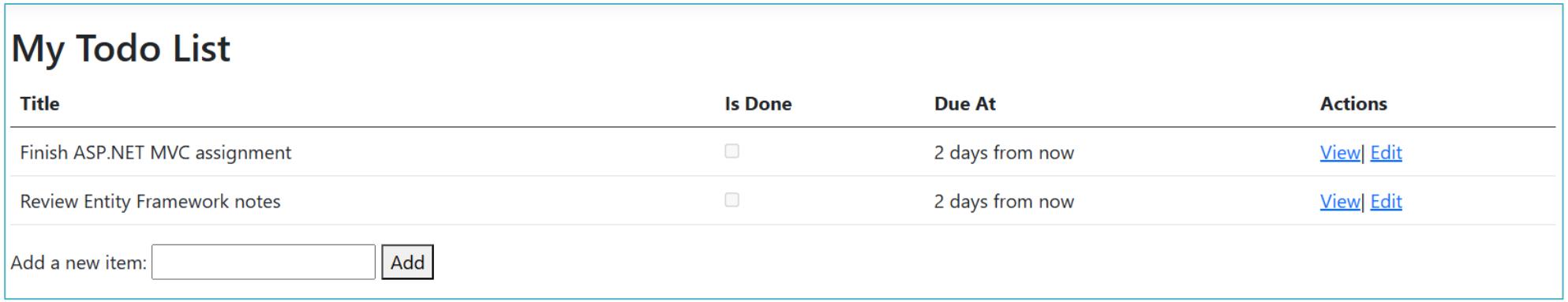
```
{
    "ConnectionStrings": {
      "DefaultConnection": "DataSource=db/TodoDb.db"
    },
    "Logging": {
      "LogLevel": {
        "Default": "Information",
        "Microsoft.AspNetCore": "Warning"
      }
    },
    "AllowedHosts": "*"
  }
```

Controller

```
[HttpGet]
0 references
public async Task<IActionResult> Index()
{
    return View(
        new TodoViewModel
        {
            TodoItems = await _todoService.GetItemsAsync()
        });
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> AddItem(TodoItem item)
{
    if (!ModelState.IsValid)
    {
        return RedirectToAction("Index");
    }

    item.DueAt = DateTime.Now.AddDays(2);

    bool successful = await _todoService.AddItemAsync(item);
    if (!successful)
    {
        return BadRequest("Couldn't add item.");
    }

    return RedirectToAction("Index");
}
```

Index View


My Todo List
Title
Is Done
Due At
Actions
Finish ASP.NET MVC assignment
2 days from now
View| Edit
Review Entity Framework notes
2 days from now
View| Edit
Add a new item:
Add

Completing To-Do Items

Goal: Make checkboxes *actually do something!*

- ✓ Users can tick items as “Done”
- ✓ Each item needs a way to send its ID to the server
- ✓ ✅ Action → Update database → Refresh list

💡 *We’ll use forms, hidden fields, and a pinch of JavaScript.*

Adding a Checkbox Form

```
<td>
    <form asp-action="MarkDone" method="POST">
        <input type="checkbox" class="done-checkbox" />
        <input type="hidden" name="id" value="@item.Id" />
    </form>
</td>
```

If we run this now...

- Checkboxes still don't work
- We could add "Submit" buttons — but that's bad UX 😅

👉 Better idea: **Submit automatically** when the checkbox is clicked!

Adding Smart Behavior with JavaScript

When checkbox is clicked:

- ✅ Disables it (can’t click twice)
- 📤 Submits the form automatically

```
$(document).ready(function () {
    $(".done-checkbox").on("click", function (e) {
        markCompleted(e.target);
    });
});

1 reference
function markCompleted(checkbox) {
    checkbox.disabled = true;

    checkbox.closest('form').submit();
}
```

Controller Logic

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> MarkDone(Guid id)
{
    if (id == Guid.Empty) return RedirectToAction(nameof(Index));

    bool successful = await _todoService.MarkDoneAsync(id);

    if (!successful) return BadRequest("Could not mark the item as completed!");

    return RedirectToAction("Index");
}
```

Adding the Service Method

```
1 reference
Task<List<TodoItem>> GetIncompleteItemsAsync();

0 references
Task<bool> MarkDoneAsync(Guid id);
```

```
1 reference
public async Task<List<TodoItem>> GetIncompleteItemsAsync()
{
    return await _context.Items
        .Where(item =>  item.IsDone == 0)
        .Select(item => new TodoItem
        {
            Id = new Guid(item.Id),
            Title = item.Title,
            DueAt = Convert.ToDateTime(item.DueAt),
            IsDone = item.IsDone == 1,
            OwnerId = item.OwnerId
        })
        .ToListAsync();
}
```

```
2 references
public async Task<bool> MarkDoneAsync(Guid id)
{
    var item = await _context.Items
        .Where(item => item.Id == id.ToByteArray())
        .SingleOrDefaultAsync();

    if (item == null) return false;

    item.IsDone = 1;

    var saveResult = await _context.SaveChangesAsync();
    return saveResult == 1;
}
```

Try It Out!

My Todo List

Completed?	Title	Is Done	Due At	Actions
☐	Finish ASP.NET MVC assignment	☐	2 days from now	View | Edit
☐	Review Entity Framework notes	☐	2 days from now	View | Edit

Add a new item: Add

Task Prepare View and Edit

Security & Identity Overview

Concept	Question It Answers	Purpose
Authentication	“Who is this user?”	Verifies identity
Authorization	“Can this user do X?”	Grants or denies access

Example:

- Authentication → Logging in
- Authorization → Access control after login

ASP.NET Core Identity

Built-in system for managing **users, passwords, and roles**.

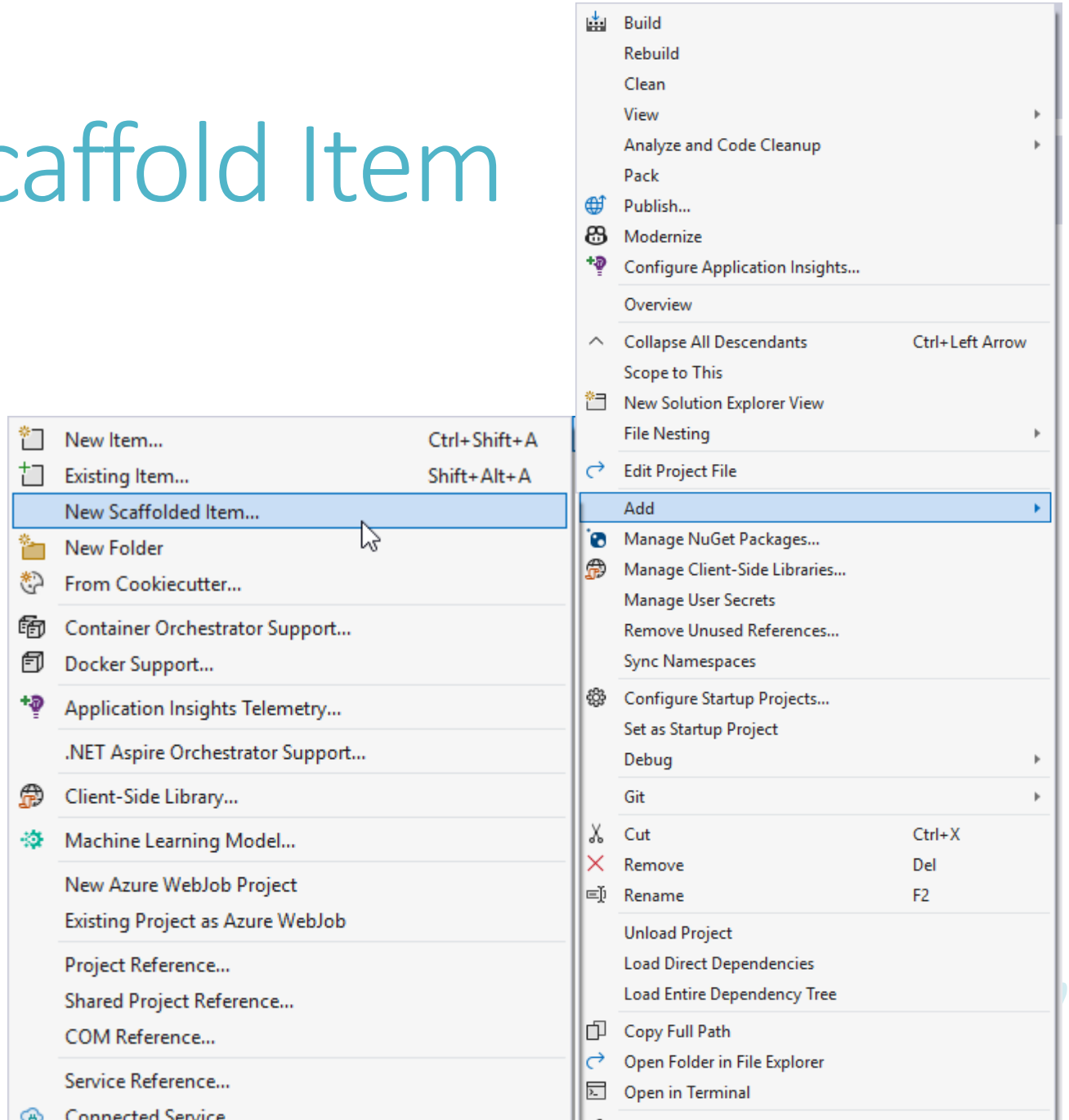
Installed automatically with the **MVC + Individual Auth** template.

Handles:

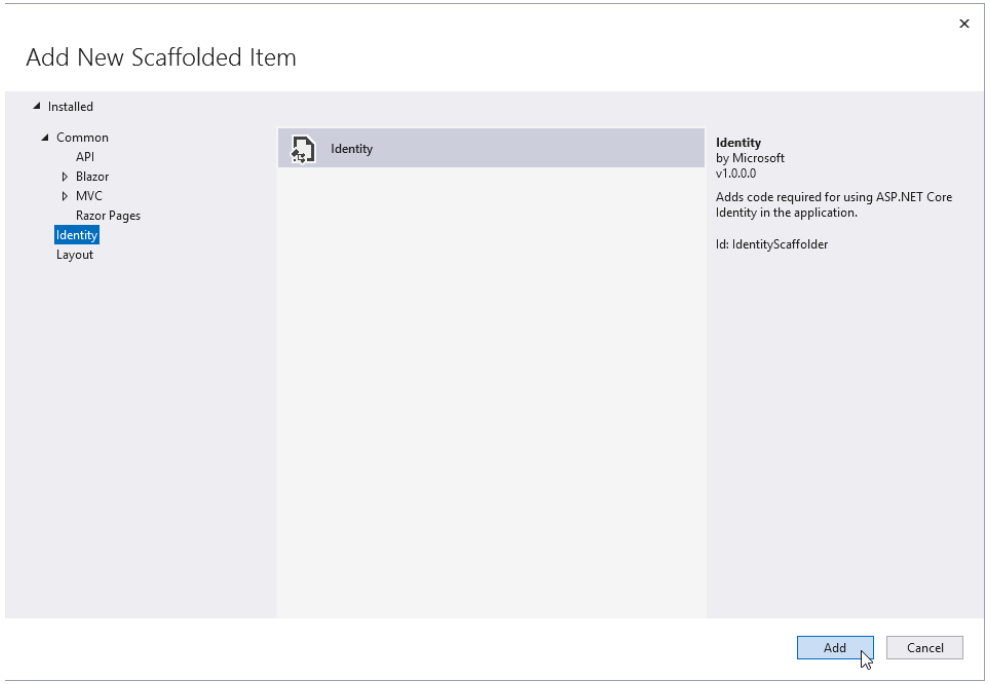
- Email/password login
- Multi-factor authentication
- Social logins (Google, Facebook, etc.)
- OAuth 2.0 / OpenID Connect

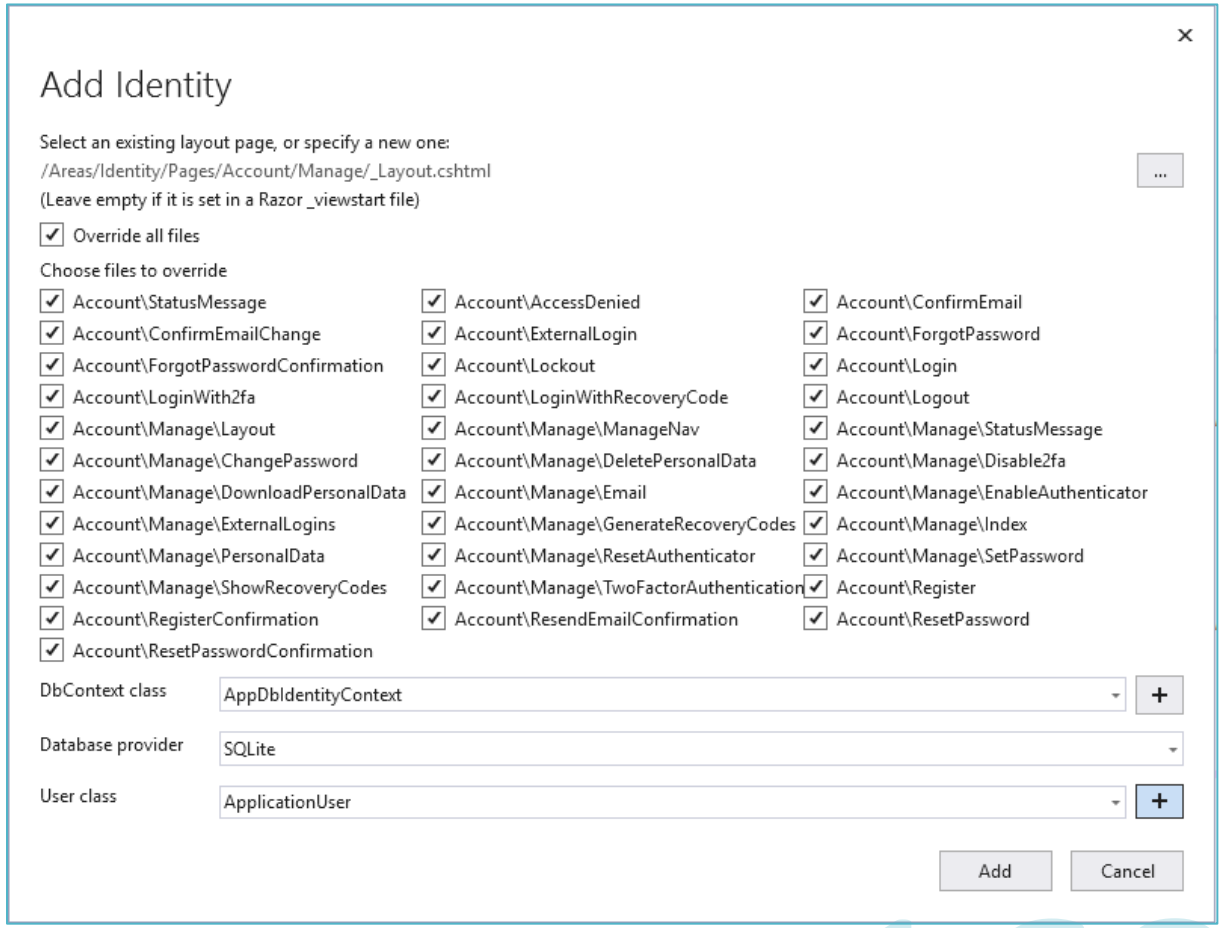
Stores hashed passwords and user data securely.

Adding Identity Scaffold Item



Add Identity





IdentityDbContext

```
namespace MyTodoApp.Mvc.Areas.Identity.Data;

6 references
public class AppDbIdentityContext : IdentityDbContext<ApplicationUser>
{
    0 references
    public AppDbIdentityContext(DbContextOptions<AppDbIdentityContext> options)
        : base(options)
    {
    }

    1 reference
    protected AppDbIdentityContext(DbContextOptions options)
        : base(options)
    {
    }
```

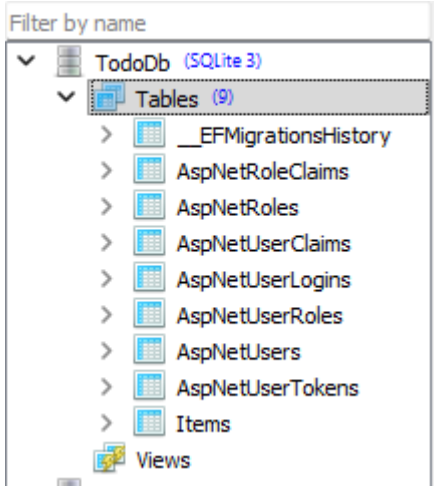
```
5 references
public partial class AppDbContext : AppDbIdentityContext
{

    0 references
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {
    }
```

Add Context and Update Database

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddDbContext<AppDbIdentityContext>(opt =>
    opt.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));
```

```
PM> add-migration initial -context AppDbIdentityContext
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
```

```
Package Manager Console
Package source: All    Default project: MyTodoApp.Mvc
  PM> Update-Database
  Build started...
  Build succeeded.
  More than one DbContext was found. Specify which one to use. Use the '-Context' paramete
  PM> Update-Database -context AppDbIdentityContext
  Build started...
  Build succeeded.
  Microsoft.EntityFrameworkCore.Database.Command[20101]
        Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
```



Program.cs

```
builder.Services
    .AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<AppDbIdentityContext>()
    .AddDefaultTokenProviders()
    .AddDefaultUI();
```

```
// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();
```

```
// Needed for Identity UI endpoints (e.g., /Identity/Account/Login)
app.MapRazorPages();

app.Run();
```

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </li>
        <partial name="_LoginPartial" />
    </ul>
</div>
```

InvalidOperationException: The entity type 'IdentityUserLogin<string>'

An unhandled exception occurred while processing the request.

InvalidOperationException: The entity type 'IdentityUserLogin<string>' requires a primary key to be defined. If you intended to use a keyless entity type, call 'HasNoKey' in 'OnModelCreating'. For more information on keyless entity types, see https://go.microsoft.com/fwlink/?linkid=2141943.

Microsoft.EntityFrameworkCore.Infrastructure.ModelValidator.ValidateNonNullPrimaryKeys(IModel model, IDiagnosticsLogger<Validation> logger)

```
2 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    OnModelCreatingPartial(modelBuilder);
}
```

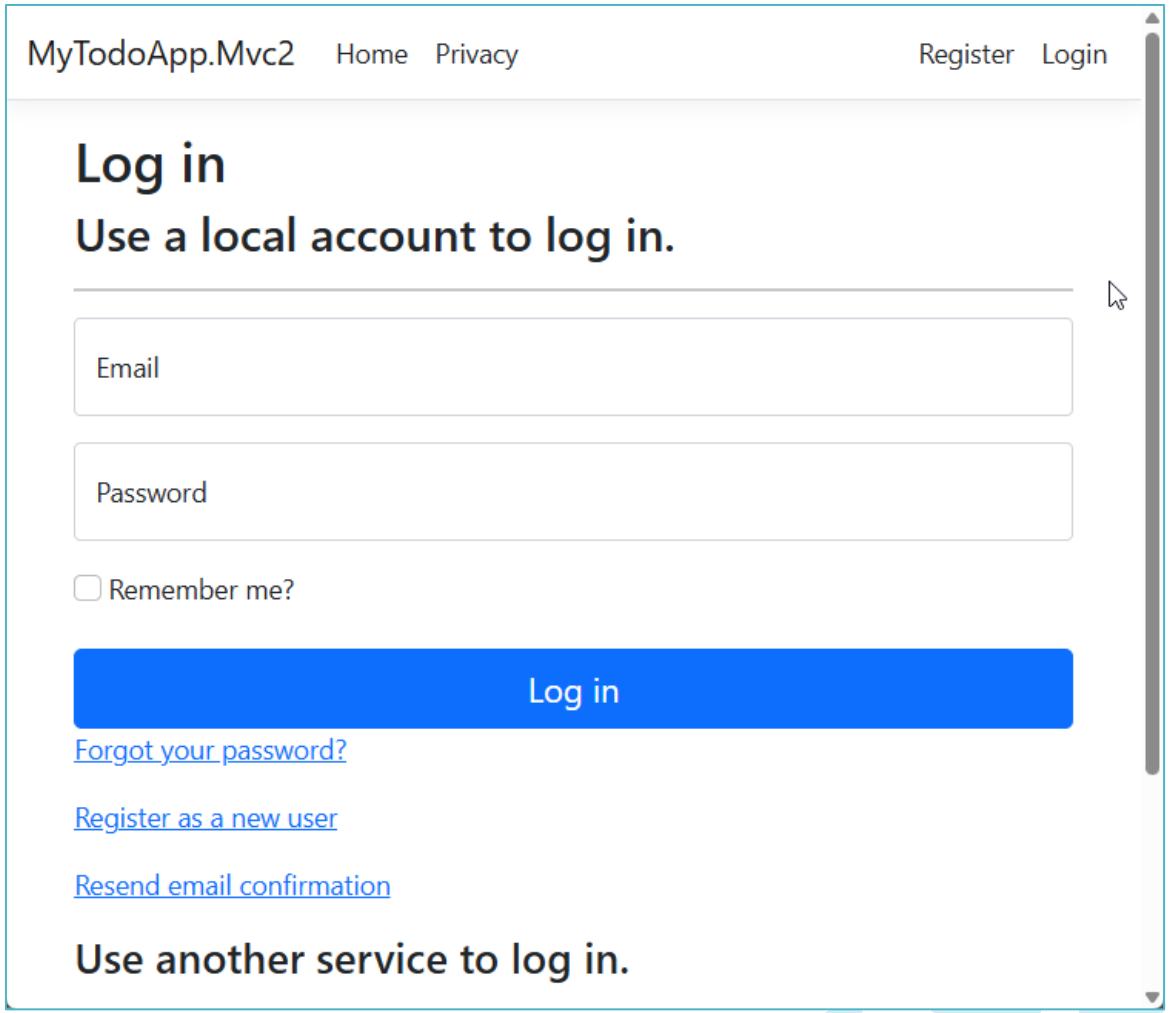
Can still Scaffold

```
PS> dotnet ef dbcontext scaffold "Data Source=db/TodoDb.db" Microsoft.EntityFrameworkCore.Sqlite   --context AppDbContex
t   --context-dir Entities   --output-dir Entities   -f   -t Items
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source code. You can a
void scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsof
t.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see https://go.microsoft.com/fwlink/?Link
Id=723263.
The column 'OwnerId' on table 'Items' should map to a property of type 'Guid', but its values are in an incompatible for
mat. Using a different type.
The column 'DueAt' on table 'Items' should map to a property of type 'DateTime', but its values are in an incompatible f
ormat. Using a different type.
```

Requiring Authentication

```
[Authorize()]
3 references
public class TodoController : Controller
{
    private readonly ITodoService _todoService;
    private readonly ILogger<TodoController> _logger;
    private readonly UserManager<ApplicationUser> _userManager;

    0 references
    public TodoController(ITodoService todoService, ILogger<TodoController> logger, UserManager<ApplicationUser> userManager)
    {
        _todoService = todoService;
        _logger = logger;
        _userManager = userManager;
    }

    2 references
    private async Task<ApplicationUser?> CurrentUser() => await _userManager.GetUserAsync(User);
```

```
<div class="navbar-collapse collapse d-sm-in
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" as
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" as
        </li>
        <partial name="_AdminActionsPartial"
    </ul>
</div>
<div class="justify-content-end">
    <partial name="_LoginPartial" />
</div>
```



Updating AddItem & MarkDone

```
2 references
public async Task<List<TodoItem>> GetIncompleteItemsAsync(ApplicationUser user)
{
    return await _context.Items
        .Where(item =>  item.IsDone == 0 && item.OwnerId == user.Id)
        .Select(item => new TodoItem
        {
            Id = new Guid(item.Id),
            Title = item.Title,
            DueAt = Convert.ToDateTime(item.DueAt),
            IsDone = item.IsDone == 1,
            OwnerId = item.OwnerId
        })
        .ToListAsync();
}
```

Updating AddItem & MarkDone

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> AddItem(TodoItem item)
{
    if (!ModelState.IsValid)
    {
        return RedirectToAction("Index");
    }

    var currentUser = await CurrentUser();

    item.DueAt = DateTime.Now.AddDays(2);
    item.OwnerId = currentUser?.Id;
    item.IsDone = false;

    (bool successful, Guid id) = await _todoService.AddItemAsync(item);
    if (!successful)
    {
        return BadRequest("Couldn't add item.");
    }

    return RedirectToAction("Index");
}
```

```
public async Task<bool> MarkDoneAsync(Guid id, ApplicationUser user)
{
    var item = await _context.Items
        .Where(item => item.Id == id.ToByteArray() && item.OwnerId == user.Id)
        .SingleOrDefaultAsync();

    if (item == null) return false;

    item.IsDone = 1;

    var saveResult = await _context.SaveChangesAsync();
    return saveResult == 1;
}
```

IdentitySeeder

```
public static async Task SeedAsync(IServiceProvider services)
{
    using var scope = services.CreateScope();

    var roleMgr = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();
    await EnsureRolesAsync(roleMgr);

    var userMgr = scope.ServiceProvider.GetRequiredService<UserManager<ApplicationUser>>();
    await EnsureTestAdminAsync(userMgr);
}

private static async Task EnsureRolesAsync(RoleManager<IdentityRole> roleManager)
{
    var alreadyExists = await roleManager.RoleExistsAsync(Constants.AdministratorRole);

    if (alreadyExists) return;

    await roleManager.CreateAsync(new IdentityRole(Constants.AdministratorRole));
}

private static async Task EnsureTestAdminAsync(UserManager<ApplicationUser> userManager)
{
    var testAdmin = await userManager.Users
        .Where(x => x.UserName == "admin@todo.local")
        .SingleOrDefaultAsync();

    if (testAdmin != null) return;

    testAdmin = new ApplicationUser { UserName = "admin@todo.local", Email = "admin@todo.local" };
    IdentityResult identity = await userManager.CreateAsync(testAdmin, "NotSecure123!!");

    await userManager.AddToRoleAsync(testAdmin, Constants.AdministratorRole);
}
```

Using IdentitySeeder

```
// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();

var app = builder.Build();

await IdentitySeeder.SeedAsync(app.Services);
```

Manage Users' Model

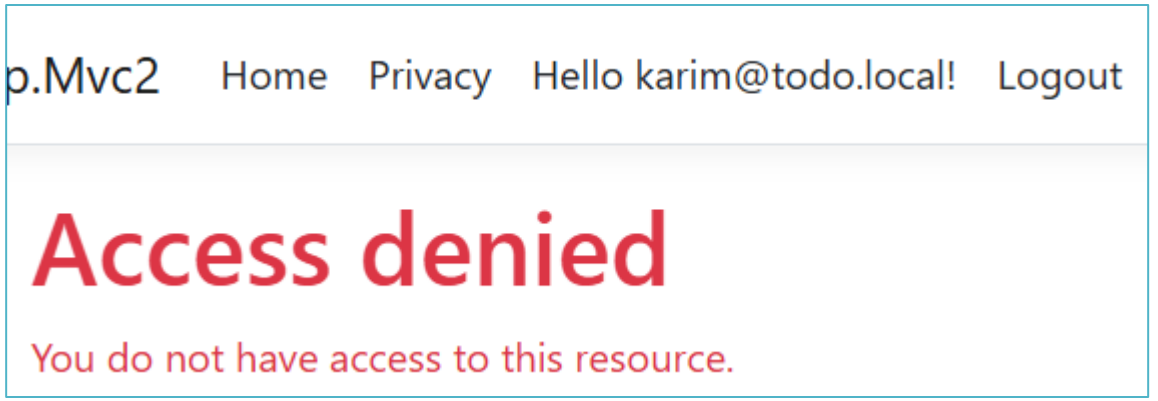
```
using MyTodoApp.Mvc.Areas.Identity.Data;

namespace MyTodoApp.Mvc.Models.ManageUsersViewModels;

public class ManageUsersViewModel
{
    public ApplicationUser[] Administrators { get; set; }
    public ApplicationUser[] Everyone { get; set; }
}
```

Manage Users

```
[Authorize(Roles = "Admin")]
public class ManageUsersController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;

    public ManageUsersController(UserManager<ApplicationUser> userManager)
    {
        _userManager = userManager;
    }

    public async Task<IActionResult> Index()
    {
        var admins = (await _userManager
            .GetUsersInRoleAsync(Constants.AdministratorRole))
            .ToArray();

        var everyone = await _userManager.Users.ToArrayAsync();

        var model = new ManageUsersViewModel
        {
            Administrators = admins,
            Everyone = everyone
        };

        return View(model);
    }
}
```

Manage User View

```
@using MyTodoApp.Mvc.Models.ManageUsersViewModels
@model ManageUsersViewModel
@{
    ViewData["Title"] = "Manage users";
}
<h2>@ViewData["Title"]</h2>
<h3>Administrators</h3>
<table class="table">
    <thead>
        <tr>
            <td>Id</td>
            <td>Email</td>
        </tr>
    </thead>
    @foreach (var user in Model.Administrators)
    {
        <tr>
            <td>@user.Id</td>
            <td>@user.Email</td>
        </tr>
    }
</table>
<h3>Everyone</h3>
<table class="table">
    <thead>
        <tr>
            <td>Id</td>
            <td>Email</td>
        </tr>
    </thead>
    @foreach (var user in Model.Everyone)
    {
        <tr>
            <td>@user.Id</td>
            <td>@user.Email</td>
        </tr>
    }
</table>
```



Partial View & Authorization in Views

Sometimes we need **authorization checks inside a view**, not just in controllers.

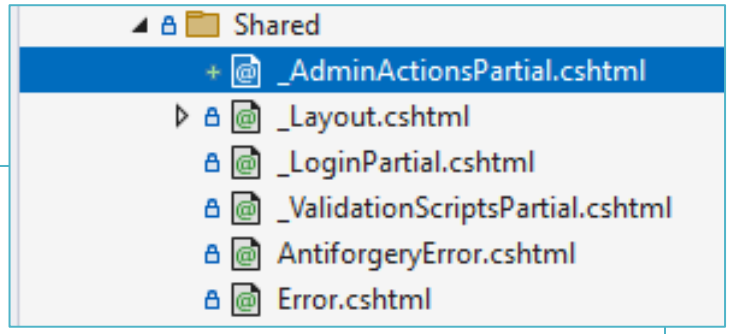
Example: show **“Manage Users”** link only for administrators.

✅ Solution: create a **partial view** to keep the layout clean.

What is a Partial View?

- A partial view is a reusable Razor file (e.g. _AdminActionsPartial.cshtml)that renders part of a page (like a menu or footer).
- Usually placed in Views/Shared/.
- Naming convention: starts with an underscore (_).

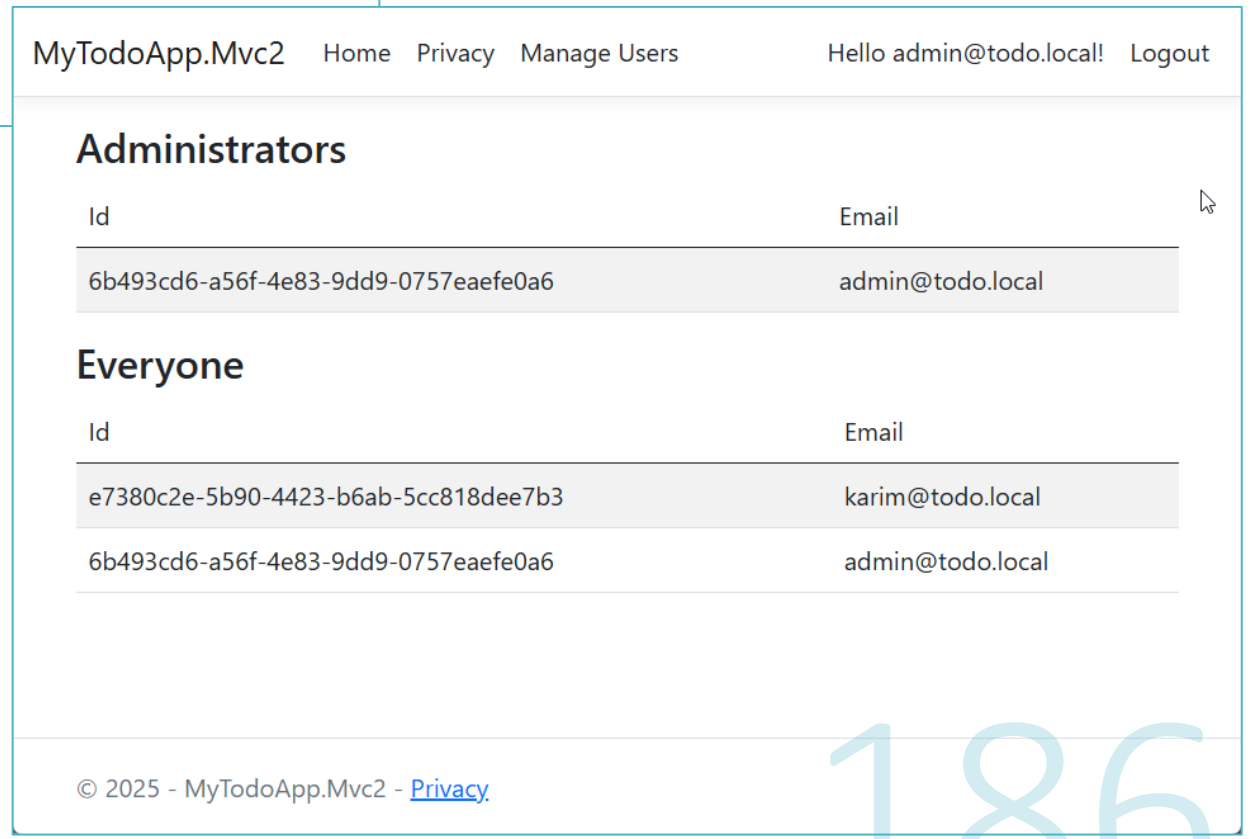
Creating the Partial View




```
@using Microsoft.AspNetCore.Identity
@using MyTodoApp.Mvc.Areas.Identity.Data

@inject SignInManager<ApplicationUser> signInManager
@inject UserManager<ApplicationUser> userManager

@if (signInManager.IsSignedIn(User))
{
    var currentUser = await userManager.GetUserAsync(User);
    var isAdmin = currentUser != null &&
                  await userManager.IsInRoleAsync(currentUser, Constants.AdministratorRole);
    if (isAdmin)
    {
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link text-dark" asp-controller="ManageUsers" asp-
action="Index">Manage Users</a>
            </li>
        </ul>
    }
}
```

Integrate the Partial in Layout

```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </li>
        <partial name="_LoginPartial" />
        @await Html.PartialAsync("_AdminActionsPartial")
    </ul>
</div>
```



Task: _TodoActionsPartial.cshtml

_TodoActionsPartial.cshtml

```
@using Microsoft.AspNetCore.Identity
@using MyTodoApp.Mvc.Areas.Identity.Data
@inject SignInManager<ApplicationUser> SignInManager

@if (SignInManager.IsSignedIn(User))
{
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link text-dark"
               asp-controller="Todo"
               asp-action="Index">Manage Todos</a>
        </li>
    </ul>
}
```

Task: _AlertPartial.cshtml

```
@if (TempData["Success"] != null)
{
    <div class="alert alert-success">@TempData["Success"]</div>
}
@if (TempData["Error"] != null)
{
    <div class="alert alert-danger">@TempData["Error"]</div>
}
```

Task: _UserProfilePartial.cshtml

```
@using Microsoft.AspNetCore.Identity
@using MyTodoApp.Mvc.Areas.Identity.Data
@inject UserManager<ApplicationUser> UserManager
@inject SignInManager<ApplicationUser> SignInManager

@if (SignInManager.IsSignedIn(User))
{
    var currentUser = await UserManager.GetUserAsync(User);


    <div class="dropdown">
        <a class="d-flex align-items-center text-decoration-none dropdown-toggle"
           href="#" data-bs-toggle="dropdown">
            <i data-lucide="user-circle" class="text-light" style="width:40px;height:40px;"></i>
            <strong>@currentUser?.UserName</strong>
        </a>
        <ul class="dropdown-menu dropdown-menu-end">
            <li><a class="dropdown-item" asp-area="Identity" asp-
page="/Account/Manage/Index">Profile</a></li>
            <li><hr class="dropdown-divider" /></li>
            <li><a class="dropdown-item" asp-area="Identity" asp-page="/Account/Logout">Logout</a></li>
        </ul>
    </div>
}
```

When to Use **Partial Views**

Use **partial views** when you want to:

- Render a **specific part of a view model**
- Reuse a **UI section** without extra logic
- Keep your main view **clean and modular**

✅ Ideal for:

- Forms, alerts, reusable form sections
- UI-only components with no data access

Use a **View Component** when:

- You need to include **business logic** or **database access**
- The section is **functionally independent** from the main page
- You want to fetch or prepare data before rendering

✅ Ideal for:

- Dashboards, summaries, widgets (e.g., Todo Stats, Recent Tasks)

Understanding View Components

- Reusable UI blocks
- Independent from main view
- Combine **business logic + rendering**
- Invoked **inside Razor views**, not via HTTP
- Think: “mini MVC controller”

Why Use View Components

Separate complex sections

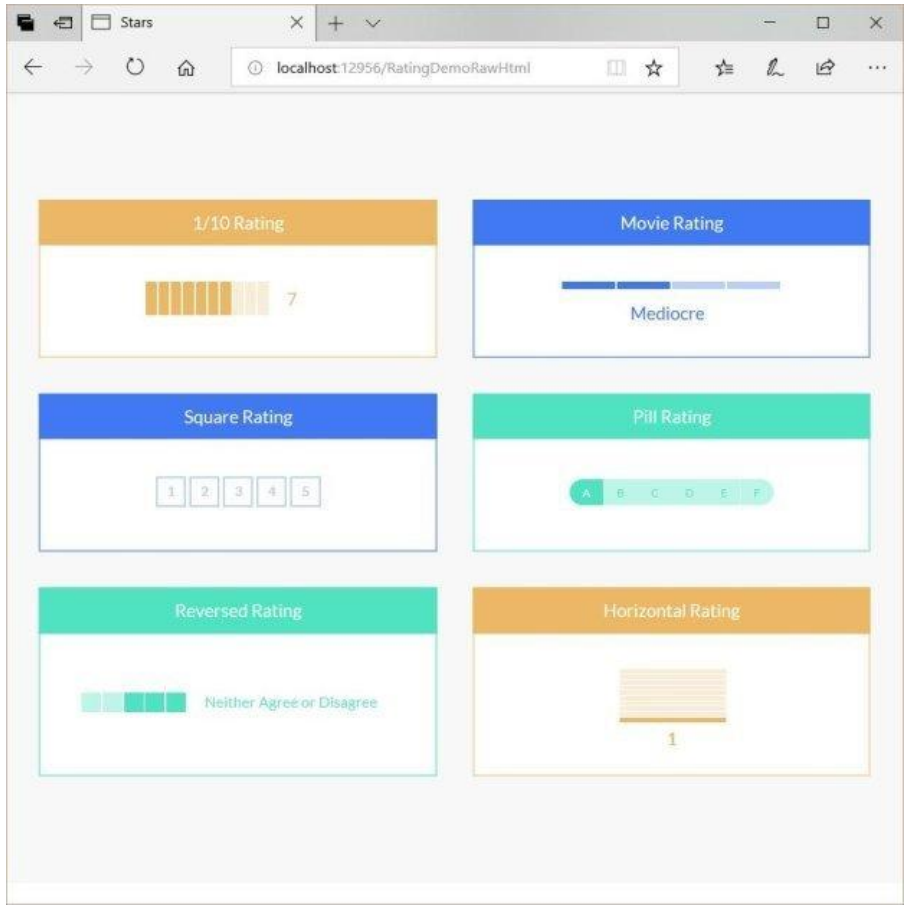
Keeps main Razor clean

Encapsulates:

- Logic
- Data access
- Rendering

Easier testing

Supports **Dependency Injection**



Kellner, P. (2018, August 27). Why You Should Use View Components, not Partial Views, in ASP.NET Core. Retrieved October 22, 2025, from Telerik Blogs website: https://www.telerik.com/blogs/why-you-should-use-view-components-not-partial-views-aspnet-core

Structure Overview

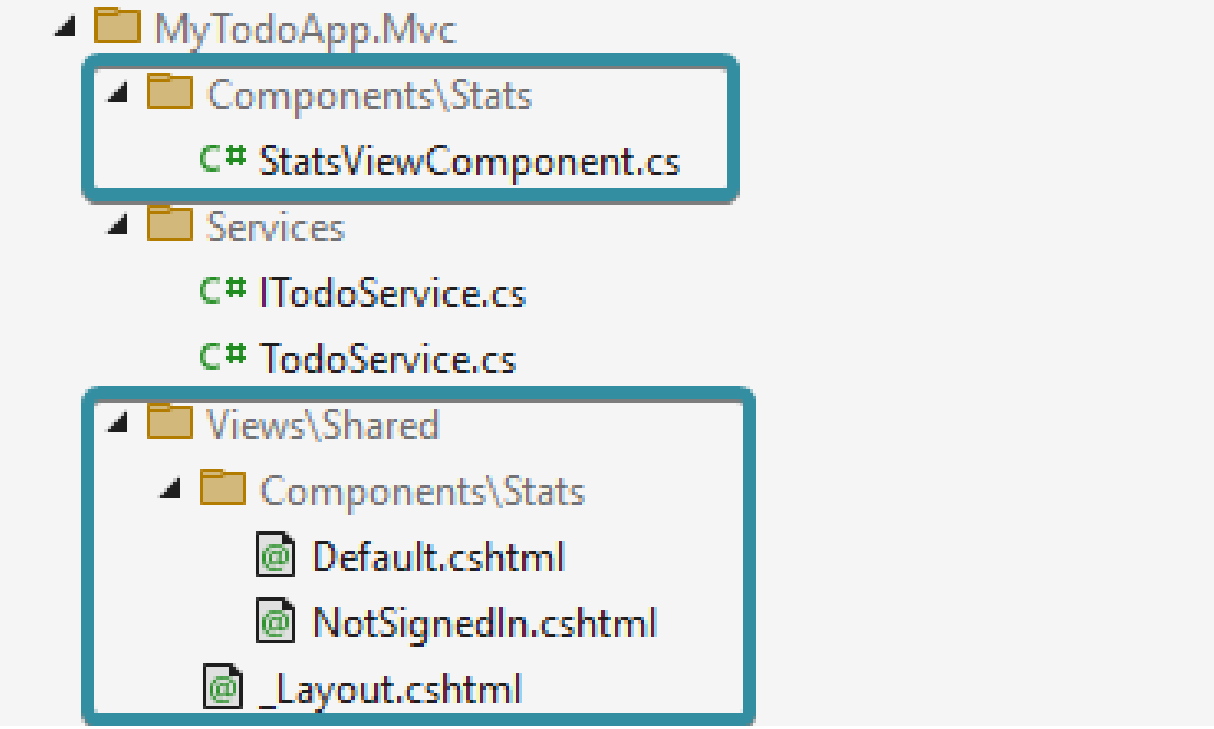
View Component Folder Layout:

/Components/StatsViewComponent.cs

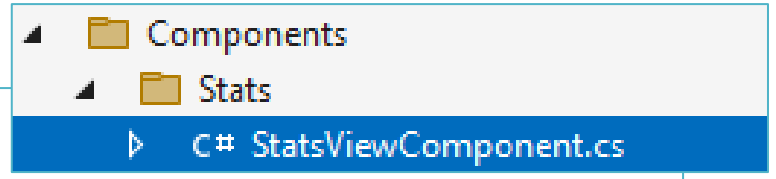
/Views/Shared/Components/Stats/Default.cshtml

Core method:

public async Task<IViewComponentResult> InvokeAsync()



StatsViewComponent

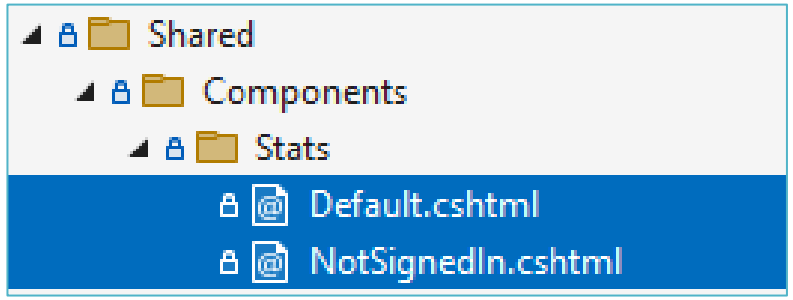


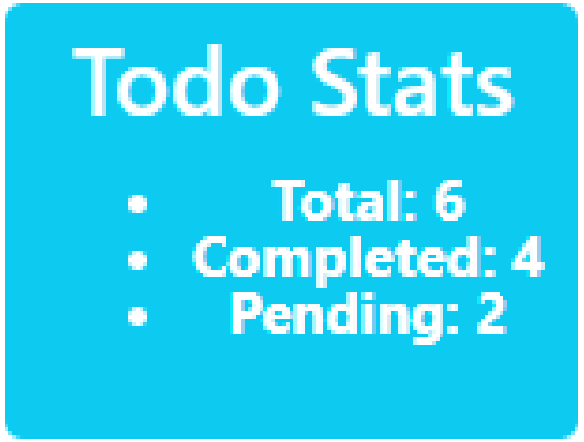

```
public class StatsViewComponent : ViewComponent
{
    private readonly ITodoService _todo;
    private readonly UserManager<ApplicationUser> _userManager;

    private async Task<ApplicationUser?> GetCurrentUserAsync()
    {
        return await _userManager.GetUserAsync(HttpContext.User);
    }

    public StatsViewComponent(ITodoService todo, UserManager<ApplicationUser> userManager)
    {
        _todo = todo;
        _userManager = userManager;
    }

     public async Task<IViewComponentResult> InvokeAsync()
     {
         if (!User.Identity.IsAuthenticated) return View("NotSignedIn");

         var currentUser = await GetCurrentUserAsync();
         var model = _todo.GetTodoStats(currentUser.Id);

         return View(model);
     }
 }
```

```
public class StatsViewModel
{
    public int Total { get; set; }
    public int Completed { get; set; }
    public int Pending { get; set; }

}
```

View Template Example

```
<div class="stats badge bg-info">
    <h5>Todo Stats</h5>
    <ul>
        <li>Total: @Model.Total</li>
        <li>Completed: @Model.Completed</li>
        <li>Pending: @Model.Pending</li>
    </ul>
</div>
```



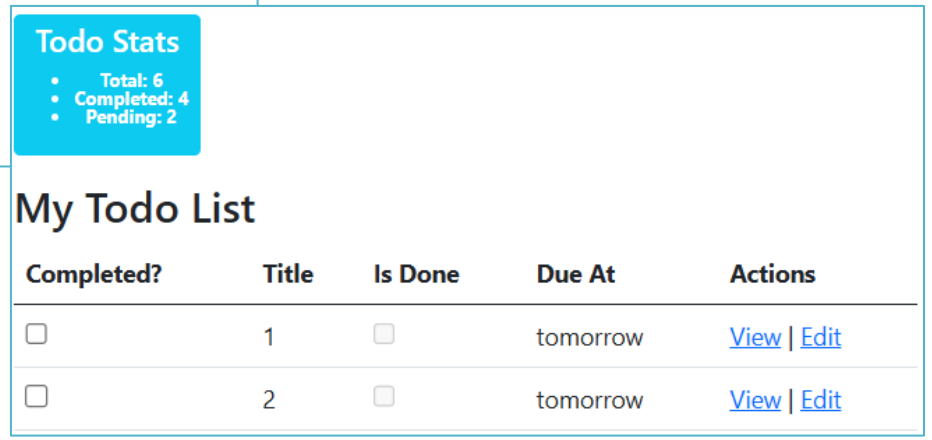


View Template Example

```
@using MyTodoApp.Mvc2
@using MyTodoApp.Mvc2.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, MyTodoApp.Mvc2
```

```
<div class="container">
    <main role="main" class="pb-3">
        <div class="mb-3">
            @await Component.InvokeAsync("Stats")
        </div>
        @RenderBody()
    </main>
</div>
```

```
<div class="mb-3">
    <vc:stats></vc:stats>
</div>
```



Accessing HttpContext Inside a View Component

View Components can access the **current request** and **HttpContext**, just like controllers.

This allows you to:

- Check user authentication
- Retrieve route or session data
- Apply conditional rendering logic

⚙️ Default Template Behavior

When you call:

return View(model);

If no specific Razor view name is provided,
the component automatically looks for a template named:

👉 **Default.cshtml**

View Component File Locations

View Component partial views are stored in specific folders.
They work like normal partials but are kept separate for clarity.

Valid Locations:

Project Type	Path
MVC Views	Views/Shared/Components/<ComponentName>/<TemplateName>.cshtml
Razor Pages	Pages/Shared/Components/<ComponentName>/<TemplateName>.cshtml

Module 7

What is a Web API?

A **Web API (Application Programming Interface)** is a **set of HTTP endpoints** that allow systems (browsers, mobile apps, or other servers) to **communicate with your application** through structured data — usually JSON.

APIs power modern apps:

- 🖥️ Web front-ends (e.g., React, MVC)
- 📱 Mobile apps
- 🌐 IoT devices and embedded systems
- 🤝 System integration between organizations

Benefits:

- Decoupled architecture
- Reusability across multiple clients
- Easier testing and scaling

REST at a Glance

REST = Representational State Transfer

Principle	Meaning
Resources	Everything is a thing (e.g., Todo, User, Order).
Representations	Data is represented (usually JSON).
Stateless	Each request is independent — no hidden session.
Uniform Interface	Uses standard HTTP methods and codes.

Example of RESTful Resource

Action	HTTP Method	URL	Result
Get all todos	GET	/api/todos	200 OK + list
Get one todo	GET	/api/todos/3	200 OK + item
Create new todo	POST	/api/todos	201 Created
Update todo	PUT	/api/todos/3	204 No Content
Delete todo	DELETE	/api/todos/3	204 No Content

Representations (JSON)

```
{
  "id": 1,
  "title": "Finish ASP.NET slides",
  "isDone": false,
  "dueAt": "2025-10-22T15:30:00Z"
}
```

Two Ways to Build APIs in ASP.NET Core

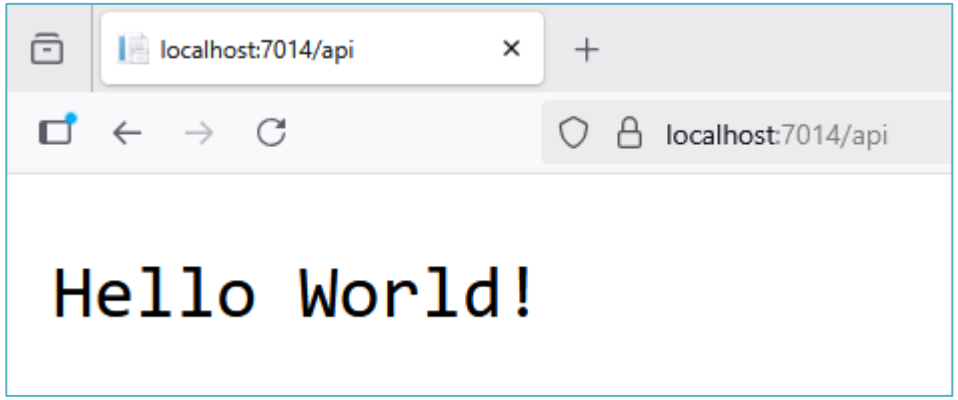
Style	Example
Minimal API	Define endpoints directly in Program.cs
Controller-based API	Use [ApiController] classes like TodosController

Minimal API Endpoint

Part	Meaning	Example
Route	URL path that clients call	/api
HTTP Verb	Type of request	MapGet (for GET requests)
Handler	What to do when called	() => "Hello World!"

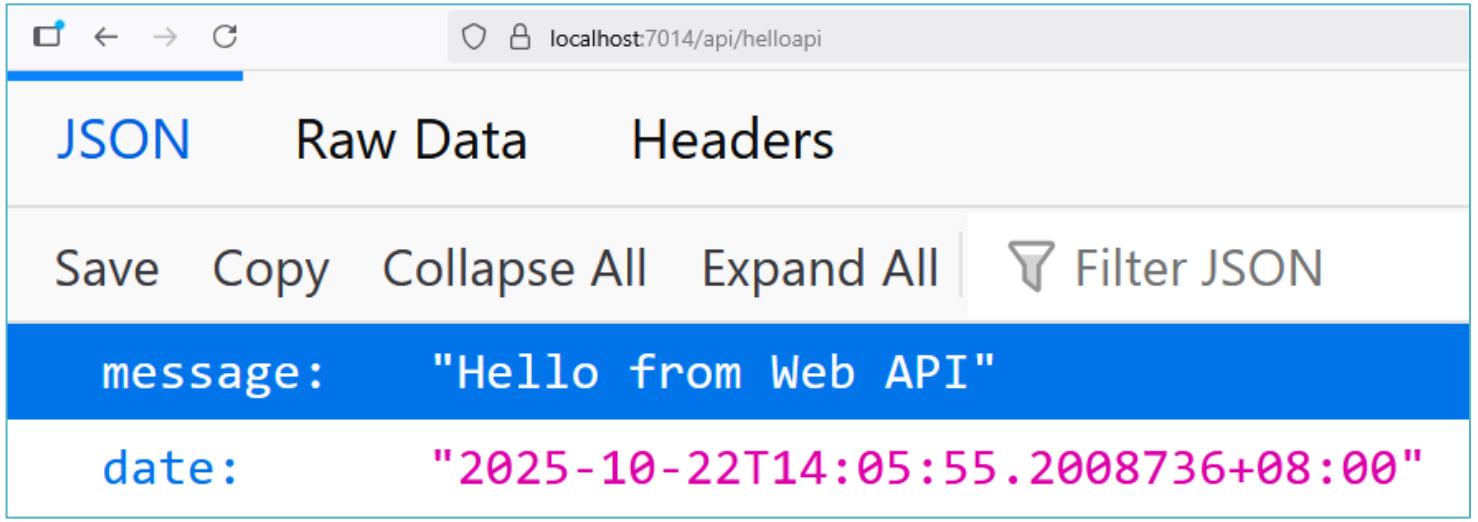
```
app.MapGet("/api", () => "Hello World!");

app.Run();
```

Step	Action
1	The ASP.NET Core pipeline receives a GET request for `/api`.
2	The route `/api` matches `app.MapGet("/api", ...)`.
3	The lambda handler runs and returns `"Hello World!"`.
4	ASP.NET Core serializes the response (as plain text or JSON).
5	The browser displays the result.



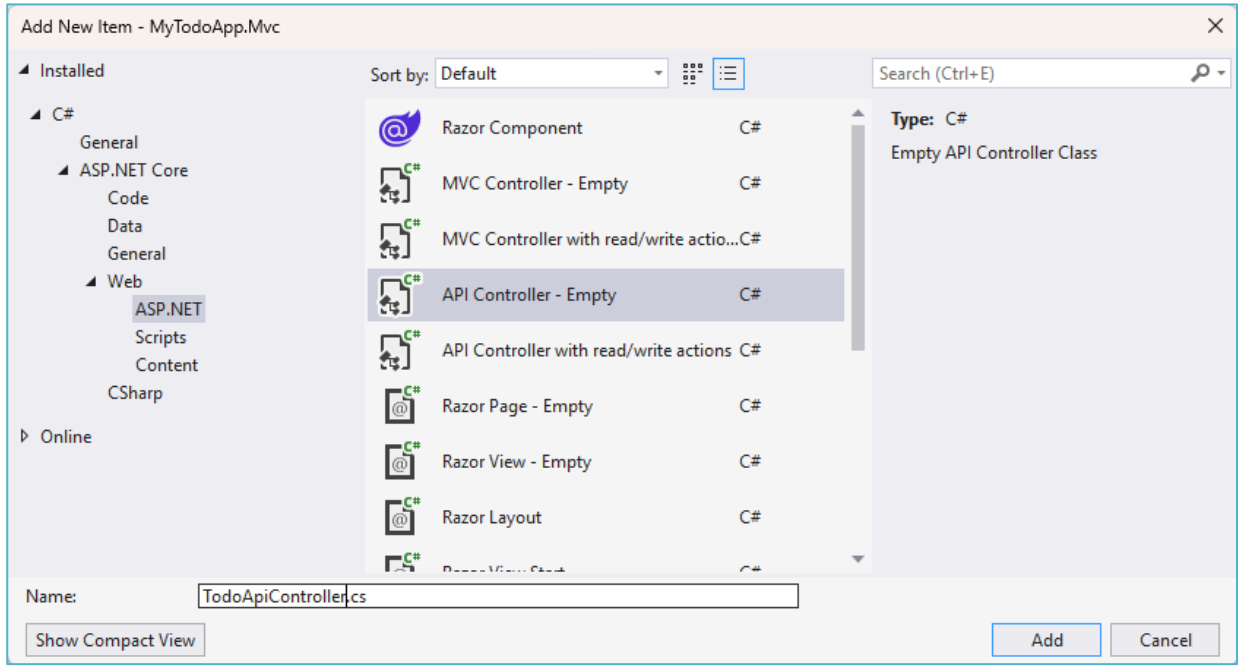
Change the Response Type

```
app.MapGet("/api", () => "Hello World!");

app.MapGet("/api/helloapi", () => new { message = "Hello from Web API", date = DateTime.Now });

app.Run();
```



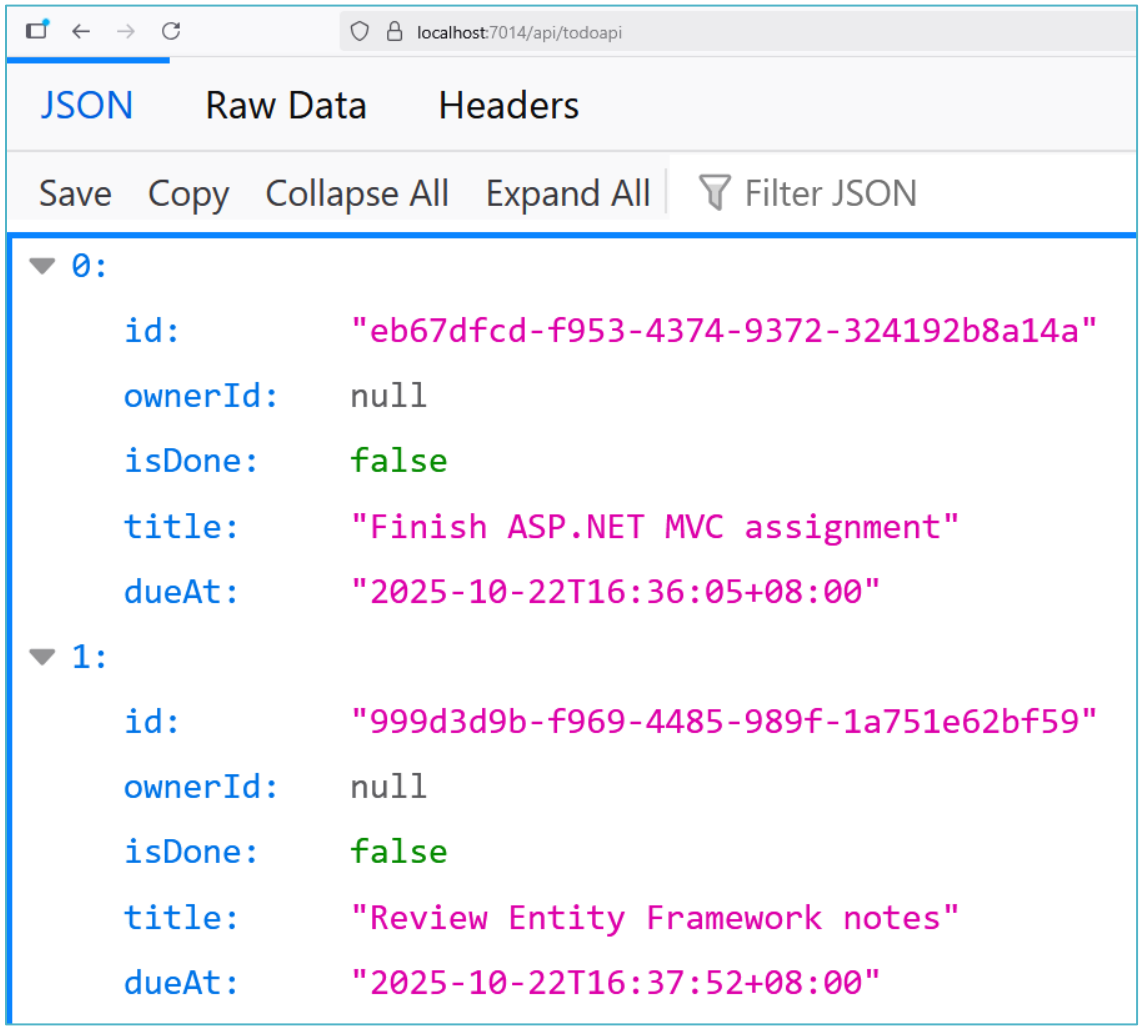
Controller-Based API

```
namespace MyTodoApp.Mvc.Controllers;

[Route("api/[controller]")]
[ApiController]
0 references
public class TodoApiController : ControllerBase
{
}
```

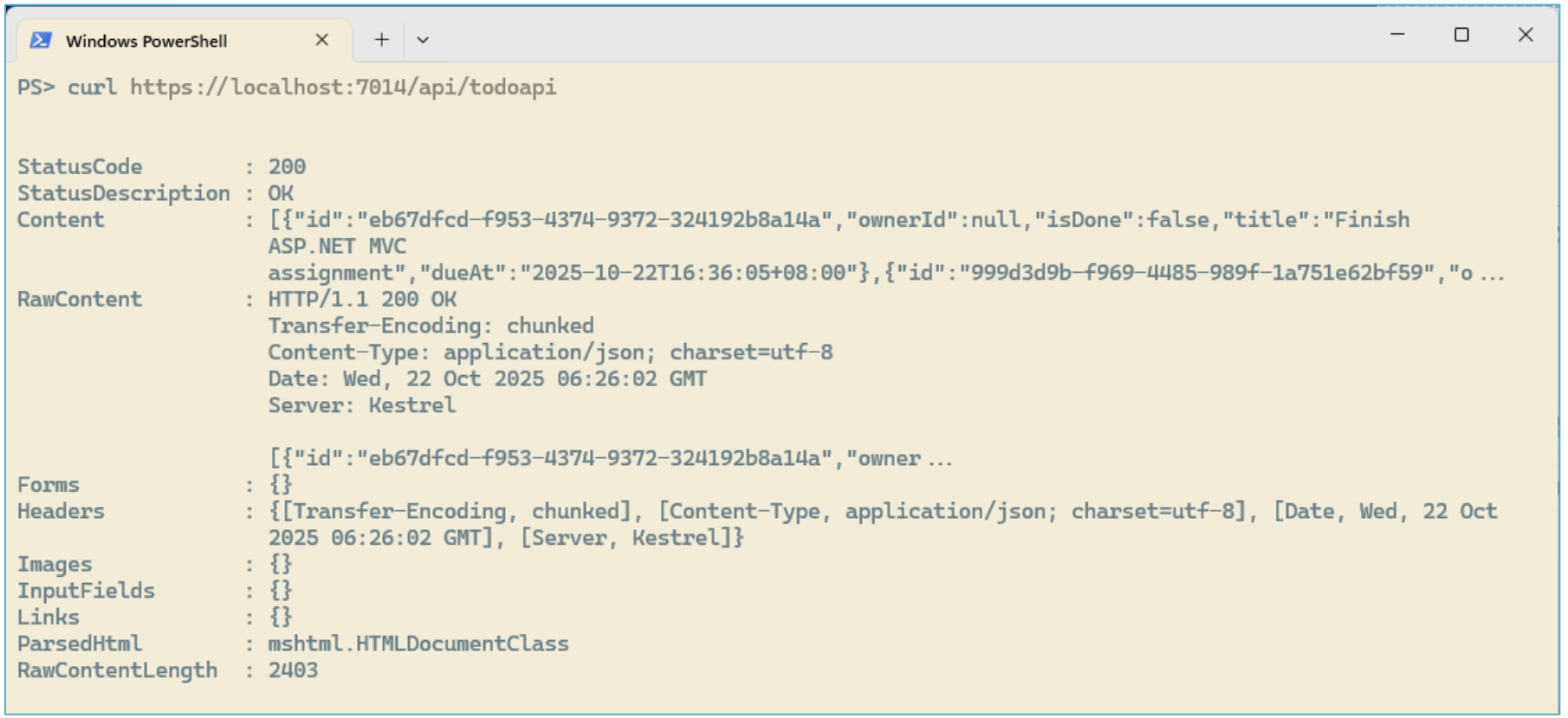


Action	Method	Path	Request Body	Success	Error
List all todos	GET	/api/todos	—	200 OK + JSON array	500 on server error
Create todo	POST	/api/todos	JSON { title, isDone?, dueAt? }	201 Created + Location header	400 validation

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class TodoApiController : ControllerBase
{
    private readonly ITodoService _svc;

    0 references
    public TodoApiController(ITodoService svc) => _svc = svc;

    [HttpGet]
    0 references
    public async Task<ActionResult<IEnumerable<TodoItem>>> GetAll()
    {
        return Ok(await _svc.GetItemsAsync());
    }
}
```

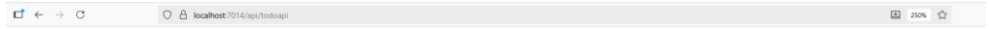


Controller-Based API



Running Two HttpGet

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class TodoApiController : ControllerBase
{
    private readonly ITodoService _svc;

    0 references
    public TodoApiController(ITodoService svc) => _svc = svc;

    [HttpGet]
    0 references
    public async Task<ActionResult<IEnumerable<TodoItem>>> GetAll()
    {
        return Ok(await _svc.GetItemsAsync());
    }

    [HttpGet]
    0 references
    public async Task<ActionResult<TodoItem>> GetItemAsync(Guid id)
    {
        return Ok(await _svc.GetItemAsync(id));
    }
}
```



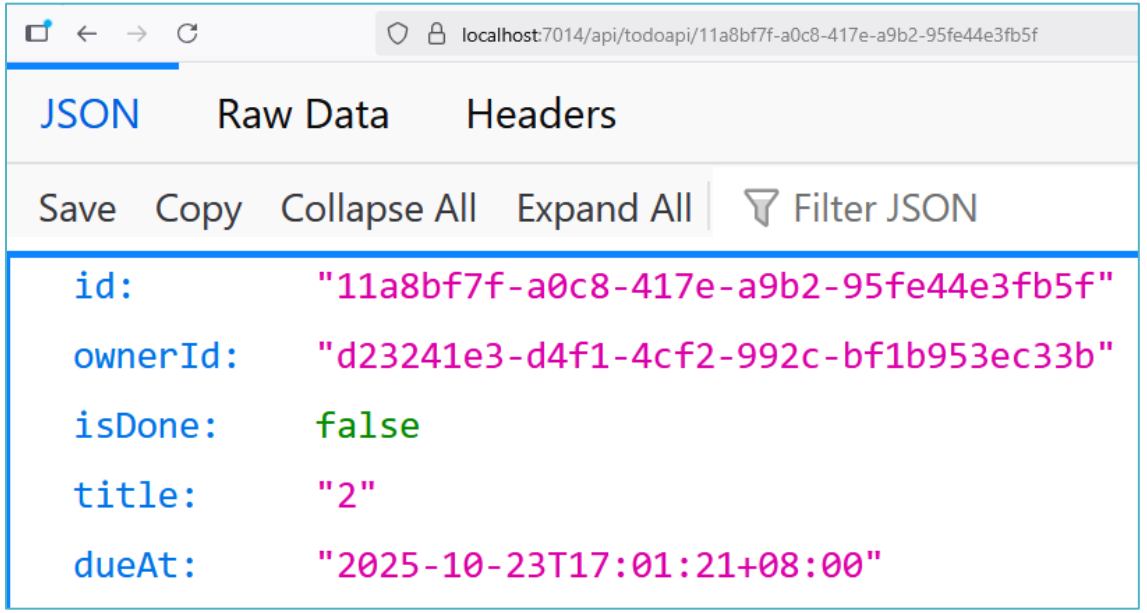
An unhandled exception occurred while processing the request.

AmbiguousMatchException: The request matched multiple endpoints. Matches:

MyTodoApp.Mvc.Controllers.TodoApiController.GetItemAsync (MyTodoApp.Mvc)
MyTodoApp.Mvc.Controllers.TodoApiController.GetAll (MyTodoApp.Mvc)

Routing

```
[HttpGet("{id:guid}")]
0 references
public async Task<ActionResult<TodoItem>> GetItemById(Guid id)
{
    var item = await _svc.GetItemAsync(id);
    if (item is null) return NotFound();    // 404 when missing

    return Ok(item);
}
```

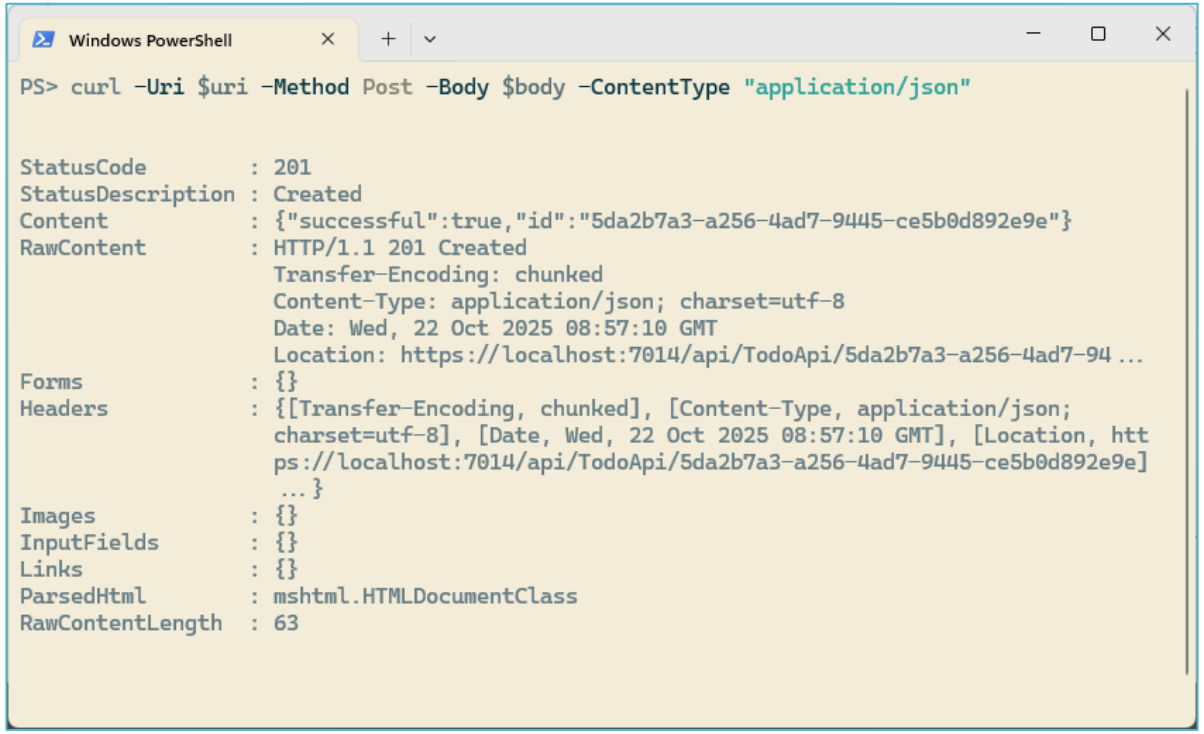


Call the Api

```
Windows PowerShell
PS> # Define the URL
PS> $uri = "https://localhost:7014/api/todoapi"
PS> # Create the JSON body
PS> $body = @{
>>     title = "Buy milk"
>>     isDone = $false
>>     dueAt = "2025-10-23T12:00:00Z"
>> } | ConvertTo-Json
PS> Invoke-RestMethod -Uri $uri -Method Post -Body $body -ContentType "application/json"

successful id
---------- --
      True 28d15079-a648-4962-ba51-4531496174b0


PS>
```

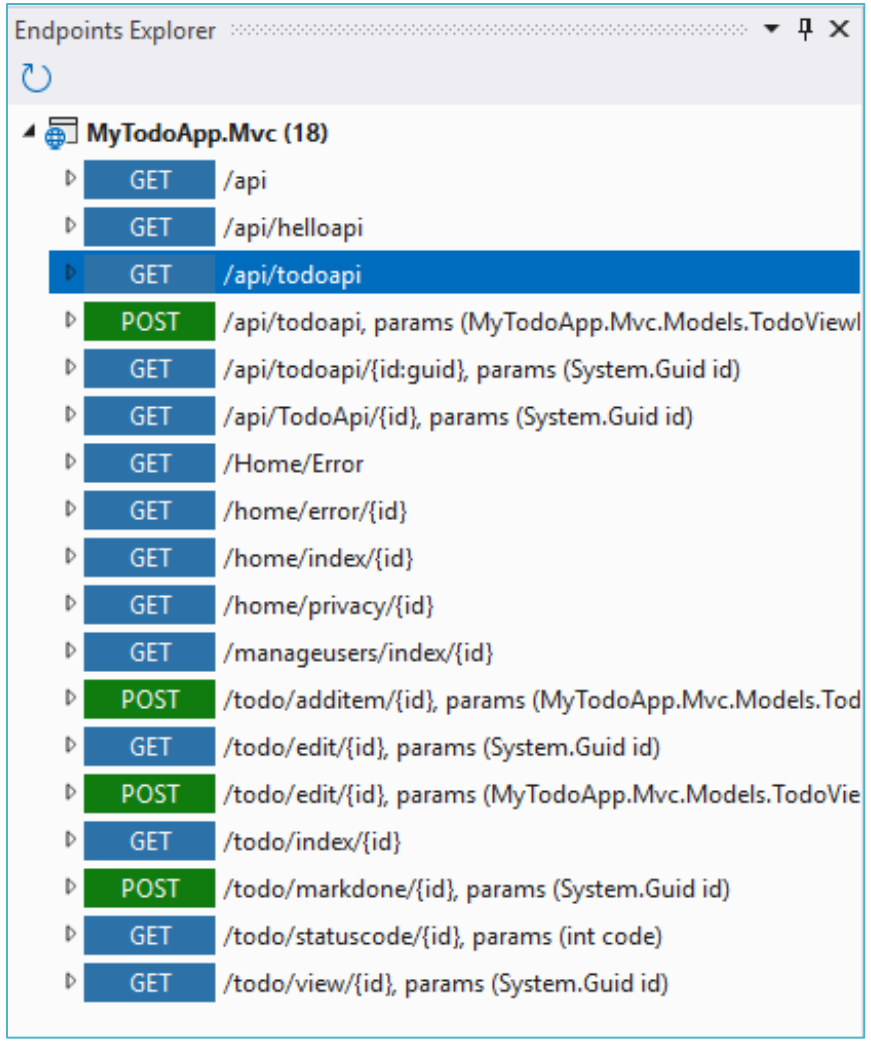
```csharp
/// <summary>
/// POST /api/todo – create a new todo.
/// </summary>
[HttpPost]
0 references
public async Task<ActionResult<(bool, Guid)>> Create([FromBody] TodoItem dto)
{
    // Create via your domain/service
    (bool successful, Guid id) = await _svc.AddItemAsync(dto);

    // If you want to support Location header with 201 Created, expose a GET-by-id:
    return CreatedAtAction(nameof(GetItemById), new { id = id }, new { successful, id });
}
```

Call the Api


Windows PowerShell
PS> curl -Uri $uri -Method Post -Body $body -ContentType "application/json"
StatusCode : 201
StatusDescription : Created
Content : {"successful":true,"id":"5da2b7a3-a256-4ad7-9445-ce5b0d892e9e"}
RawContent : HTTP/1.1 201 Created
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Date: Wed, 22 Oct 2025 08:57:10 GMT
Location: https://localhost:7014/api/TodoApi/5da2b7a3-a256-4ad7-94 ...
Forms : {}
Headers : {[Transfer-Encoding, chunked], [Content-Type, application/json; charset=utf-8], [Date, Wed, 22 Oct 2025 08:57:10 GMT], [Location, https://localhost:7014/api/TodoApi/5da2b7a3-a256-4ad7-9445-ce5b0d892e9e] ...}
Images : {}
InputFields : {}
Links : {}
ParsedHtml : mshtml.HTMLDocumentClass
RawContentLength : 63


Windows PowerShell
PS> curl -Uri "https://localhost:7014/api/todoapi/28d15079-a648-4962-ba51-4531496174b0" -Method GET
StatusCode : 200
StatusDescription : OK
Content : {"id":"28d15079-a648-4962-ba51-4531496174b0","ownerId":null,"isDone":false,"title":"Buy milk","dueAt":"2025-10-23T12:00:00+00:00"}
RawContent : HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Date: Wed, 22 Oct 2025 08:55:41 GMT
Server: Kestrel
{"id":"28d15079-a648-4962-ba51-4531496174b0","ownerI ...
Forms : {}
Headers : {[Transfer-Encoding, chunked], [Content-Type, application/json; charset=utf-8], [Date, Wed, 22 Oct 2025 08:55:41 GMT], [Server, Kestrel]}
Images : {}
InputFields : {}
Links : {}
ParsedHtml : mshtml.HTMLDocumentClass
RawContentLength : 130

Test Web API




```
@MyTodoApp.Mvc_HostAddress = https://localhost:7014

Send request | Debug
GET {{MyTodoApp.Mvc_HostAddress}}/api/todoapi

###
```

Status: 200 OK Time: 32.2 ms Size: 2927 bytes

Formatted **Raw** **Headers** **Request**

Body

application/json; charset=utf-8, 2927 bytes


```
[
  {
    "id": "eb67dfcd-f953-4374-9372-324192b8a14a",
    "ownerId": null,
    "isDone": false,
    "title": "Finish ASP.NET MVC assignment",
    "dueAt": "2025-10-22T16:36:05+08:00"
  },
  {
    "id": "999d3d9b-f969-4485-989f-1a751e62bf59",
    "ownerId": null,
    "isDone": false,
    "title": "Review Entity Framework notes",
    "dueAt": "2025-10-22T16:37:52+08:00"
  },
```

Test Web API

```
Send request | Debug
POST {{MyTodoApp.Mvc_HostAddress}}/api/todoapi
Content-Type: application/json

{
  "title": "Buy milk",
  "isDone": false,
  "dueAt": "2025-10-23T12:00:00Z"
}
```

Name	Value
Date	Wed, 22 Oct 2025 09:13:05 GMT
Server	Kestrel
Location	https://localhost:7014/api/TodoApi/b3d9c30d-f160-4631-a532-f7125454a12f
Transfer-Encoding	chunked
Content-Type	application/json; charset=utf-8
Content-Length	63

Status: 201 Created Time: 24.27 ms Size: 63 bytes

Formatted **Raw** **Headers** **Request**

Body

application/json; charset=utf-8, 63 bytes

```
{
  "successful": true,
  "id": "b3d9c30d-f160-4631-a532-f7125454a12f"
}
```

Test Web API

```
Send request | Debug
GET {{MyTodoApp.Mvc_HostAddress}}/todo/index/

###
```

Status: 200 OK Time: 88.87 ms Size: 6784 bytes

Formatted **Raw** **Headers** **Request**

Body

text/html; charset=utf-8, 6784 bytes

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Log in - MyTodoApp.Mvc</title>
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="/css/site.css?v=pAGv4ietcJNk_EwsQZ5BN9-K4MuNYS2a9wl4Jw-
    <link rel="stylesheet" href="/MyTodoApp.Mvc.styles.css?v=NIs2dJfM6njYNc1UyNKWFBjauaf
</head>
<body>
    <header b-a497qte5te>
        <nav b-a497qte5te class="navbar navbar-expand-sm navbar-toggleable-sm navbar-lig
            <div b-a497qte5te class="container-fluid">
                <a class="navbar-brand" href="/">MyTodoApp.Mvc</a>
                <button b-a497qte5te class="navbar-toggler" type="button" data-bs-toggle
                        aria-expanded="false" aria-label="Toggle navigation">
```

Test Web API

```
// PUT /api/todos/{id}
[HttpPut()]
0 references
public async Task<IActionResult> UpdateAsync([FromBody] TodoItem item)
{
    bool ok = await _svc.UpdateItemAsync(item);
    if (!ok) return NotFound();      // item with this id didn't exist

    return NoContent();              // 204 on successful update
}
```

```
Send request | Debug
PUT {{MyTodoApp.Mvc_HostAddress}}/api/todoapi
Content-Type: application/json

{
    "id": "71261a69-3239-409c-9858-aee82b306a7f",
    "ownerId": null,
    "isDone": false,
    "title": "Buy milk1",
    "dueAt": "2025-10-23T20:00:00+08:00"
}
```

Status: 204 No Content Time: 54.71 ms Size: 0 bytes

Formatted Raw Headers Request

Body

0 bytes

Status: 404 Not Found Time: 266.33 ms Size: 162 bytes

Formatted Raw Headers Request

Body

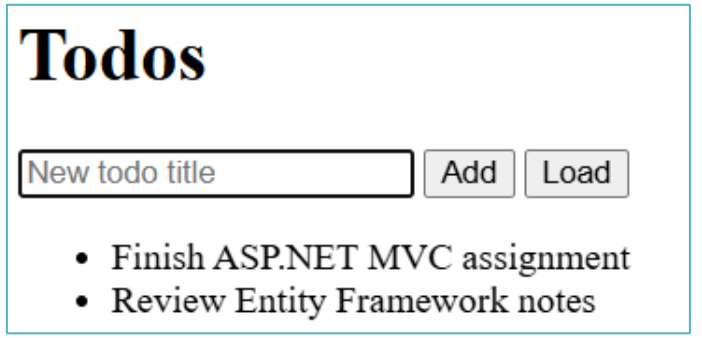
application/problem+json; charset=utf-8, 162 bytes

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",
  "title": "Not Found",
  "status": 404,
  "traceId": "00-3a2978ee86e39e6d42017ad99d401cdc-d3afdc27917e1912-00"
}
```

Implement MarkDone & Delete

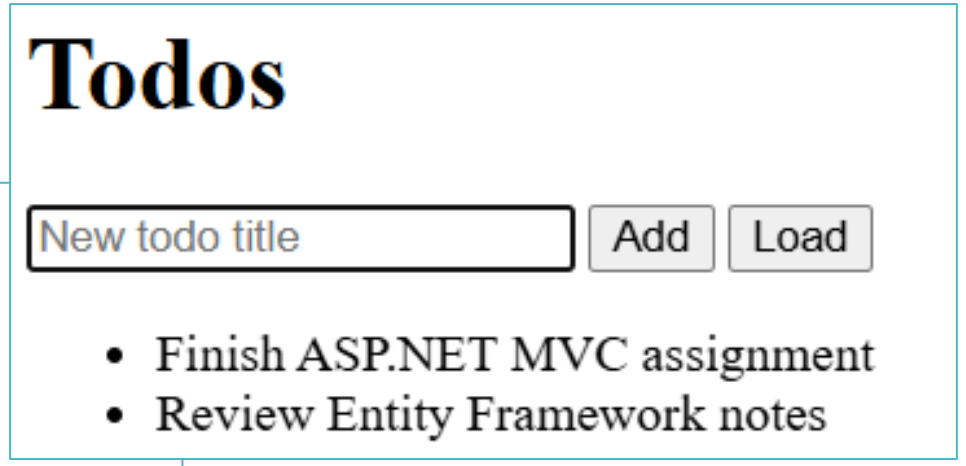
Call Api in JavaScript

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Fetch Demo</title>
</head>
<body>
    <h1>Todos</h1>
    <input id="title" placeholder="New todo title">
    <button id="add">Add</button>
    <button id="load">Load</button>
    <ul id="list"></ul>
    <script>
        const API = '/api/todos'; // change if needed
        const $list = document.getElementById('list');

        async function load() {
            const res = await fetch(API);
            const data = await res.json();
            $list.innerHTML = data.map(t => `<li>${t.title} ${t.isDone ? '(done)' : ''}</li>`).join('');
        }

        async function add() {
            const title = document.getElementById('title').value.trim();
            if (!title) return;

            // create a date two days from now
            const due = new Date();
            due.setDate(due.getDate() + 2);

            await fetch(API, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ title, isDone: false, dueAt: due.toISOString() })
            });
            document.getElementById('title').value = '';
            load();
        }
        document.getElementById('load').onclick = load;
        document.getElementById('add').onclick = add;
        load(); // initial
    </script>
</body>
</html>
```



Call Api in JavaScript

```
async function add() {
    const title = document.getElementById('title').value.trim();
    if (!title) return;

    // create a date two days from now
    const due = new Date();
    due.setDate(due.getDate() + 2);

    fetch(API, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ title, isDone: false, dueAt: due.toISOString() })
    })
        .then(res => {
            if (!res.ok) throw new Error(`HTTP ${res.status}`);
            console.log('Todo added successfully!');
            document.getElementById('title').value = '';
            load();
        })
        .catch(err => {
            console.error('Fetch failed:', err);
            alert('❌ Failed to add todo.');
        });

    document.getElementById('title').value = '';
    load();
}
```

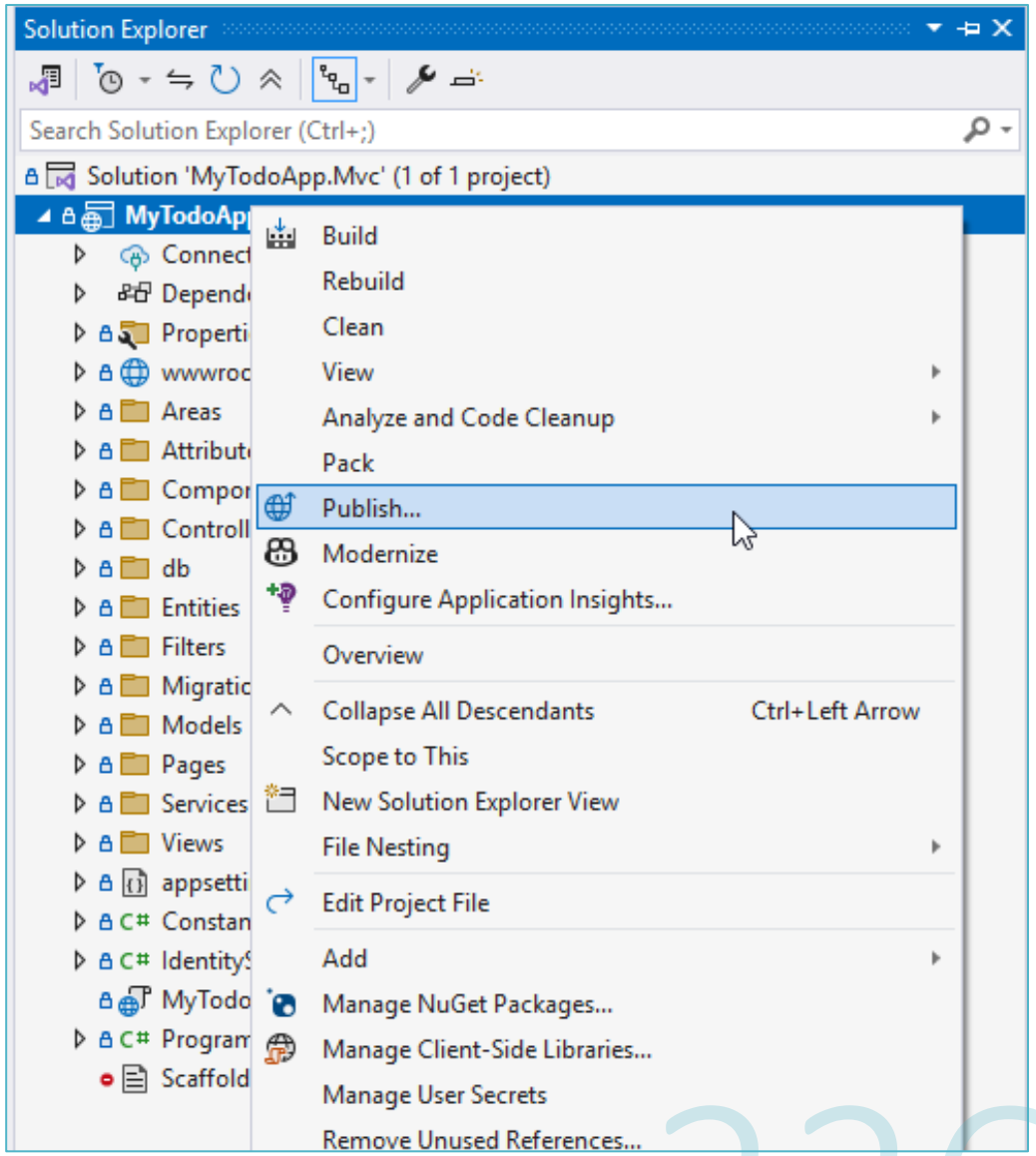


Call Api in JavaScript

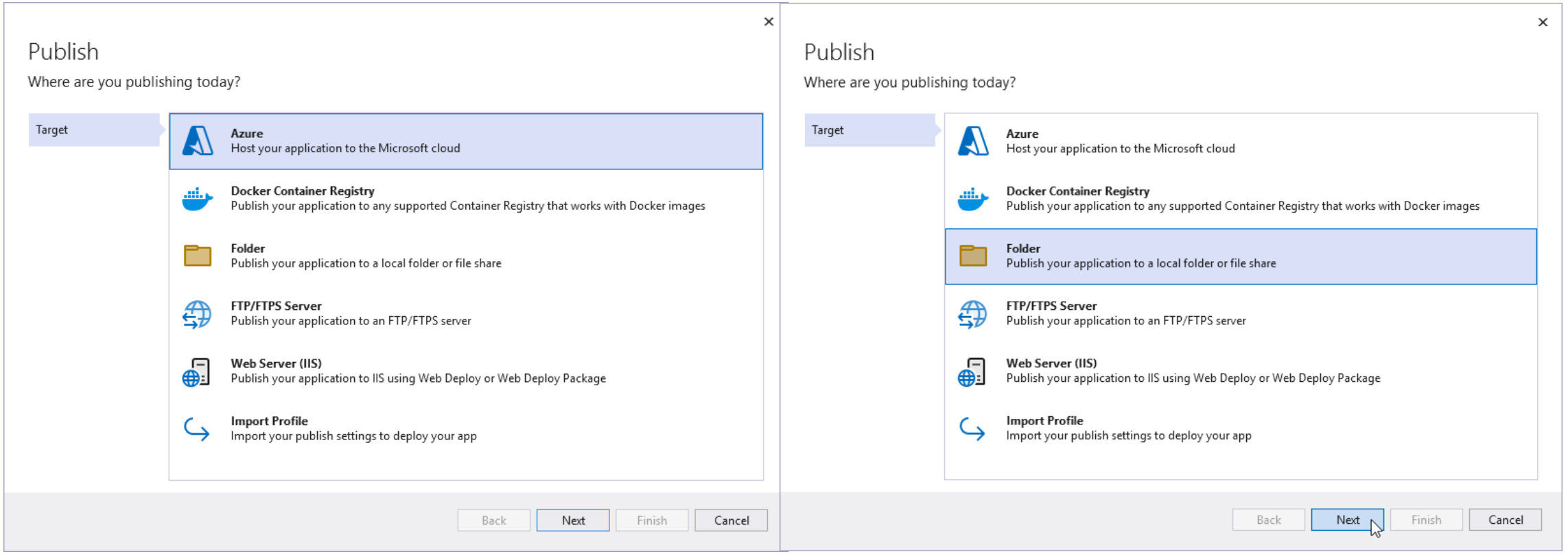
```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
async function add() {
    const title = document.getElementById('title').value.trim();
    if (!title) return;

    // create a date two days from now
    const due = new Date();
    due.setDate(due.getDate() + 2);

    $.ajax({
        url: "/api/todos",
        type: "POST",                       // HTTP method
        contentType: "application/json",  // sending JSON
        data: JSON.stringify({
            title: title,
            isDone: false,
            dueAt: due.toISOString()
        }),
        success: function (result) {
            console.log("Todo added:", result);
            $("#title").val("");
            load(); // reload list
        },
        error: function (xhr, status, error) {
            console.error("Error:", xhr.responseText);
            alert("Failed to add todo: " + error);
        }
    });
}
```

Module 8

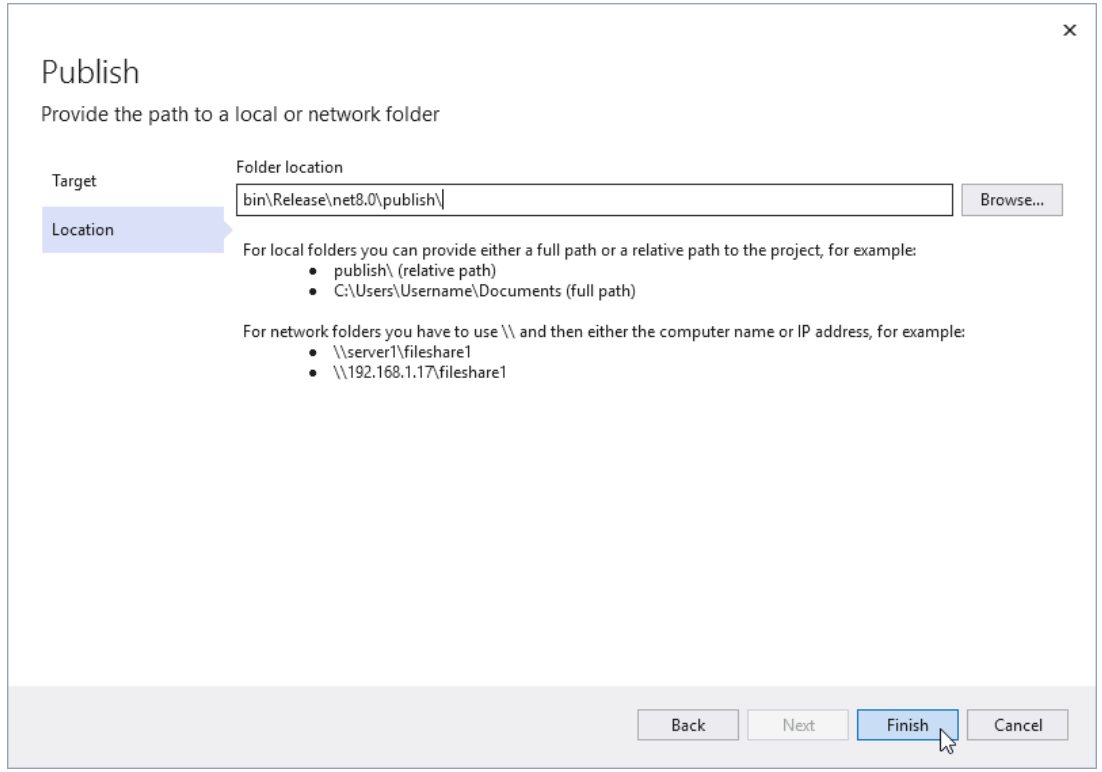
Publish

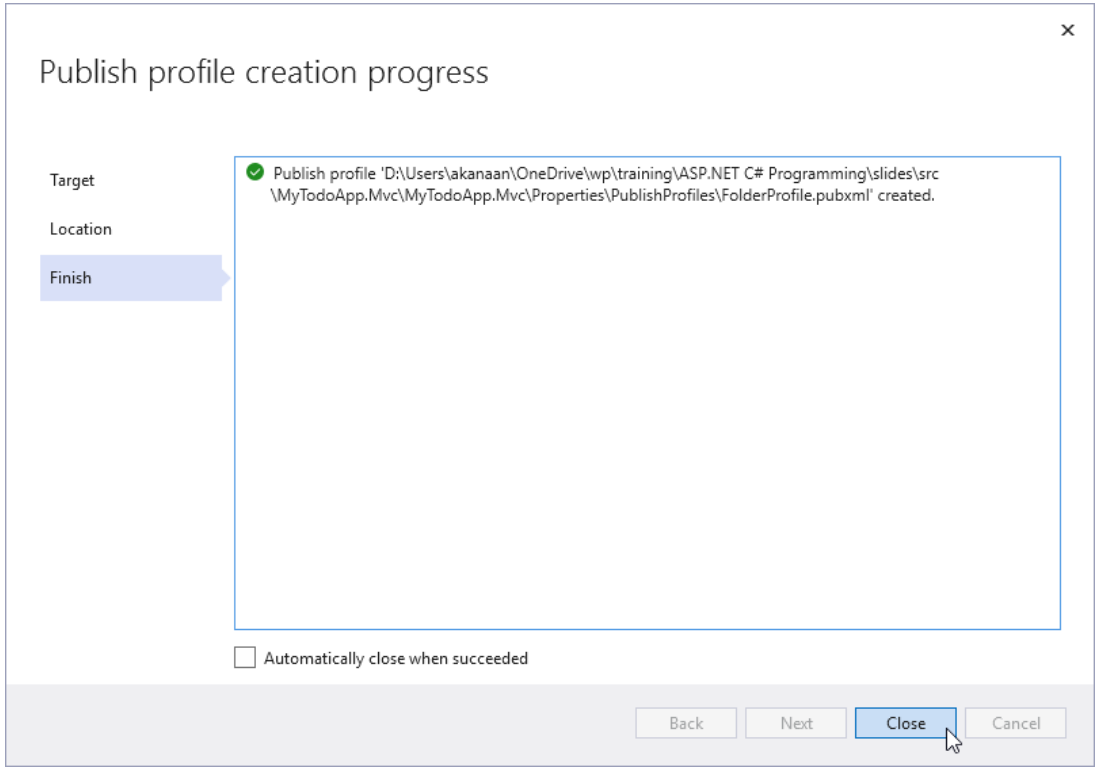

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MyTodoApp.Mvc' (1 of 1 project)
Build
Rebuild
Clean
View
Analyze and Code Cleanup
Pack
Publish...
Modernize
Configure Application Insights...
Overview
Collapse All Descendants
Ctrl+Left Arrow
Scope to This
New Solution Explorer View
File Nesting
Edit Project File
Add
Manage NuGet Packages...
Manage Client-Side Libraries...
Manage User Secrets
Remove Unused References...
Connect
Depend
Properti
wwwroc
Areas
Attribut
Compor
Controll
db
Entities
Filters
Migratic
Models
Pages
Services
Views
appsetti
Constan
Identity
MyTodo
Program
Scaffold

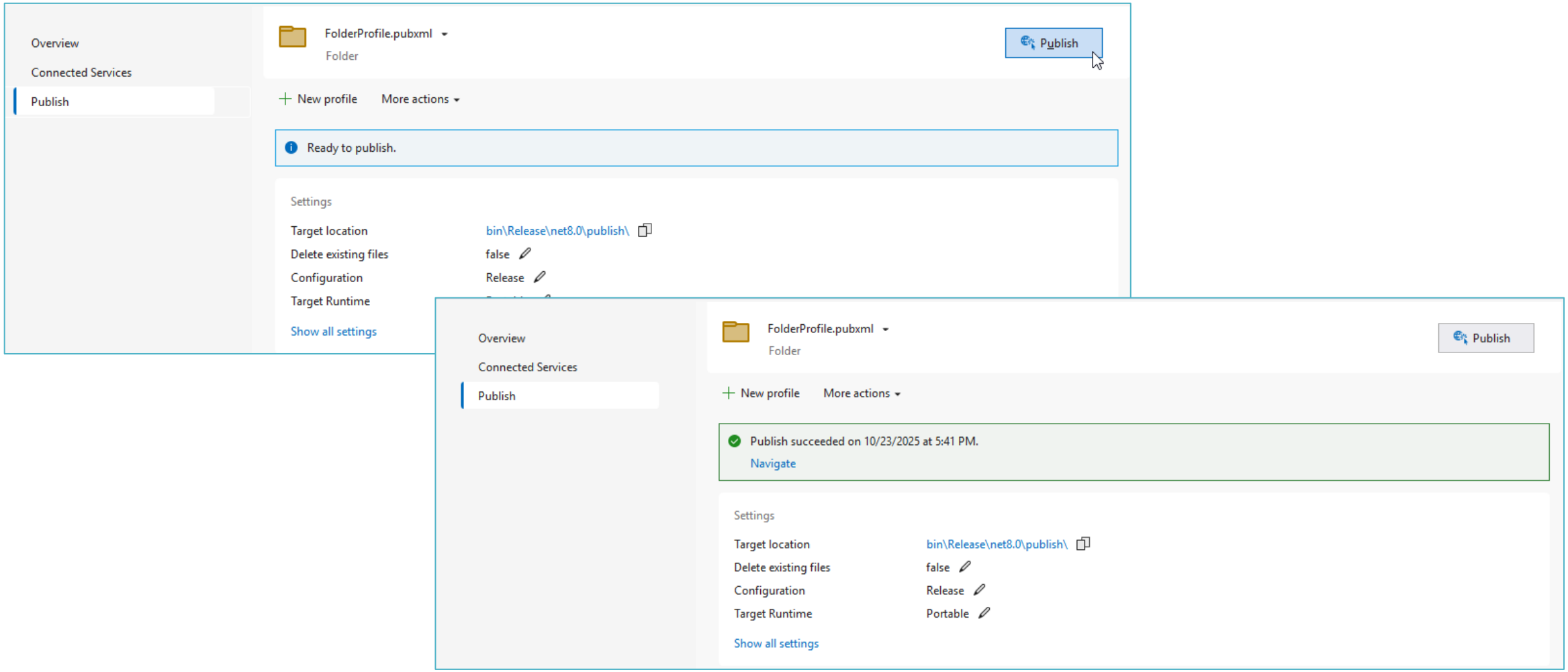
Publish


Publish
Where are you publishing today?
Target
Azure
Host your application to the Microsoft cloud
Docker Container Registry
Publish your application to any supported Container Registry that works with Docker images
Folder
Publish your application to a local folder or file share
FTP/FTPS Server
Publish your application to an FTP/FTPS server
Web Server (IIS)
Publish your application to IIS using Web Deploy or Web Deploy Package
Import Profile
Import your publish settings to deploy your app
Back
Next
Finish
Cancel
Publish
Where are you publishing today?
Target
Azure
Host your application to the Microsoft cloud
Docker Container Registry
Publish your application to any supported Container Registry that works with Docker images
Folder
Publish your application to a local folder or file share
FTP/FTPS Server
Publish your application to an FTP/FTPS server
Web Server (IIS)
Publish your application to IIS using Web Deploy or Web Deploy Package
Import Profile
Import your publish settings to deploy your app
Back
Next
Finish
Cancel

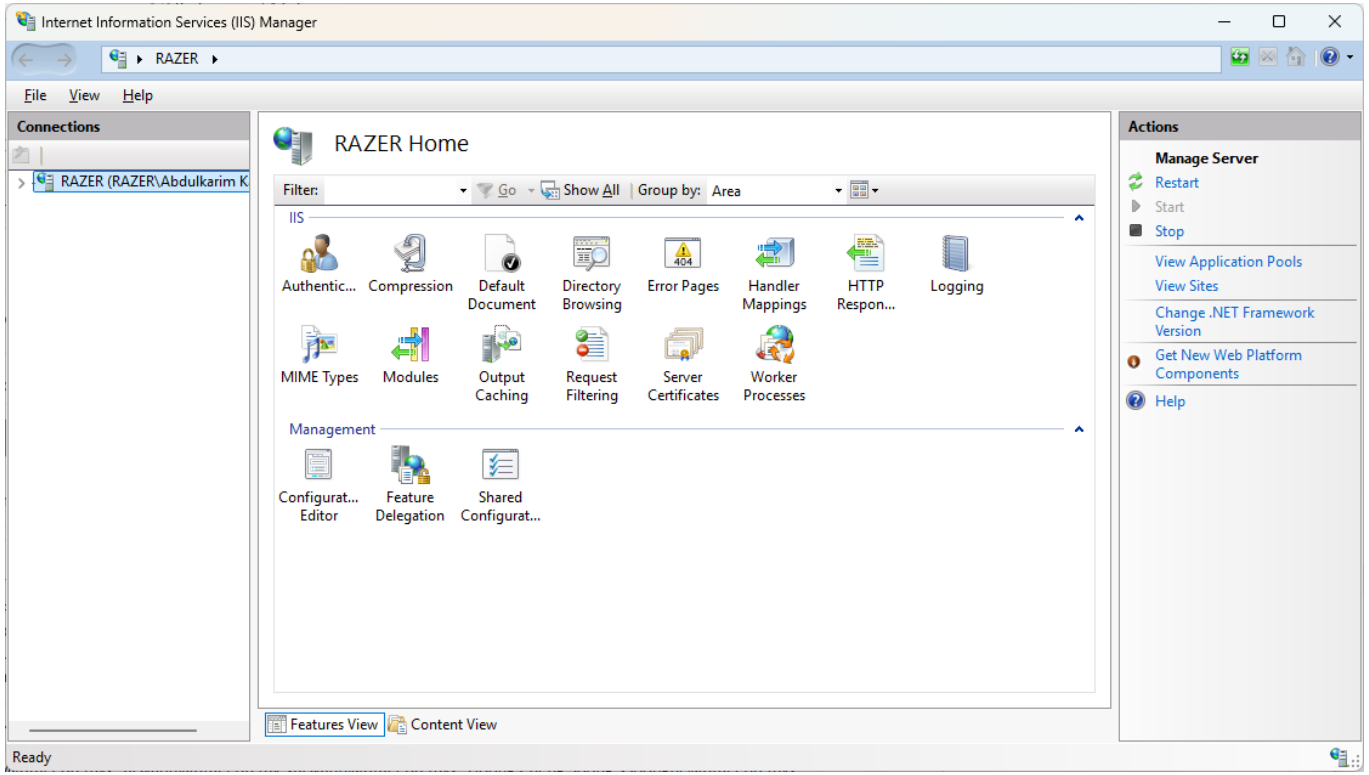
Publish

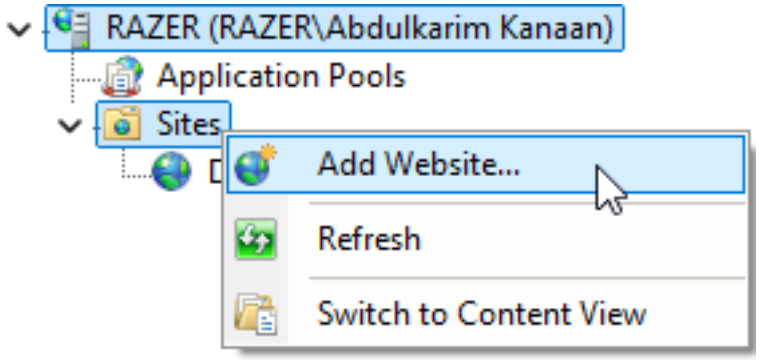

Publish
Provide the path to a local or network folder
Target
Location
Folder location
bin\Release\net8.0\publish\
Browse...
For local folders you can provide either a full path or a relative path to the project, for example:
publish\ (relative path)
C:\Users\Username\Documents (full path)
For network folders you have to use \\ and then either the computer name or IP address, for example:
\\server1\fileshare1
\\192.168.1.17\fileshare1
Back
Next
Finish
Cancel


Publish profile creation progress
Target
Location
Finish
Publish profile 'D:\Users\akanaan\OneDrive\wp\training\ASP.NET C# Programming\slides\src\MyTodoApp.Mvc\MyTodoApp.Mvc\Properties\PublishProfiles\FolderProfile.pubxml' created.
Automatically close when succeeded
Back
Next
Close
Cancel

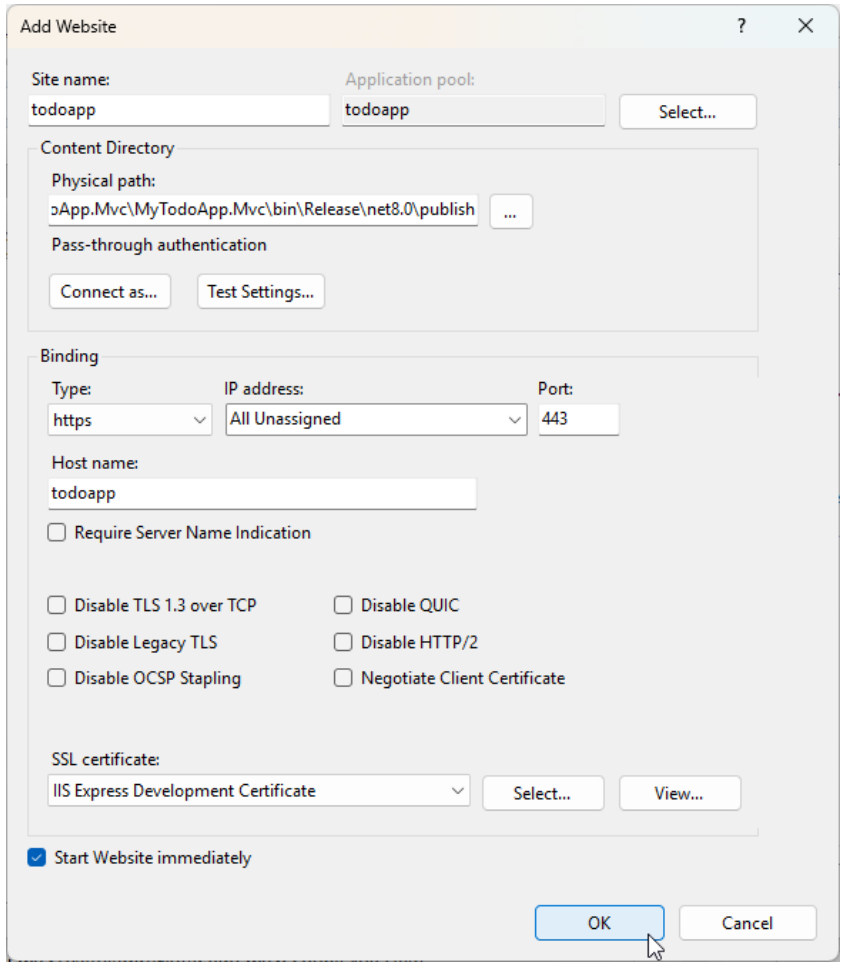

Overview
Connected Services
Publish
FolderProfile.pubxml
Folder
Publish
New profile
More actions
Ready to publish.
Settings
Target location
bin\Release\net8.0\publish\
Delete existing files
false
Configuration
Release
Target Runtime
Show all settings
Overview
Connected Services
Publish
FolderProfile.pubxml
Folder
Publish
New profile
More actions
Publish succeeded on 10/23/2025 at 5:41 PM.
Navigate
Settings
Target location
bin\Release\net8.0\publish\
Delete existing files
false
Configuration
Release
Target Runtime
Portable
Show all settings

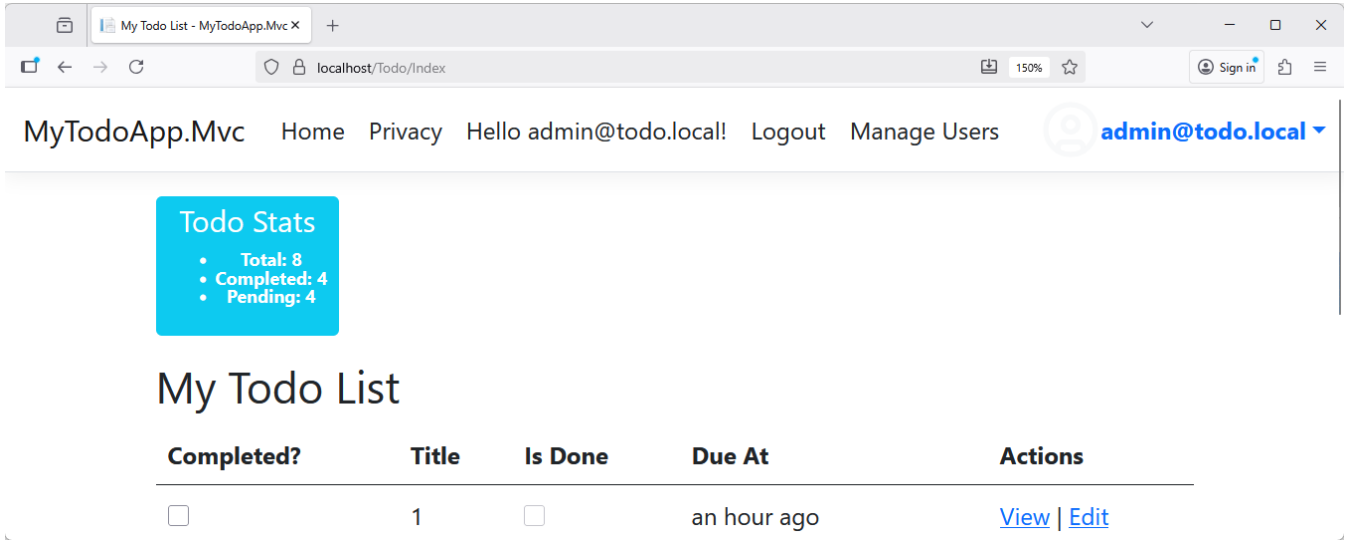
IIS


Internet Information Services (IIS) Manager
RAZER
File View Help
Connections
RAZER (RAZER\Abdulkarim K
RAZER Home
Filter:
Go
Show All
Group by: Area
IIS
Authentic...
Compression
Default Document
Directory Browsing
Error Pages
Handler Mappings
HTTP Respon...
Logging
MIME Types
Modules
Output Caching
Request Filtering
Server Certificates
Worker Processes
Management
Configurat... Editor
Feature Delegation
Shared Configurat...
Features View
Content View
Actions
Manage Server
Restart
Start
Stop
View Application Pools
View Sites
Change .NET Framework Version
Get New Web Platform Components
Help
Ready


RAZER (RAZER\Abdulkarim Kanaan)
Application Pools
Sites
Add Website...
Refresh
Switch to Content View

Running the App





Certifications & Credentials

Abdulkarim Kanaan Jebna

🎓 IBM Data Science Professional Certificate

🤖 DeepLearning.AI TensorFlow Developer Certificate

💼 Microsoft Certified Professional (MCP)

🖥️ Microsoft Certified Technology Specialist (MCTS)

Abdulkarim M. Jamal Kanaan


Linked-In: a-kanaan


abdulkarim@utar.edu.my


a.kanaan@msn.com


017-2290877

Assistant Professor, GT

Research: R&D, Machine Learning, Deep Learning, Technopreneurship, Management Information System

Training: Visualization, Artificial Intelligence

System Architect: .NET Framework – Windows Application, Web Application

Programming Skills: C# & Python