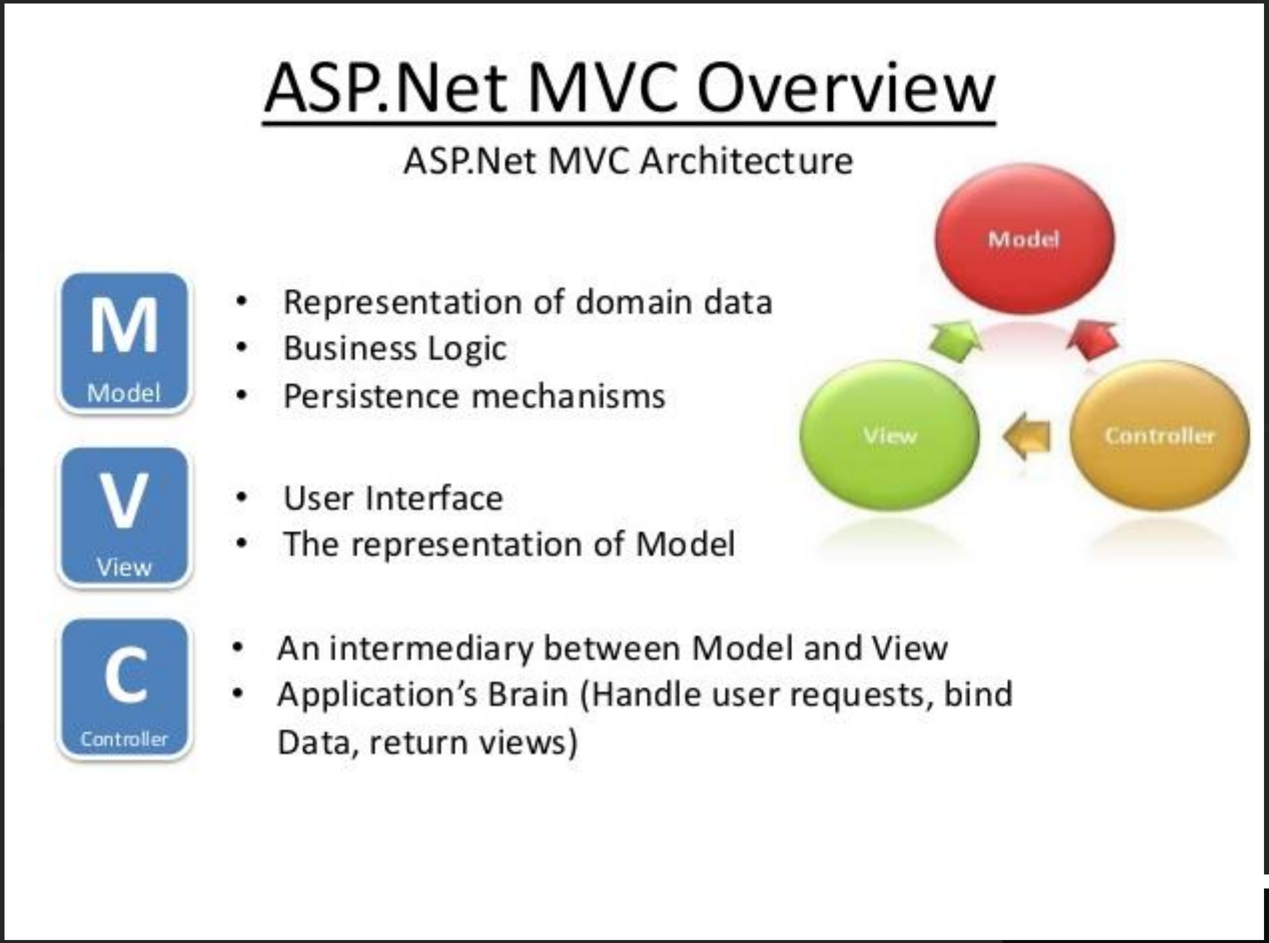
COMP 2139
How to develop a Single-Page Web Application

Agenda

- Describe how to use Visual Studio to create an ASP.NET core project
- List the name of six folders that are included in an MVC web application
- Describe how a controller and its action methods work
- Describe how you can use the **ViewBag** property to transfer data from a controller to a view
- Distinguish between a Razor code block and a Razor expression
- Describe how to use the **Startup.cs** file to configure the HTTP Request and response pipeline for a simple ASP.NET Code MVC web app
- Distinguish between a model class and a controller class
- Describe the purpose of a Razor view imports page
- Describe how to use the @model directive
- Describe how to use the **asp-controller** and **asp-action** tag helpers
- Describe the use of HttpGet and HttpPost
- Distinguish between a Razor layout and a Razor View
- Describe the purpose of a Razor view start
- Describe validating data within ASP.NET Core

MVC

Model-View-Controller



ASP.Net MVC Overview

ASP.Net MVC Architecture

M Model

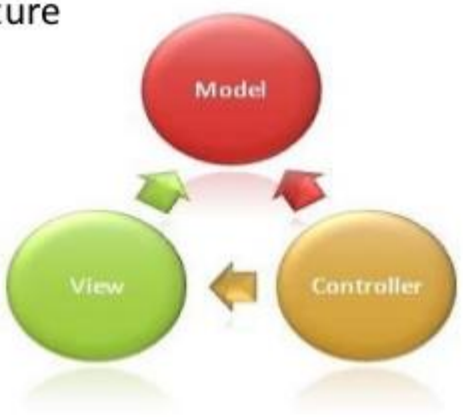
- Representation of domain data
- Business Logic
- Persistence mechanisms

V View

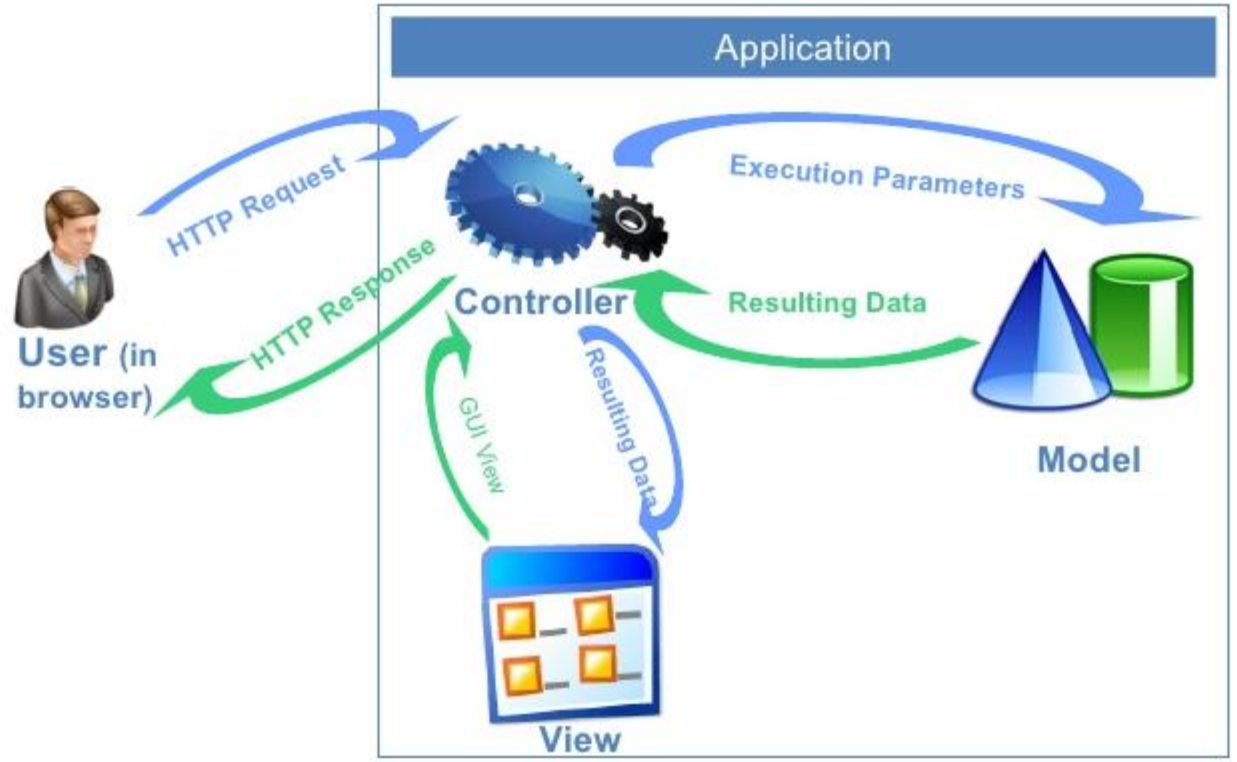
- User Interface
- The representation of Model

C Controller

- An intermediary between Model and View
- Application's Brain (Handle user requests, bind Data, return views)

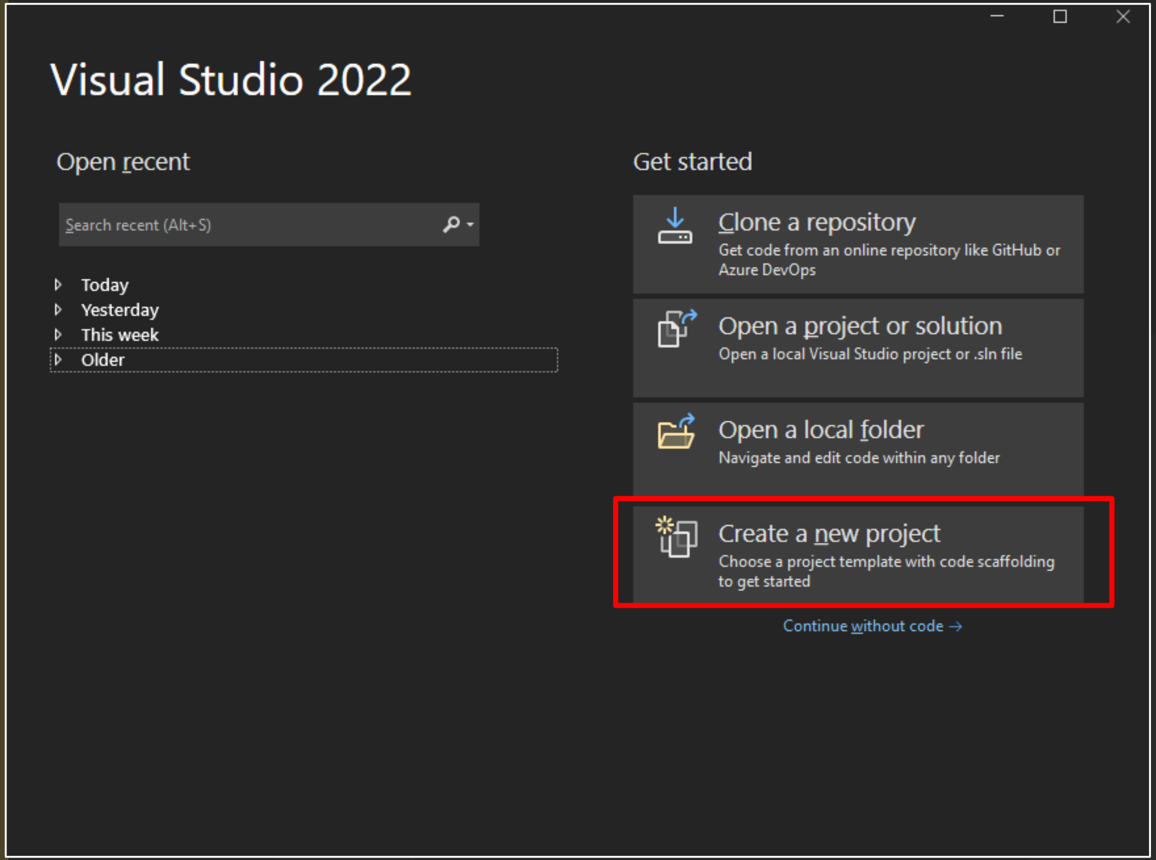


How ASP.NET MVC works



Creating a New Web Application

The Dialog for starting a new Web Application



Visual Studio 2022

Open recent

Search recent (Alt+S)

- Today
- Yesterday
- This week
- Older

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

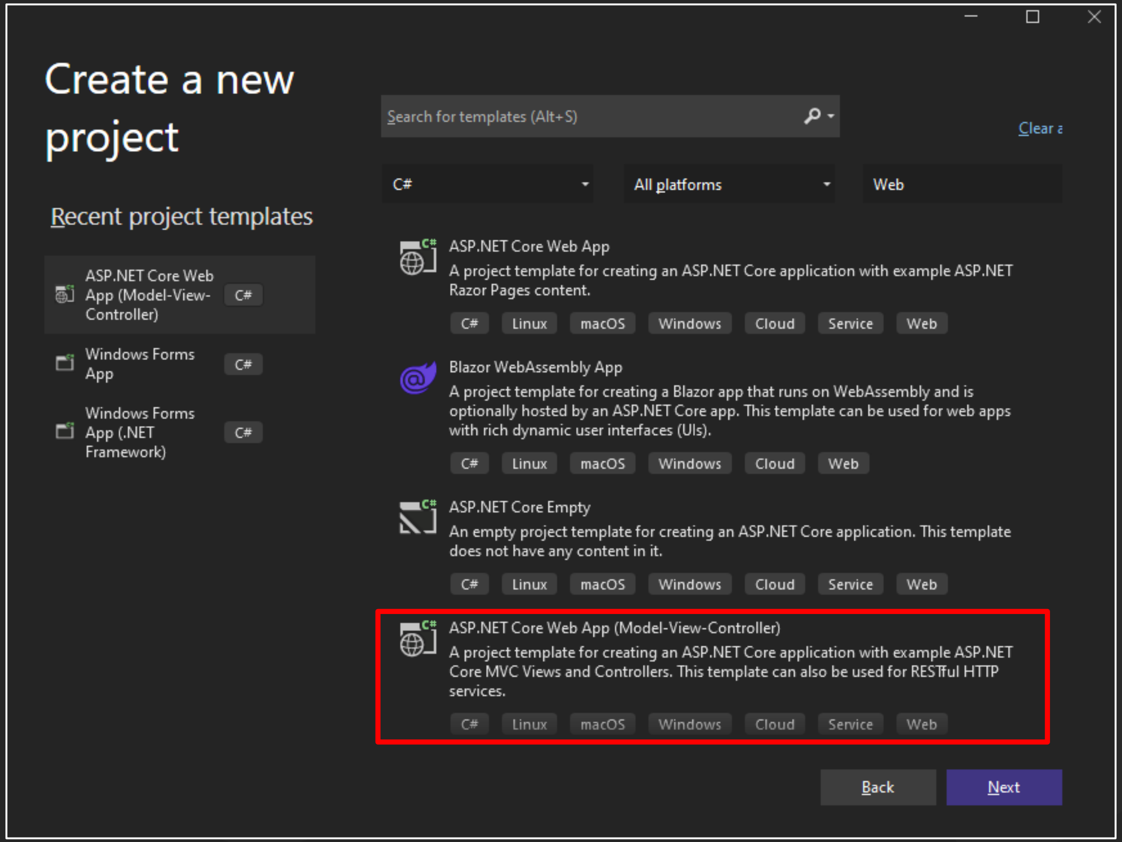
Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

1. Start Visual Studio.
2. From the menu system, select **File→New→Project**.
3. Select the **ASP.NET Core Web Application** item and click the **Next** button.
4. Enter a project name.
5. Specify the location (folder). To do that, you can click the Browse button.
6. If necessary, edit the solution name and click the Create button.

Project Templates Dialog



Create a new project

Search for templates (Alt+S)

Clear a

C# | All platforms | Web

Recent project templates

- ASP.NET Core Web App (Model-View-Controller) C#
- Windows Forms App C#
- Windows Forms App (.NET Framework) C#

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

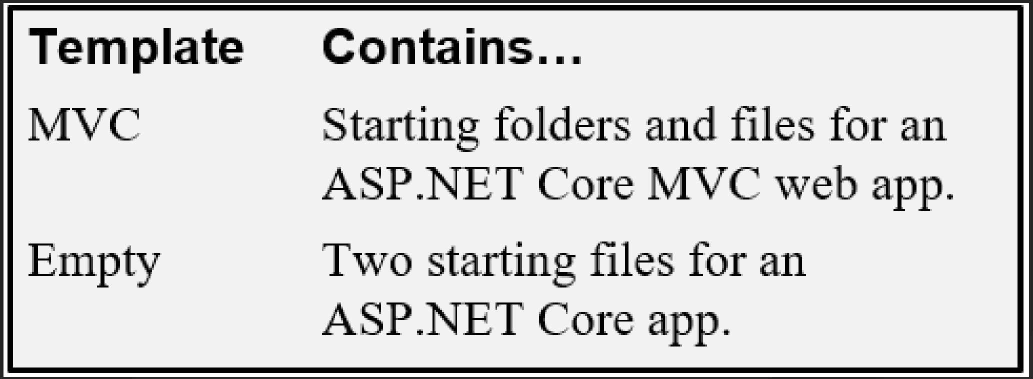
Blazor WebAssembly App
A project template for creating a Blazor app that runs on WebAssembly and is optionally hosted by an ASP.NET Core app. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web

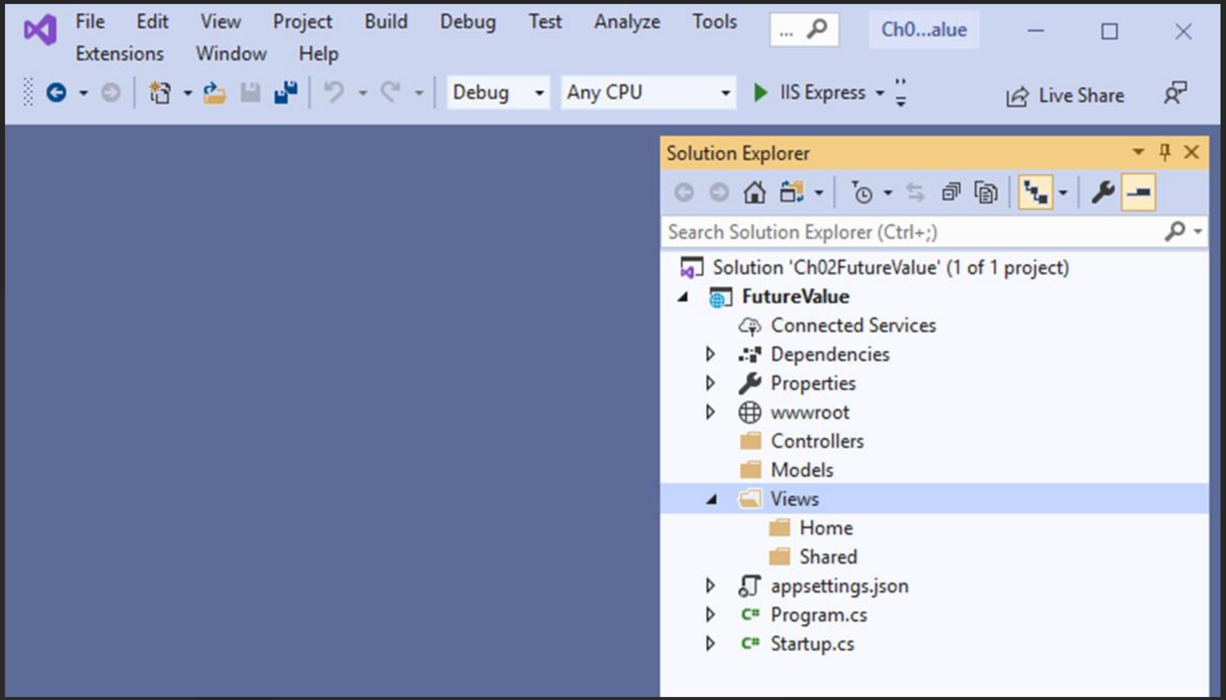
ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
C# Linux macOS Windows Cloud Service Web

Use the resulting dialog to select the **Web Application (Model-View-Controller)** template or the Empty template.

Templates for this Course



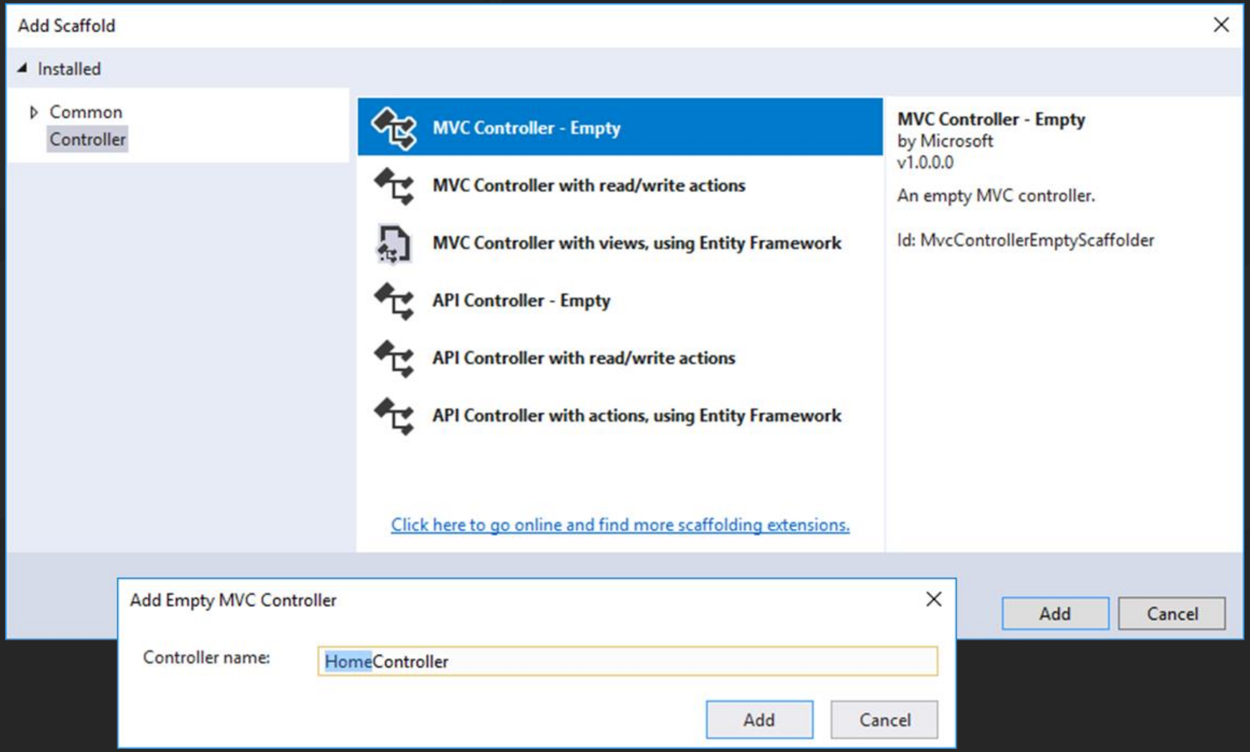
Template	Contains…
MVC	Starting folders and files for an ASP.NET Core MVC web app.
Empty	Two starting files for an ASP.NET Core app.



Solution 'Ch02FutureValue' (1 of 1 project)
- FutureValue
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - Controllers
  - Models
  - Views
    - Home
    - Shared
  - appsettings.json
  - Program.cs
  - Startup.cs

Visual Studio dialog for adding a Controller

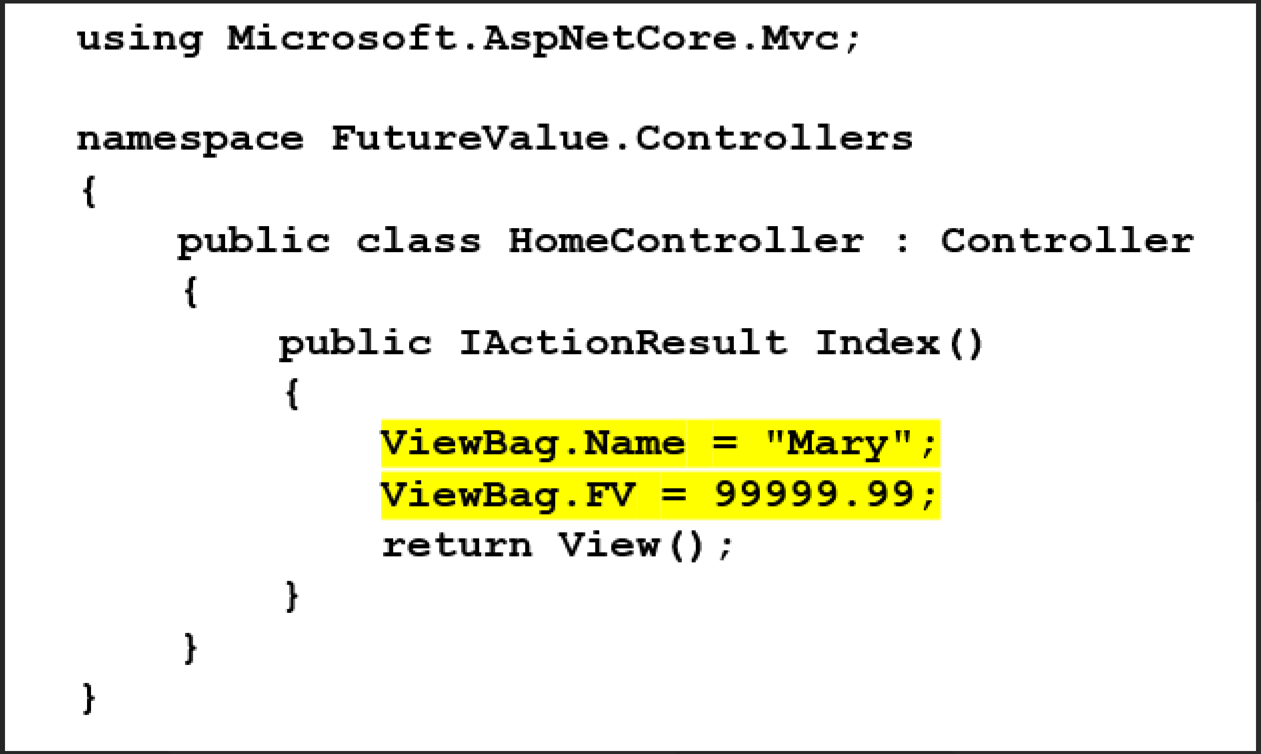
1. Right-click the **Controllers** folder and select **Add→Controller**.
2. In the **Add Scaffold** dialog, select "**MVC Controller – Empty**" and click Add.
3. In the Add Empty MVC Controller dialog, name the controller and click Add.


Add Scaffold
Installed
Common
Controller
MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
API Controller - Empty
API Controller with read/write actions
API Controller with actions, using Entity Framework
Click here to go online and find more scaffolding extensions.
Add
Cancel
Add Empty MVC Controller
Add
Cancel

MVC Controller - Empty
by Microsoft
v1.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder

Controller name: HomeController

Example Controller (HomeController.cs)

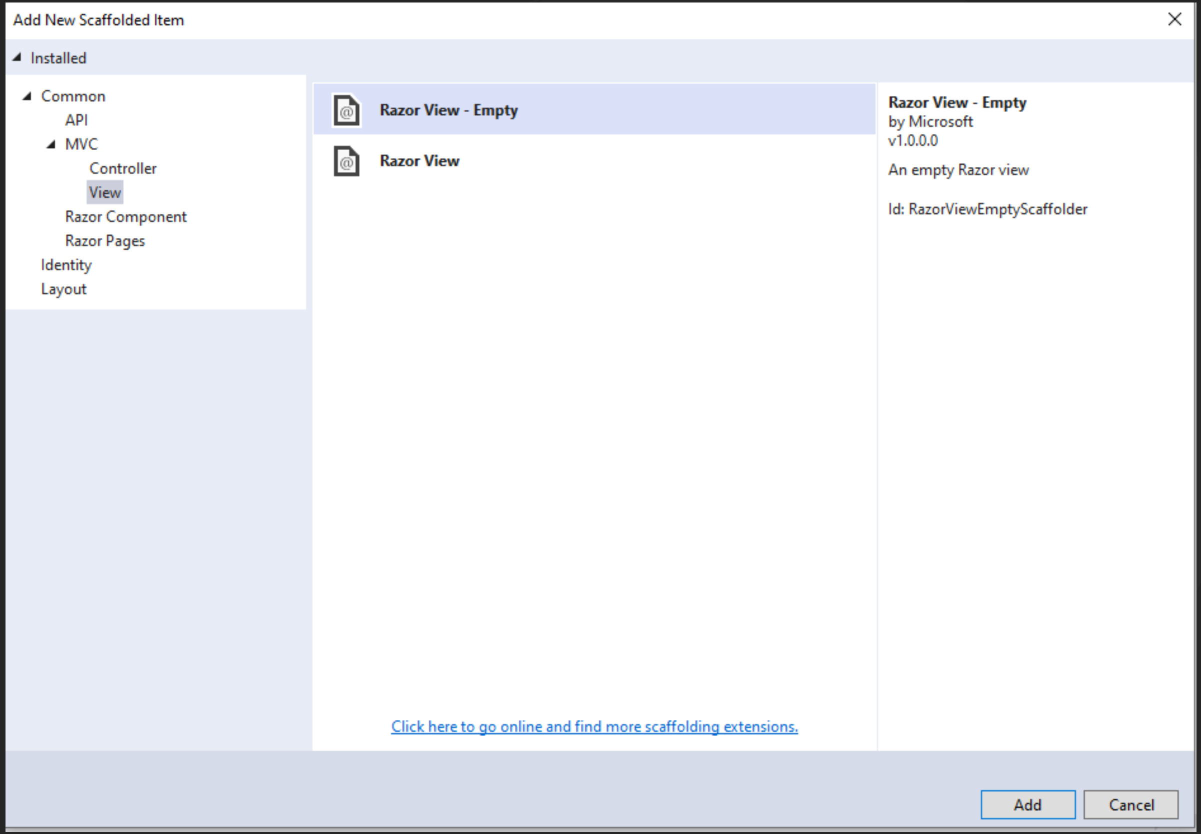


```
using Microsoft.AspNetCore.Mvc;

namespace FutureValue.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            ViewBag.Name = "Mary";
            ViewBag.FV = 99999.99;
            return View();
        }
    }
}
```

- A method of a controller runs in response to an HTTP action (ex GET or POST) is know as an action method
- The ViewBag is automatically available to controller and views, it uses dynamic properties to get/set value
- A View() method returns a ViewResult object for the view associated with an action method.

Adding a Razor view

1. In the Solution Explorer, right-click the Views/Home folder and select Add→View.
2. In the resulting dialog, enter the name of the view (ex "Index").
3. Click the Add button.



Add New Scaffolded Item

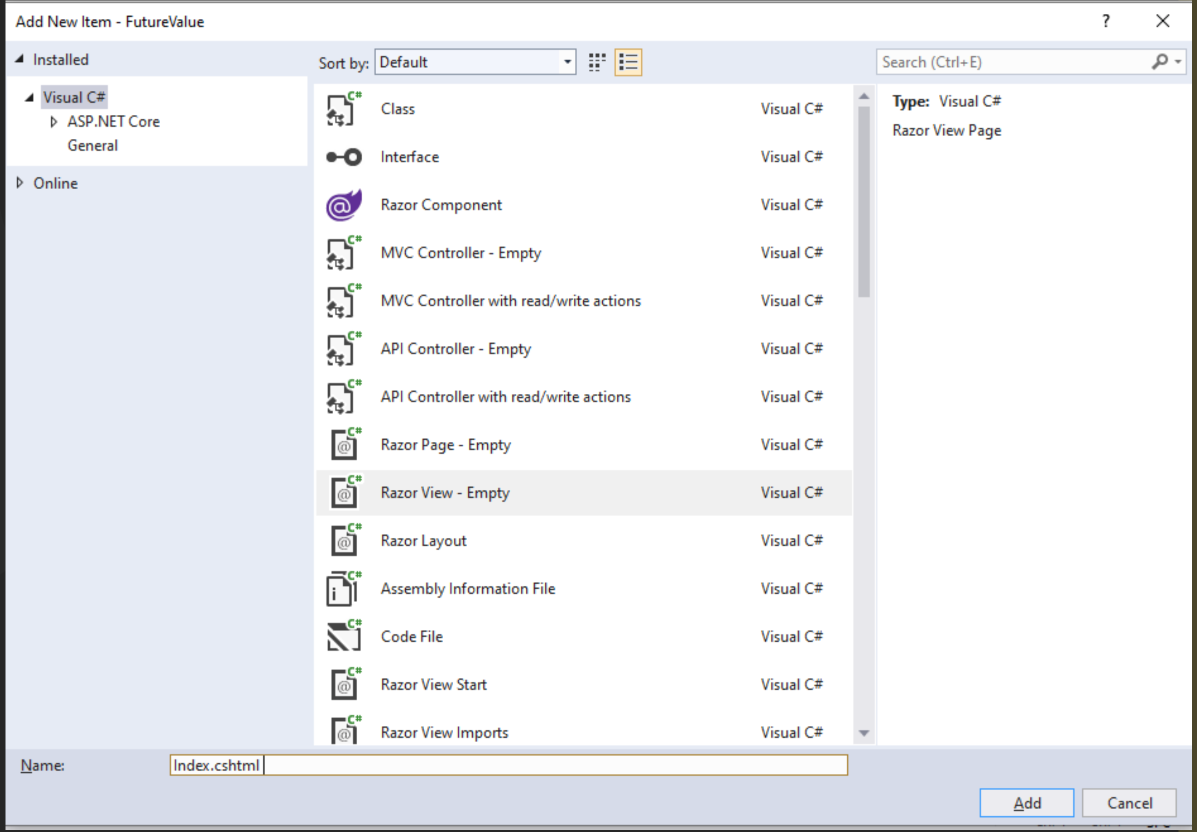
- Installed
  - Common
    - API
    - MVC
      - Controller
      - View
    - Razor Component
    - Razor Pages
  - Identity
  - Layout

Razor View - Empty
Razor View

Razor View - Empty
by Microsoft
v1.0.0.0
An empty Razor view
Id: RazorViewEmptyScaffolder

Click here to go online and find more scaffolding extensions.

Add | Cancel



Add New Item - FutureValue

- Installed
  - Visual C#
    - ASP.NET Core
    - General
- Online

Sort by: Default

Search (Ctrl+E)

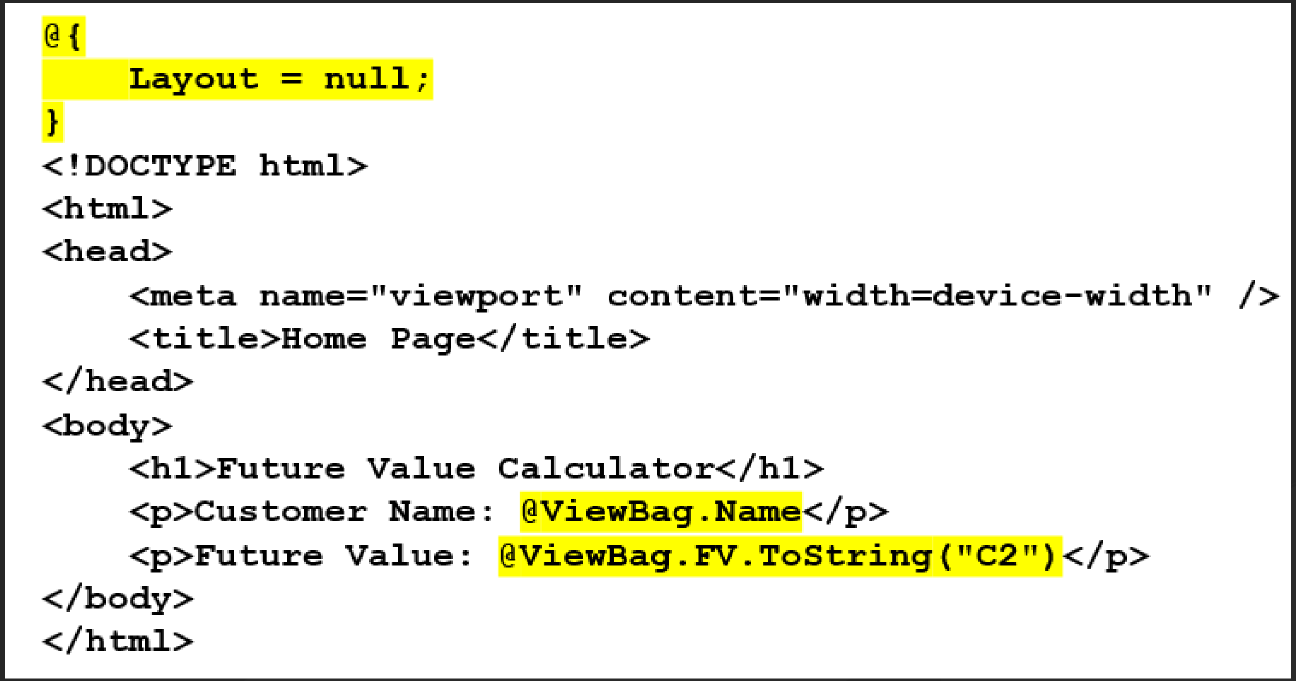
Class	Visual C#
Interface	Visual C#
Razor Component	Visual C#
MVC Controller - Empty	Visual C#
MVC Controller with read/write actions	Visual C#
API Controller - Empty	Visual C#
API Controller with read/write actions	Visual C#
Razor Page - Empty	Visual C#
Razor View - Empty	Visual C#
Razor Layout	Visual C#
Assembly Information File	Visual C#
Code File	Visual C#
Razor View Start	Visual C#
Razor View Imports	Visual C#

Type: Visual C#
Razor View Page

Name: Index.cshtml

Add | Cancel

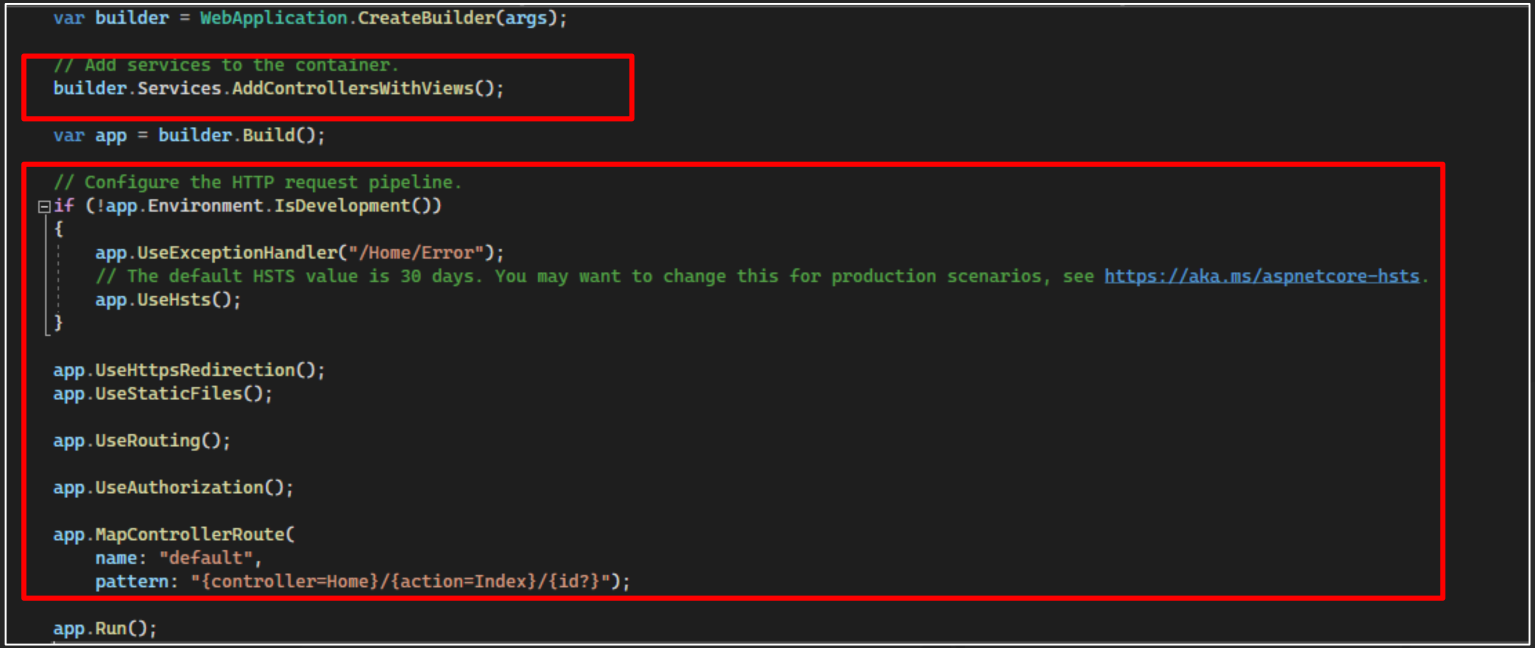
The Home/Index.cshtml view



```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Home Page</title>
</head>
<body>
    <h1>Future Value Calculator</h1>
    <p>Customer Name: @ViewBag.Name</p>
    <p>Future Value: @ViewBag.FV.ToString("C2")</p>
</body>
</html>
```

- A Razor view contains both C# and HTML. Thats why its extension is .cshtml.
- The Razor View Engine uses server-side code to embed C# code within HTML elements.
- To execute one or more C# statement, you declare a Razor code block by coding the @sign followed by a pair for braces { }
- To evaluate an expression and display a result you can code a Razor expression by coding the @ sign and then coding the expression.

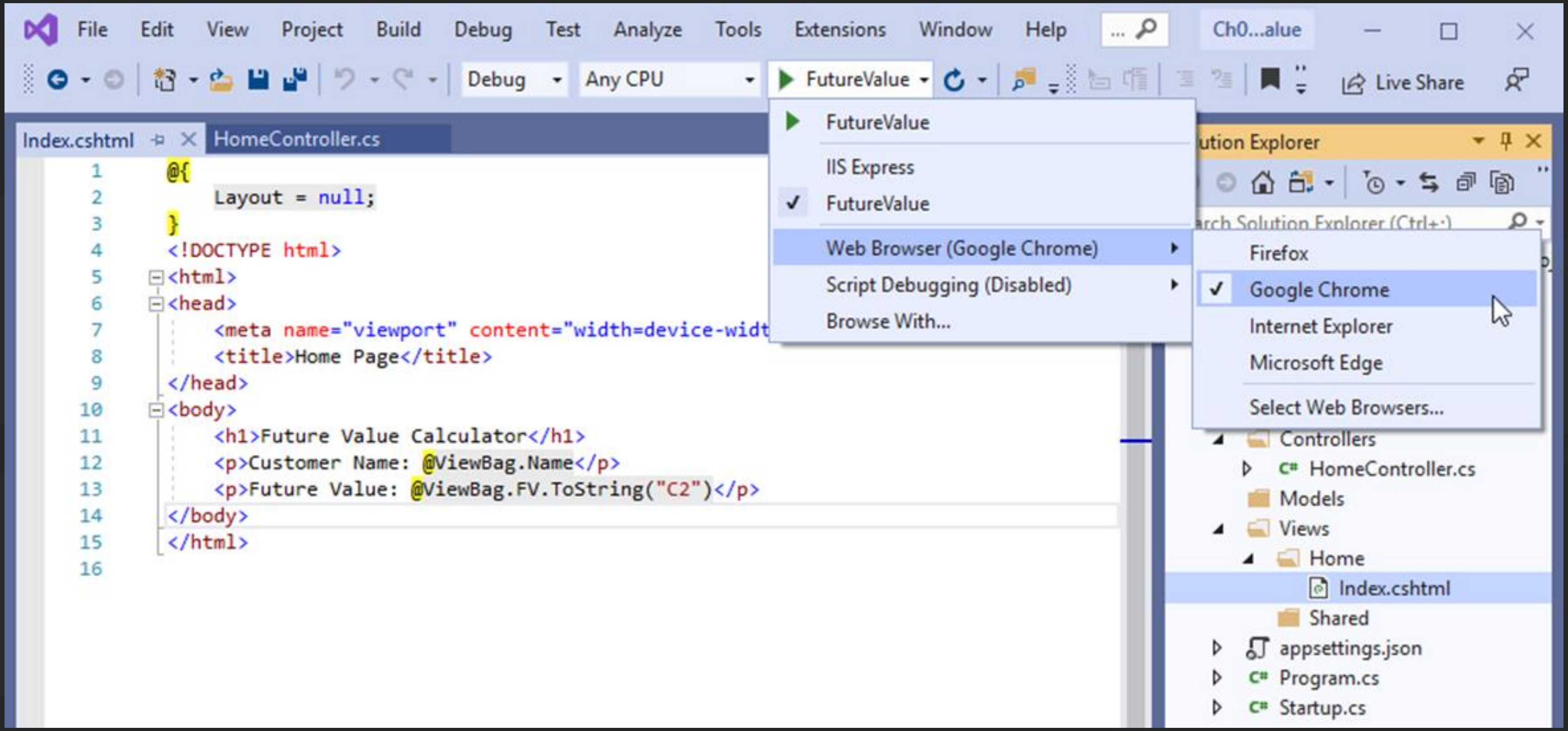
The Program.cs after it has been edited (page1)



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

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

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

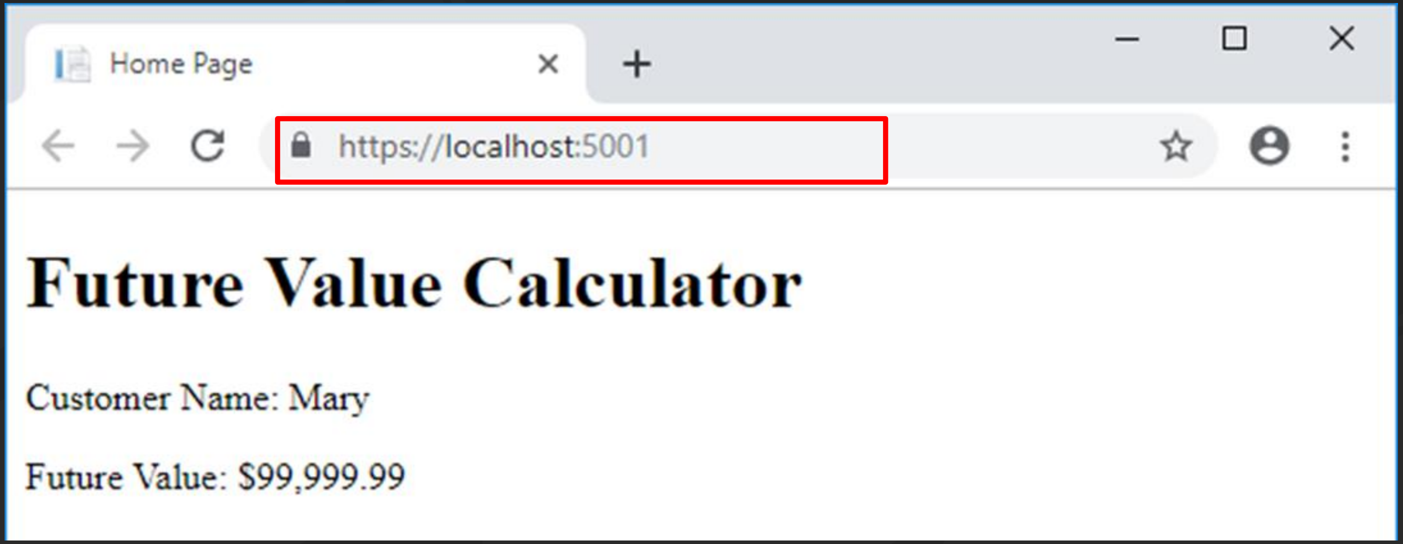
- The Program.cs file contains the code that configures the middleware for the HTTP request pipeline.
- In Program.cs we add services and configure the Applications pipeline.. Check the web hosting environment is a dev env. If so it configure the middleware for dev etc..
- The MapControllerRoute (in this example) sets the default controller to the HomeController, and sets default action to Index() action

Running and Application in Visual Studio



```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-widt
    <title>Home Page</title>
</head>
<body>
    <h1>Future Value Calculator</h1>
    <p>Customer Name: @ViewBag.Name</p>
    <p>Future Value: @ViewBag.FV.ToString("C2")</p>
</body>
</html>
```

Example: The Future Value Application

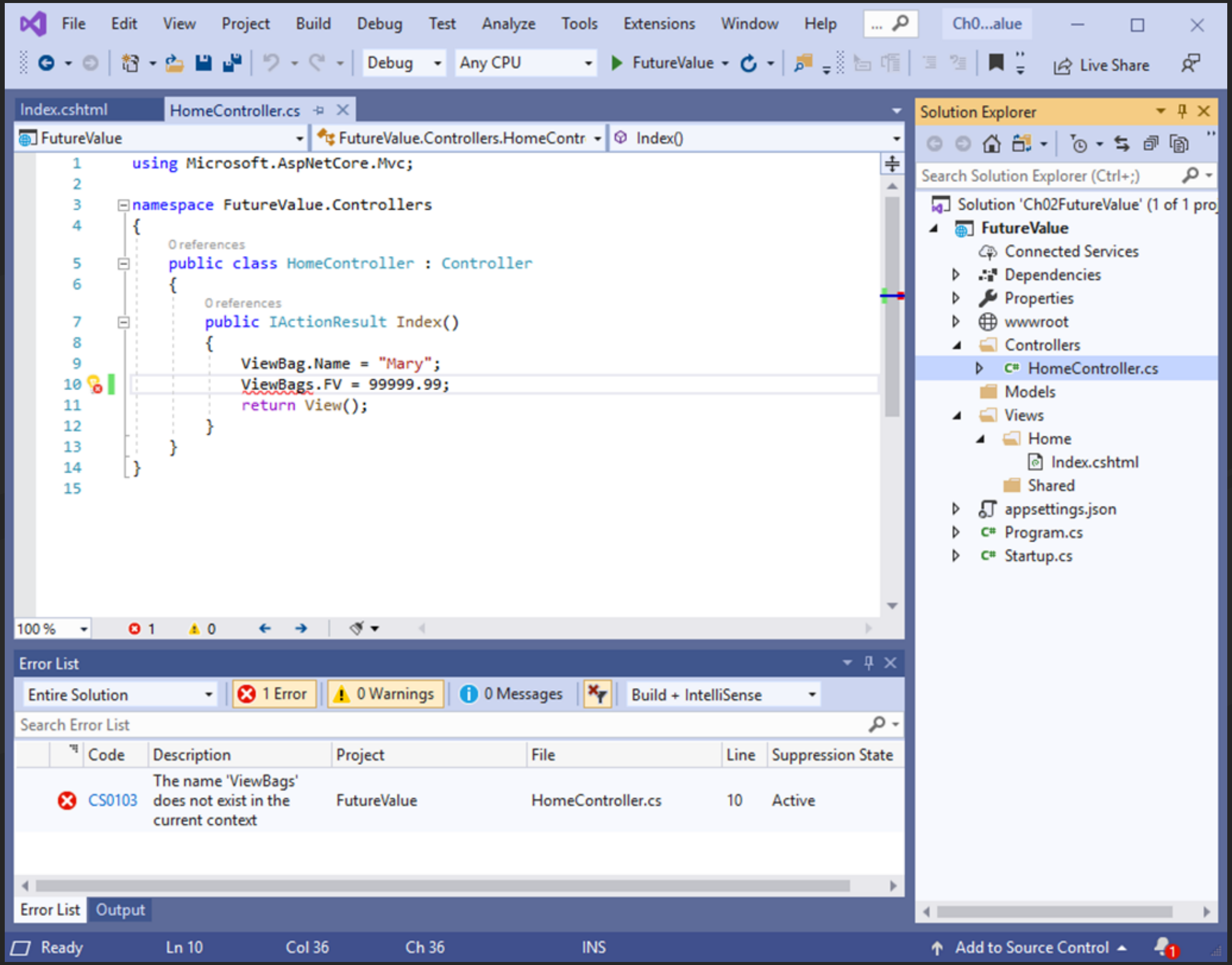


Future Value Calculator

Customer Name: Mary

Future Value: $99,999.99

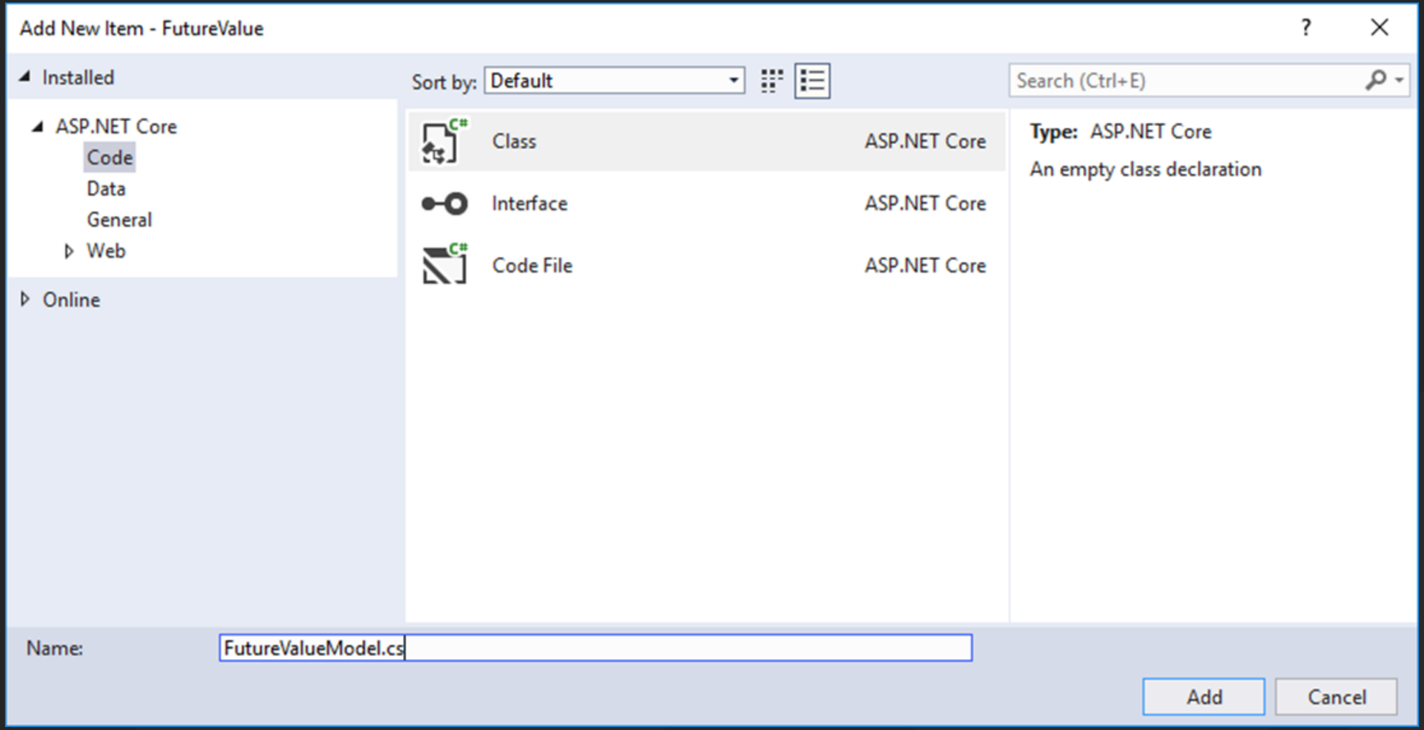
The Error List window in Visual Studio



```
using Microsoft.AspNetCore.Mvc;

namespace FutureValue.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public IActionResult Index()
        {
            ViewBag.Name = "Mary";
            ViewBags.FV = 99999.99;
            return View();
        }
    }
}
```

	Code	Description	Project	File	Line	Suppression State
⊗	CS0103	The name 'ViewBags' does not exist in the current context	FutureValue	HomeController.cs	10	Active

The dialog for adding a class



Class	ASP.NET Core
Interface	ASP.NET Core
Code File	ASP.NET Core

Type: ASP.NET Core

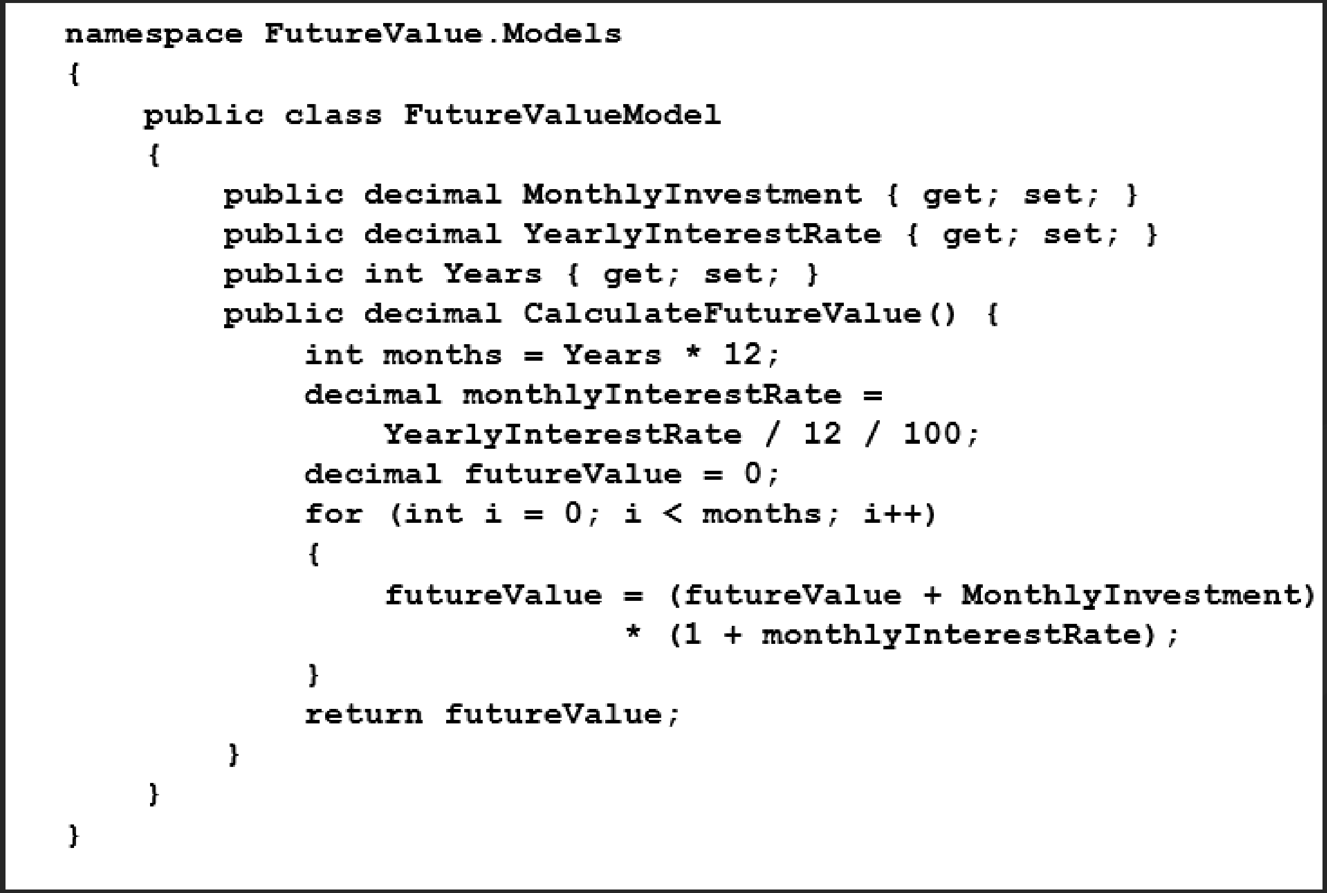
An empty class declaration

Name: FutureValueModel.cs

How to add a file for a model class

1. In the Solution Explorer, right-click the Models folder and select Add→Class.
2. In the resulting dialog, enter the name of the class, and click the Add button.

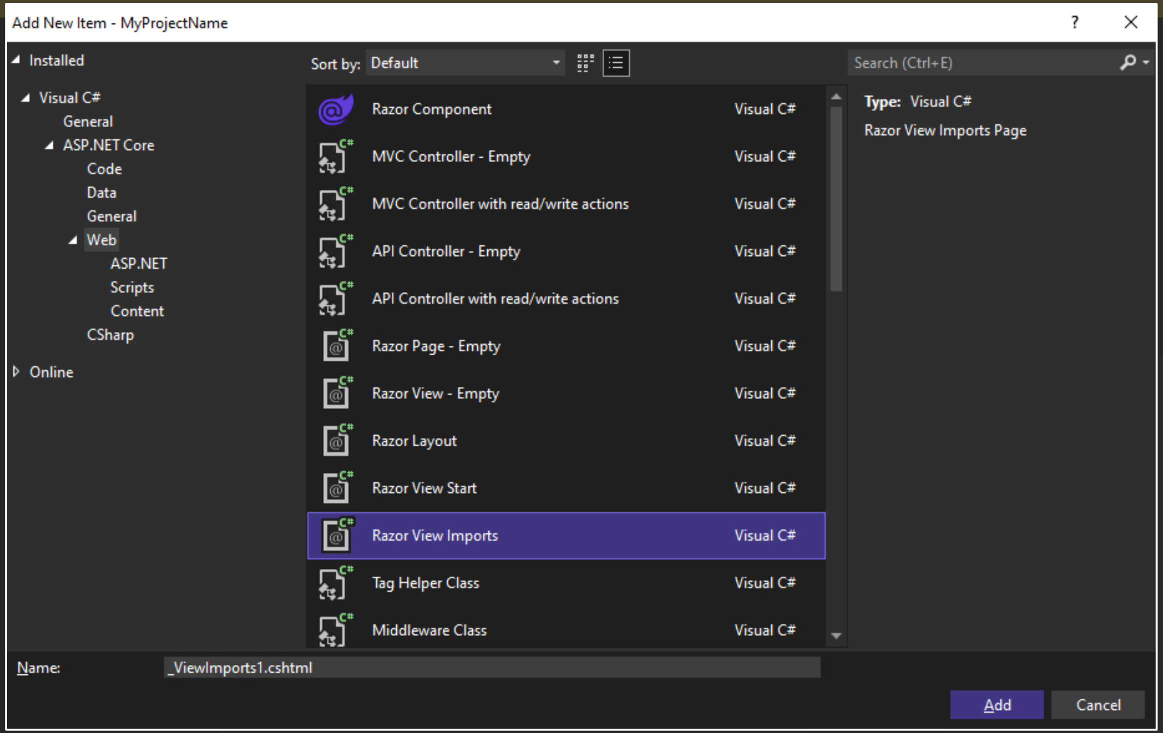
The FutureValueModel class



```
namespace FutureValue.Models
{
    public class FutureValueModel
    {
        public decimal MonthlyInvestment { get; set; }
        public decimal YearlyInterestRate { get; set; }
        public int Years { get; set; }
        public decimal CalculateFutureValue() {
            int months = Years * 12;
            decimal monthlyInterestRate =
                YearlyInterestRate / 12 / 100;
            decimal futureValue = 0;
            for (int i = 0; i < months; i++)
            {
                futureValue = (futureValue + MonthlyInvestment)
                            * (1 + monthlyInterestRate);
            }
            return futureValue;
        }
    }
}
```

- A model is a regular C# class that models the data for the application. The class for a model is typically stored in the Models folder
- A model can’t have the same name as the namespace

The dialog for adding a Razor View imports page


Add New Item - MyProjectName

Installed
- Visual C#
  - General
  - ASP.NET Core
    - Code
    - Data
    - General
    - Web
      - ASP.NET
      - Scripts
      - Content
    - CSharp
- Online

Sort by: Default

Razor Component	Visual C#
MVC Controller - Empty	Visual C#
MVC Controller with read/write actions	Visual C#
API Controller - Empty	Visual C#
API Controller with read/write actions	Visual C#
Razor Page - Empty	Visual C#
Razor View - Empty	Visual C#
Razor Layout	Visual C#
Razor View Start	Visual C#
Razor View Imports	Visual C#
Tag Helper Class	Visual C#
Middleware Class	Visual C#

Search (Ctrl+E)

Type: Visual C#
Razor View Imports Page

Name: _ViewImports1.cshtml

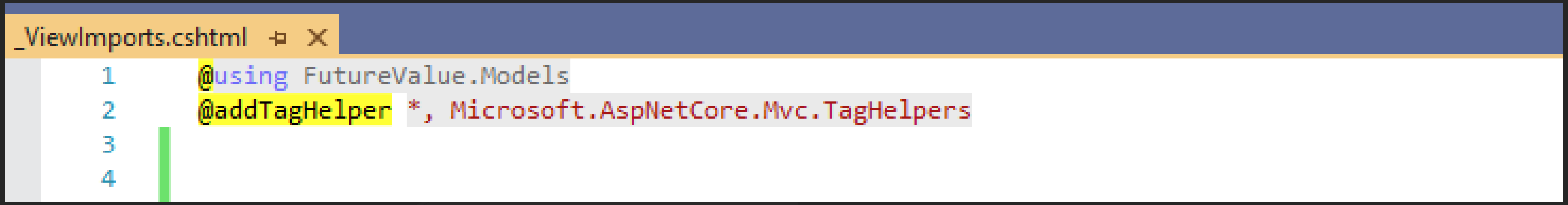
Add Cancel

- Most applications include a Razor view imports page.
- The Razor view imports page make it easier to work with model classes and tag helpers that are available from ASP.Net core MVC

How to add a Razor view imports page

1. In the Solution Explorer, right-click the Views folder and select Add→New Item.
2. In the resulting dialog, select the Installed→ASP.NET Core→Web category, select the Razor View Imports item, and click the Add button.

Example: The Views/_ViewImport.cshtml


_ViewImports.cshtml

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


```

A Razor view imports page makes it easier to work with…

- Model classes.
- Tag helpers.

Common tag helpers for forms

Tag helper	HTML tags
• asp-for	<label> <input>
• asp-action	<form> <a>
• asp-controller	<form> <a>

Tag Helper	HTML tags	Description
asp-for	<label><input>	Binds the HTML element to the specified model property
asp-action	<form><a>	Specifies the action for the URL. If no controller is specified , MVC uses the current controller
asp-controller	<form><a>	Specifies the controller for the URL

A strongly-typed Index view with tag helpers (part 1)

```
@model FutureValueModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Future Value Calculator</title>
</head>

<body>
    <h1>Future Value Calculator</h1>
    <form asp-action="Index" method="post">
        <div>
           <label asp-for="MonthlyInvestment">
               Monthly Investment:</label>
           <input asp-for="MonthlyInvestment" />
        </div>
```

- You can use @model directive to bind the model to the view. This kind of view is strongly-typed
- The ASP.NET Core MVC tag helpers are used to automatically generate attributes for some HTML elements.
- They are also used to bind HTML elements to the properties of the object that's the model for the view.

A strongly-typed Index view with tag helpers (part 2)

```
        <div>
           <label asp-for="YearlyInterestRate">
               Yearly Interest Rate:</label>
           <input asp-for="YearlyInterestRate" />
        </div>
        <div>
           <label asp-for="Years">Number of Years:</label>
           <input asp-for="Years" />
        </div>
        <div>
           <label>Future Value:</label>
           <input value="@ViewBag.FV.ToString("C2")" readonly>
        </div>
        <button type="submit">Calculate</button>
        <a asp-action="Index">Clear</a>
    </form>
</body>
</html>
```

Attributes that indicate the HTTP Verb an action method handles

```
HttpGet
HttpPost
```

Methods for returning a view from a controller

```
View()
View(model)
```

Attribute	**Description**
HttpGet	Specifies that the action method handles a **GET** request (**Default**).
HttpPost	Specifies that the action method handles a **POST** request.

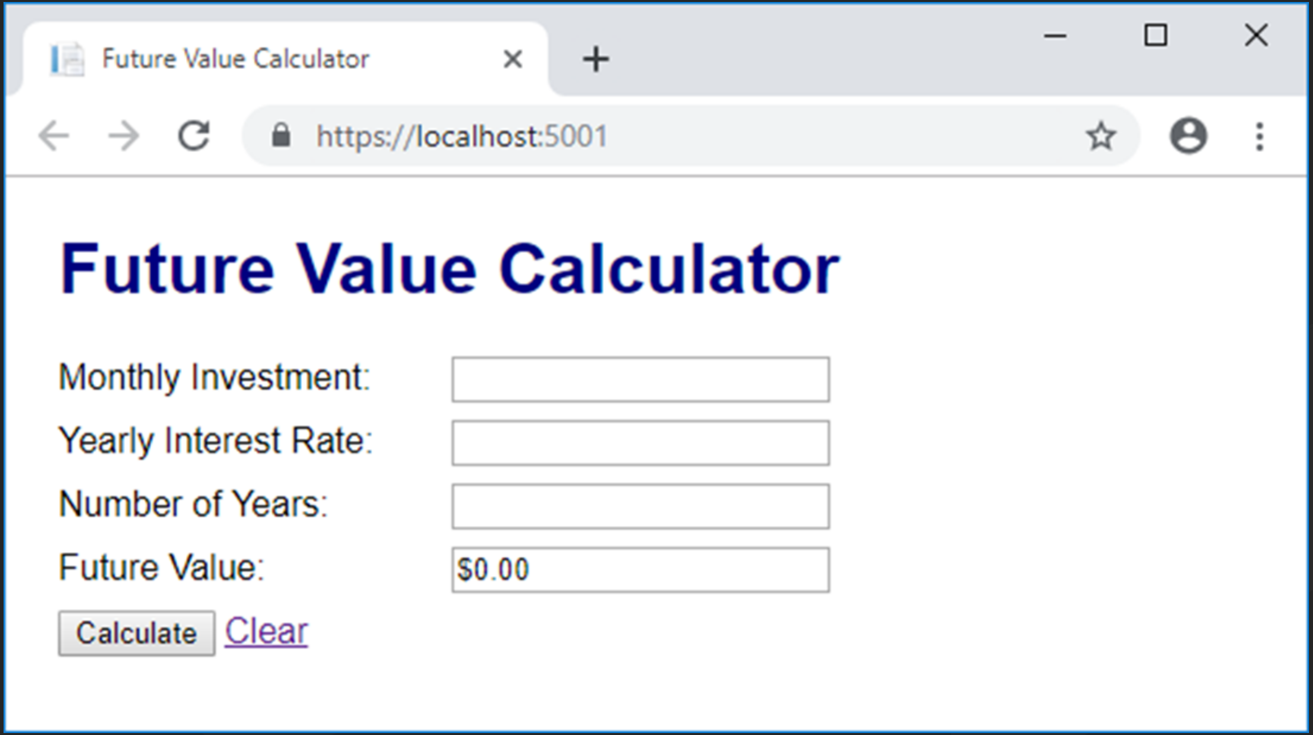
method	**Description**
View()	Returns the view that corresponds to the ***current*** controller and action.
View(model)	Passes the specified model to the view that corresponds to the current controller and action so the view can bind to the model

An overloaded Index() action method

```
using Microsoft.AspNetCore.Mvc;
using FutureValue.Models;

public class HomeController : Controller
{
    [HttpGet]
    public IActionResult Index()
    {
        ViewBag.FV = 0;
        return View();
    }

    [HttpPost]
    public IActionResult Index(FutureValueModel model)
    {
        ViewBag.FV = model.CalculateFutureValue();
        return View(model);
    }
}
```

Future Value Application

Application After GET Request

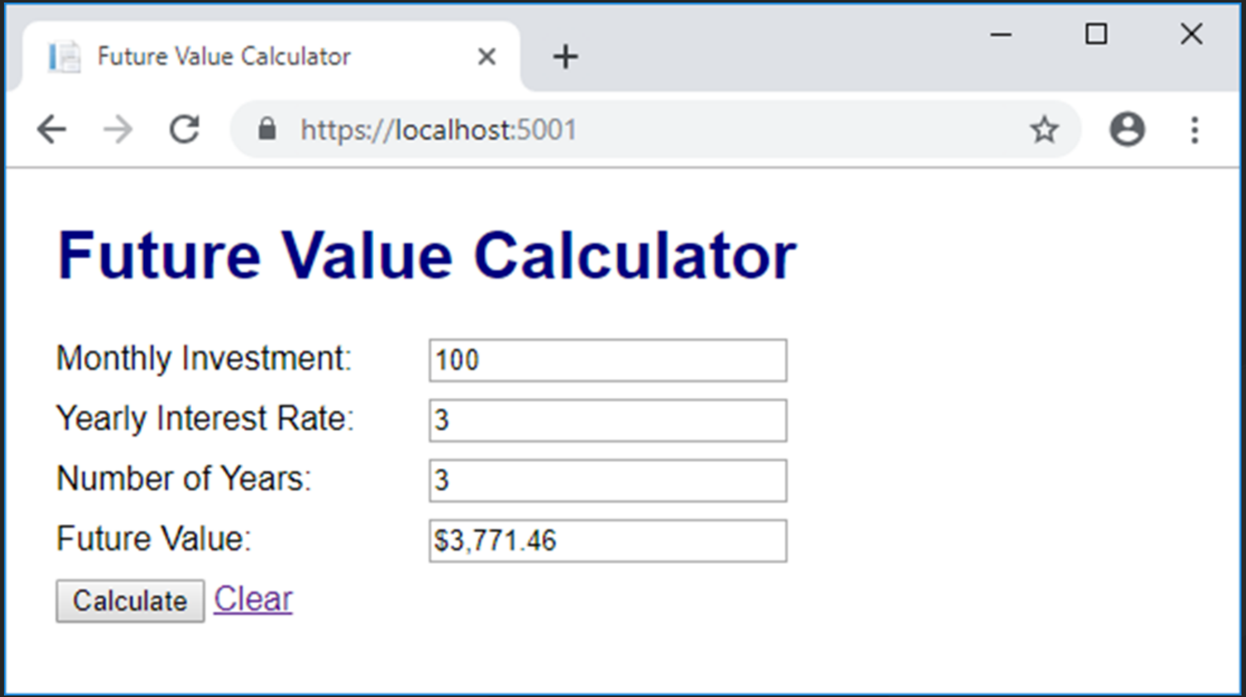


Future Value Calculator

Monthly Investment:	
Yearly Interest Rate:	
Number of Years:	
Future Value:	$0.00

- When the future value application starts, it sends a GET request to the index action of the Home Controller
- When the user click the clear link, the application sends a Get request to the Index() action of the HomeController

Application After POST Request


Future Value Calculator

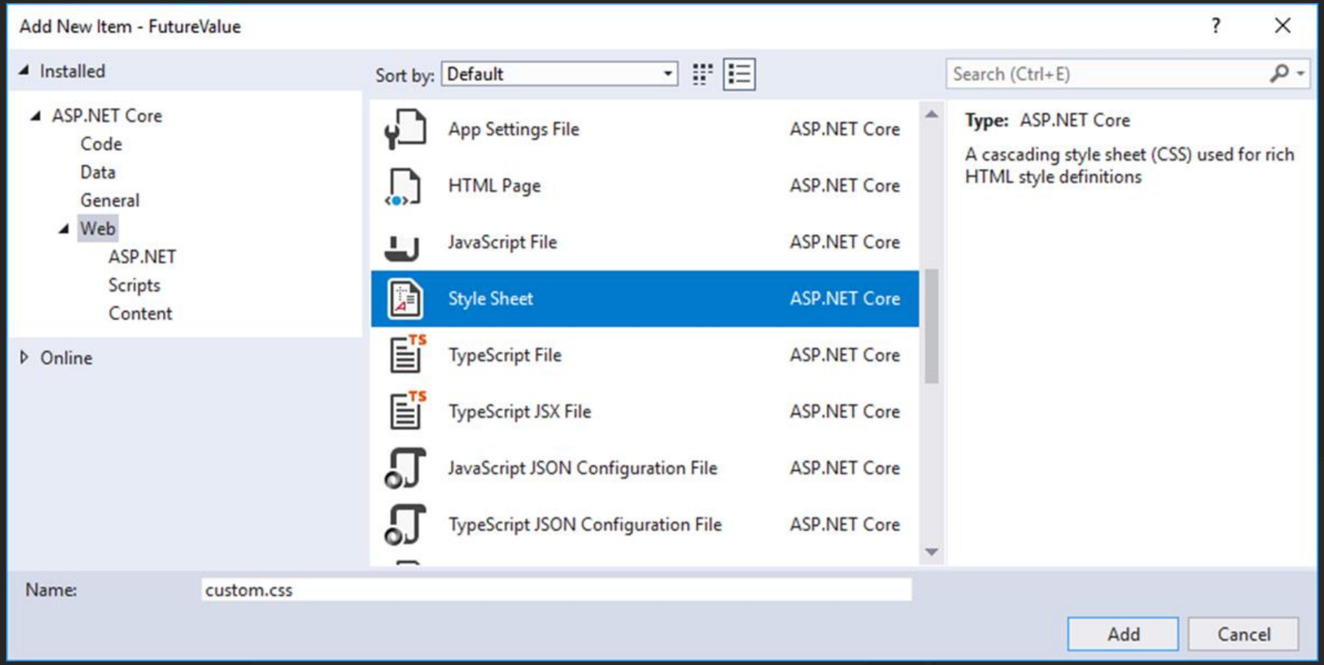
https://localhost:5001

Future Value Calculator

Monthly Investment:	100
Yearly Interest Rate:	3
Number of Years:	3
Future Value:	$3,771.46

- When the user clicks the calculate button, the application sends a POST request to the Index() action of the HomeController.
- If the user has filled out the form correctly this automatically set the three properties if the model object.

The dialog for adding a CSS style sheet


Add New Item - FutureValue
Installed
ASP.NET Core
Code
Data
General
Web
ASP.NET
Scripts
Content
Online
Sort by: Default
Search (Ctrl+E)
App Settings File ASP.NET Core
HTML Page ASP.NET Core
JavaScript File ASP.NET Core
Style Sheet ASP.NET Core
TypeScript File ASP.NET Core
TypeScript JSX File ASP.NET Core
JavaScript JSON Configuration File ASP.NET Core
TypeScript JSON Configuration File ASP.NET Core
Add
Cancel

Type: ASP.NET Core

A cascading style sheet (CSS) used for rich HTML style definitions

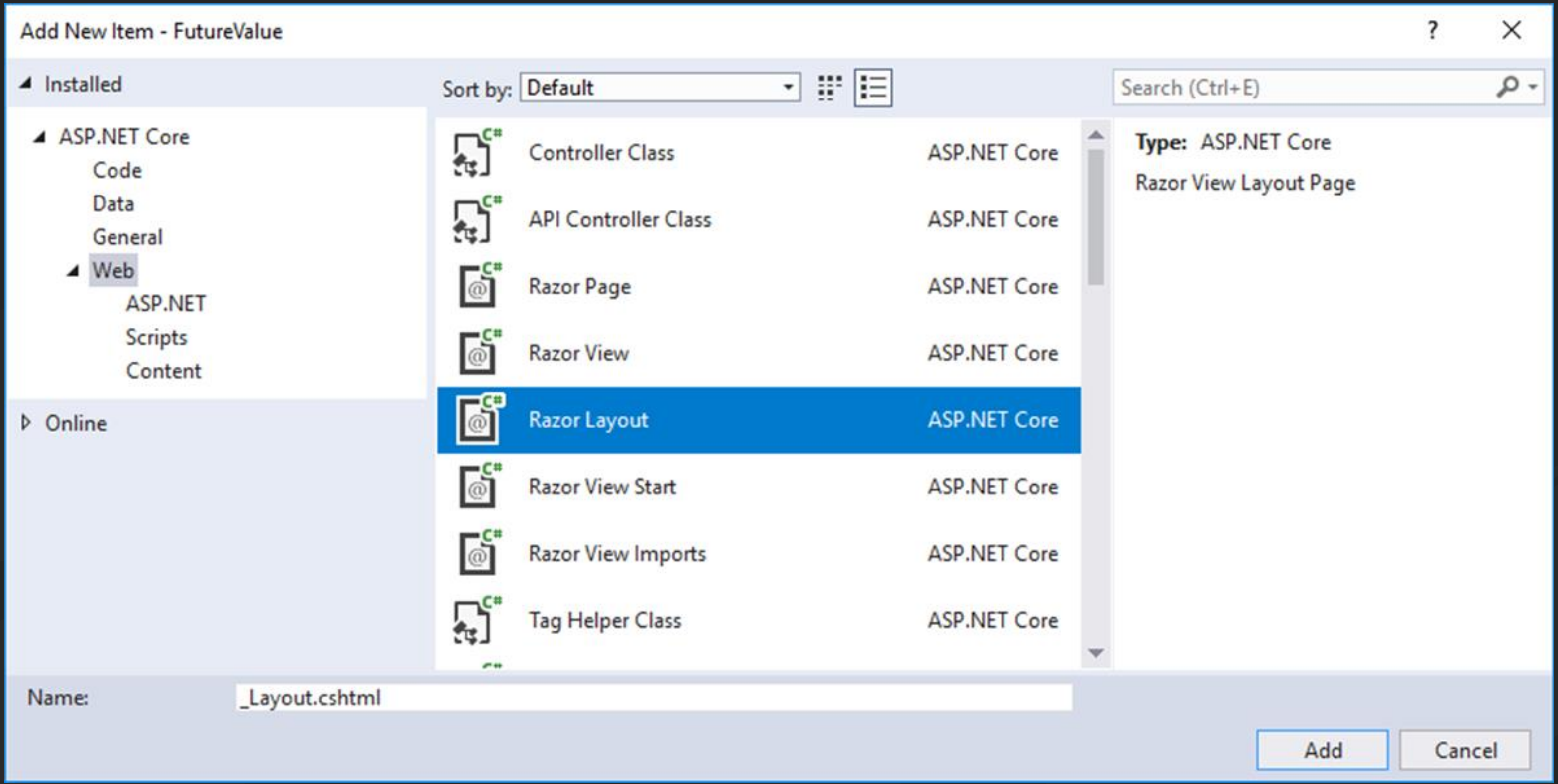
Name: custom.css

1. If the wwwroot/css folder doesn't exist, create it.
2. Right-click the wwwroot/css folder and select Add→New Item.
3. Select the ASP.NET Core→Web category, select the Style Sheet item, enter a name for the CSS file, and click the Add button.

Example css file (custom.css)

```css
body {
    padding: 1em;
    font-family: Arial, Helvetica, sans-serif;
}

h1 {
    margin-top: 0;
    color: navy;
}

label {
    display: inline-block;
    width: 10em;
    padding-right: 1em;
}

div {
    margin-bottom: .5em;
}
```

The dialog for adding a Razor layout, view start, or view


Add New Item - FutureValue

Installed

- ASP.NET Core
  - Code
  - Data
  - General
  - Web
    - ASP.NET
    - Scripts
    - Content
- Online

Sort by: Default

Search (Ctrl+E)

Controller Class	ASP.NET Core
API Controller Class	ASP.NET Core
Razor Page	ASP.NET Core
Razor View	ASP.NET Core
Razor Layout	ASP.NET Core
Razor View Start	ASP.NET Core
Razor View Imports	ASP.NET Core
Tag Helper Class	ASP.NET Core

Type: ASP.NET Core

Razor View Layout Page

Name: _Layout.cshtml

How to add a Razor layout

1. Right-click the Views/Shared folder and select Add→New Item.
2. Select the ASP.NET Core→Web category, select the Razor Layout item, and click the Add button.

The dialog for adding a Razor layout, view start, or view…

How to add a Razor layout

1. Right-click the **Views/Shared folder** and select **Add→New Item**.
2. Select the **ASP.NET Core→Web category**, select the **Razor Layout item**, and click the Add button.

How to add a Razor view start

1. Right-click the Views folder (not the Views/Shared folder) and select Add→New Item.
2. Select the ASP.NET Core→Web category, select the Razor View Start item, and click the Add button.

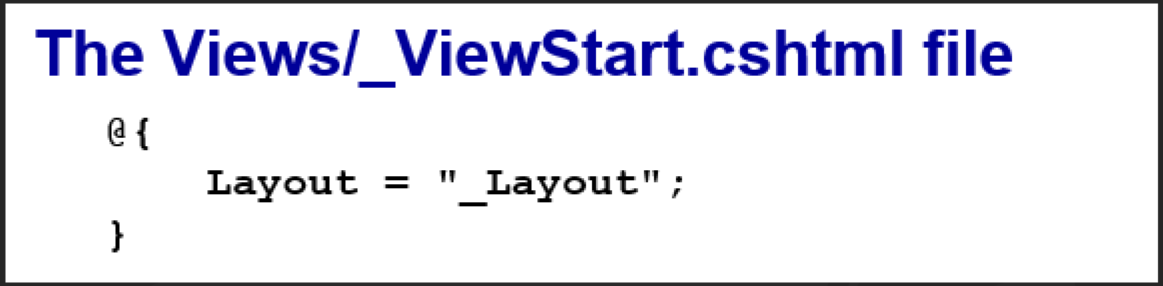
How to add a Razor view

1. Right-click the folder for the view (Views/Home, for example) and select Add→View.
2. Use the dialog from figure 2-5 to specify the name for the view.
3. If you're using a layout that has a view start, select the "Use a layout page" item but don't specify a name for the layout page.

The Views/Shared/_Layout.cshtml

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    <link rel="stylesheet" href="~/css/custom.css" />
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

The Views/_ViewStart.cshtml


The Views/_ViewStart.cshtml file

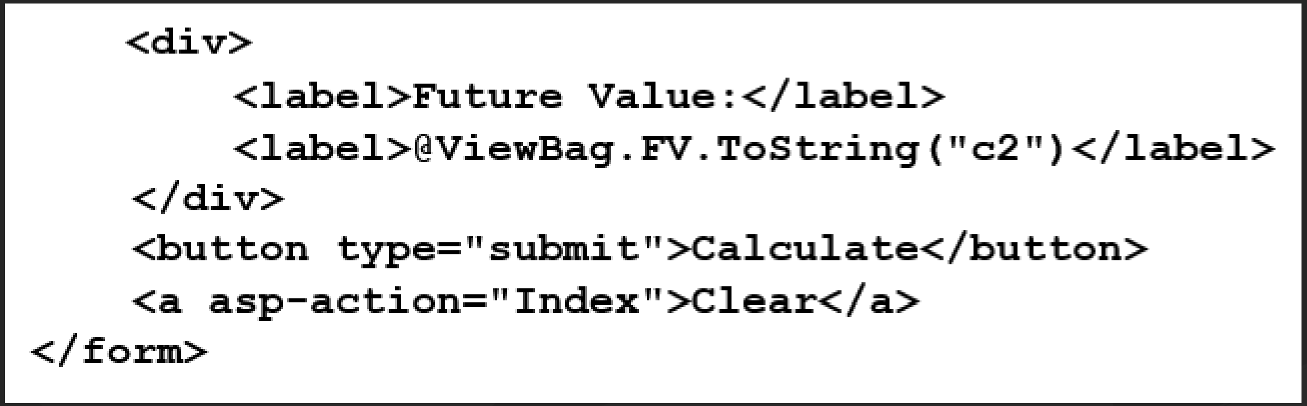
```
@{
    Layout = "_Layout";
}
```

- You can use the Razor _ViewStart toi set the default layout for all the view in your application.
- However, yu can also use the layout property of a view to override the default value.

The View/Home/Index.cshtml file (part 1)

```
@model FutureValueModel
@{
    ViewBag.Title = "Future Value Calculator";
}
<h1>Future Value Calculator</h1>
<form asp-action="Index" method="post">
    <div>
        <label asp-for="MonthlyInvestment">
            Monthly Investment:</label>
        <input asp-for="MonthlyInvestment" />
    </div>
    <div>
        <label asp-for="YearlyInterestRate">
            Yearly Interest Rate:</label>
        <input asp-for="YearlyInterestRate" />
    </div>
    <div>
        <label asp-for="Years">Number of Years:</label>
        <input asp-for="Years" />
    </div>
```

The View/Home/Index.cshtml file (part 2)



```
    <div>
        <label>Future Value:</label>
        <label>@ViewBag.FV.ToString("c2")</label>
    </div>
    <button type="submit">Calculate</button>
    <a asp-action="Index">Clear</a>
</form>
```

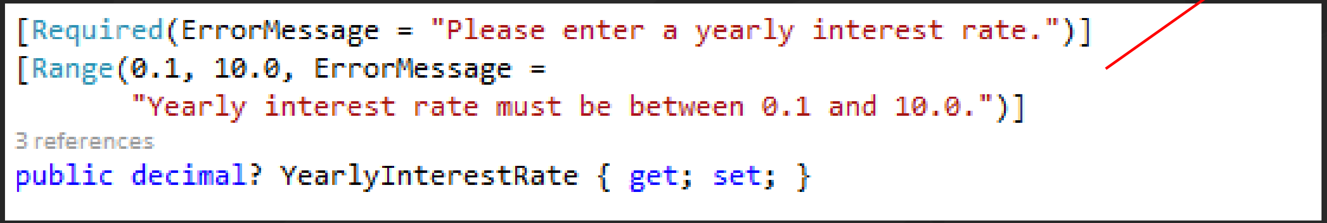
DataAnnotations

How to import the DataAnnotations namespace

```
using System.ComponentModel.DataAnnotations;
```

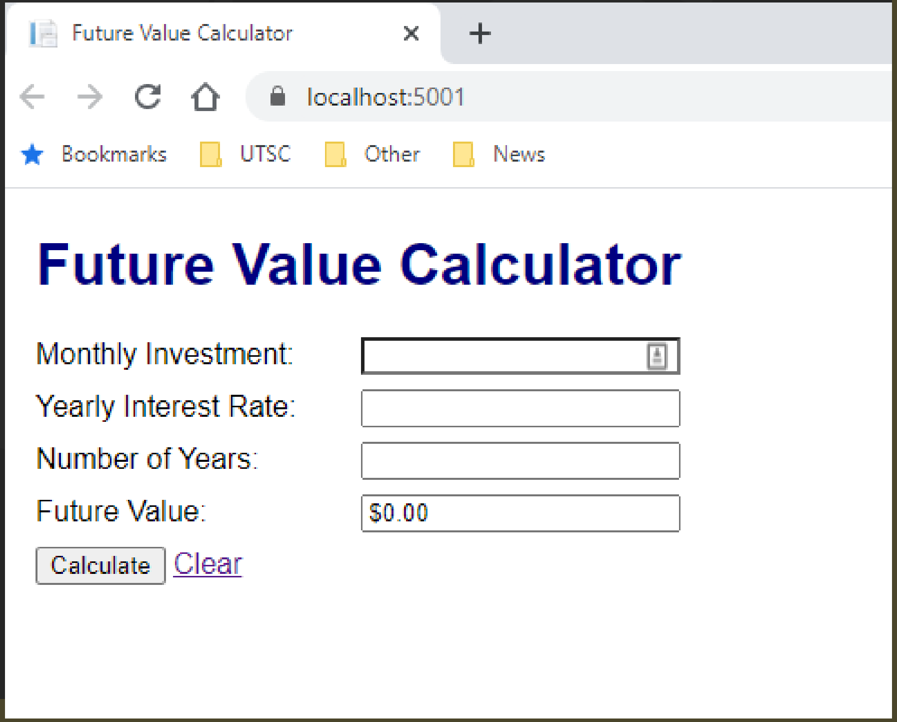
Two common validation attributes

```
Required
Range(min, max)
```



```
[Required(ErrorMessage = "Please enter a yearly interest rate.")]
[Range(0.1, 10.0, ErrorMessage =
      "Yearly interest rate must be between 0.1 and 10.0.")]
3 references
public decimal? YearlyInterestRate { get; set; }
```

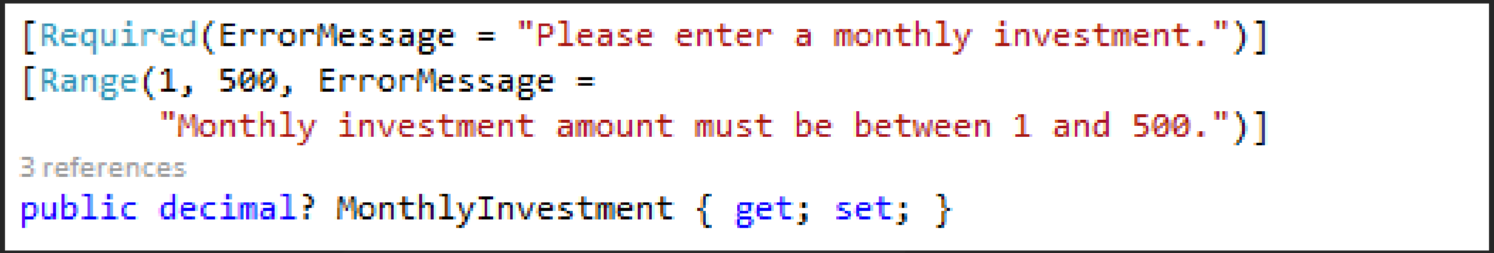
Data Validation



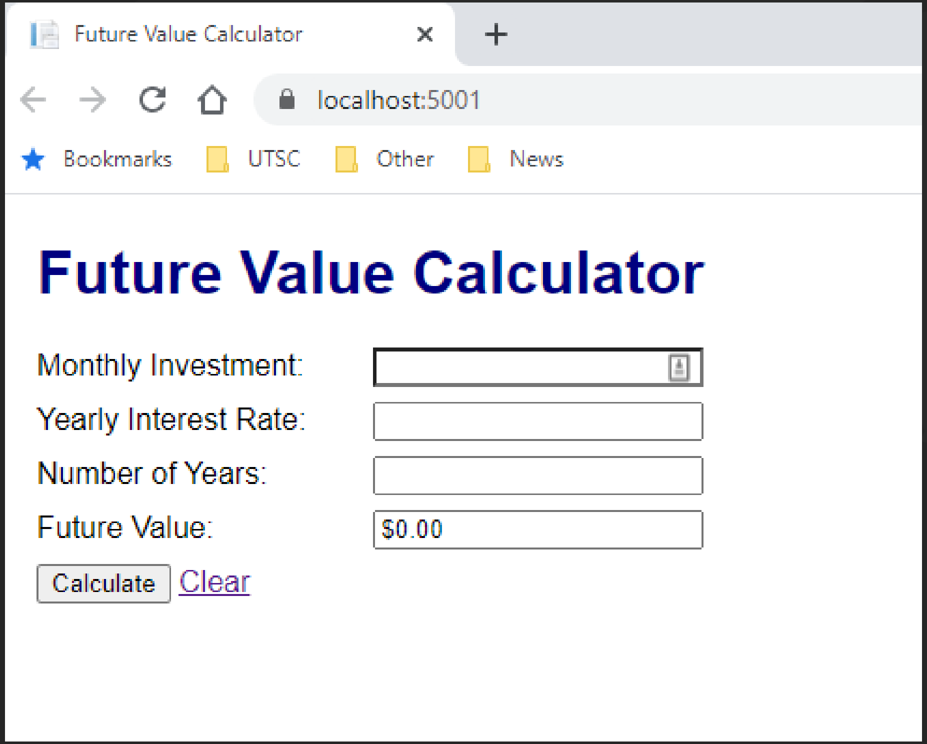
Future Value Calculator

Monthly Investment:	
Yearly Interest Rate:	
Number of Years:	
Future Value:	$0.00

Model Property with two validation attributes



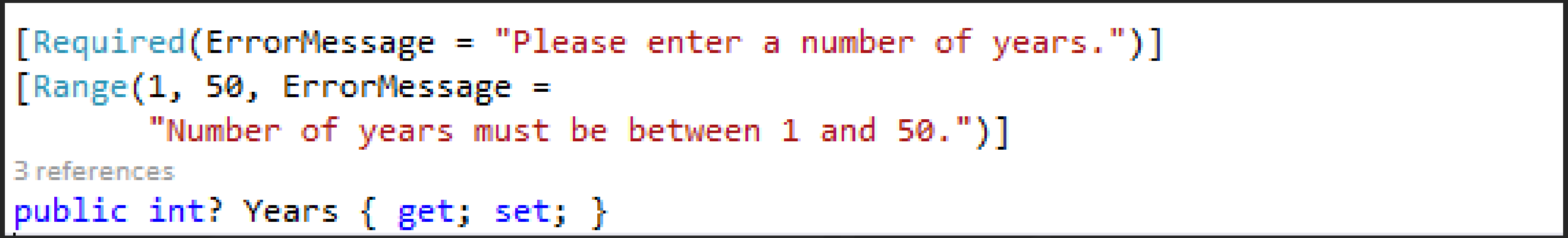
```
[Required(ErrorMessage = "Please enter a monthly investment.")]
[Range(1, 500, ErrorMessage =
       "Monthly investment amount must be between 1 and 500.")]
3 references
public decimal? MonthlyInvestment { get; set; }
```



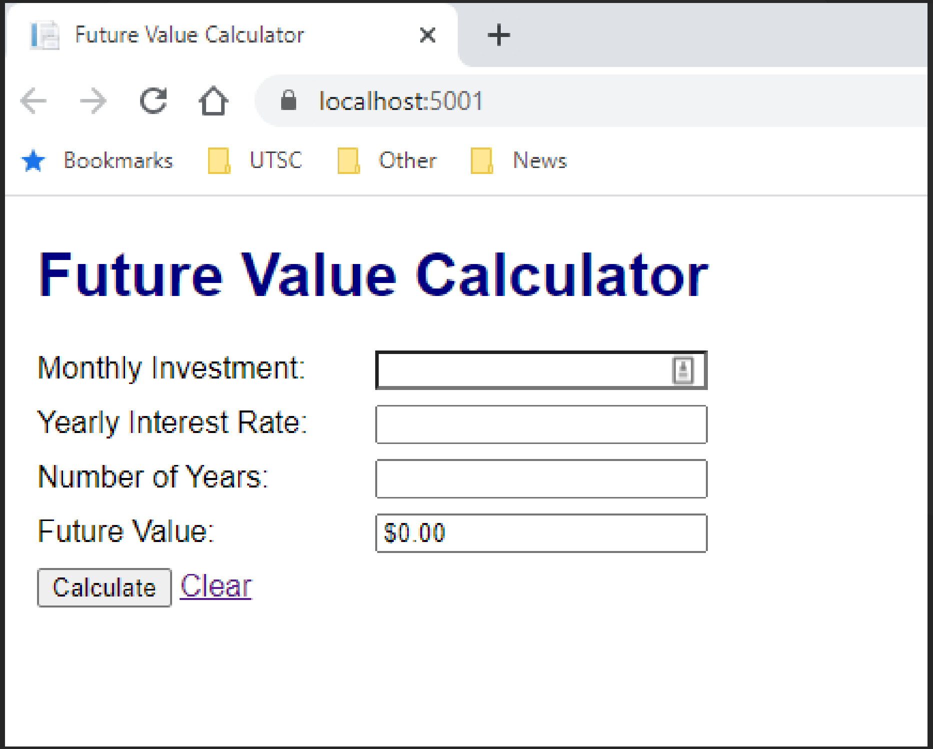
Future Value Calculator

Monthly Investment:	
Yearly Interest Rate:	
Number of Years:	
Future Value:	$0.00

Model Property with a user-friendly error message



```
[Required(ErrorMessage = "Please enter a number of years.")]
[Range(1, 50, ErrorMessage =
       "Number of years must be between 1 and 50.")]
3 references
public int? Years { get; set; }
```



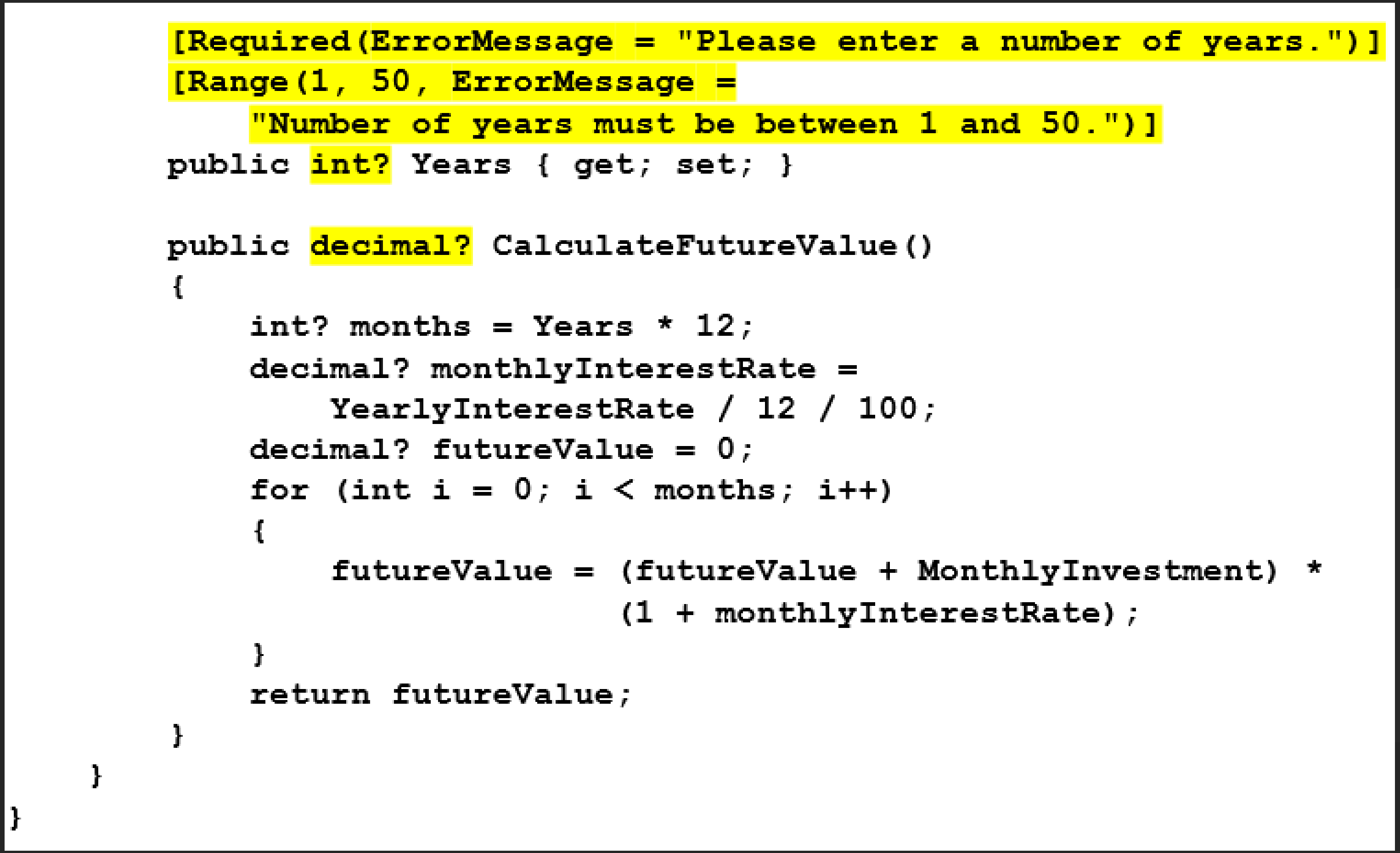
Future Value Calculator

Monthly Investment:	
Yearly Interest Rate:	
Number of Years:	
Future Value:	$0.00

Example Model Class with data validation attributes (part 1)

```
using System.ComponentModel.DataAnnotations;

namespace FutureValue.Models
{
    public class FutureValueModel
    {
        [Required(ErrorMessage =
            "Please enter a monthly investment.")]
        [Range(1, 500, ErrorMessage =
            "Monthly investment amount must be between 1 and 500.")]
        public decimal? MonthlyInvestment { get; set; }

        [Required(ErrorMessage =
            "Please enter a yearly interest rate.")]
        [Range(0.1, 10.0, ErrorMessage =
            "Yearly interest rate must be between 0.1 and 10.0.")]
        public decimal? YearlyInterestRate { get; set; }
```

Example Model Class with data validation attributes (part 2)



```
        [Required(ErrorMessage = "Please enter a number of years.")]
        [Range(1, 50, ErrorMessage =
            "Number of years must be between 1 and 50.")]
        public int? Years { get; set; }

        public decimal? CalculateFutureValue()
        {
            int? months = Years * 12;
            decimal? monthlyInterestRate =
                YearlyInterestRate / 12 / 100;
            decimal? futureValue = 0;
            for (int i = 0; i < months; i++)
            {
                futureValue = (futureValue + MonthlyInvestment) *
                              (1 + monthlyInterestRate);
            }
            return futureValue;
        }
    }
}
```

An action method that checks for invalid data

```
[HttpPost]
public IActionResult Index(FutureValueModel model)
{
    if (ModelState.IsValid)
    {
        ViewBag.FV = model.CalculateFutureValue();
    }
    else
    {
        ViewBag.FV = 0;
    }
    return View(model);
}
```

A view that displays a summary of validation messages

```
<form asp-action="Index" method="post">
    <div asp-validation-summary="All"></div>
    <div>
        <label asp-for="MonthlyInvestment">
            Monthly Investment:</label>
        <input asp-for="MonthlyInvestment" />
    </div>
    <!-- rest of input form -->
</form>
```

Questions?