Module 5

Developing Page Content

infoSupport
Solid Innovator

Module Overview

- Creating Page Content with Razor Syntax
- Using HTML Helpers and Tag Helpers
- Reusing Code

Lesson 1: Creating Page Content with Razor Syntax

- Adding Page Content
- Differentiating Server-Side Code from HTML
- Features of the Razor Syntax
- Demonstration: How to Use the Razor Syntax
- Dependency Injection

Adding Page Content

- Page Content handle the presentation logic
- Page Content files have a .cshtml extension

Differentiating Server-Side Code from HTML

- Razor identifies server-side code by looking for the **@** symbol
- Razor distinguishes the server-side code from the HTML content that is sent to the browser unchanged

```
<body>
    @for (int i = 0; i < 5; i++)
    {
        <span>@i</span>
    }
</body>
```

Using the @ Symbol

In Razor syntax, the **@** symbol has various uses. You can:

- Use **@** to identify server-side C# code
- Use **@@** to render an @ symbol in an HTML page
- Use **@:** to explicitly declare a line of text as content and not code
- Use **<text>**to explicitly declare several lines of text as content and not code

Features of the Razor Syntax

A sample code block displaying the features of Razor

```
@* Some more Razor examples *@
<span>
    Price including Sale Tax: @ViewBag.Price * 1.2
</span>
<span>
    Price including Sale Tax: @(ViewBag.Price * 1.2)
</span>
@{
    int i = 5;
    int j = 6;
    int z = i + j;
    @z
}
```

Demonstration: How to Use the Razor Syntax

In this demonstration, you will learn how to:

- Add code to the Page Content by using the Razor syntax

Dependency Injection

- ASP.NET Core supports dependency injection into page content
- You can inject a service using the **@inject** directive
  - `@inject <type> <instance name>`

Lesson 2: Using HTML Helpers and Tag Helpers

- Introduction to HTML Helpers and Tag Helpers
- Using HTML Action Helpers
- Demonstration: How to Use HTML Helpers
- Using Tag Helpers
- Demonstration: How to Use Tag Helpers

Introduction to HTML Helpers and Tag Helpers

- HTML helpers:
  - Use a Razor syntax
  - Make it easier to identify areas of code
  - Does not require explicit enabling of the feature

- Tag helpers:
  - Use an HTML-like syntax, as well as tag properties
  - Require explicit usage of a directive
  - Create more easily legible HTML with less immediately apparent code

Using HTML Action Helpers

- **Html.ActionLink()**

```
@Html.ActionLink("Click here to view photo 1",
   "Display", new { id = 1 })
```

```
<a href="/photo/display/1">
   Click here to view photo 1
</a>
```

- **Url.Action()**

```
<img alt="This image came from an Action"
   src="@Url.Action("GetImage", new { id = 1 })" />
```

```
<img
   alt="This image came from an action"
   src="/photo/getimage/1" })"
/>
```

Demonstration: How to Use HTML Helpers

In this demonstration, you will learn how to:

- Use the **Html.ActionLink** helper to navigate from one action to another action
- Use the **Html.ActionLink** helper to pass a parameter to an action
- Use the **Url.Action** helper to generate a path to an action

Using Tag Helpers

- Tag helpers are an alternative to HTML helpers
- Tag helpers look like regular HTML elements
- The following HTML helper and tag helper produce the same HTML:

HTML helper:

```
@Html.ActionLink("Press me", "AnotherAction")
```

Tag helper:

```
<a asp-action="AnotherAction">Press me</a>

<a asp-page="photos/edit/1">Edit Photo 1</a>
```

Using the @addTagHelper Directive

- To use tag helpers, you need to add the **@addTagHelper** directive to a page

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

- To make tag helper available to all pages, add the **@addTagHelper** directive to the **_ViewImports.cshtml** file

Demonstration: How to Use Tag Helpers

In this demonstration, you will learn how to:

- Use the anchor tag helper to navigate from one page to another page
- Pass a parameter to a handler by using the anchor tag helper
- Make tag helpers available to pages by using the **_ViewImports.cshtml** file

Lesson 3: Reusing Code

- Creating Partial Views
- Using Partial Views
- Demonstration: How to Create and Use Partial Views
- Creating View Components
- Using View Components
- Invoking View Components with Parameters
- Demonstration: How to Create and Use View Components

Creating Partial Views

- Use partial views to render identical HTML content in different locations of your web application
- Often created inside the **/Views/Shared** folder
- By convention, the names of partial views are prefixed with an underscore
  - For example: _MyPartialView.cshtml

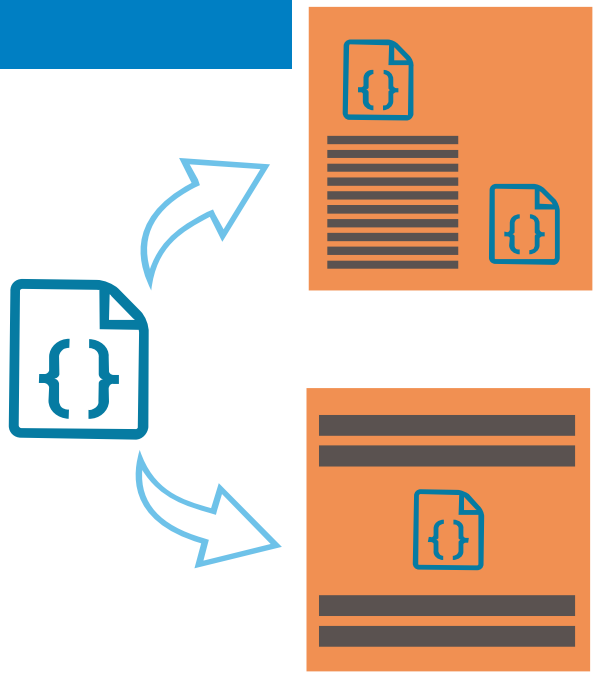
```
<p>In partial view</p>
```

Using Partial Views

You can use the **Html.PartialAsync()** method or the <partial> tag helper to render a partial view within a page content

```
@await Html.PartialAsync("_MyPartialView")
Now in page
<partial name="_MyPartialView" />
```

In partial view
Now in page
In partial view



Demonstration: How to Create and Use Partial Views

In this demonstration, you will learn how to:

- Add a partial view to a Web application
- Fill the content of the partial view
- Embed a partial view in a Page by using the `<partial>` tag helper

Using the ViewBag Property

Adding Information:

```
ViewBag.Message = "some text";

ViewBag.ServerTime = DateTime.Now;
```

Retrieving Information:

```
<p>

   Message is: @ViewBag.Message

</p>

<p>

   Server time is: @ViewBag.ServerTime.ToString()

</p>
```

Creating View Components

- You can use view components to render identical or similar HTML content in different locations
- A view component consists of two parts:
  - A class
    - Should be public and non-abstract
    - Usually derived from the **ViewComponent** base class
    - Should have a method called **InvokeAsync**, which defines its logic
  - A view
    - Can be 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

A View Component Example

- Content of a view component class located in a file named **MyViewComponent.cs**:

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

- Content of a view component view located in a file named **Default.cshtml**:

  some text

Using View Components

You can include a view component in a view by:

- using the **@Component.InvokeAsync** method:

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

```
some text
```

- using a tag helper:

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

```
some text
```

Invoking View Components with Parameters

- **A view component that gets a parameter:**

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

- **Pass a parameter to the view component:**

```
@await Component.InvokeAsync("My", 5)
<vc:My param="5"></vc:My>
```

Demonstration: How to Create and Use View Components

In this demonstration, you will learn how to:

- Add a view component class to a web application
- Add a view component view to a web application
- Embed the view component in a Page

Lab: Developing Page Content

- Exercise 1: Adding Pages to a Web Application
- Exercise 2: Adding a Partial View
- Exercise 3: Adding a View Component

Estimated Time: 60 minutes