ASP.NET CORE MVC MODULE 05
DEVELOPING VIEWS

Summer 2021 – Web Development using ASP .Net Core MVC

At this point in the course, you may not yet have a complete understanding of the **models**.
In a complete MVC application, **controllers**, **views**, and **models** are tightly integrated.
Models are covered in Module 6, so please be patient! We'll get there next time.

MAIN SOURCES FOR THESE SLIDED

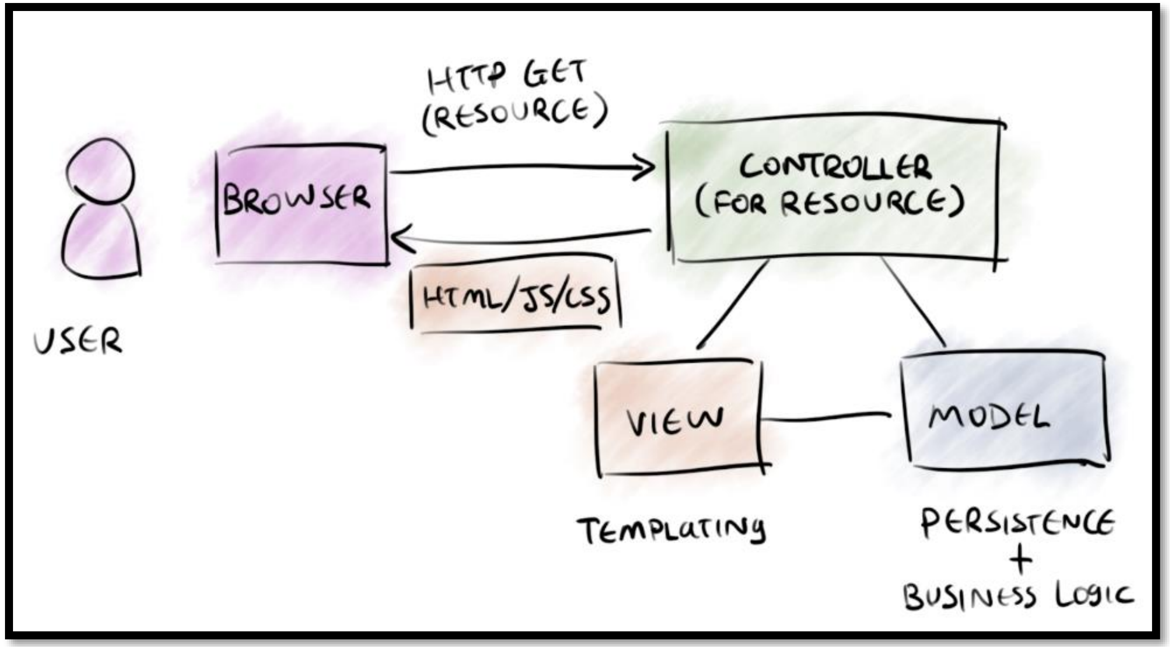
- Unless otherwise specified, the main sources for these slides are:
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications ←for homework
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0 ←for “textbook”

RECOMMENDED READING AHEAD

- On your own, you may want to check out the following:
  - Page 1621: Get started with ASP.NET Core MVC
  - Page 1630: Part 2, add a controller to an ASP.NET Core MVC app
  - Page 1642: Part 3, add a view to an ASP.NET Core MVC app
  - Page 1658: Part 4, add a model to an ASP.NET Core MVC app
  - Page 1691: Part 5, work with a database in an ASP.NET Core MVC app
  - Page 1704: Part 6, controller methods and views in ASP.NET Core
  - Page 1718: Part 7, add search to an ASP.NET Core MVC app
  - Page 1729: Part 8, add a new field to an ASP.NET Core MVC app (… see migrations)
  - Page 1734: Part 9, add validation to an ASP.NET Core MVC app

SOME ESSENTIAL MVC CONCEPTS – REVISITED

- Here is a great picture to help you visualize the MVC model



Naming is important: MVC relies on **convention** over **configuration**

- a **request** is sent from a **web browser**.
  - for example: **http://www.mysite.com/student/show/1**

- a **controller** object is instantiated to respond to this **request**.
  - for the above, the **StudentController** will be instantiated
  - the **URL routing** determines which **controller** & **action** will handler the request

- an **action** method is then called by the **controller**.
  - for the above, a **Show** action will be selected
  - a **model binder** determines the values passed to the **action** as **parameters** (e.g. **1**).
  - the **action** may create a new instance of a **model** class.
  - this **model** object may be passed to a **view** to display results.

- a **view** will produce the output that is sent back to the **browser**.
  - The output could be HTML+CSS+JS file (most often), or JSON, XML, text, files, …

- Side note: where does middleware fit into this diagram?

THE MVC ARCHITECTURAL PATTERN

- See page 1630

- Models:
  - classes representing **the data** of an application.
  - often, model objects will retrieve/store data from/in a **database**.

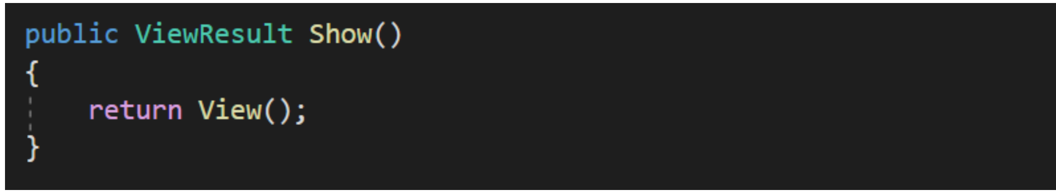
- Views:
  - are components that **display the user interface**(UI).
  - typically, they display the **model** data.

- Controllers:
  - classes that **handle requests** from browsers.
  - may retrieve **model** data.
  - often call **view** templates to send back a response to the browser requests.

- Notes:
  - Typically, we have one **controller** class for each **model** class. (Student ← model, StudentController ← controller)
  - Each **controller** can have **multiple** views (often, each **action** has its own **view**)

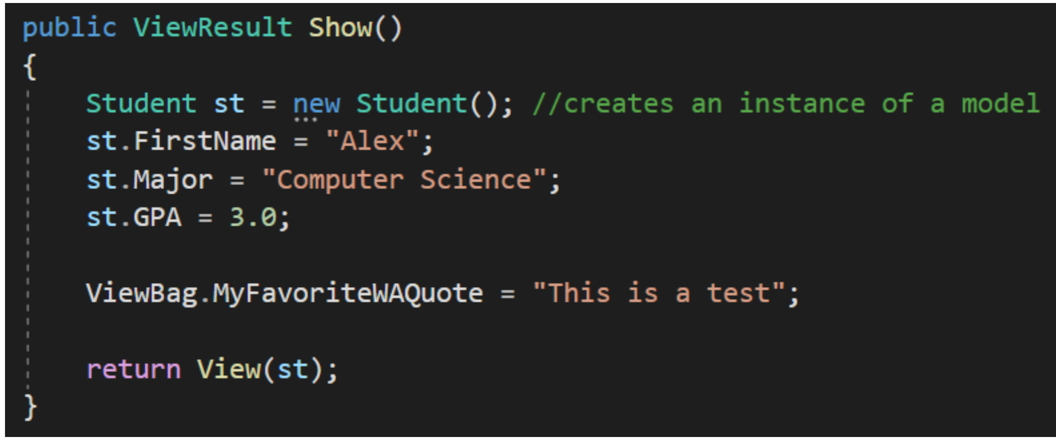
VIEWS (VIEW TEMPLATES)

- When an **action** returns a **view** file … those are **Razor view files** (or **Razor-based view templates**)



```
public ViewResult Show()
{
    return View();
}
```

- These **view** files have a .cshtml extension
- They can contain both C# and HTML
  - C# code is server side only
  - Before being sent to the client who made the request, Razor will use the C# code to render your final html page.
  - As such, only HTML is being sent to the client.



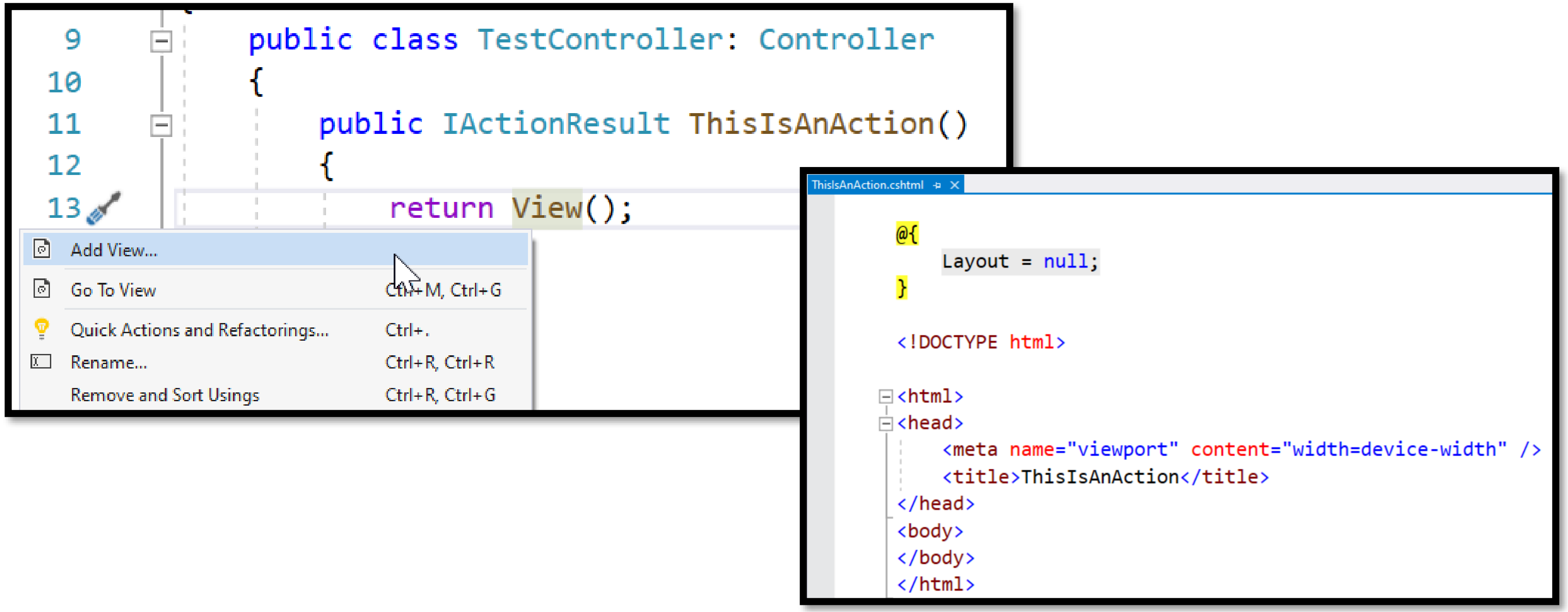
```
public ViewResult Show()
{
    Student st = new Student(); //creates an instance of a model
    st.FirstName = "Alex";
    st.Major = "Computer Science";
    st.GPA = 3.0;

    ViewBag.MyFavoriteWAQuote = "This is a test";

    return View(st);
}
```

- Examples of **views** for a **StudentController** class:
  - **Show** ← would display a student (details, a photo, …)
  - **Show** ← would display a list of all students (how would we differentiate between this and the previous view?)
  - **Create** ← would allow clients to create a new student, and add it to our database
  - **Edit** ← would allow clients to edit an existing student
  - **Delete** ← would allow clients to delete an existing student

- Source: page 210

ADDING VIEWS TO YOUR PROJECT

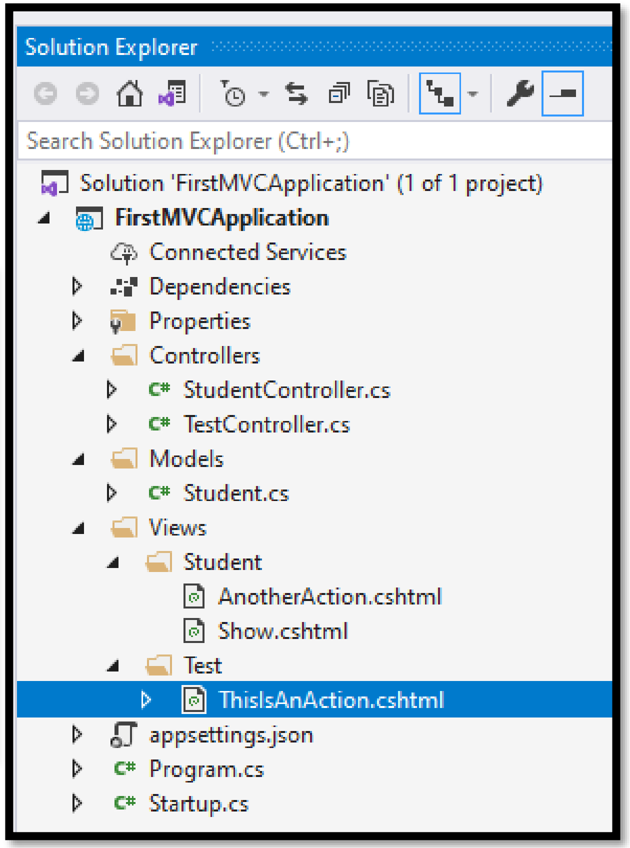
- To create a **view** file: right-click an **action**, and then select **Add View**.
  - Then choose, for example, **Razor View** and accept the defaults.
  - In our example, for the *ThisIsAnAction* action that was part of the *TestController*, a view with the name *ThisIsAnAction.cshtml* was created inside *Views* > *Test* folder.



```
    public class TestController: Controller
    {
        public IActionResult ThisIsAnAction()
        {
            return View();
```

Add View...
Go To View   Ctrl+M, Ctrl+G
Quick Actions and Refactorings...   Ctrl+.
Rename...   Ctrl+R, Ctrl+R
Remove and Sort Usings   Ctrl+R, Ctrl+G
ThisIsAnAction.cshtml

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>
</body>
</html>
```



- Please pay attention to the location the views are added:
  - All views are inside the **Views** folder.
  - The views used by **StudentController** controller, are inside **Views > Student** folder
  - The views used by **TestController** controller, are inside **Views > Test** folder

ADDING VIEWS TO YOUR PROJECT

- Another way create a **view** file: right-click a selected folder then select **Add,** then select **View**.

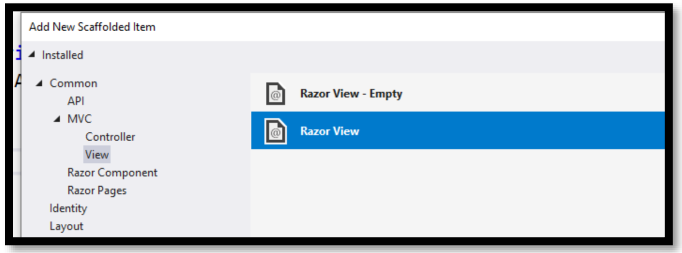
- If the project does not have the structure highlighted earlier, then first create those folders.

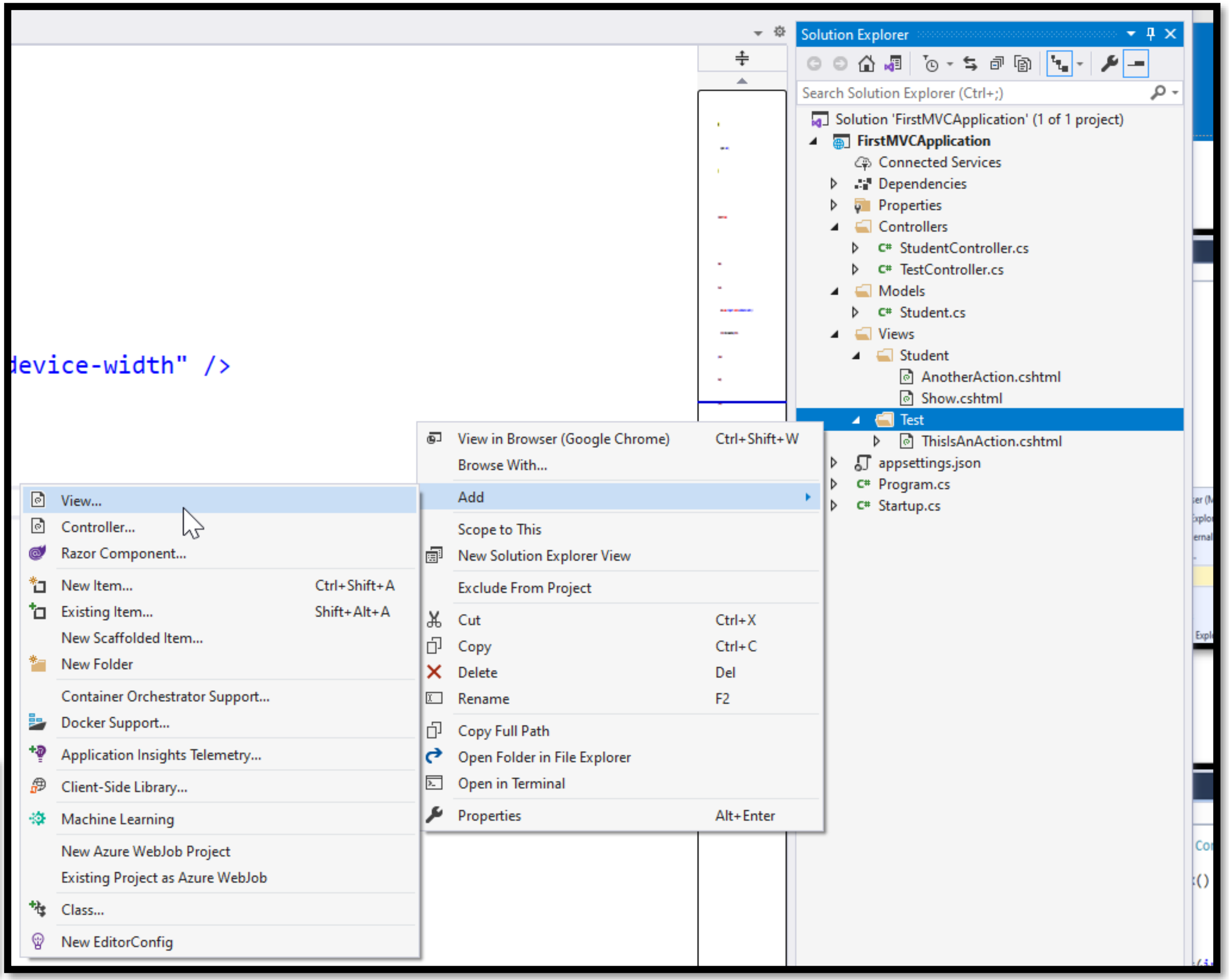
- All views must be inside **Views** folder.

- In Views folder, views used only by a **TestController** must be inside a **Test** subfolder

- In Views folder, views used only by a **StudentController** must be inside a **Student** subfolder

- And so on …





RAZOR SYNTAX

- "Razor is a markup syntax for embedding **server-based** code into **webpages**."
  - Files containing **Razor** usually have a **.cshtml** file extension

- **Razor** uses @ to identify server-side C# code.
  - You only need to mark the **start** of a Razor code expression with the @ symbol. Razor infers the **end** of it
    - When an @ symbol is followed by a Razor reserved **keyword**, it transitions into Razor-specific markup.
    - Otherwise, it transitions into plain C#.

- Let's start

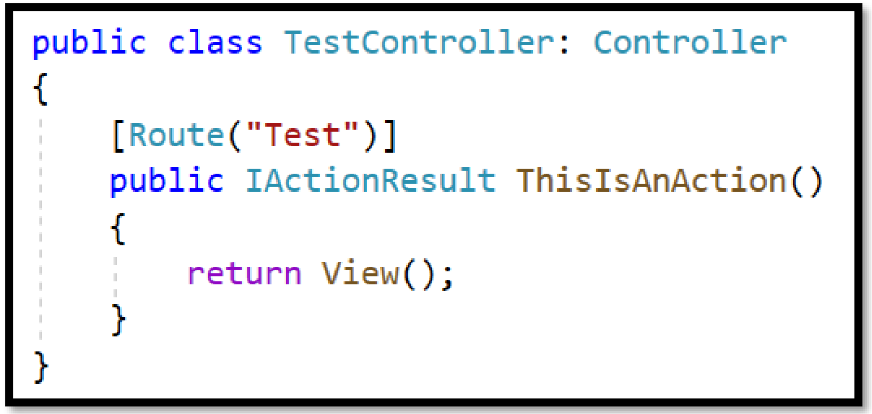
- To escape the @ symbol in Razor, use a second @ symbol
  - Example: <h1> @@User </h1> will be rendered as: < h1> @User </ h1>

- Note: Razor will not modify HTML attributes and content containing email addresses
  - Example: <h1>user@domain.com </h1> will be rendered as: <h1>user@domain.com </h1>

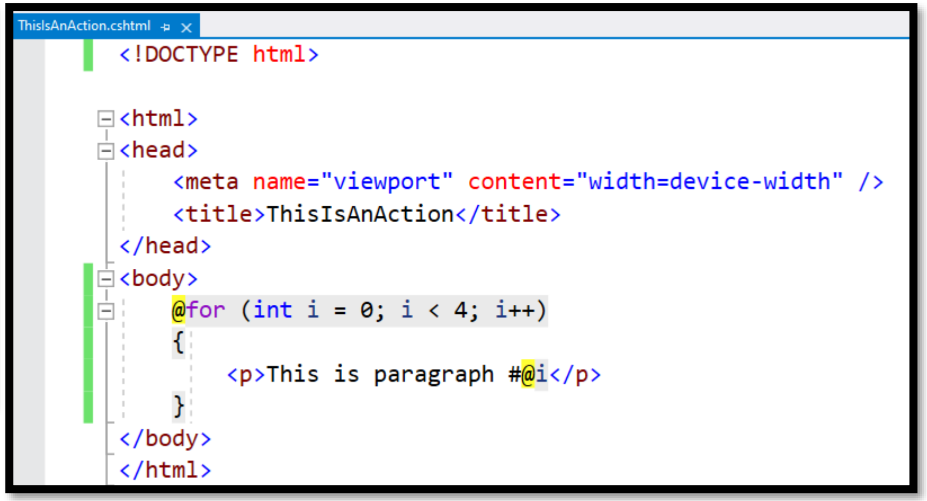
- Source / See also pages 2729-2743

IMPLICIT RAZOR EXPRESSIONS

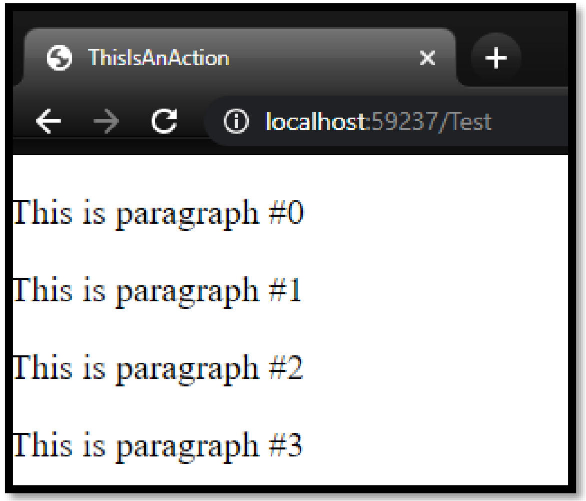
- Implicit Razor expressions start with @ then add C# code



```
public class TestController: Controller
{
    [Route("Test")]
    public IActionResult ThisIsAnAction()
    {
        return View();
    }
}
```

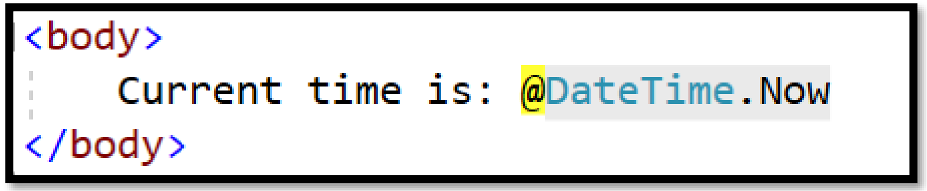


```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>
    @for (int i = 0; i < 4; i++)
    {
        <p>This is paragraph #@i</p>
    }
</body>
</html>
```

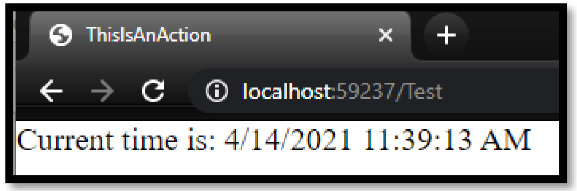


```
This is paragraph #0

This is paragraph #1

This is paragraph #2

This is paragraph #3
```

- Here is another example:



```
<body>
    Current time is: @DateTime.Now
</body>
```



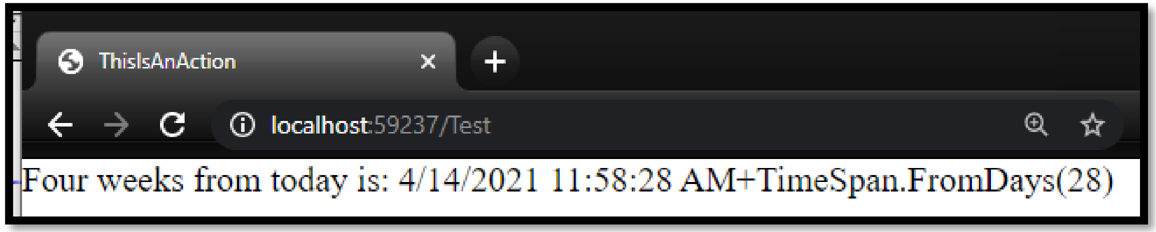
```
Current time is: 4/14/2021 11:39:13 AM
```

- Implicit expressions cannot handle C# generics (for example List<int> myList)
  - the <> characters are interpreted as HTML tags
  - solution: use **explicit Razor expression** or a **Razor code block** (seen below)

- Source / See also pages 2729-2743

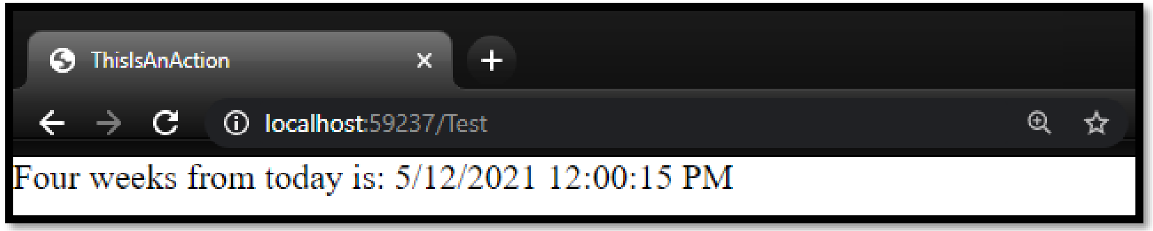
EXPLICIT RAZOR EXPRESSIONS

- Explicit Razor expressions use @ and a set of parentheses (with C# code inside): @(…)
  - `Four weeks from today is: @DateTime.Now+TimeSpan.FromDays(28)`



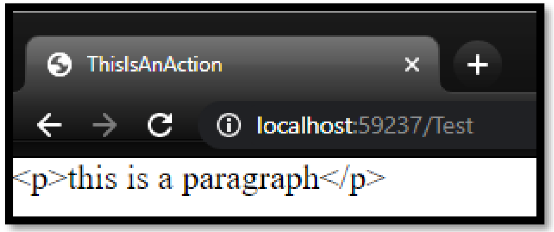
```
Four weeks from today is: 4/14/2021 11:58:28 AM+TimeSpan.FromDays(28)
```

  - `Four weeks from today is: @(DateTime.Now+TimeSpan.FromDays(28))`

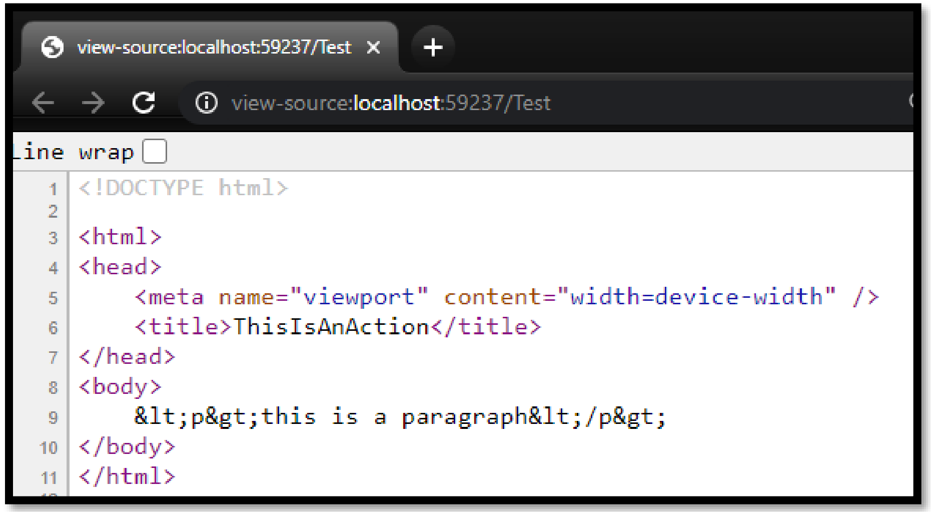


```
Four weeks from today is: 5/12/2021 12:00:15 PM
```

- C# expressions that evaluate to strings are **HTML encoded**
  - Example: `@("<p>this is a paragraph</p>")`



```
<p>this is a paragraph</p>
```

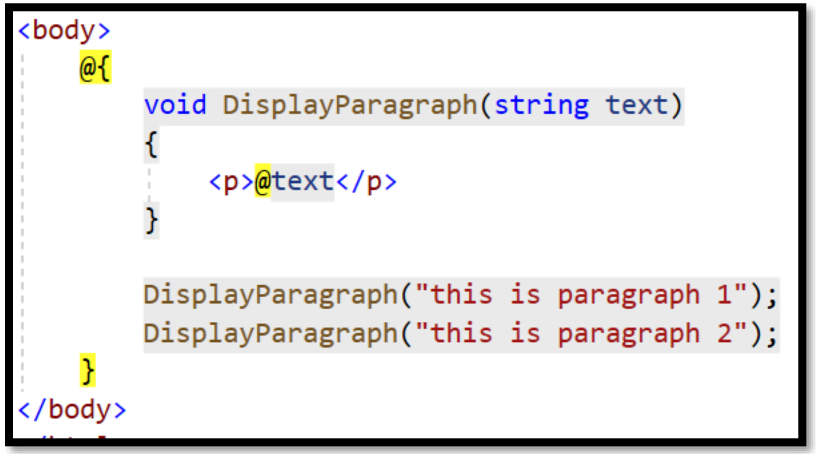

view-source:localhost:59237/Test
view-source:localhost:59237/Test
Line wrap □

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>
    &lt;p&gt;this is a paragraph&lt;/p&gt;
</body>
</html>
```

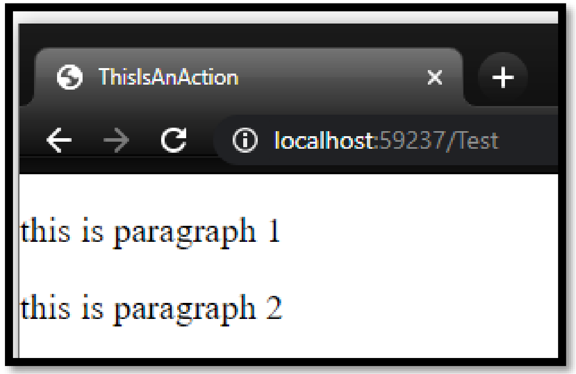
- Source / See also pages 2729-2743

RAZOR CODE BLOCKS

- Razor code blocks use @ and a set of {} (with C# code inside): @{…}
  - Use to write multiple lines of server-side code

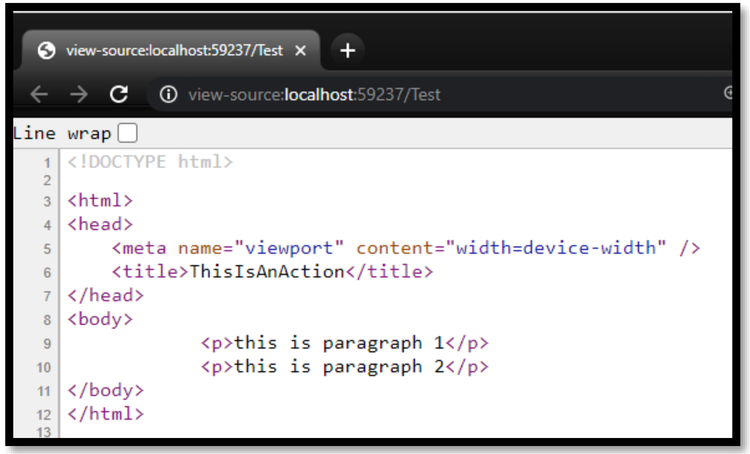


```
<body>
    @{
        void DisplayParagraph(string text)
        {
            <p>@text</p>
        }

        DisplayParagraph("this is paragraph 1");
        DisplayParagraph("this is paragraph 2");
    }
</body>
```



this is paragraph 1

this is paragraph 2

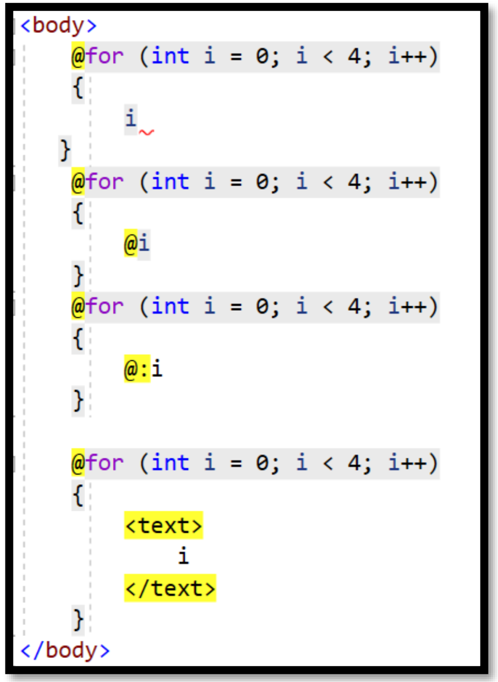


view-source:localhost:59237/Test
view-source:localhost:59237/Test

Line wrap □

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>
            <p>this is paragraph 1</p>
            <p>this is paragraph 2</p>
</body>
</html>
```

- Use @: to explicitly declare **a line of text** as *content* and not *code*
  - Useful to fix errors when Razor misinterprets content as code
  - Use **<text>** to explicitly declare several lines of text as content and not code

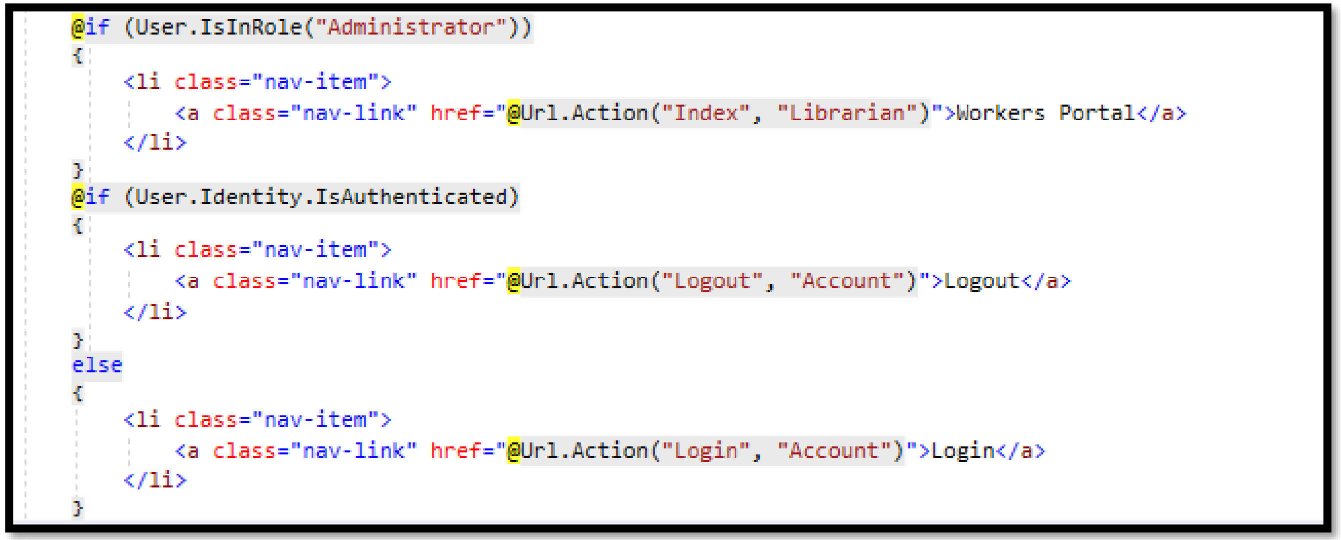


```
<body>
    @for (int i = 0; i < 4; i++)
    {
        i
    }
    @for (int i = 0; i < 4; i++)
    {
        @i
    }
    @for (int i = 0; i < 4; i++)
    {
        @:i
    }

    @for (int i = 0; i < 4; i++)
    {
        <text>
            i
        </text>
    }
</body>
```

- Source / See also pages 2729-2743

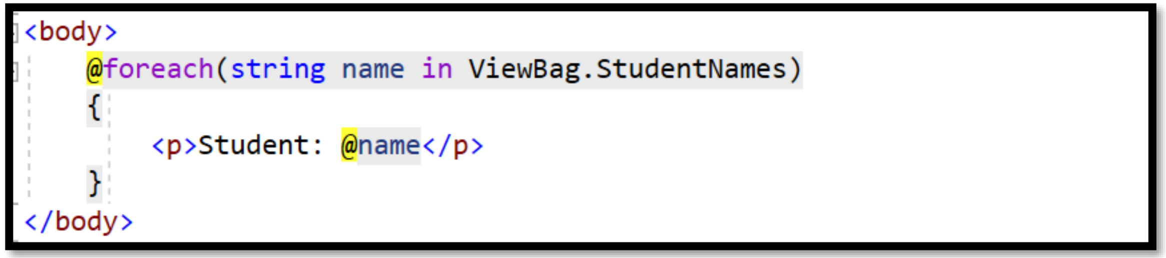
CONTROL STRUCTURES & OTHER

- Conditionals: @if, else if, else, and @switch
  - For example:



```
@if (User.IsInRole("Administrator"))
{
    <li class="nav-item">
        <a class="nav-link" href="@Url.Action("Index", "Librarian")">Workers Portal</a>
    </li>
}
@if (User.Identity.IsAuthenticated)
{
    <li class="nav-item">
        <a class="nav-link" href="@Url.Action("Logout", "Account")">Logout</a>
    </li>
}
else
{
    <li class="nav-item">
        <a class="nav-link" href="@Url.Action("Login", "Account")">Login</a>
    </li>
}
```

- Looping: @for, @foreach, @while, and @do while



```
<body>
    @foreach(string name in ViewBag.StudentNames)
    {
        <p>Student: @name</p>
    }
</body>
```

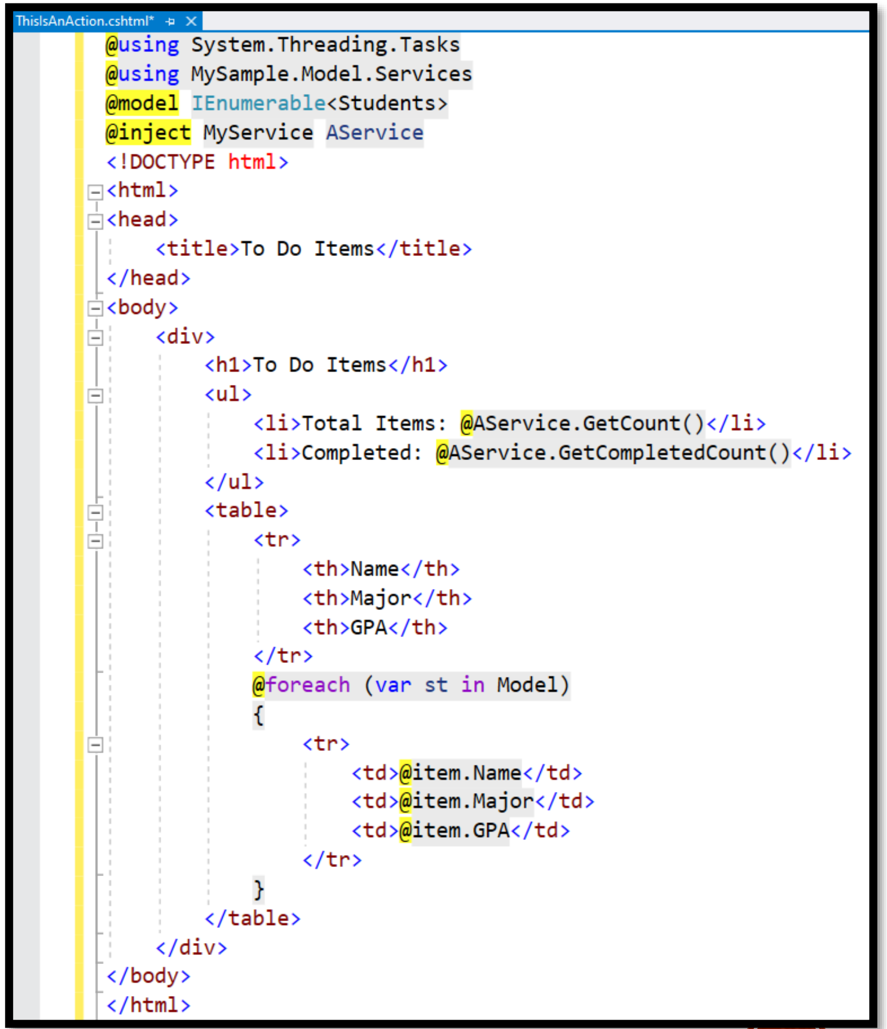
- Error handling: @try, catch, finally
- Comments: @* … *@
  - Razor comments are removed by the server before the contents are sent to the client. TEST IT!
  - C# comments (// and /*…*/) are also supported
  - HTML comments <!-- HTML comment --> will be sent to the client
- Source / See also pages 2729-2743

DIRECTIVES (MORE DETAILS COMING SOON!)

- @inject allows a Razor Page to **inject a service** (from the service container) into a **view**.
    - **@inject** <type> <instance name>

- @model specifies the type of the **model** passed (from an **action**) to a **view**
    - **@model** TypeNameOfModel
    - Be careful to Model vs model! (we'll see more later!)

- @using adds C# ***using** directive* to a **view**
    - **@using** NamespaceName



```
@using System.Threading.Tasks
@using MySample.Model.Services
@model IEnumerable<Students>
@inject MyService AService
<!DOCTYPE html>
<html>
<head>
    <title>To Do Items</title>
</head>
<body>
    <div>
        <h1>To Do Items</h1>
        <ul>
            <li>Total Items: @AService.GetCount()</li>
            <li>Completed: @AService.GetCompletedCount()</li>
        </ul>
        <table>
            <tr>
                <th>Name</th>
                <th>Major</th>
                <th>GPA</th>
            </tr>
            @foreach (var st in Model)
            {
                <tr>
                    <td>@item.Name</td>
                    <td>@item.Major</td>
                    <td>@item.GPA</td>
                </tr>
            }
        </table>
    </div>
</body>
</html>
```

- Source / See also pages 2729-2743, and 1824 ← check this out for more details!

IN-CLASS DEMO

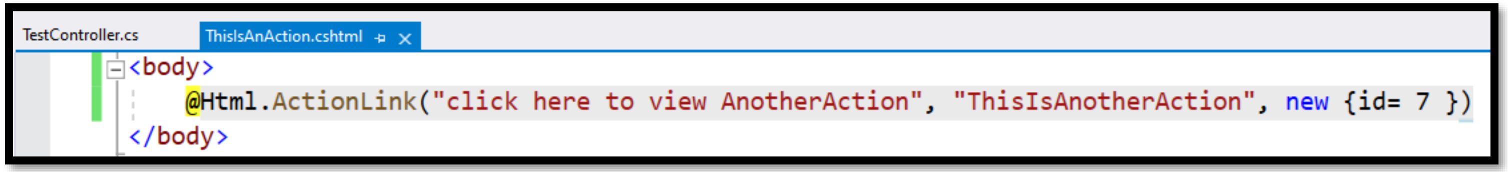
Demonstration: How to Use the Razor Syntax

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_DEMO.md#demonstration-how-to-use-the-razor-syntax

HTML HELPERS

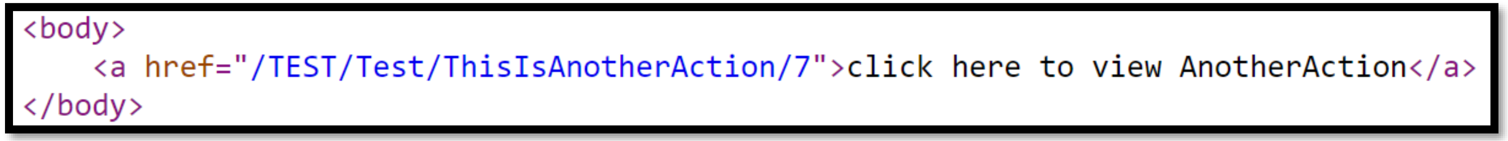
- HTML Helpers are C# methods used inside views to return **strings** (essentially small pieces of HTML)
  - MVC has built-in Helpers methods, but one can also create custom ones.
  - We'll see more HTML helpers in the next module!

- Html.ActionLink() will return an **<a> element** with a link to an **action**.
  - Example:



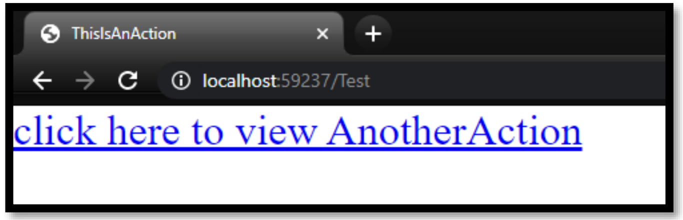
```
<body>
    @Html.ActionLink("click here to view AnotherAction", "ThisIsAnotherAction", new {id= 7 })
</body>
```

  - Rendered as:



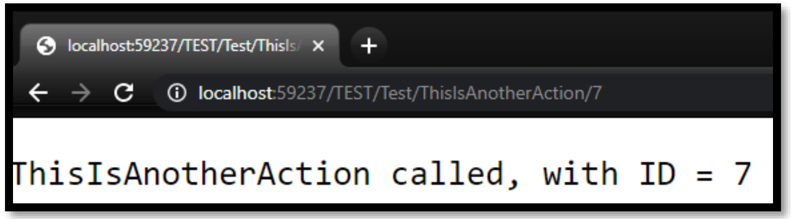
```
<body>
    <a href="/TEST/Test/ThisIsAnotherAction/7">click here to view AnotherAction</a>
</body>
```

  - Seen as:



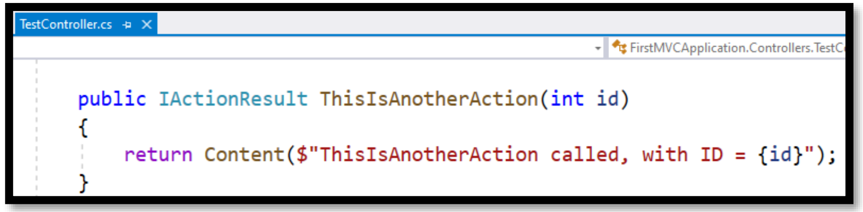
click here to view AnotherAction

  - When users click on it:



localhost:59237/TEST/Test/ThisIs
localhost:59237/TEST/Test/ThisIsAnotherAction/7

```
ThisIsAnotherAction called, with ID = 7
```

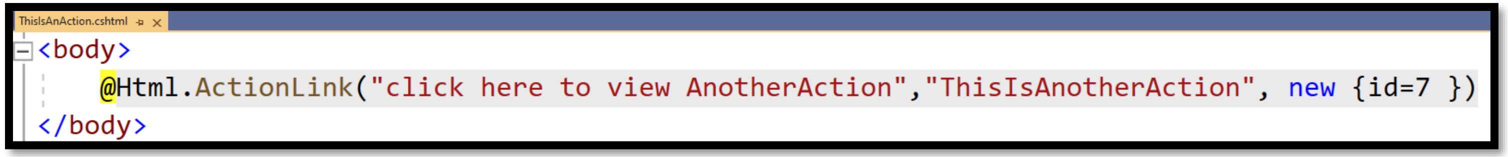


```
public IActionResult ThisIsAnotherAction(int id)
{
    return Content($"ThisIsAnotherAction called, with ID = {id}");
}
```

- Sources:
  - pages 120 +, 1790
  - https://www.c-sharpcorner.com/article/html-helpers-in-asp-net-mvc-5/

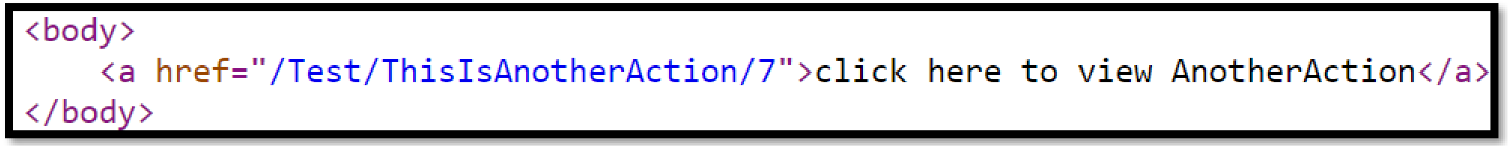
HTML HELPERS

- Html.ActionLink() will return an **<a> element** with a link to an **action**.
  - Example:



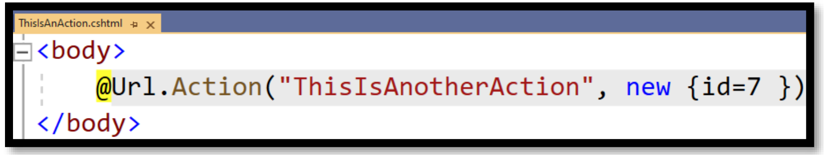
```
<body>
    @Html.ActionLink("click here to view AnotherAction","ThisIsAnotherAction", new {id=7 })
</body>
```

  - Rendered as:



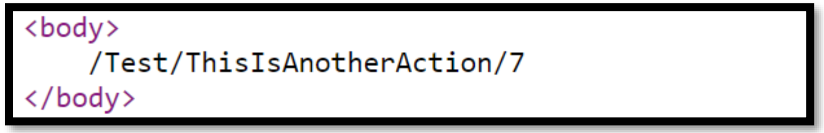
```
<body>
    <a href="/Test/ThisIsAnotherAction/7">click here to view AnotherAction</a>
</body>
```

- If you only need the URL, not the entire <a> element (useful for <img>) then use Url.Action() instead
  - Example:



```
<body>
    @Url.Action("ThisIsAnotherAction", new {id=7 })
</body>
```

  - Rendered as:



```
<body>
    /Test/ThisIsAnotherAction/7
</body>
```

- Sources:
  - pages 120+, 1790
  - https://www.c-sharpcorner.com/article/html-helpers-in-asp-net-mvc-5/

HTML HELPERS – IF TIME

- Example:
- Create an **action**
- Add [Route(“Test”)] to it
- Create a corresponding **view**.

- Create a second **action**
- Make this action simply return **contents**

- In the view above, add HTML helpers that will create URLs to the second action.
- Test both:
  - Html.ActionLink
  - Url.Action

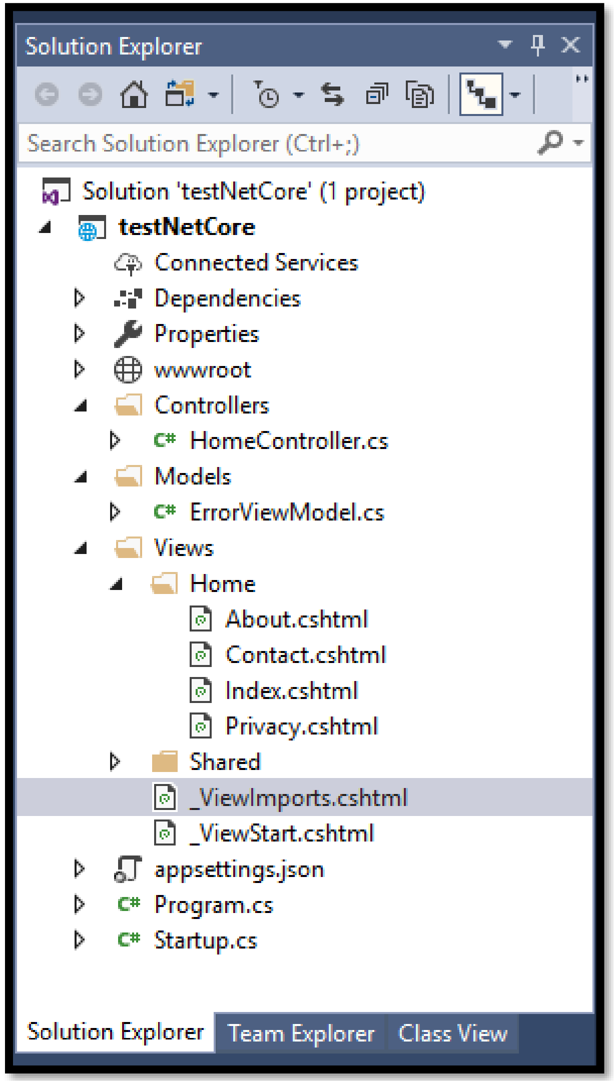
- Test how links generated by those helpers change when you change routing!

TAG HELPERS

- **Tag helpers** are an alternative to **HTML helpers**.
  - They produce the same result!
    - **Tag helpers** use a more HTML-like syntax
    - **HTML helpers** use a more C#-like syntax

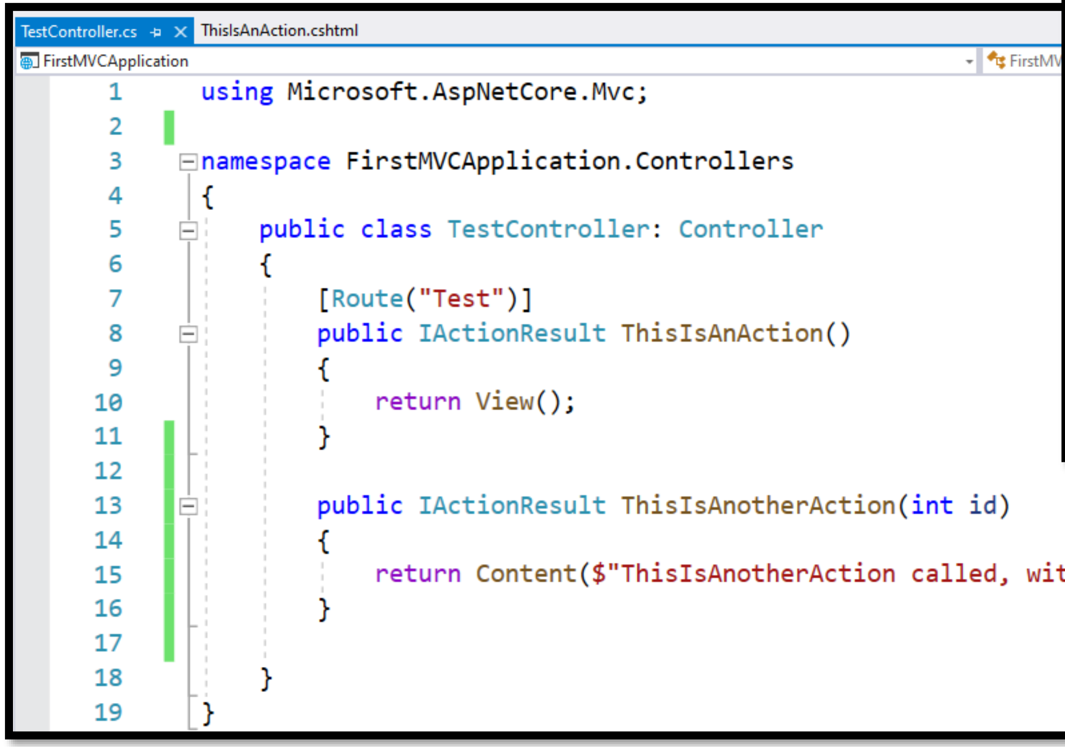
- To use a **Tag Helper** inside a view,
  one must use the following **@addTagHelper** directive:
  - `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`
    - This can be added in <u>every **view**</u> that uses them
    - Alternatively, once can add this directive <u>once</u>, in the project's _**ViewImports.cshtml**, then use them in every view

- See also pages 2743+, 2756+

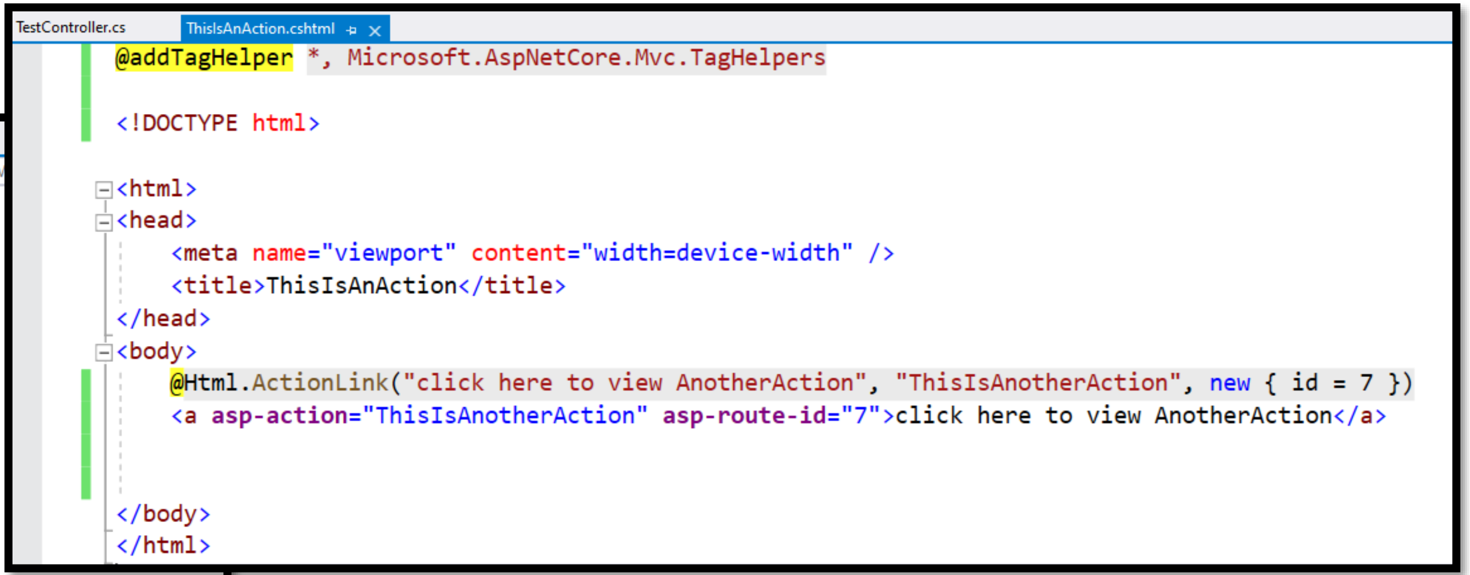


TAG HELPERS

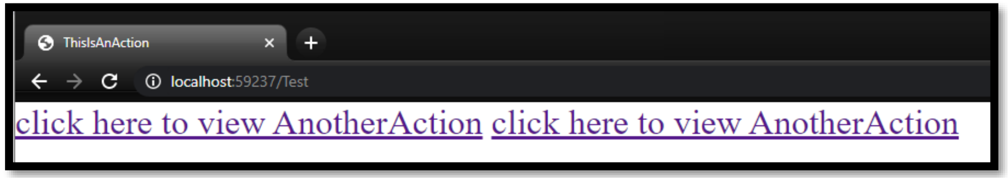
- Example:



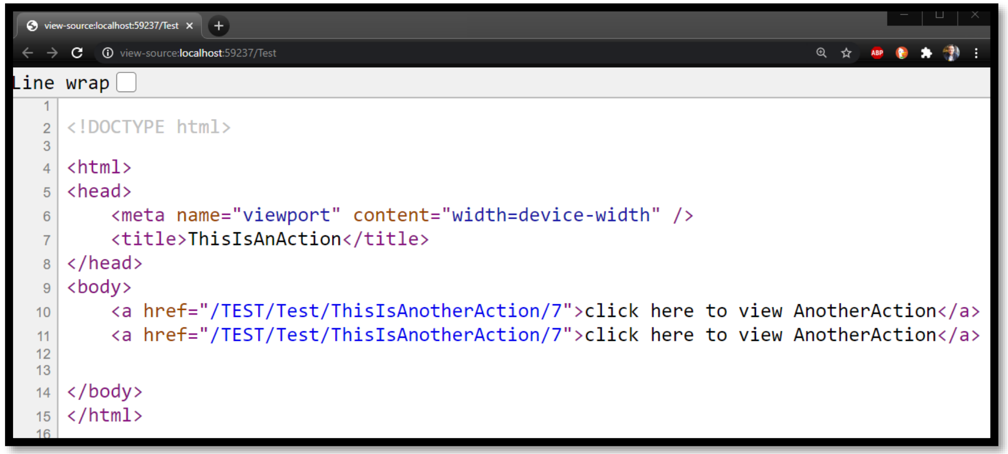
```
using Microsoft.AspNetCore.Mvc;

namespace FirstMVCApplication.Controllers
{
    public class TestController: Controller
    {
        [Route("Test")]
        public IActionResult ThisIsAnAction()
        {
            return View();
        }

        public IActionResult ThisIsAnotherAction(int id)
        {
            return Content($"ThisIsAnotherAction called, with ID = {id}");
        }

    }
}
```



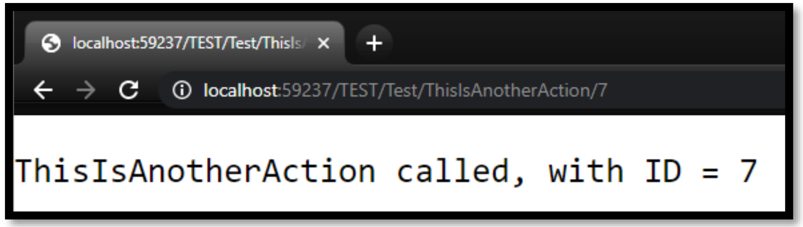
```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>
    @Html.ActionLink("click here to view AnotherAction", "ThisIsAnotherAction", new { id = 7 })
    <a asp-action="ThisIsAnotherAction" asp-route-id="7">click here to view AnotherAction</a>


</body>
</html>
```



click here to view AnotherAction click here to view AnotherAction


view-source:localhost:59237/Test
view-source:localhost:59237/Test
Line wrap □

```

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>
    <a href="/TEST/Test/ThisIsAnotherAction/7">click here to view AnotherAction</a>
    <a href="/TEST/Test/ThisIsAnotherAction/7">click here to view AnotherAction</a>


</body>
</html>
```


localhost:59237/TEST/Test/ThisIs
localhost:59237/TEST/Test/ThisIsAnotherAction/7

```
ThisIsAnotherAction called, with ID = 7
```

IN-CLASS DEMO

Demonstration: How to Use HTML Helpers

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_DEMO.md#demonstration-how-to-use-html-helpers

IN-CLASS DEMO

Demonstration: How to Use Tag Helpers

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_DEMO.md#demonstration-how-to-use-tag-helpers

PARTIAL VIEWS

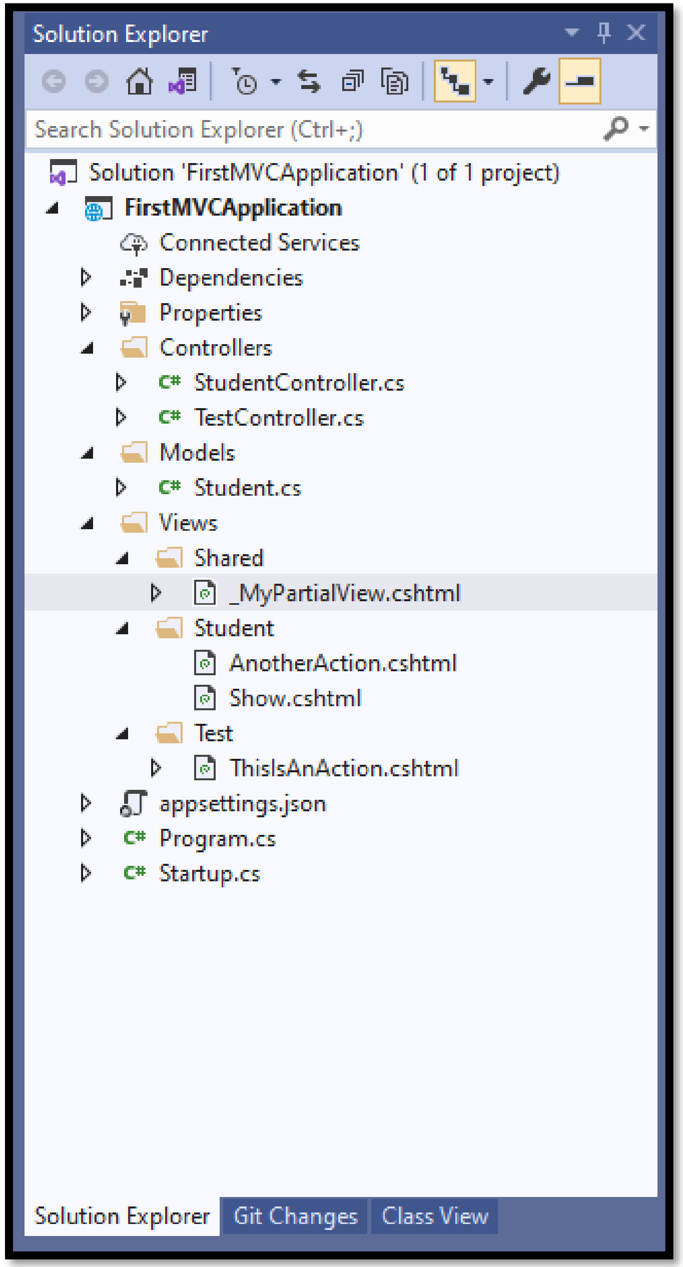
- **Partial views** are **reusable** parts for **views.**
  - A **partial view** renders **only a portion of HTML content**, which you can then insert into several **views** at run time.
  - They help reduce code duplication.
  - Example: you may want to display, in several views, the *most popular items*.

- Create a **partial view** by using the **Add View** dialog box, in the same way that you create any other **view.**
  - Convention: names of partial views are usually prefixed with an **underscore**
  - Partial views are often created inside the **/Views/Shared** folder
  - For example: _MyPartialView.cshtml
  - Sample content: <p>This is my first partial view</p>

- Just like **views** (more details later!), **partial views** can be:
  - **Strongly-typed**: has a declaration of the **@model** directive at the top of the file
  - **Dynamic**: it does not have the **@model** directive (details in the next module)

- Use **Html.PartialAsync to render a partial view within another view** file

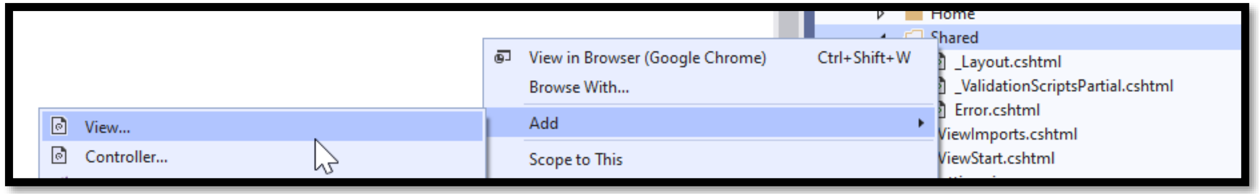
- Source: page 1745, 1754+, see also 1760 (passing a model …)

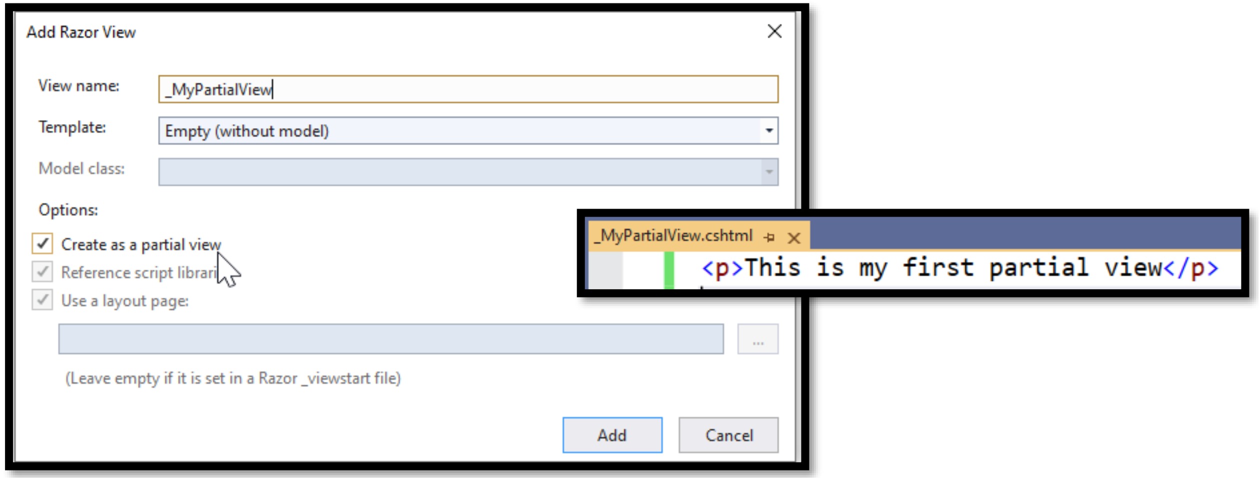


```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'FirstMVCApplication' (1 of 1 project)
  FirstMVCApplication
    Connected Services
    Dependencies
    Properties
    Controllers
      StudentController.cs
      TestController.cs
    Models
      Student.cs
    Views
      Shared
        _MyPartialView.cshtml
      Student
        AnotherAction.cshtml
        Show.cshtml
      Test
        ThisIsAnAction.cshtml
    appsettings.json
    Program.cs
    Startup.cs
```

PARTIAL VIEWS - EXAMPLE

- **Create** a **partial view**




Add Razor View

View name: _MyPartialView

Template: Empty (without model)

Model class:

Options:

☑ Create as a partial view

☑ Reference script librari

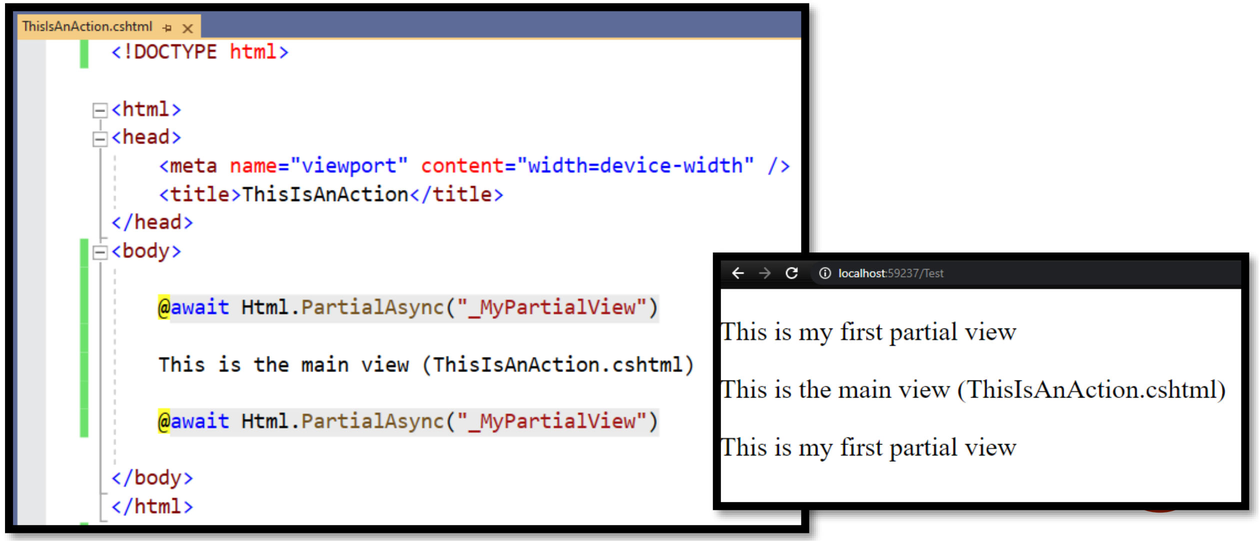
☑ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

_MyPartialView.cshtml

```
<p>This is my first partial view</p>
```

- **Use** the **partial view**


ThisIsAnAction.cshtml

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ThisIsAnAction</title>
</head>
<body>

    @await Html.PartialAsync("_MyPartialView")

    This is the main view (ThisIsAnAction.cshtml)

    @await Html.PartialAsync("_MyPartialView")

</body>
</html>
```

localhost:59237/Test

```
This is my first partial view

This is the main view (ThisIsAnAction.cshtml)

This is my first partial view
```

PARTIAL VIEWS – IF TIME

- If time:

- In a controller action, set some values inside the **ViewBag**
- Then use that value inside a **View**
- Then use that value inside a **Partial View**

IN-CLASS DEMO

Demonstration: How to Create and Use Partial Views

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_DEMO.md#demonstration-how-to-create-and-use-partial-views

VIEW COMPONENTS – ONLY IF TIME

- View components are an alternative to **partial views**
  - they also allow you to reduce repetitive code,
  - but they're appropriate for view content that requires code to run on the server in order to render the webpage.

- They are particularly useful when the rendered content requires **database interaction**

- "**View components** are intended anywhere you have reusable rendering logic that's too complex for a **partial view**, such as:
  - Dynamic navigation menus
  - [...]
  - Shopping cart
  - Recently published articles
  - Sidebar content on a typical blog"

- Source: page 1745+, 3072+

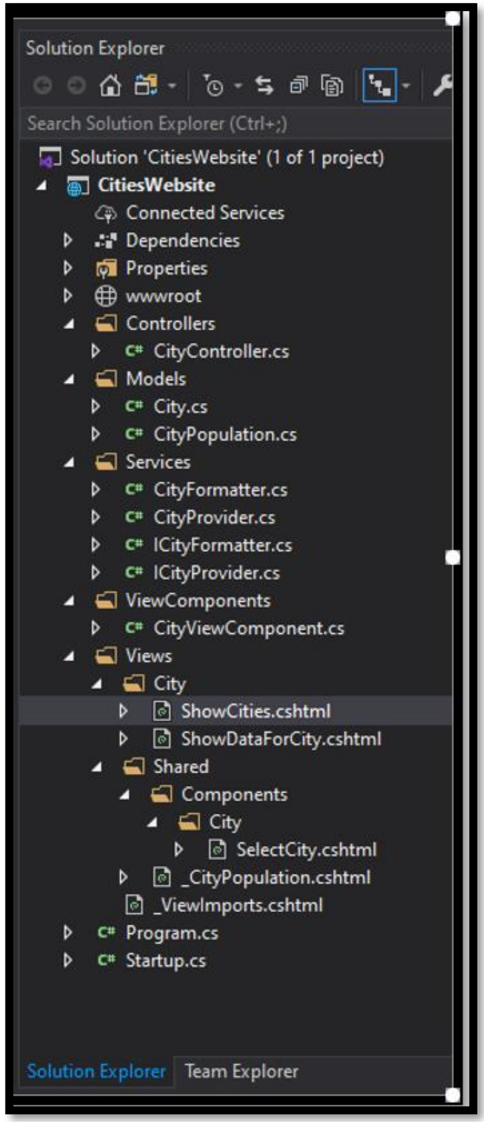
VIEW COMPONENTS – ONLY IF TIME

- A view component consists of two parts:

- **A class**
  - Usually derived from the **ViewComponent** base class
  - Recommended to locate this class in a folder named **ViewComponents**
  - Should have a method called **InvokeAsync**, which defines its logic

- **A view**
  - Located in a folder under **Views\Shared\Components** folder
  - The name of the folder should be the same as the name of the view component class without the **ViewComponent** suffix
    - For **City**ViewComponent, the view will be inside Views\Shared\Components\City
    - For **My**ViewComponent, the view will be in a folder named Views\Shared\Components\My

- Source: page 1745+, 3072+



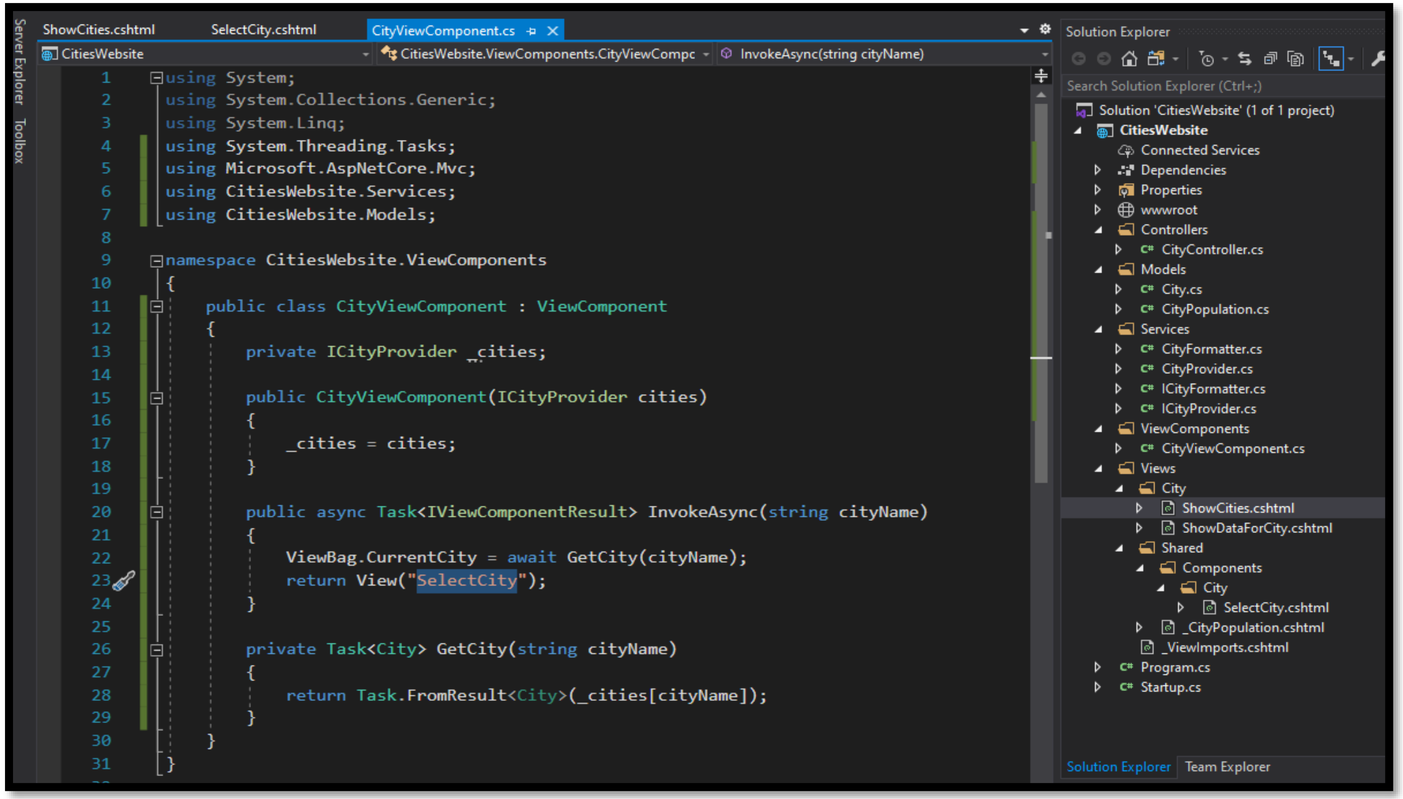
Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'CitiesWebsite' (1 of 1 project)
  CitiesWebsite
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      CityController.cs
    Models
      City.cs
      CityPopulation.cs
    Services
      CityFormatter.cs
      CityProvider.cs
      ICityFormatter.cs
      ICityProvider.cs
    ViewComponents
      CityViewComponent.cs
    Views
      City
        ShowCities.cshtml
        ShowDataForCity.cshtml
      Shared
        Components
          City
            SelectCity.cshtml
        _CityPopulation.cshtml
      _ViewImports.cshtml
    Program.cs
    Startup.cs
```

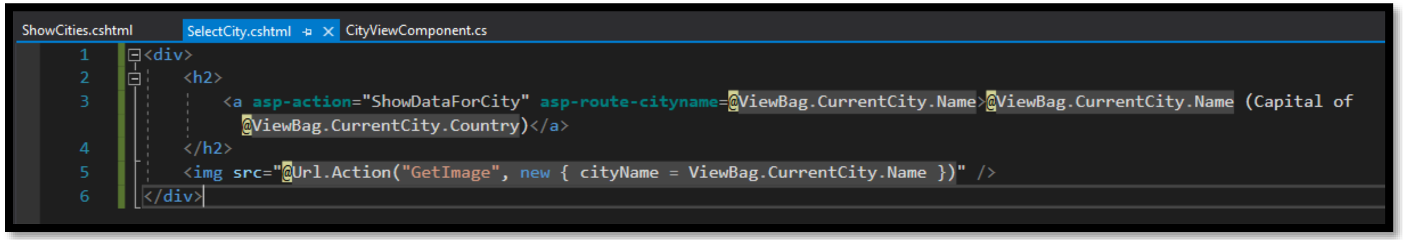
Solution Explorer Team Explorer

EXTRA – PART OF YOUR OPTIONAL HOMEWORK

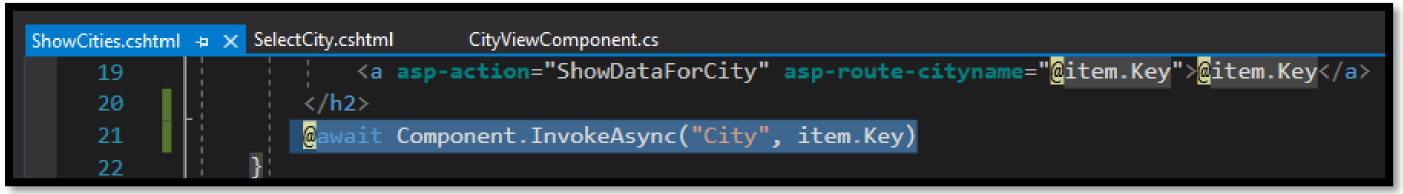


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using CitiesWebsite.Services;
using CitiesWebsite.Models;

namespace CitiesWebsite.ViewComponents
{
    public class CityViewComponent : ViewComponent
    {
        private ICityProvider _cities;

        public CityViewComponent(ICityProvider cities)
        {
            _cities = cities;
        }

        public async Task<IViewComponentResult> InvokeAsync(string cityName)
        {
            ViewBag.CurrentCity = await GetCity(cityName);
            return View("SelectCity");
        }

        private Task<City> GetCity(string cityName)
        {
            return Task.FromResult<City>(_cities[cityName]);
        }
    }
}
```

```
Solution 'CitiesWebsite' (1 of 1 project)
  CitiesWebsite
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
      CityController.cs
    Models
      City.cs
      CityPopulation.cs
    Services
      CityFormatter.cs
      CityProvider.cs
      ICityFormatter.cs
      ICityProvider.cs
    ViewComponents
      CityViewComponent.cs
    Views
      City
        ShowCities.cshtml
        ShowDataForCity.cshtml
      Shared
        Components
          City
            SelectCity.cshtml
        _CityPopulation.cshtml
      _ViewImports.cshtml
    Program.cs
    Startup.cs
```

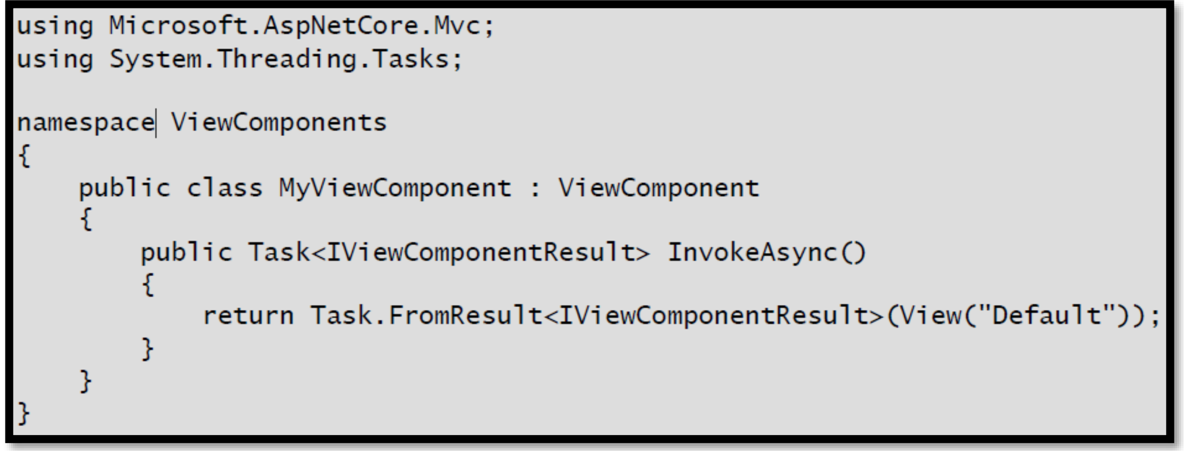


```
<div>
    <h2>
        <a asp-action="ShowDataForCity" asp-route-cityname=@ViewBag.CurrentCity.Name>@ViewBag.CurrentCity.Name (Capital of
         @ViewBag.CurrentCity.Country)</a>
    </h2>
    <img src="@Url.Action("GetImage", new { cityName = ViewBag.CurrentCity.Name })" />
</div>
```



```
                <a asp-action="ShowDataForCity" asp-route-cityname="@item.Key">@item.Key</a>
            </h2>
            @await Component.InvokeAsync("City", item.Key)
        }
```

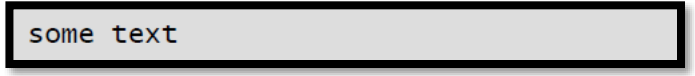
- Example of **a view component** class named **MyViewComponent**:
  - Below, the name of the partial view that will be rendered is **Default**:



```
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace ViewComponents
{
    public class MyViewComponent : ViewComponent
    {
        public Task<IViewComponentResult> InvokeAsync()
        {
            return Task.FromResult<IViewComponentResult>(View("Default"));
        }
    }
}
```

class

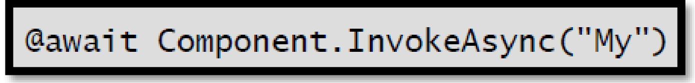
- Content of a view component view located in a file named **Default.cshtml**:
  - It is located in the **Views\Shared\Components\My\Default.cshtml** file.



```
some text
```

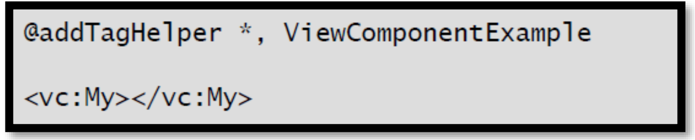
view

- To use a **view component** one can include it in a **view**:
  - You can include a **view component** in a **view** by using the **@Component.InvokeAsync** method



```
@await Component.InvokeAsync("My")
```

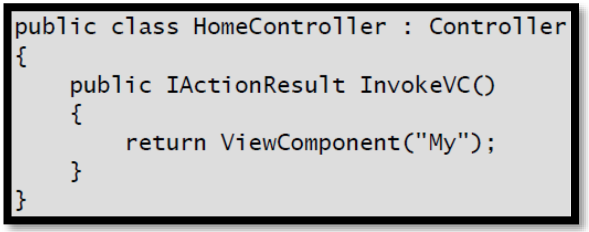
  - Alternatively, one can use a **tag helper** to include the **view component** in a view



```
@addTagHelper *, ViewComponentExample

<vc:My></vc:My>
```

uses ...

  - When the **view** containing the code above is sent to the user, it will contain the text “some text”.
- Invoking **ViewComponent** from a Controller action ➔ ➔ ➔

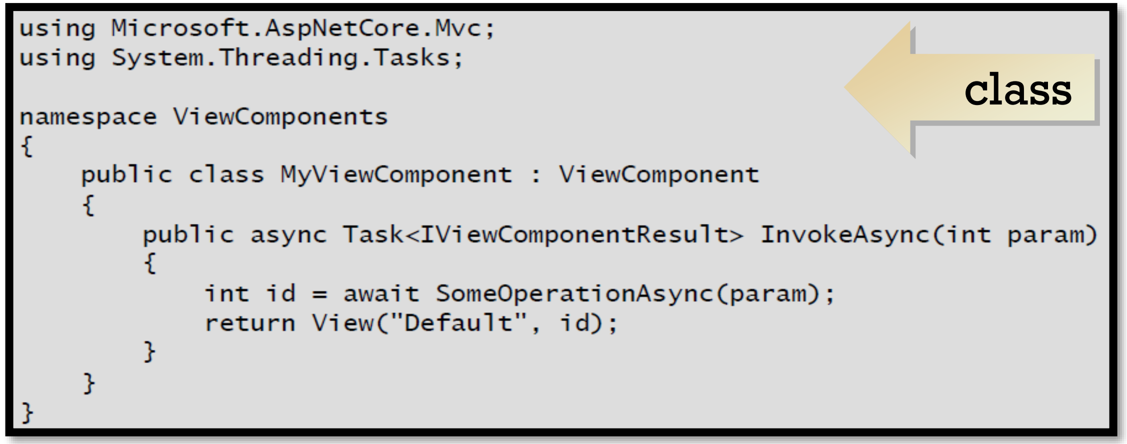


```
public class HomeController : Controller
{
    public IActionResult InvokeVC()
    {
        return ViewComponent("My");
    }
}
```

INVOKING VIEW COMPONENTS WITH PARAMETERS

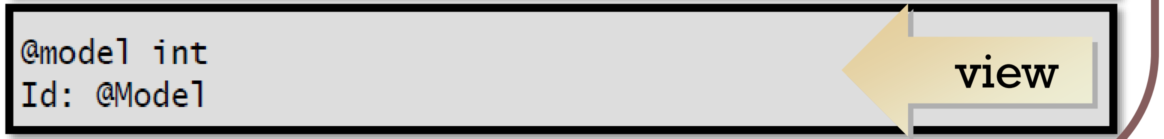
- The **InvokeAsync** method can take **any number of parameters**.
  - Those parameters will be passed when the **view component** is invoked in a **view** or in a **controller**

- Example:
  - The **view component** class:



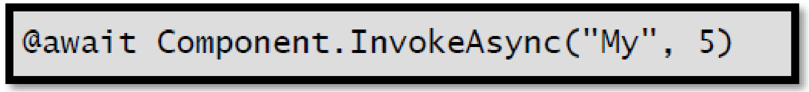
```
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace ViewComponents
{
    public class MyViewComponent : ViewComponent
    {
        public async Task<IViewComponentResult> InvokeAsync(int param)
        {
            int id = await SomeOperationAsync(param);
            return View("Default", id);
        }
    }
}
```

  - Default View for **component view**:



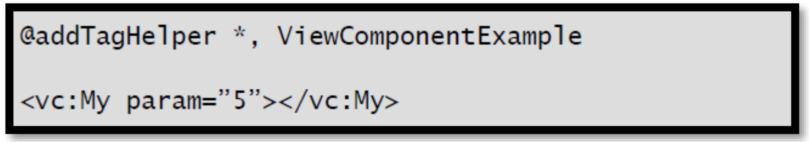
```
@model int
Id: @Model
```

  - Passing a parameter from a **view** to the **view component**:



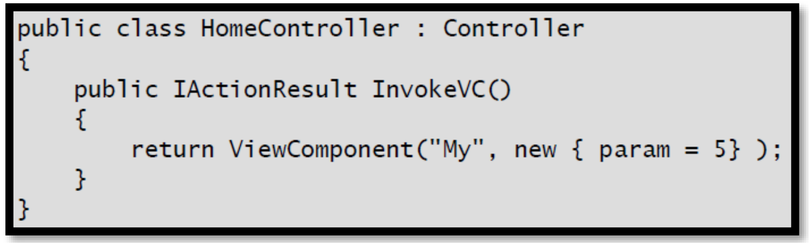
```
@await Component.InvokeAsync("My", 5)
```

  - Passing a parameter from a **view** to the **view component**:
    - Using a **tag helper**



```
@addTagHelper *, ViewComponentExample

<vc:My param=”5”></vc:My>
```

  - Passing a parameter from a **controller** to the **view component:**



```
public class HomeController : Controller
{
    public IActionResult InvokeVC()
    {
        return ViewComponent("My", new { param = 5} );
    }
}
```

- EXTRA: see also: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/view-components?view=aspnetcore-3.1

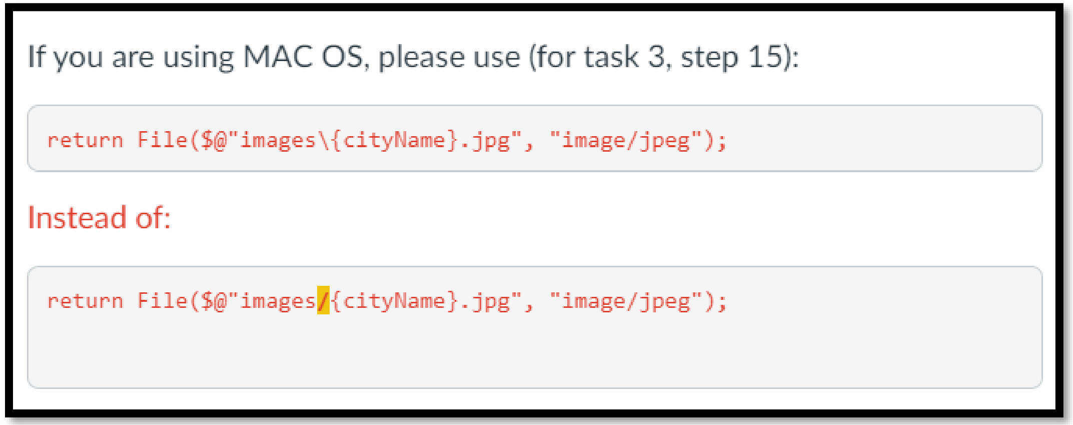
IN-CLASS DEMO – SKIP IF NEEDED

Demonstration: How to Create and Use View Components

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_DEMO.md#demonstration-how-to-create-and-use-view-components

LAB/HOMEWORK: DEVELOPING VIEWS

- **Module 05**
  - Exercise 1: Adding Views to an MVC Application
  - Exercise 2: Adding a Partial View
  - Exercise 3: Adding a View Component ← OPTIONAL!



If you are using MAC OS, please use (for task 3, step 15):

```
return File($@"images\{cityName}.jpg", "image/jpeg");
```

Instead of:

```
return File($@"images/{cityName}.jpg", "image/jpeg");
```

- You will find the **high-level** steps on the following page:
  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_LAB_MANUAL.md

- You will find the **detailed** steps on the following page:
  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD05_LAK.md

- For your homework submit one zipped folder with your complete solution.