.NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – Workshop*PLUS*

< Engineer Name >

Customer Engineer

v3.1

Module 5: Views & Razor Pages

Module Overview

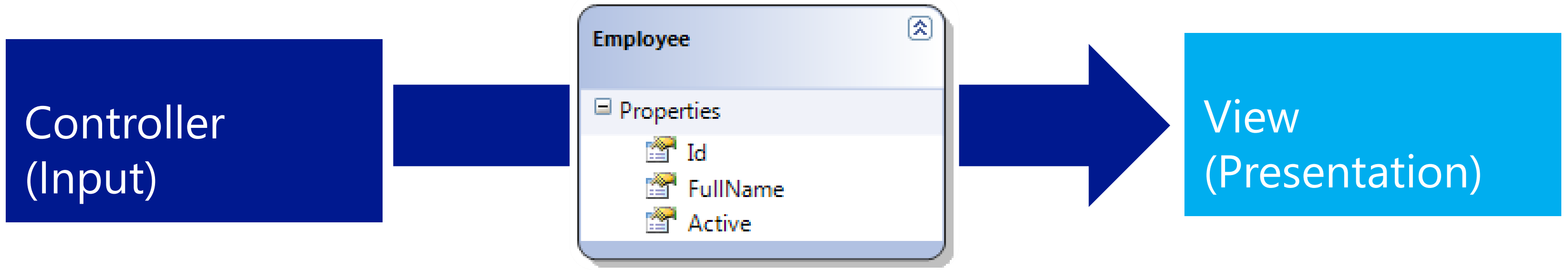
Module 5: Razor Pages & MVC Views

Section 1: View Fundamentals

Lesson: Role of Views

View

- Components that display the application's user interface
- Responsible for transforming a model into a format presentable to user
    - For web pages, View transforms the model contents to HTML



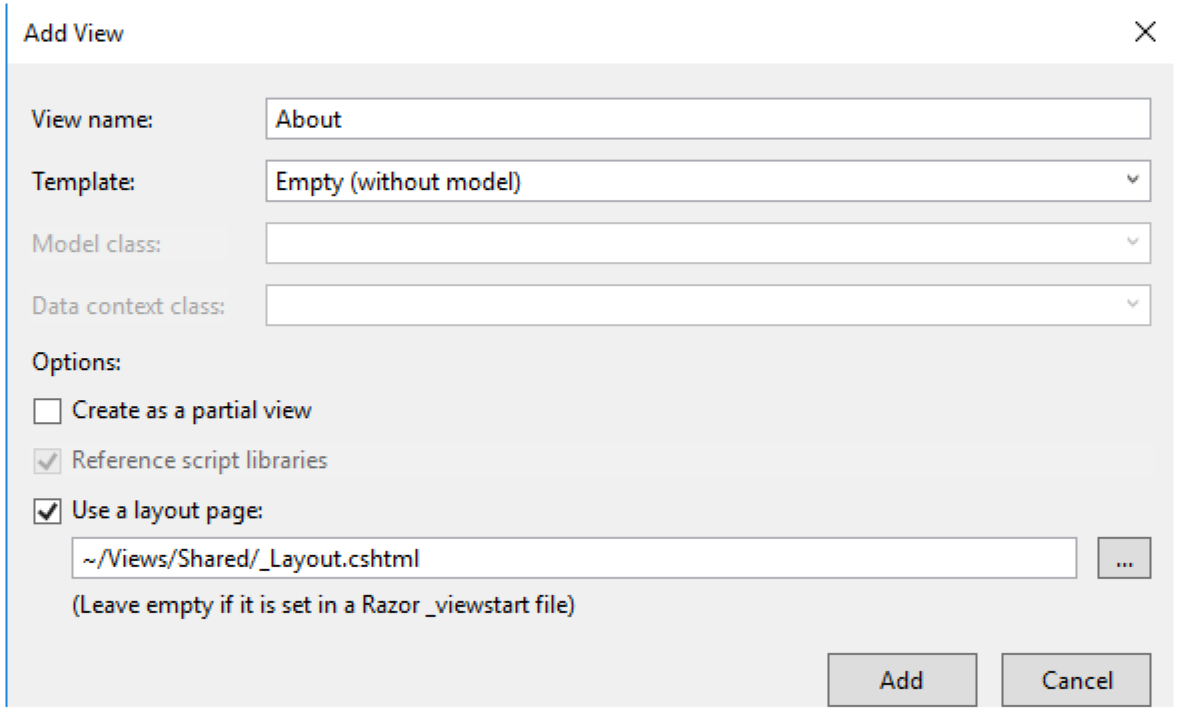
Role of a View

- View takes model data as input, and outputs it in user presentable form (for example, HTML)

- Example:
  1. User sends a URL request with query string values
  2. Controller is triggered against the request
  3. Controller handles query-string values
  4. Controller passes the values to the model
  5. Model uses the value to query the database and returns the results
  6. Controller selects a View to render the UI
  7. Controller returns the View to requesting browser

View Creation

- Views are named according to view engine
  - Razor: *.cshtml or *.vbhtml (for classic asp.net)
  - View can be created through:
    - Solution Explorer
    - Action Method



Add View

View name: About

Template: Empty (without model)

Model class:

Data context class:

Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [x] Use a layout page:

~/Views/Shared/_Layout.cshtml

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

Specifying Views

- Select View using default convention

```
public ActionResult About()
{
    ViewBag.Message = "Your app description page.";
    return View();
}
```

Views > Home > About.cshtml

- Select a particular view

```
public ActionResult About()
{
    ViewBag.Message = "Your app description page.";
    return View("AboutCompany");
}
```

Views > Home > AboutCompany.cshtml

- Select view from a different directory structure

```
public ActionResult About()
{
    ViewBag.Message = "Your app description page.";
    return View("~/Views/Home/Company/About.cshtml");
}
```

Views > Home > Company > About.cshtml

Module 5: Razor Pages & MVC Views

Section 1: Razor View Engine

Lesson: Razor Pages

Razor Pages - I

- Page-focused scenarios

- **@page** directive
    - makes the file into an MVC action (**.cshtml**)
    - handles requests directly, without going through a controller

- PageModel class keeps code clean in different file (**.cshtml.cs**)
    - By convention, razor page file and class have the same name
    - Example: Create.cshtml, Create.cshtml.cs

- **@model** represents the PageModel class implemented.

- Code file helps to implement methods to handle request sync or async:
    - OnGet, OnPost,
    - OnGetAsync, OnPostAsync
    - OnGet…, OnPost…

Razor Pages - II

- Located in **Pages** folder
  - /Pages/Index.cshtml
  - /Pages/Index.cshtml.cs

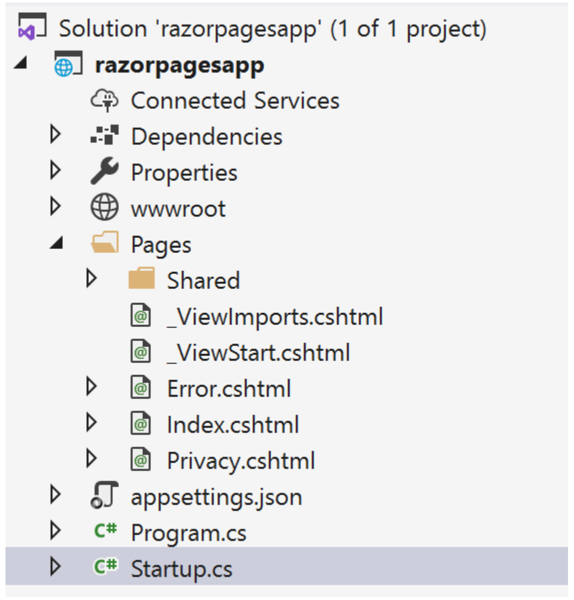
- Url determined by file path

File name and path	**matching URL**
/Pages/Index.cshtml	`/` or `/Index`
/Pages/Contact.cshtml	`/Contact`
/Pages/Store/Contact.cshtml	`/Store/Contact`
/Pages/Store/Index.cshtml	`/Store` or `/Store/Index`

Razor Pages - III

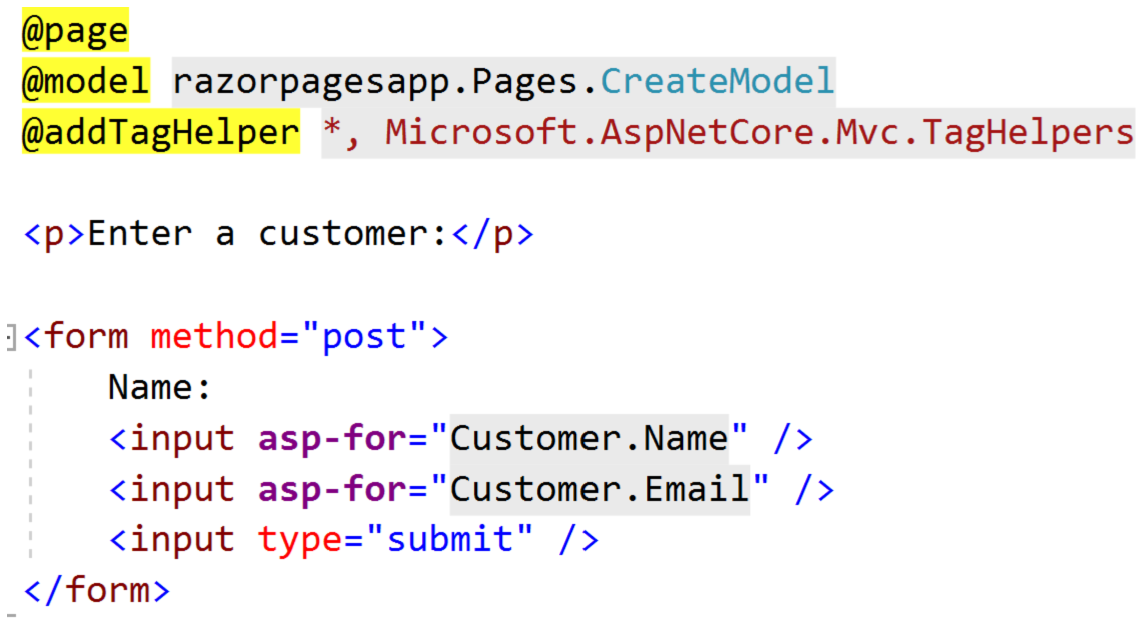
- In Startup class in Startup.cs file:

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddRazorPages();
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
        …
        app.UseEndpoints(endpoints =>
        {
                endpoints.MapRazorPages();
        });
}
```



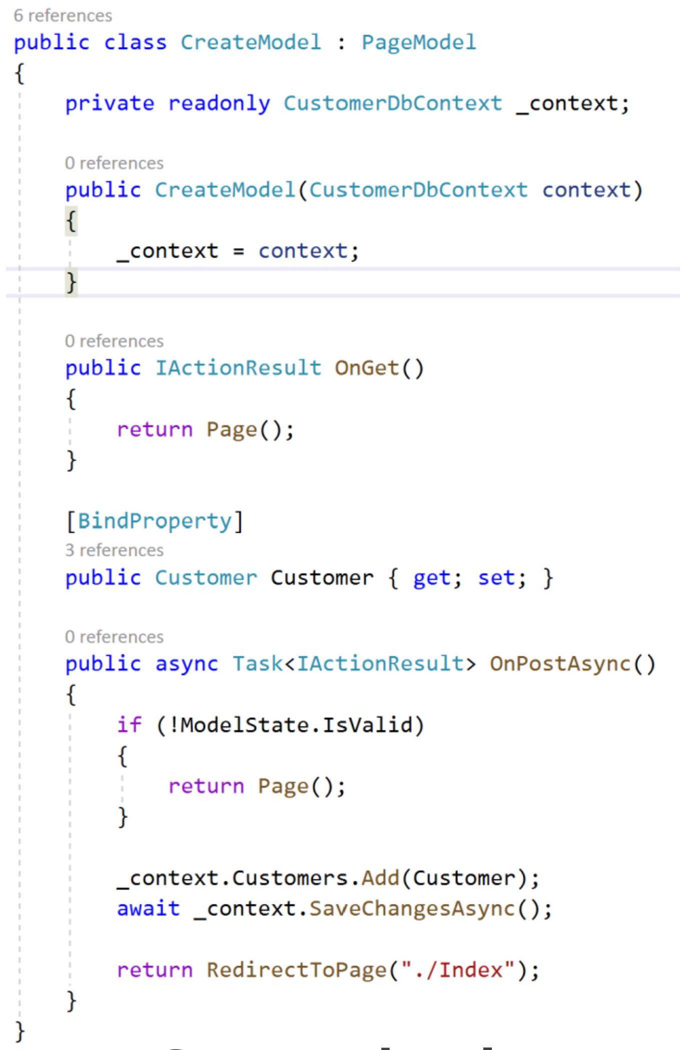
```
Solution 'razorpagesapp' (1 of 1 project)
  razorpagesapp
    Connected Services
    Dependencies
    Properties
    wwwroot
    Pages
      Shared
      _ViewImports.cshtml
      _ViewStart.cshtml
      Error.cshtml
      Index.cshtml
      Privacy.cshtml
    appsettings.json
    Program.cs
    Startup.cs
```

Razor Pages - IV



```
@page
@model razorpagesapp.Pages.CreateModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<p>Enter a customer:</p>

<form method="post">
    Name:
    <input asp-for="Customer.Name" />
    <input asp-for="Customer.Email" />
    <input type="submit" />
</form>
```

Create.cshtml



```
6 references
public class CreateModel : PageModel
{
    private readonly CustomerDbContext _context;

    0 references
    public CreateModel(CustomerDbContext context)
    {
        _context = context;
    }

    0 references
    public IActionResult OnGet()
    {
        return Page();
    }

    [BindProperty]
    3 references
    public Customer Customer { get; set; }

    0 references
    public async Task<IActionResult> OnPostAsync()
    {
        if (!ModelState.IsValid)
        {
            return Page();
        }

        _context.Customers.Add(Customer);
        await _context.SaveChangesAsync();

        return RedirectToPage("./Index");
    }
}
```

Create.cshtml.cs

Demo: Razor Pages

Module 5: Razor Pages & MVC Views

Section 1: View Fundamentals

Lesson: Passing Data to Views

ViewData

- Represents a container to pass data from a Controller to View and vice versa

- ViewData exposes an instance of *ViewDataDictionary*

- Data passed from Controller to View using ViewData
  - `ViewData["color"] = "Red";`

- Data accessed from View
  - `@ViewData ["color"]`

ViewBag

- Represents a dynamic wrapper around ViewData
  - `ViewData["Color"] > ViewBag.Color`

- ViewBag only works with valid C# identifiers
  - `ViewData["Car Color"] = "Red";`
  - ~~ViewBag.Car Color;~~

- ViewBag dynamic value cannot be used in extension methods
  - ~~`@Html.TextBox("Name", ViewBag.Color);`~~
  - `@Html.TextBox("Name", ViewData["Color"]);`

TempData

- Temporary Data
- Passing data between the current and next HTTP requests
- Data passed from Controller to View using TempData
  - `TempData["color"] = "Red";`
- Data accessed from View
  - `@TempData["color"]`
- TempData object could yield results differently than expected because the next request origin cannot be guaranteed!

TempData

- In this example, message is stored in TempData, but it is not available for all the Action methods calls.
- Privacy could raise an exception.
- Use method **TempData.Peek** or **TempData.Keep** to retain values for next request.

```
public IActionResult Index()
{
    TempData["message"] = "Message from index";
    return View();
}

0 references
public IActionResult FirstCall()
{
    //Message is used. Next request can call this data
    ViewBag.Message = TempData.Peek("message");
    //use TempData.Keep() to retain the values in TempData dictionary

    return View();
}

0 references
public IActionResult SecondCall()
{
    //when this request ends, message is not available anymore
    ViewBag.Message = TempData["message"];
    return View();
}

0 references
public IActionResult Privacy()
{
    //will throw an exception if request is made after SecondCall
    ViewBag.Message = TempData["message"];
    return View();
}
```

Strongly Typed Views

- Strongly typed to the type TModel
- Contains Model property
- Enables compile time code checking

Strongly Typed View

Controller

```
public ActionResult Detail() {
    …
    return View(person);
}
```

View

```
@model App.Models.Person
@Model.Name
@Model.Age
```

vs.

Standard View

Controller

```
public ActionResult Detail() {
    …
    return View();
}
```

View

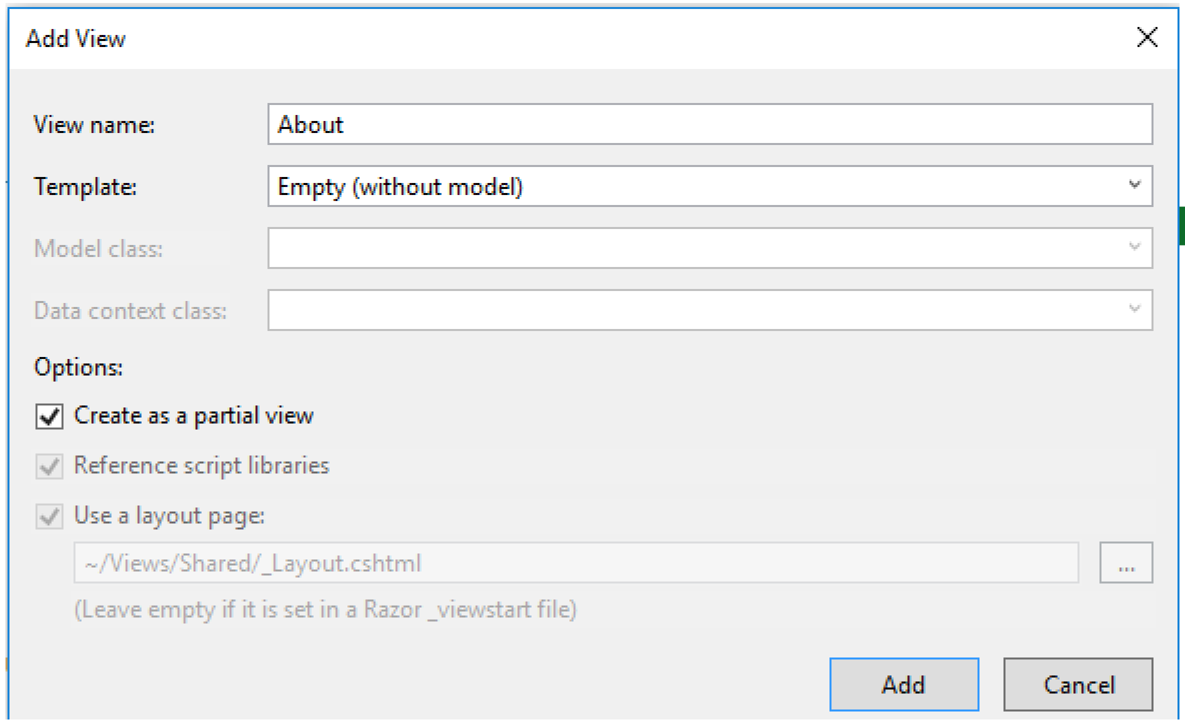
```
@ViewData["Name"]
@ViewData["Age"]
```

Partial View

- Reusable component filled with content and code
  - Theoretically plays the same role as *web controls* in ASP.NET web pages
- Useful in various scenarios:
  - Logon dialog box
  - Time widget to display time on all views of the application
- Can be rendered inside layout or regular views
- Uses ViewData and ViewBag to share data
- Partial view render:

```
<div>
    @Html.Partial("_FeaturedProduct")
</div>
```

Partial View (continued)


Add View

View name: About

Template: Empty (without model)

Model class:

Data context class:

Options:

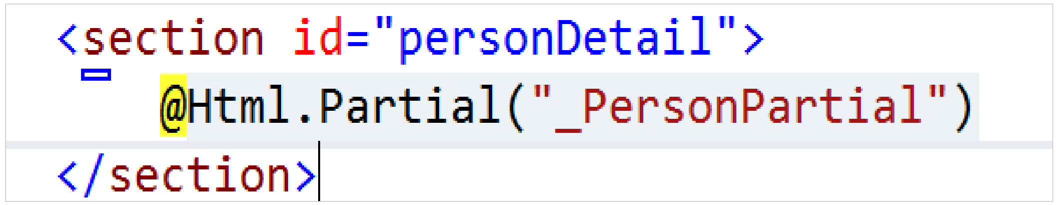
[✓] Create as a partial view

[✓] Reference script libraries

[✓] Use a layout page:

~/Views/Shared/_Layout.cshtml

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


```
<section id="personDetail">
    @Html.Partial("_PersonPartial")
</section>
```

Module 5: Razor Pages & MVC Views

Section 1: View Fundamentals

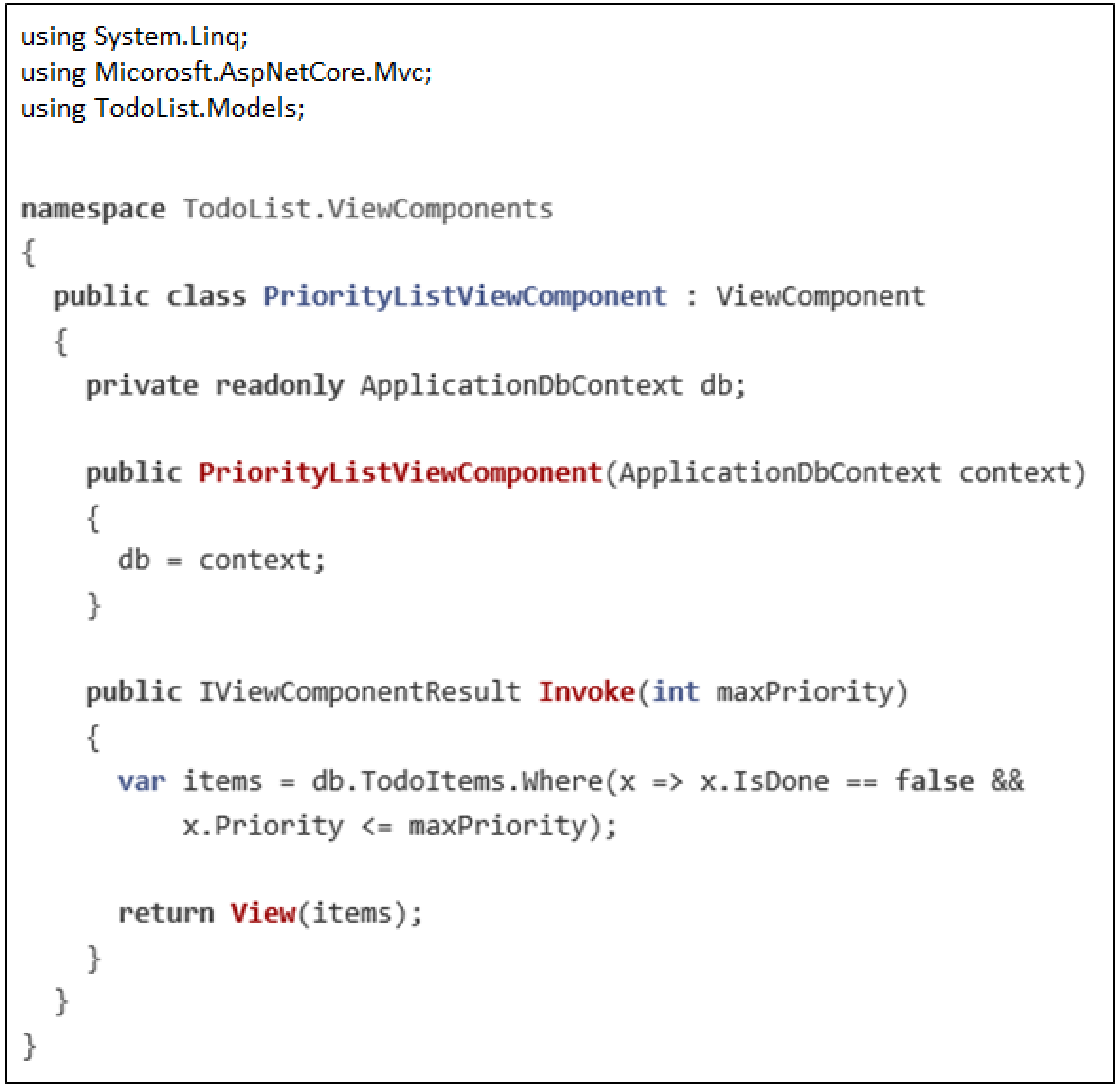
Lesson: View Components

View Component

- Similar to partial views (Partial View does not have a “code-behind”)
- Introduced in ASP.NET MVC Core
- Responds like a mini-controller, responsible for rendering a chunk
- Example scenarios for use:
  - Dynamic navigation menus
  - Tag cloud (where it queries the database)
  - Logon panel
  - Shopping cart
  - Sidebar content on a blog
- Does not use model binding; takes input data parameter

View Component [Class]

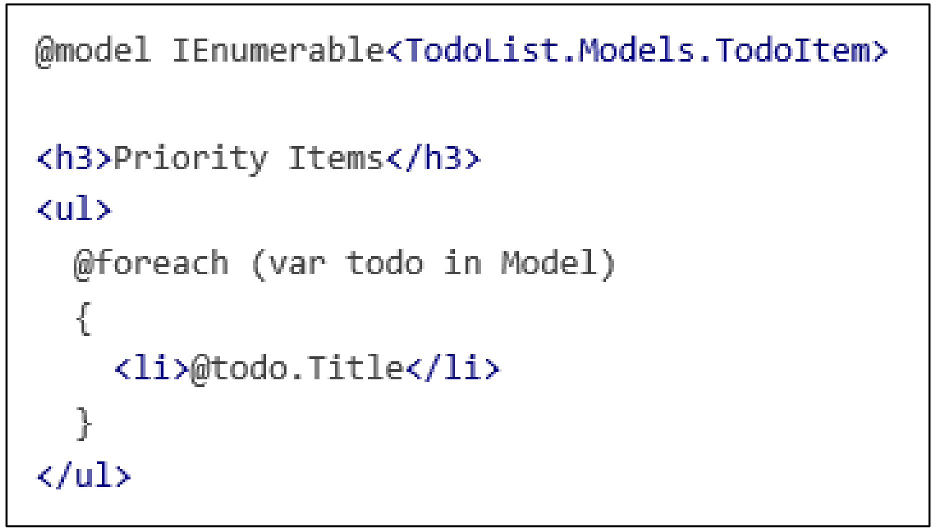
- Derive from *ViewComponent*
- Decorate with *[ViewComponent]* attribute
- Derive from a class with *[ViewComponent]* attribute
- Class name ending with the suffix *ViewComponent*
- Public, non-nested, and non-abstract class like Controllers



```
using System.Linq;
using Micorosft.AspNetCore.Mvc;
using TodoList.Models;


namespace TodoList.ViewComponents
{
  public class PriorityListViewComponent : ViewComponent
  {
    private readonly ApplicationDbContext db;

    public PriorityListViewComponent(ApplicationDbContext context)
    {
      db = context;
    }

    public IViewComponentResult Invoke(int maxPriority)
    {
      var items = db.TodoItems.Where(x => x.IsDone == false &&
          x.Priority <= maxPriority);

      return View(items);
    }
  }
}
```

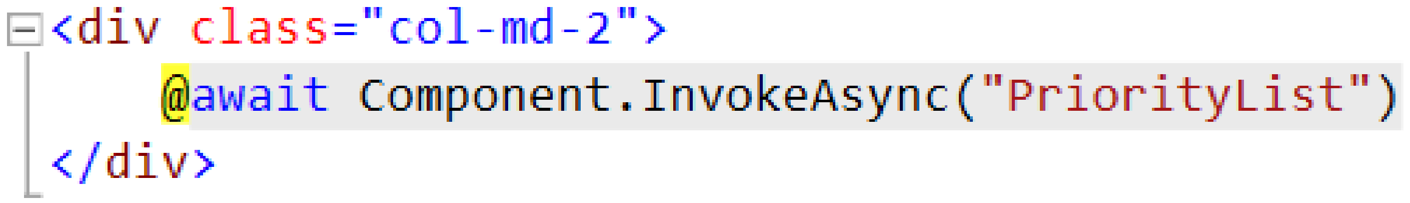
PriorityListViewComponent.cs

View Component [View]



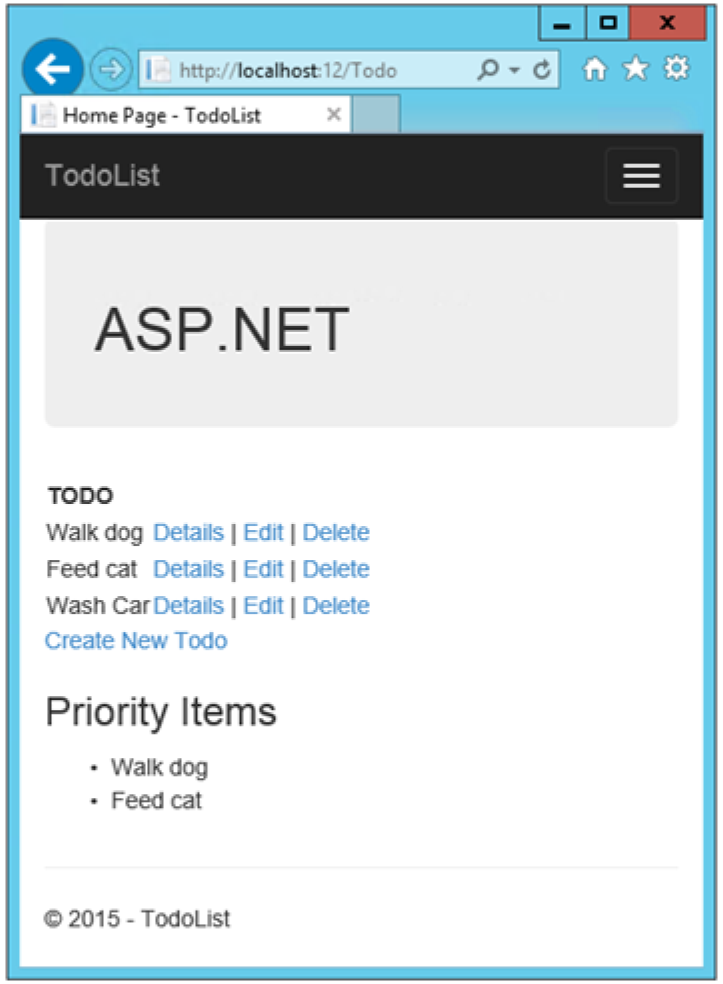
```
@model IEnumerable<TodoList.Models.TodoItem>

<h3>Priority Items</h3>
<ul>
  @foreach (var todo in Model)
  {
    <li>@todo.Title</li>
  }
</ul>
```

Views\Todo\Components\PriorityList\Default.cshtml



```
<div class="col-md-2">
      @await Component.InvokeAsync("PriorityList")
</div>
```

Views\todo\index.cshtml



http://localhost:12/Todo
Home Page - TodoList
TodoList
ASP.NET
TODO
Walk dog Details | Edit | Delete
Feed cat Details | Edit | Delete
Wash Car Details | Edit | Delete
Create New Todo
Priority Items
- Walk dog
- Feed cat

© 2015 - TodoList

View using View Component

Asynchronous View Component

```
public class PriorityListViewComponent : ViewComponent
{
  private readonly ApplicationDbContext db;

  public PriorityListViewComponent(ApplicationDbContext context)
  {
    db = context;
  }

  // Synchronous Invoke removed.

  public async Task<IViewComponentResult> InvokeAsync(int maxPriority, bool isDone)
  {
    var items = await GetItemsAsync(maxPriority, isDone);
    return View(items);
  }

  private Task<IQueryable<TodoItem>> GetItemsAsync(int maxPriority, bool isDone)
  {
    return Task.FromResult(GetItems(maxPriority, isDone));
  }
  private IQueryable<TodoItem> GetItems(int maxPriority, bool isDone)
  {
    var items = db.TodoItems.Where(x => x.IsDone == isDone &&
        x.Priority <= maxPriority);

    string msg = "Priority <= " + maxPriority.ToString() +
           " && isDone == " + isDone.ToString();
    ViewBag.PriorityMessage = msg;

    return items;
  }
}
```

Demo: View Components

Module 5: Razor Pages & MVC Views

Section 2: Razor View Engine

Lesson: Razor View Engine

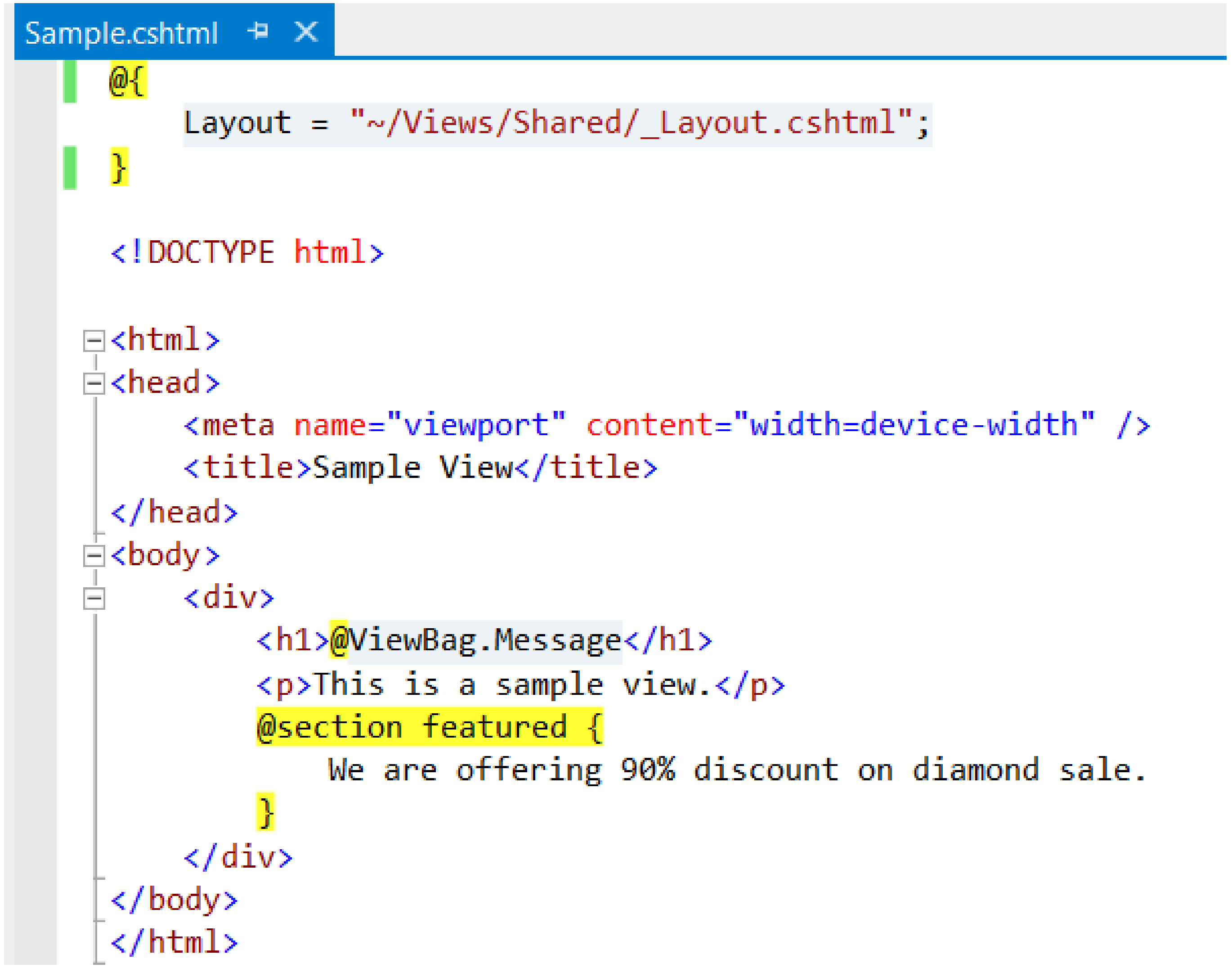
View Engines

- ASP.NET MVC comes with Razor view engine by default
  - ASPX view engine not supported by ASP.NET Core MVC
- Other view engines:
  - Brail
  - NDjango
  - NHaml
  - NVelocity
  - SharpTiles
  - Spark
  - StringTemplate
  - XSLT

Razor View Engine

- Clean, lightweight, and simple view engine for ASP.NET MVC
- Default view engine for ASP.NET MVC 3.0 onwards
- Minimizes the amount of syntax and extra characters
- Reduces syntax between code and view markup
- Full IntelliSense support in Visual Studio

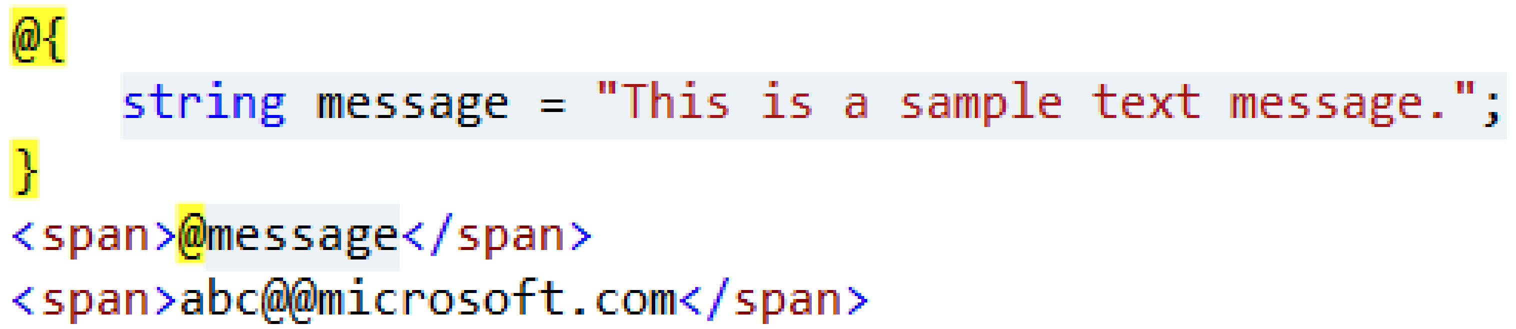
Razor View


Sample.cshtml

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Sample View</title>
</head>
<body>
    <div>
        <h1>@ViewBag.Message</h1>
        <p>This is a sample view.</p>
        @section featured {
            We are offering 90% discount on diamond sale.
        }
    </div>
</body>
</html>
```

Code Expressions

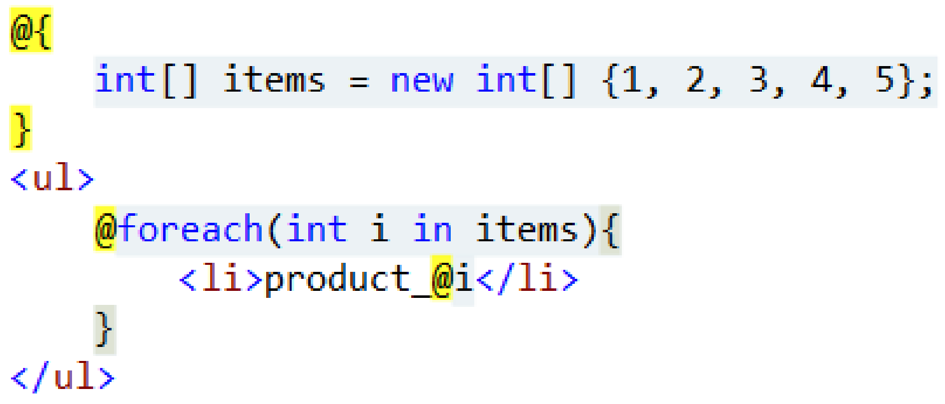
- ‘**@**’ sign used for transition from markup to code and back
- @@ used as an escape sequence



```
@{
    string message = "This is a sample text message.";
}
<span>@message</span>
<span>abc@@microsoft.com</span>
```

Code Blocks

- Razor supports code blocks within a view
- Code blocks may automatically be transformed into markup



```
@{
    int[] items = new int[] {1, 2, 3, 4, 5};
}
<ul>
    @foreach(int i in items){
        <li>product_@i</li>
    }
</ul>
```

Razor Syntax

Razor Syntax
Implicit code expression `<span>@model.Message</span>`
Explicit code expression `<span>ISBN@(isbn)</span>`

Unencoded code expression
`<span>`
`    @Html.Raw(model.AlertMessage)`
`</span>`

Code block
`@{`
`    int x = 567;`
`    string s = “Microsoft”;`
`}`

Razor Syntax (continued)

Razor Syntax

Code and markup
`@foreach(var item in items) {`
`   <span>Item No.@item.Id </span>`
`}`

Code and plain text
`@if(showMessage) {`
`    <text>`
`        Text Message.`
`    </text>`
`}`
 	
Or
 	
`@if(showMessage) {`
`    @:Text Message.`
`}`

Razor Syntax (continued)

Razor Syntax
Comments @* Multi-line comment Product name: @ViewBag.Product *@

Demo: Razor View Engine

HTML Encoding

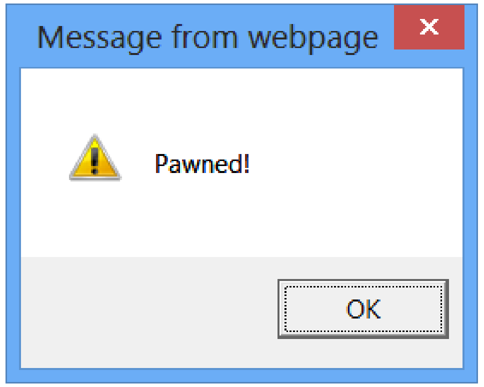
- Razor expressions are always HTML encoded!
  - Defense against Cross-Site Scripting (XSS) attack, etc.

```
@{string alert = "<script>alert('Pawned!')</script>";}
<span>@alert</span>
```

<script>alert('Pawned!')</script>

- Use Html.Raw( ) for showing HTML markup

```
@{string alert = "<script>alert('Pawned!')</script>";}
<span>@Html.Raw(alert)</span>
```



Demo: Importance of HTML Encoding

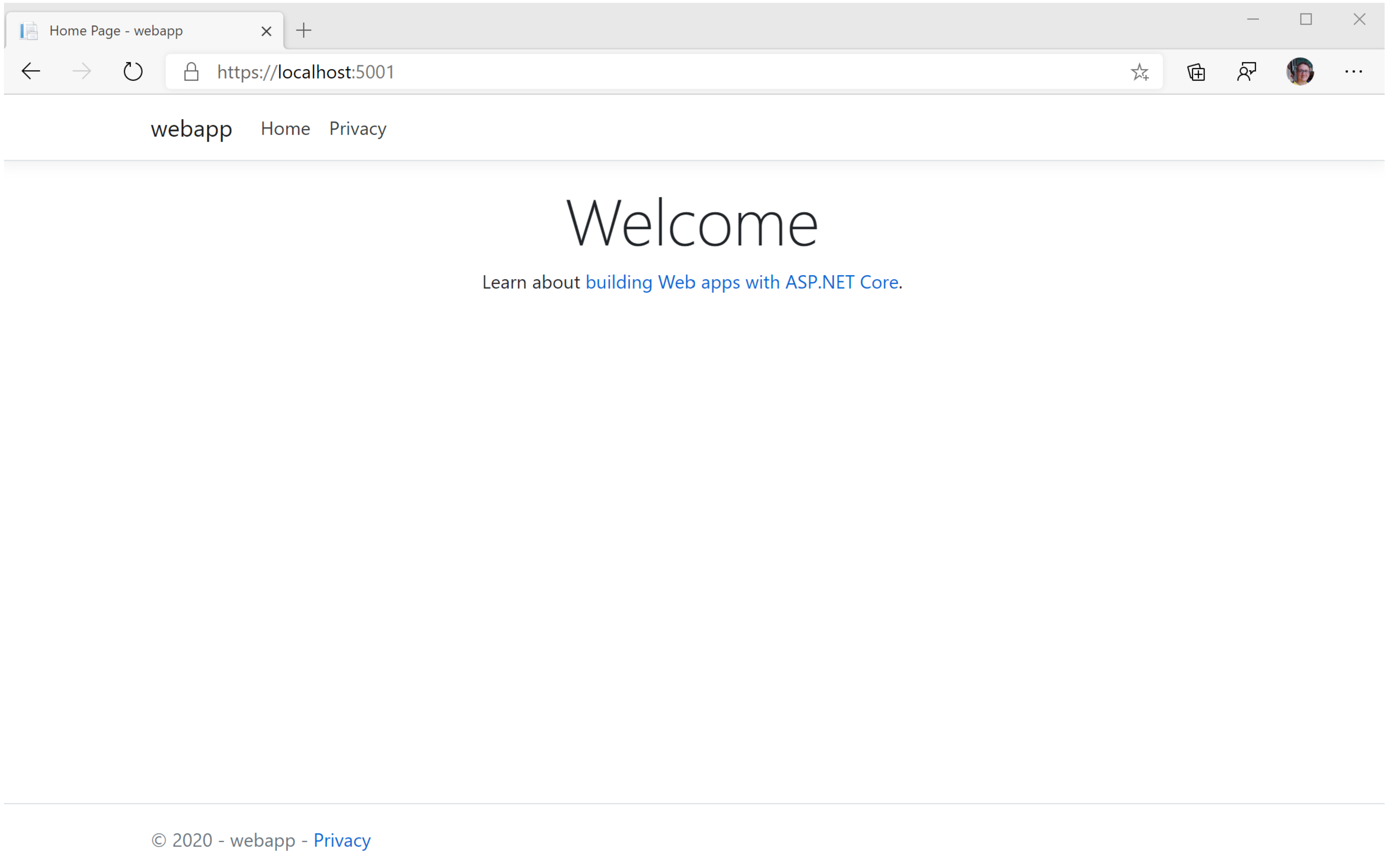
Demo: Model Binding

Module 5: Razor Pages & MVC Views

Section 2: Razor View Engine

Lesson: Layouts and Sections

Layouts – Default ASP.NET MVC Template


Home Page - webapp

https://localhost:5001

webapp Home Privacy

Welcome

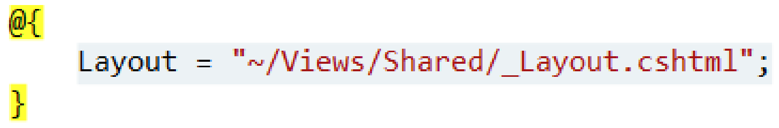
Learn about building Web apps with ASP.NET Core.

© 2020 - webapp - Privacy

Layouts

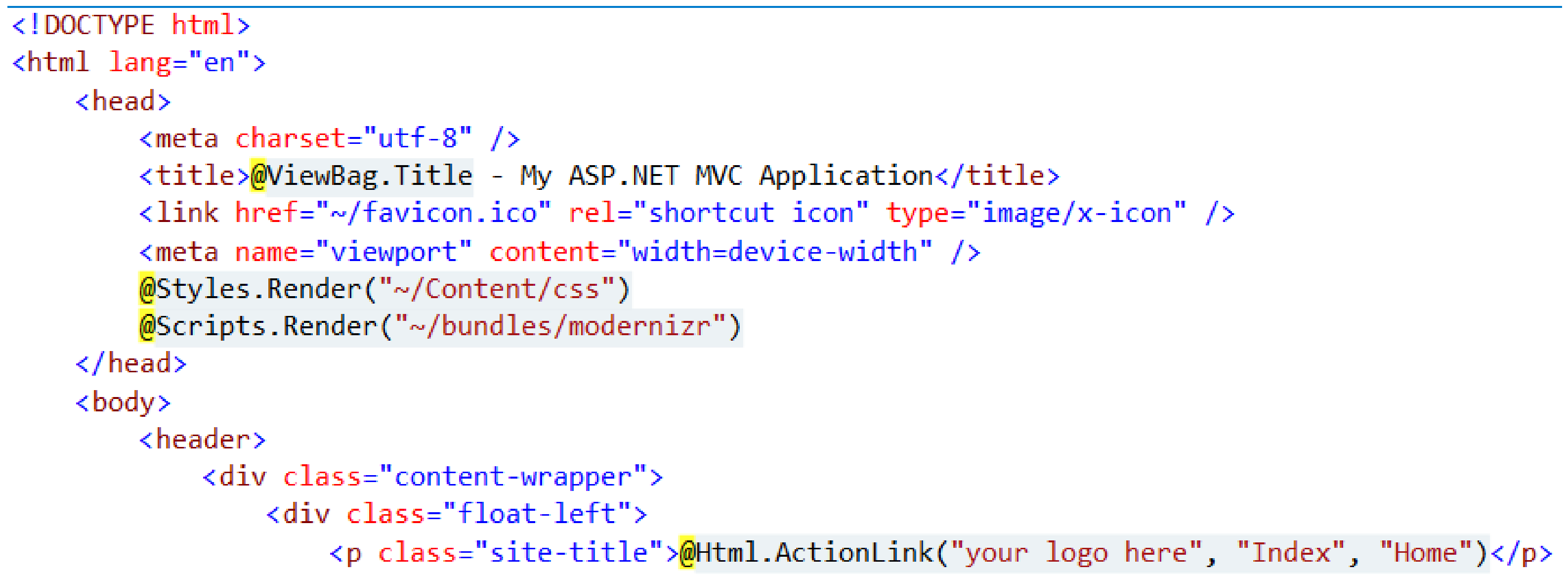
- Layouts are to views what Master Pages are to web pages in ASP.NET
- Layout defines a common template for ASP.NET MVC site
- @RenderBody( ) defines placeholder for view body

_ViewStart.cshtml



```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

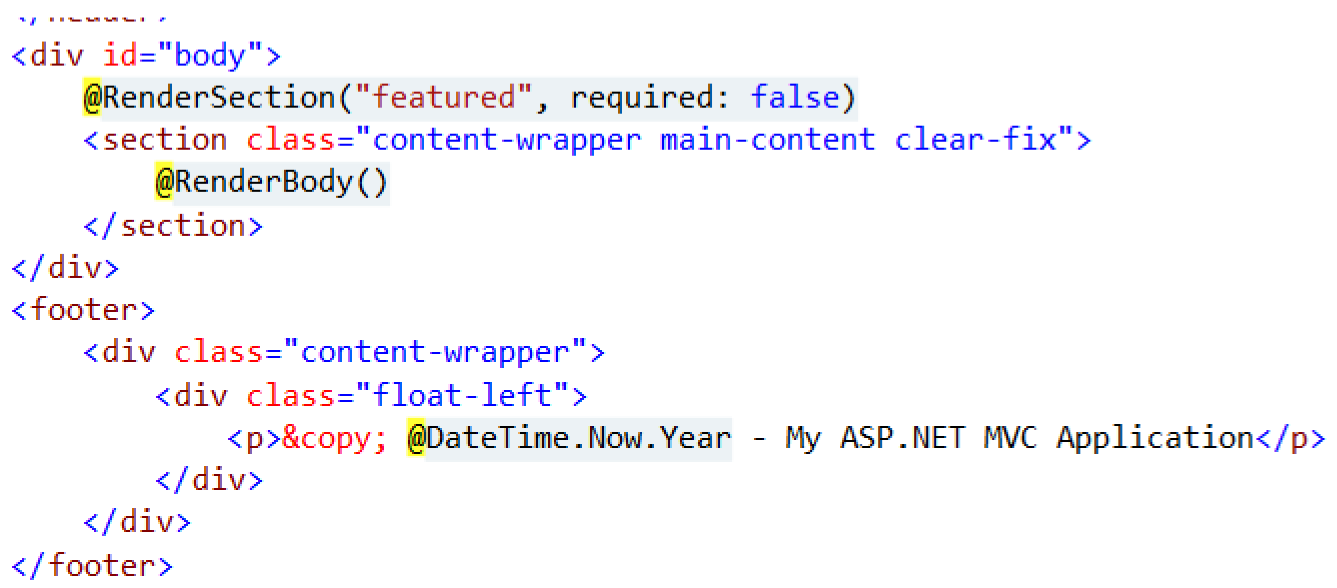
_Layout.cshtml



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>@ViewBag.Title - My ASP.NET MVC Application</title>
        <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        <meta name="viewport" content="width=device-width" />
        @Styles.Render("~/Content/css")
        @Scripts.Render("~/bundles/modernizr")
    </head>
    <body>
        <header>
            <div class="content-wrapper">
                <div class="float-left">
                    <p class="site-title">@Html.ActionLink("your logo here", "Index", "Home")</p>
```

Layout Sections

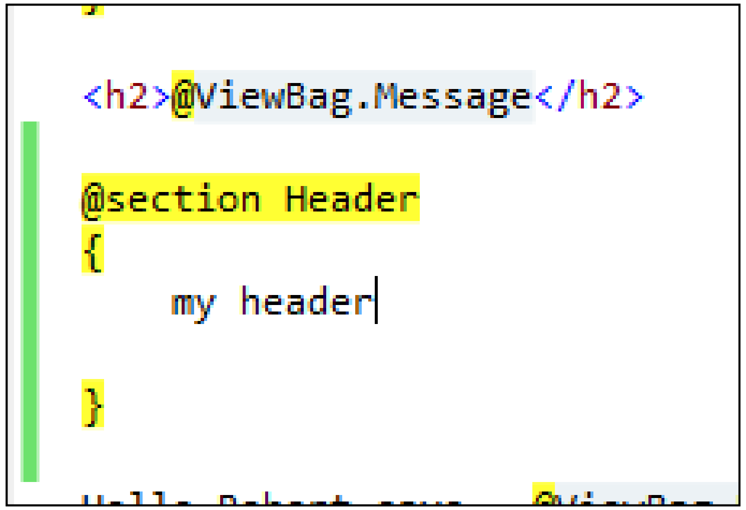
- Layout may have multiple sections
- View must provide content for all layout sections, unless explicitly made optional
- @RenderSection(…) defines placeholder for layout sections



```
<div id="body">
    @RenderSection("featured", required: false)
    <section class="content-wrapper main-content clear-fix">
        @RenderBody()
    </section>
</div>
<footer>
    <div class="content-wrapper">
        <div class="float-left">
            <p>&copy; @DateTime.Now.Year - My ASP.NET MVC Application</p>
        </div>
    </div>
</footer>
```

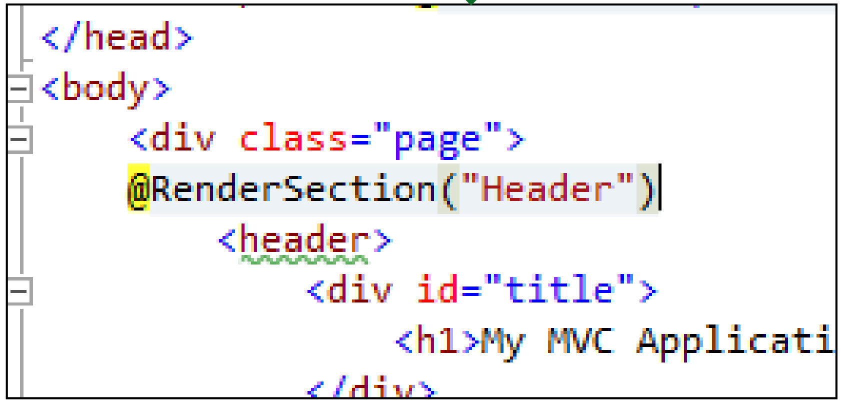
Sections

- A view can define only the sections that are referred to in the layout



```
<h2>@ViewBag.Message</h2>

@section Header
{
    my header

}
```

index.cshtml

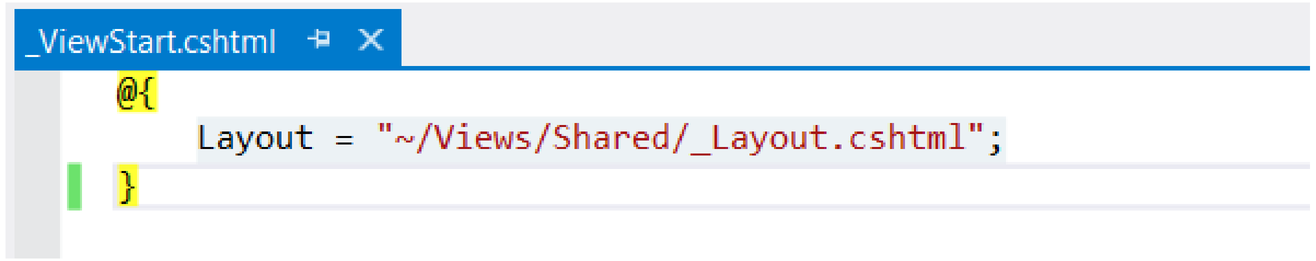


```
</head>
<body>
    <div class="page">
    @RenderSection("Header")
        <header>
            <div id="title">
                <h1>My MVC Applicati
```

_layout.cshtml

ViewStart

- _ViewStart.cshtml is used to include the same layout in all views by default
- Default layout can be overridden for specific views
    - Blank layout property means no layout has been defined

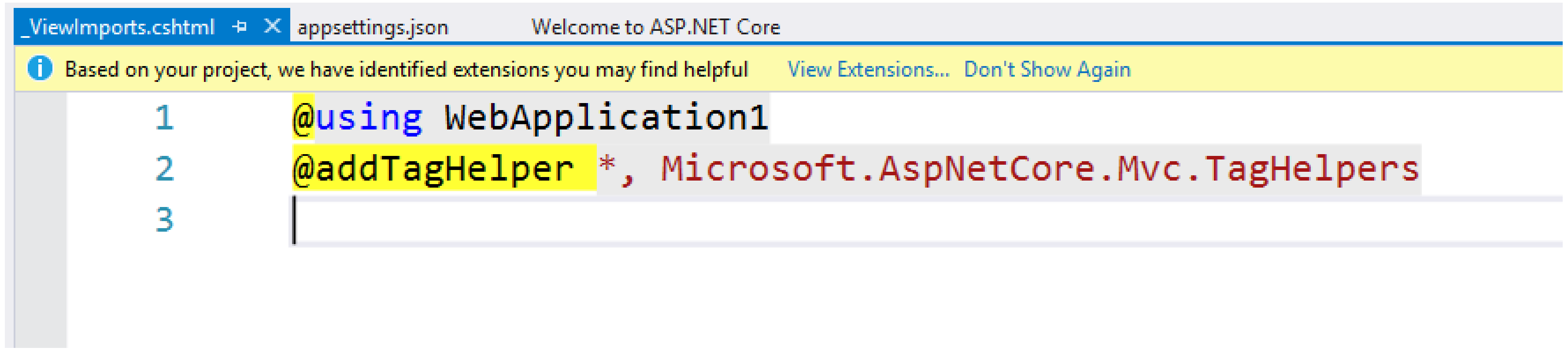


```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

_ViewStart.cshtml

View Imports

- _ViewImports.cshtml is used to import all the namespaces used by Views
- Views can add specific imports in respective files
- Tag Helper global scope is set here



```
@using WebApplication1
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

```

_ViewImports.cshtml

Demo: Layout and Sections

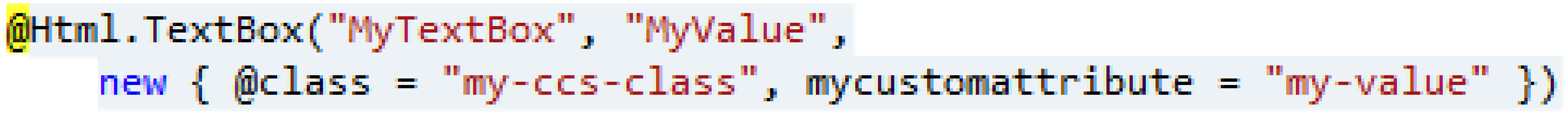
Module 5: Razor Pages & MVC Views

Section 2: Razor View Engine

Lesson: HTML Helpers, Display, and Editor Templates

Built-in HTML Helpers

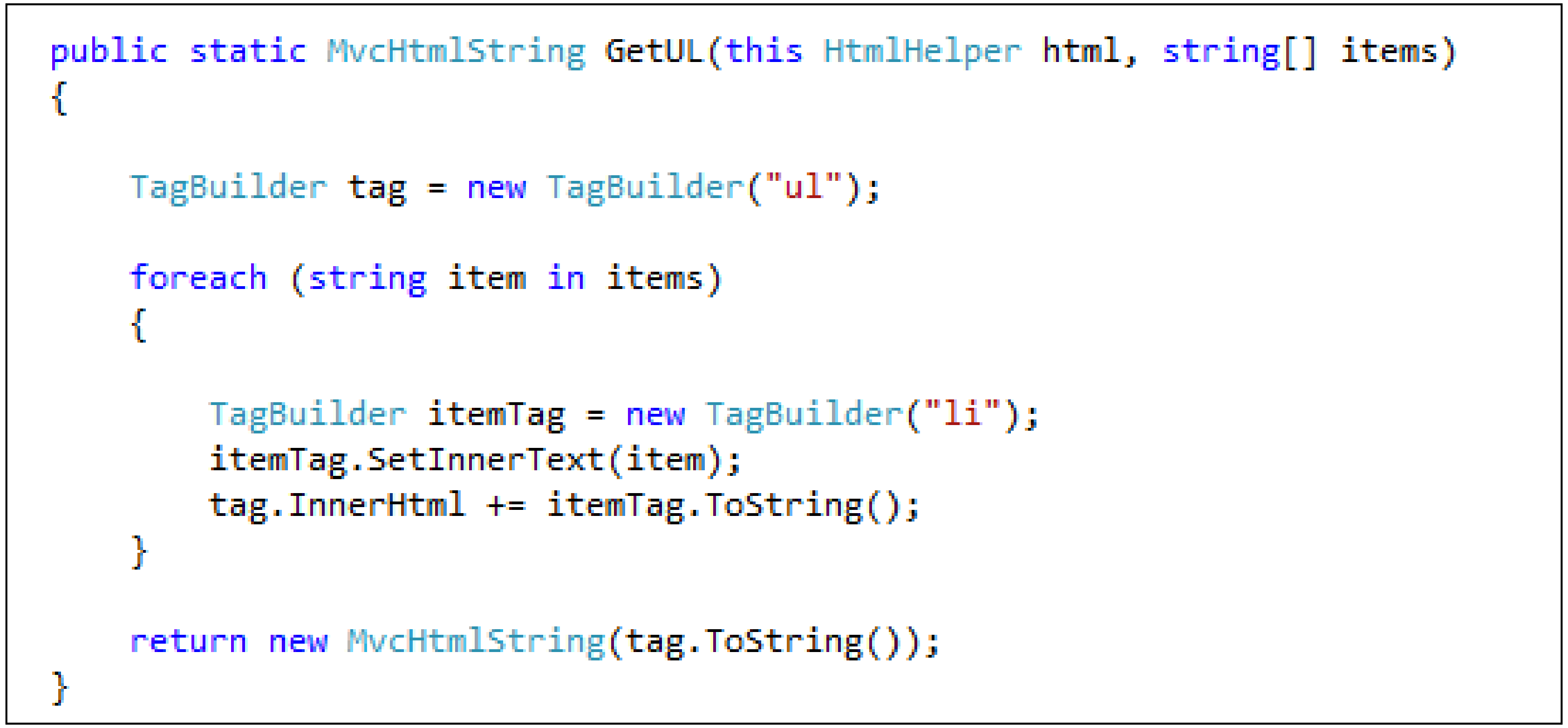
- Html.**CheckBox**("myCheckbox", false)
- Html.**Hidden**("myHidden", "val")
- Html.**RadioButton**("myRadiobutton", "val", true)
- Html.**Password**("myPassword", "val")
- Html.**TextArea**("myTextarea", "val", 5, 20, null)
- Html.**TextBox**("myTextbox", "val")



```
@Html.TextBox("MyTextBox", "MyValue",
    new { @class = "my-ccs-class", mycustomattribute = "my-value" })
```

HTML Helpers

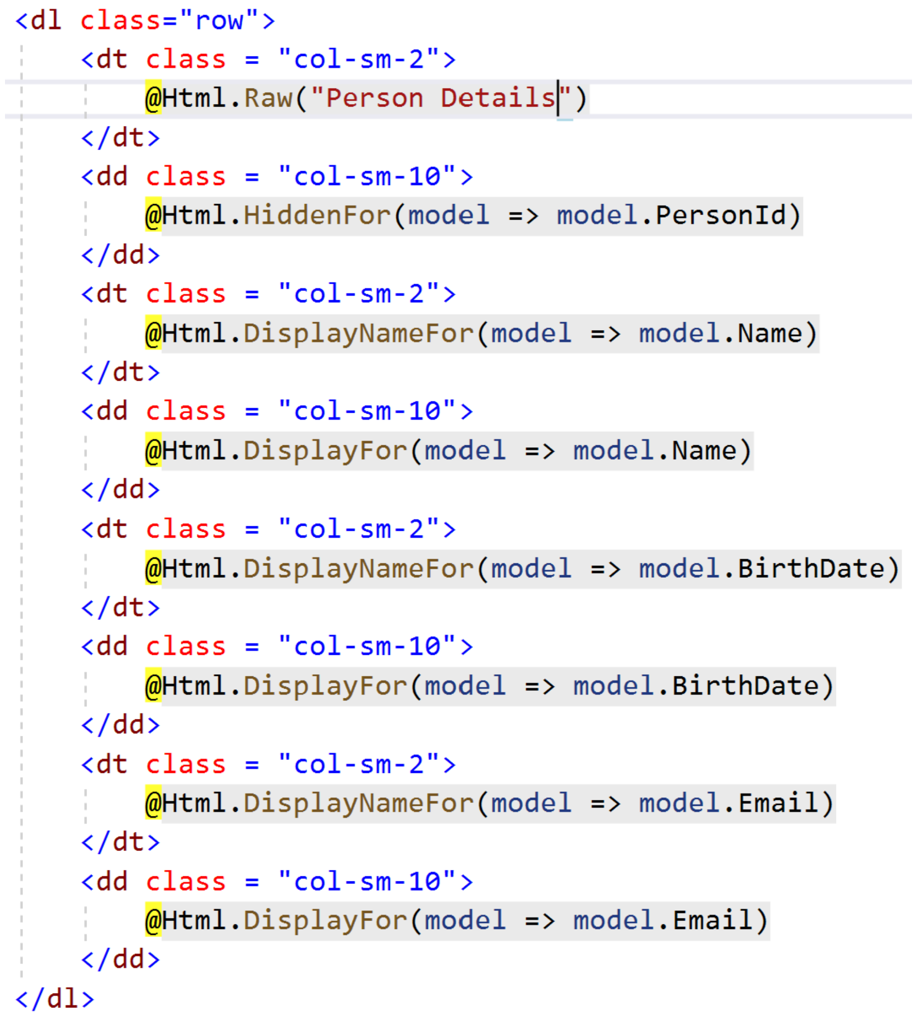
- External helpers are like regular extension methods and it takes the first parameter to HtmlHelper object



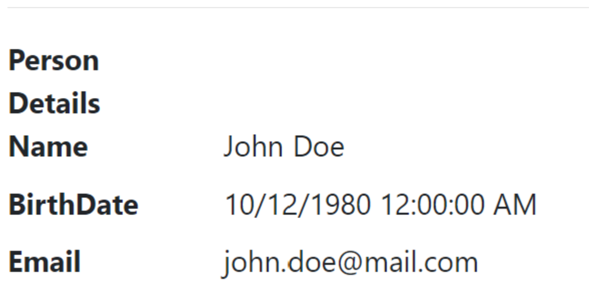
```
public static MvcHtmlString GetUL(this HtmlHelper html, string[] items)
{

    TagBuilder tag = new TagBuilder("ul");

    foreach (string item in items)
    {

        TagBuilder itemTag = new TagBuilder("li");
        itemTag.SetInnerText(item);
        tag.InnerHtml += itemTag.ToString();
    }

    return new MvcHtmlString(tag.ToString());
}
```

Built-in Display Templates

- EmailAddress
- HiddenInput
- HTML
- Text and Raw
- URL
- Collection
- Boolean
- Decimal
- String
- Object



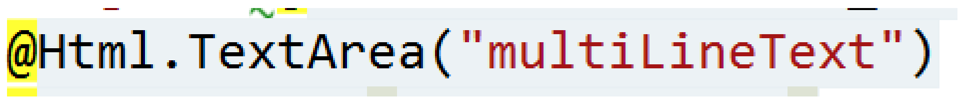
```
<dl class="row">
    <dt class = "col-sm-2">
        @Html.Raw("Person Details")
    </dt>
    <dd class = "col-sm-10">
        @Html.HiddenFor(model => model.PersonId)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Name)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Name)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.BirthDate)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.BirthDate)
    </dd>
    <dt class = "col-sm-2">
        @Html.DisplayNameFor(model => model.Email)
    </dt>
    <dd class = "col-sm-10">
        @Html.DisplayFor(model => model.Email)
    </dd>
</dl>
```



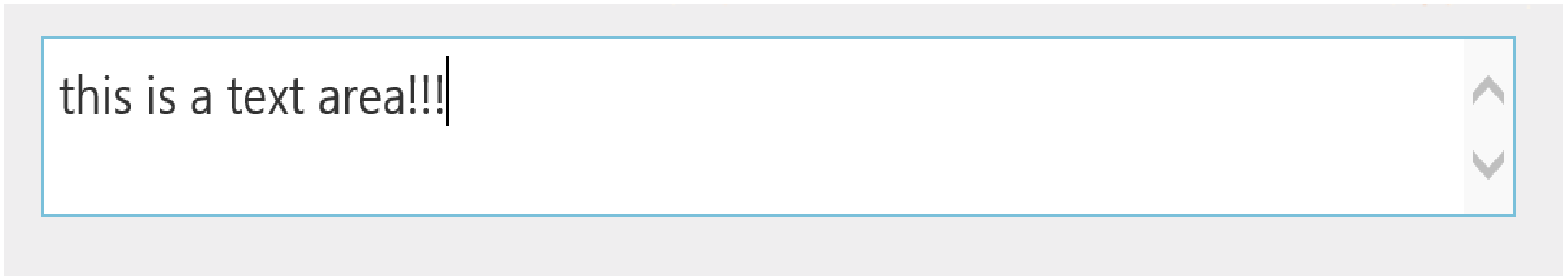
Person Details	
Name	John Doe
BirthDate	10/12/1980 12:00:00 AM
Email	john.doe@mail.com

Built-in Editor Templates

- HiddenInput
- MultilineText
- Password
- Text
- Collection
- Boolean
- Decimal
- String
- Object

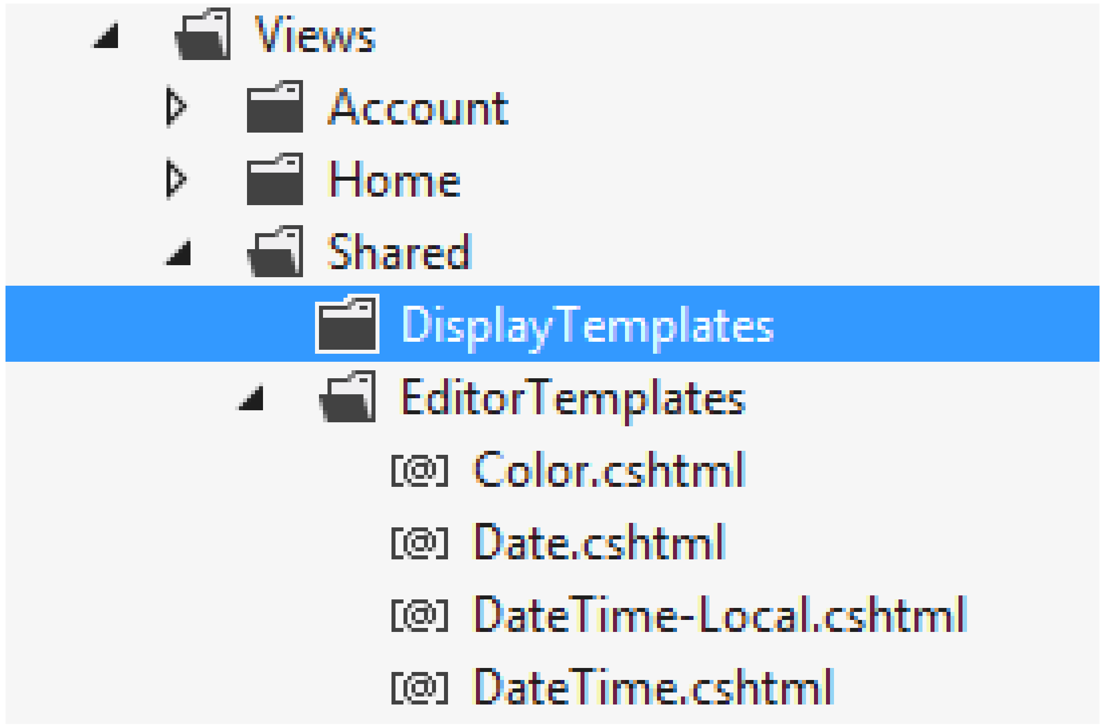


```
@Html.TextArea("multiLineText")
```



this is a text area!!!

Display and Editor Templates



```
◢ Views
  ▷ Account
  ▷ Home
  ◢ Shared
      DisplayTemplates
    ◢ EditorTemplates
        Color.cshtml
        Date.cshtml
        DateTime-Local.cshtml
        DateTime.cshtml
```

Demo: Editor

Module 5: Razor Pages & MVC Views

Section 2: Razor View Engine

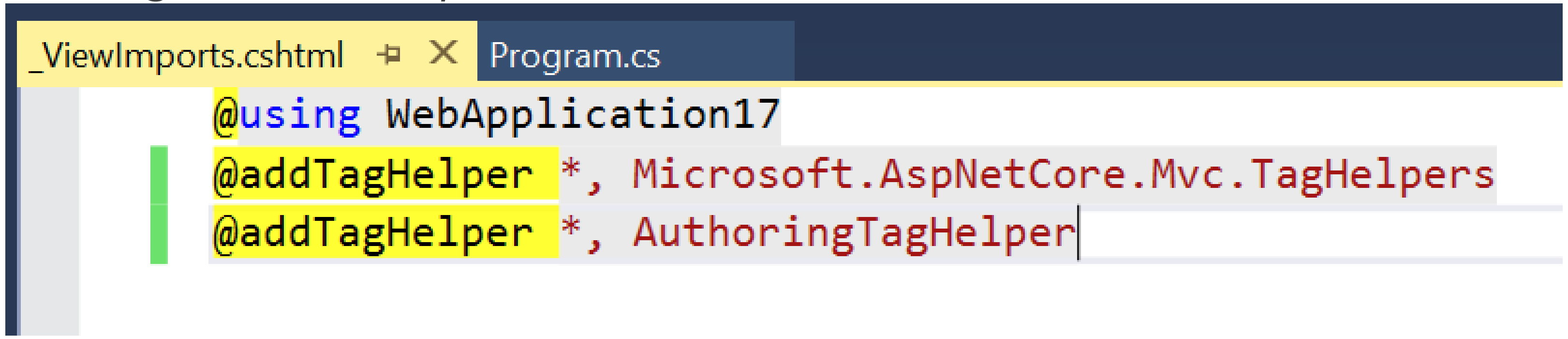
Lesson: Tag Helpers

Tag Helpers

- Enable the server-side code to participate in creating and rendering the HTML elements in Razor
- HTML-friendly development experience
- Rich IntelliSense environment for creating HTML and Razor markup
- Produces maintainable code using information available on server
    - ImageTagHelper appends version number to image name to resolve caching
- Visual Studio Tooling enabled by **Microsoft.AspNetCore.Tooling.Razor** NuGet package

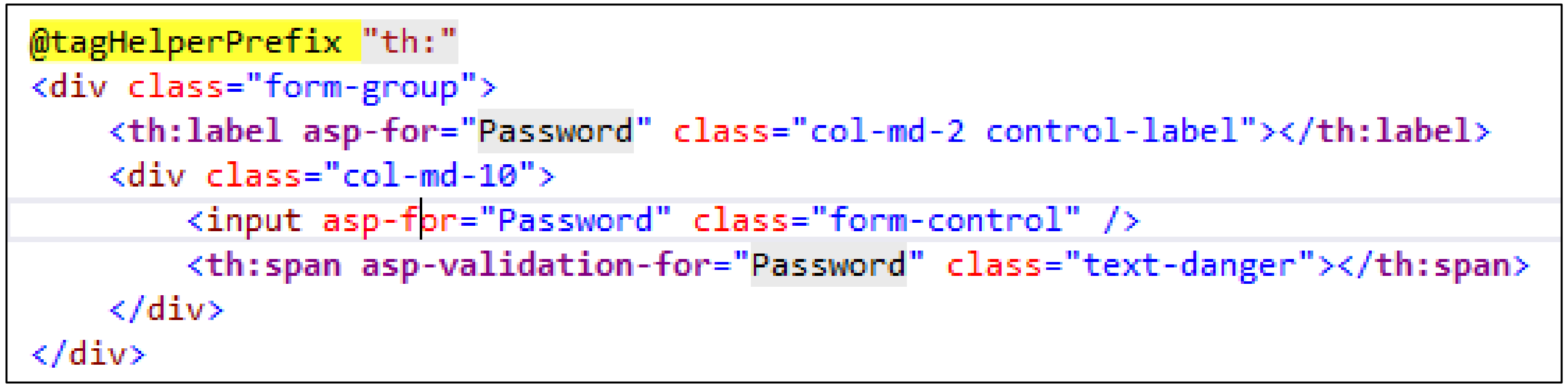
Tag Helper Scope

- *@addTagHelper* makes Tag Helpers available
  - Including it in *_ViewImports.cshtml* makes them available in all the views



```
@using WebApplication17
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, AuthoringTagHelper
```

- *@removeTagHelper* removed a previously added Tag Helper
- *@tagHelperPrefix* specifies tag prefix to enable Tag helper support



```
@tagHelperPrefix "th:"
<div class="form-group">
    <th:label asp-for="Password" class="col-md-2 control-label"></th:label>
    <div class="col-md-10">
        <input asp-for="Password" class="form-control" />
        <th:span asp-validation-for="Password" class="text-danger"></th:span>
    </div>
</div>
```

Microsoft.AspNet.Mvc.TagHelpers

- Default Tag Helpers in **Microsoft.AspNetCore.Mvc.TagHelpers** package
  - Anchor
  - Cache
  - Image
  - Input
  - Validation
  - Link
  - Select
  - Label
  - Form – Automatically adds AntiForgery token
  - Custom

- Source Code: https://github.com/aspnet/Mvc/tree/dev/src/Microsoft.AspNetCore.Mvc.TagHelpers

HTML helpers vs Tag Helpers

- Tag Helper

```
<input asp-for="UserName" />
```

- HTML Helper

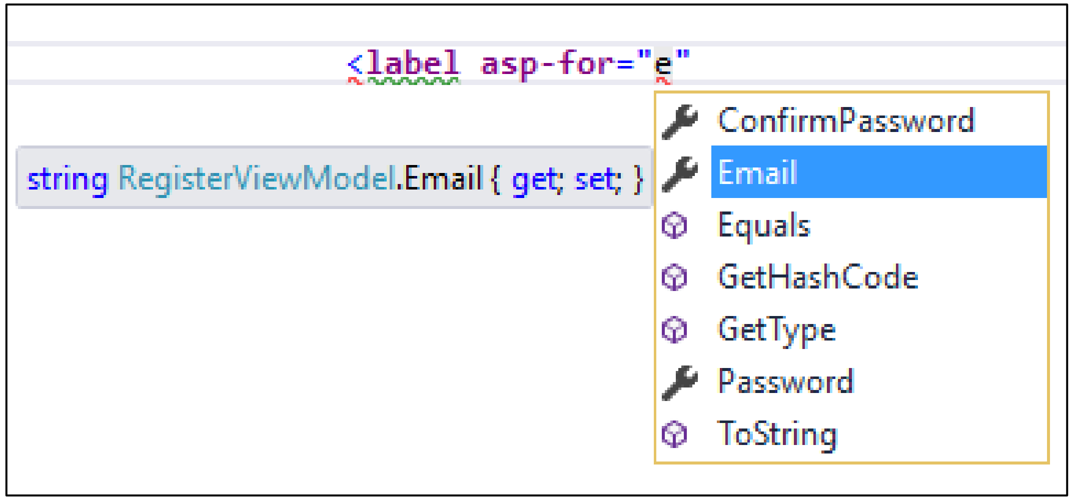
```
@Html.EditorFor(l => l.UserName)
```

- Result

```
<input name="UserName" class="text-box single-line“
 id="UserName" type="text" value="">
```

Tag Helpers vs. HTML Helper

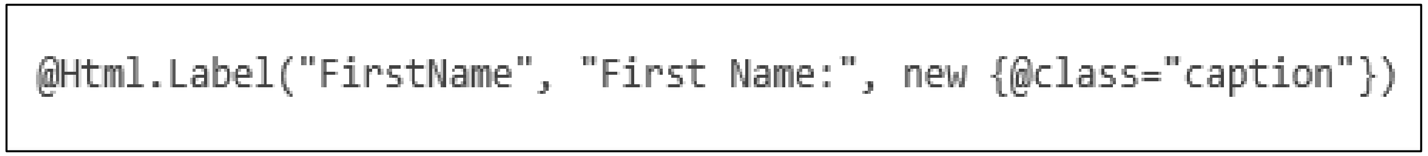
- **Tag Helper**
  - IntelliSense
  - Distinct font and clean code
  - Assists in writing robust and maintainable code
  - No need to learn C# syntax for UX designers



```
<label asp-for="e"
string RegisterViewModel.Email { get; set; }
```

Tag Helper

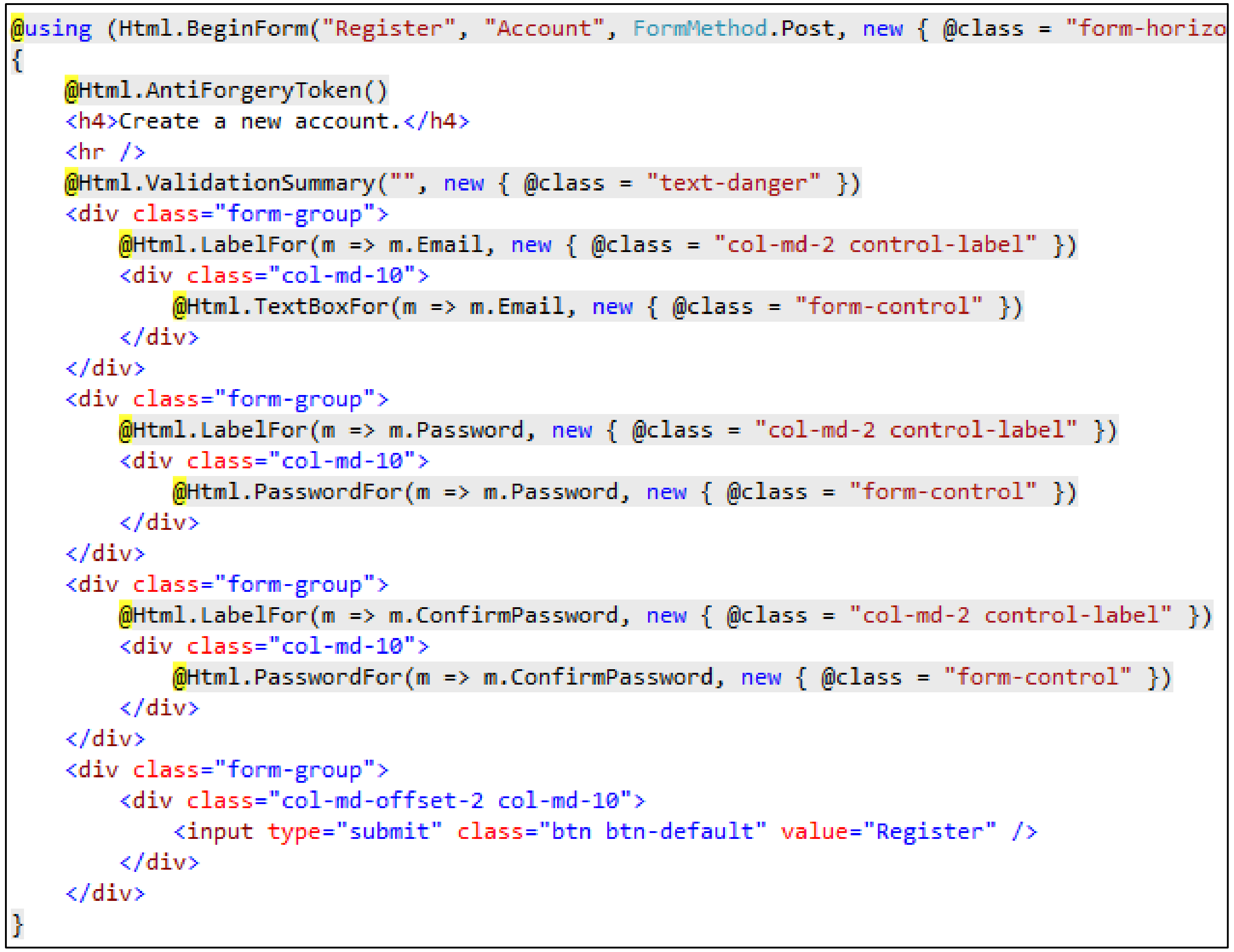
- **HTML Helper**
  - Lack of full IntelliSense support
  - Crowded code
  - Lack of maintainability, for example, image caching
  - C# knowledge is required



```
@Html.Label("FirstName", "First Name:", new {@class="caption"})
```

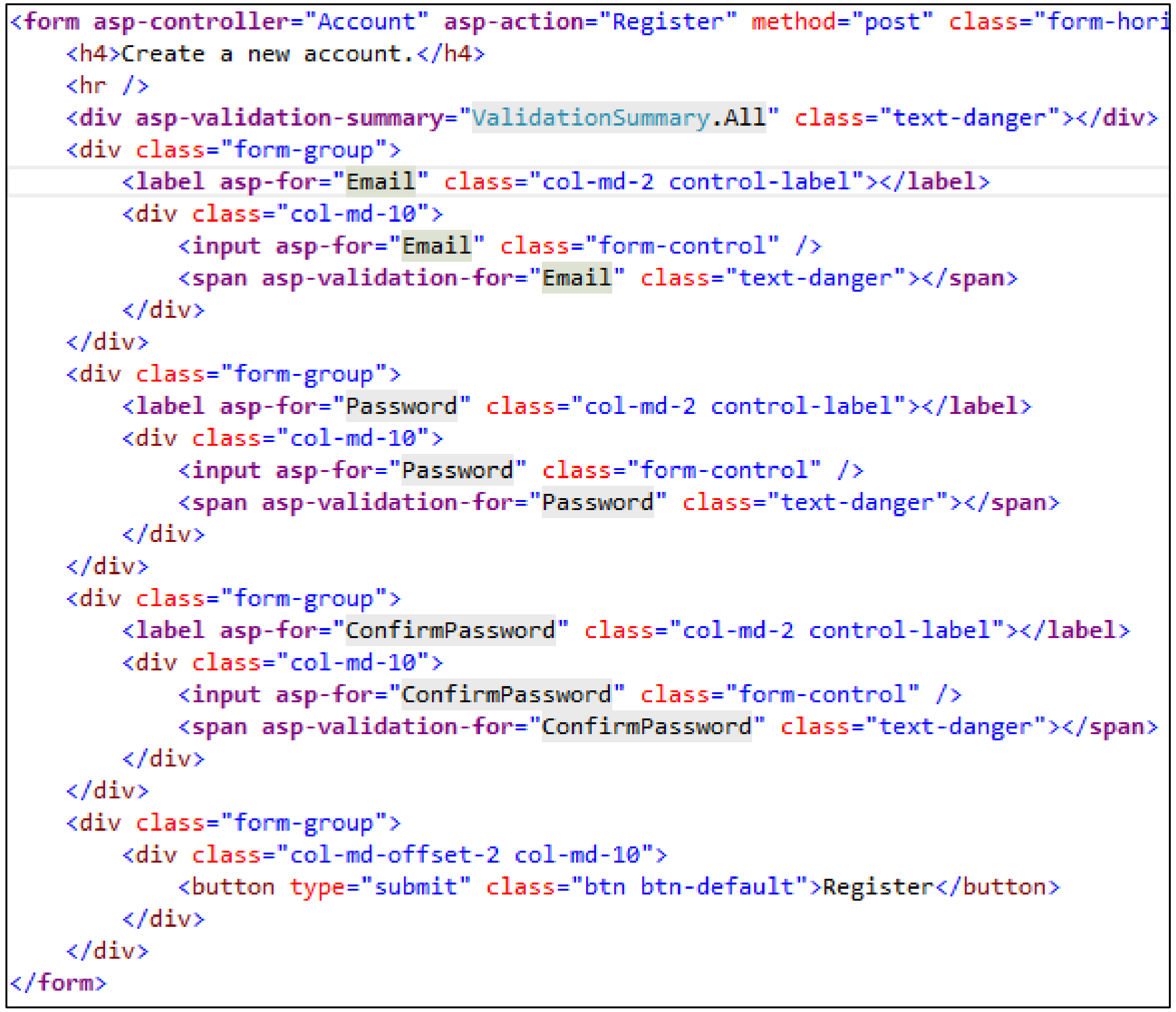
HTML Helper

Register View with HTML Helpers



```
@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizo
{
    @Html.AntiForgeryToken()
    <h4>Create a new account.</h4>
    <hr />
    @Html.ValidationSummary("", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" class="btn btn-default" value="Register" />
        </div>
    </div>
}
```

Register View with Tag Helpers



```
<form asp-controller="Account" asp-action="Register" method="post" class="form-hori
    <h4>Create a new account.</h4>
    <hr />
    <div asp-validation-summary="ValidationSummary.All" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Email" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="Email" class="form-control" />
            <span asp-validation-for="Email" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="Password" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="Password" class="form-control" />
            <span asp-validation-for="Password" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <label asp-for="ConfirmPassword" class="col-md-2 control-label"></label>
        <div class="col-md-10">
            <input asp-for="ConfirmPassword" class="form-control" />
            <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <button type="submit" class="btn btn-default">Register</button>
        </div>
    </div>
</form>
```

Label Tag Helper

```
public class SimpleViewModel
{
    [Display(Name = "Email Address")]
    public string Email { get; set; }
}

<label asp-for="Email"></label>

<label for="Email">Email Address</label>
```

Select Tag Helper

```
public class SimpleViewModel
{
    public IEnumerable<string> CountryCodes { get; set; }
}
```

```
<select
    asp-for="CountryCodes"
    asp-items="ViewBag.Countries">
</select>
```

```
<select name="CountryCodes"
        id="CountryCodes"
        multiple="multiple">
    <option selected="selected" value="CA">
        Canada
    </option>
    <option value="USA">United States</option>
    <option value="--">Other</option>
</select>
```

Form Tag Helper

```
<form asp-controller="Account"
      asp-action="Login“
      asp-route-customparam="myvalue"></form>
```

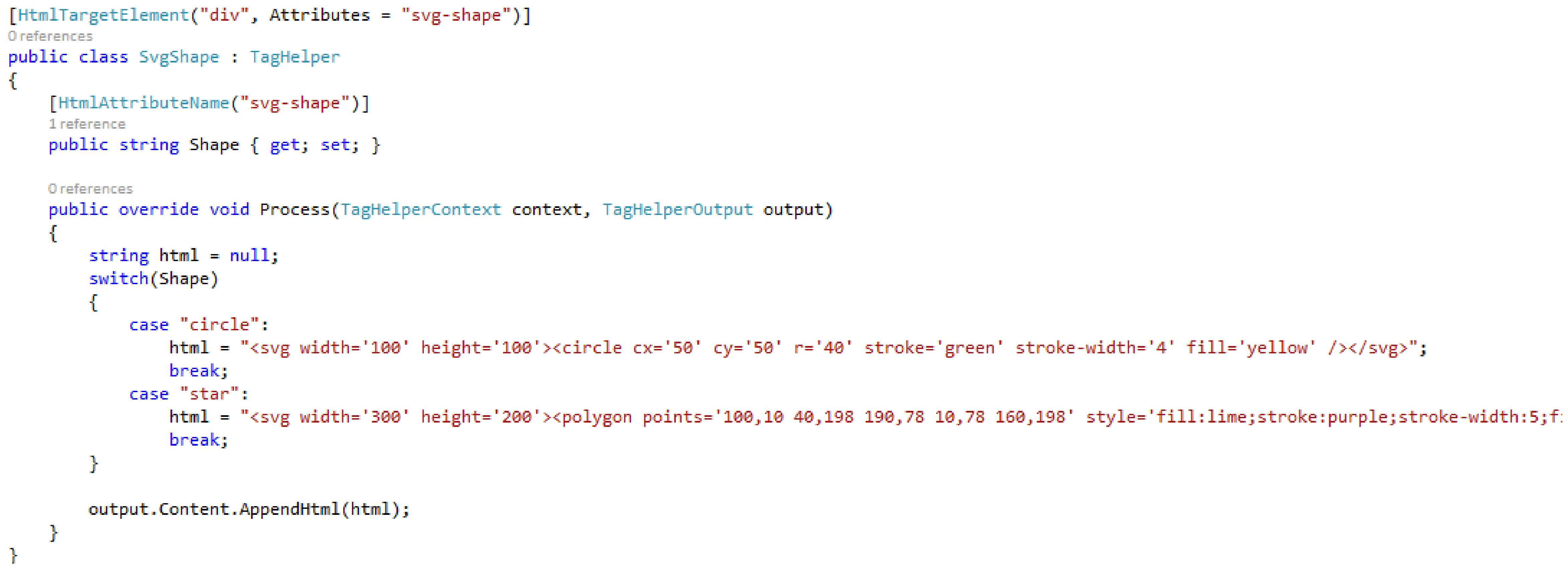
```
<form action="/Account/Login?customparam=myvalue" method="post">
    <input name="RequestVerificationToken" type="hidden"
value="CfDJ8AFtmUdx-
b5MkQvAyGYbjFmMGSMv0Fmk7gG4RqGXlkNV6yqKqj6fgqnOh4TLT6ZnWSaqtAbKkg
pEB20lvfkc2iOKZKIqt3tJ4Jij8DjmatTrZo-
DKVOLwwOzj3kB8VKpFwc0rQMjaJTTC_gVv5f0vAg">
</form>
```

Automatic Anti-Forgery Token!

Link Tag helper

```
<a asp-controller="Product"
   asp-action="Display"
   asp-route-id="@ViewBag.ProductId">
    View Details
</a>



<a href="/Product/Display/1">View Details</a>
```

Custom Tag Helper



```
[HtmlTargetElement("div", Attributes = "svg-shape")]
0 references
public class SvgShape : TagHelper
{
    [HtmlAttributeName("svg-shape")]
    1 reference
    public string Shape { get; set; }

    0 references
    public override void Process(TagHelperContext context, TagHelperOutput output)
    {
        string html = null;
        switch(Shape)
        {
            case "circle":
                html = "<svg width='100' height='100'><circle cx='50' cy='50' r='40' stroke='green' stroke-width='4' fill='yellow' /></svg>";
                break;
            case "star":
                html = "<svg width='300' height='200'><polygon points='100,10 40,198 190,78 10,78 160,198' style='fill:lime;stroke:purple;stroke-width:5;f
                break;
        }

        output.Content.AppendHtml(html);
    }
}
```

Custom Tag Helper

```
@addTagHelper "*, WebApplication3"



<div svg-shape="circle" ></div>
```

Demo: Tag Helpers

Module 5: Razor Pages & MVC Views

Section 2: Razor View Engine

Lesson: Service Injection in Views

Service Injection in Views

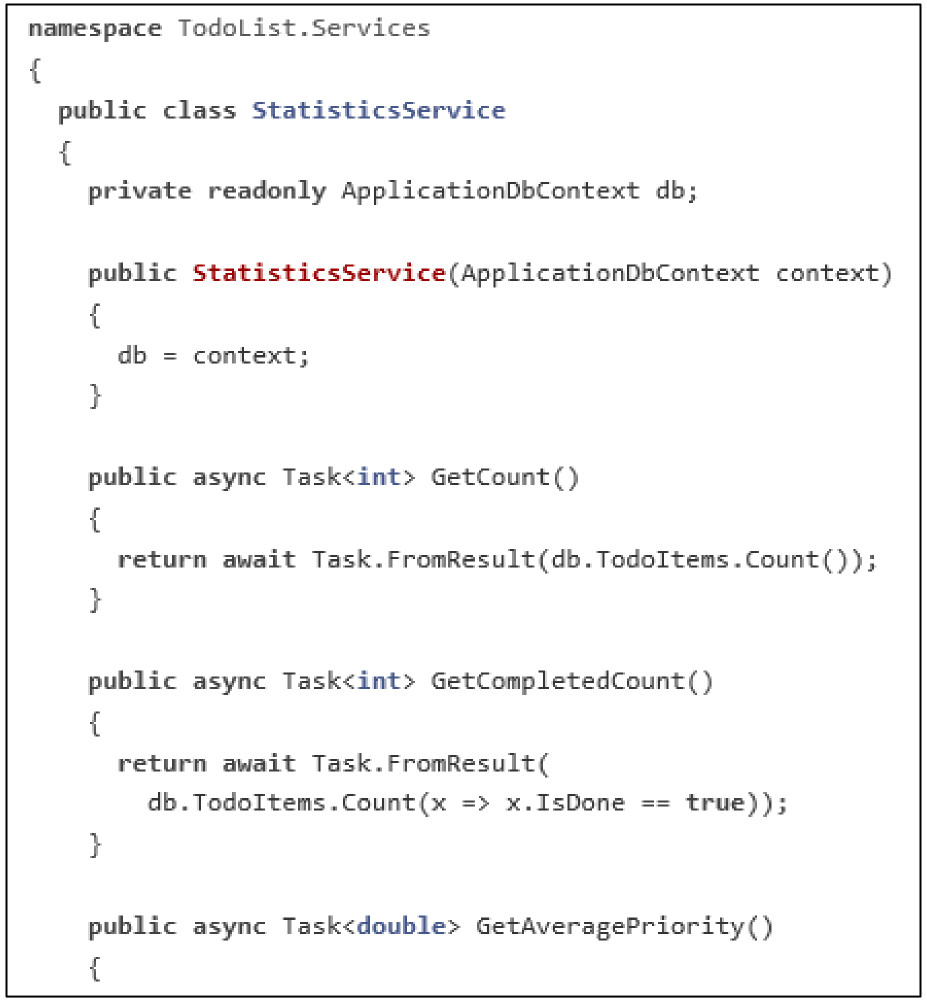
- **@inject** used for injecting dependencies in Views
- Service needs to be registered first with Inversion of Controller (IoC) container

```
public void ConfigureServices(IServiceCollection services)
{
    //...
    //Code removed for brevity
    //Add MVC services to the services container
    services.AddControllersWithViews();
    services.AddTransient<StatisticsService>();
}
```

- *@inject* markup code at the top of view

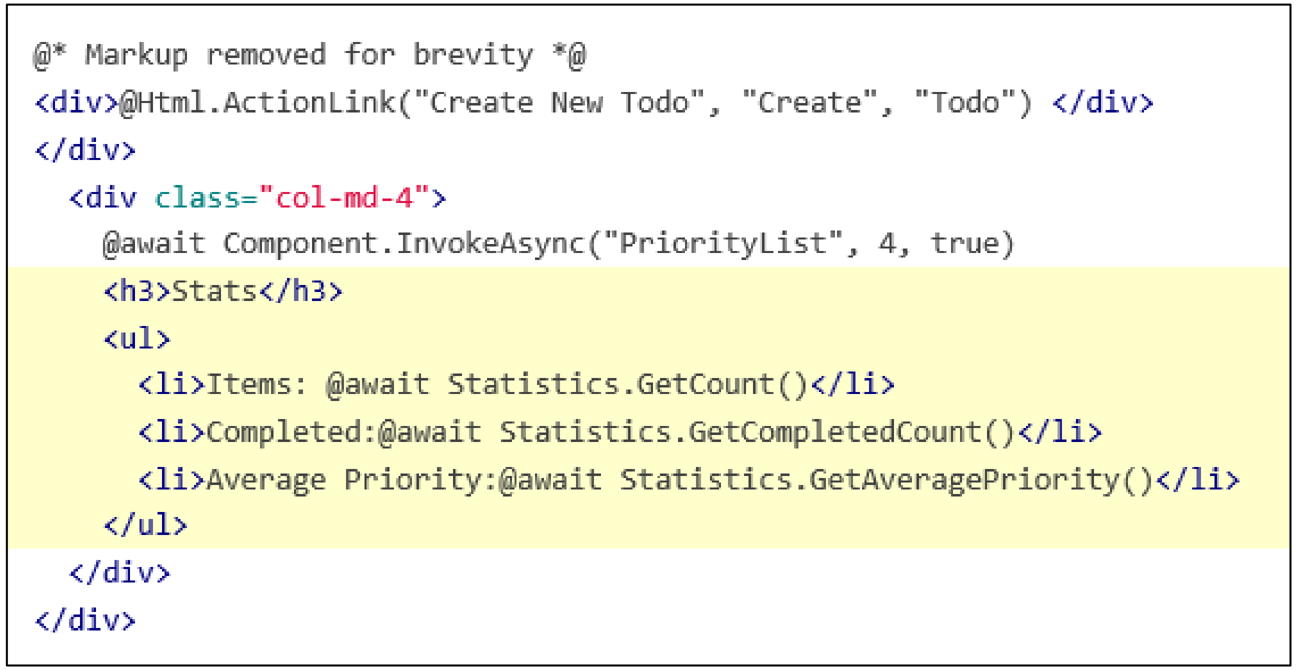
```
@inject StatisticsService StatsService
```

Injected Service Definition and Consumption



```
namespace TodoList.Services
{
  public class StatisticsService
  {
    private readonly ApplicationDbContext db;

    public StatisticsService(ApplicationDbContext context)
    {
      db = context;
    }

    public async Task<int> GetCount()
    {
      return await Task.FromResult(db.TodoItems.Count());
    }

    public async Task<int> GetCompletedCount()
    {
      return await Task.FromResult(
        db.TodoItems.Count(x => x.IsDone == true));
    }

    public async Task<double> GetAveragePriority()
    {
```

Services\StatisticsService.cs



```
@* Markup removed for brevity *@
<div>@Html.ActionLink("Create New Todo", "Create", "Todo") </div>
</div>
  <div class="col-md-4">
    @await Component.InvokeAsync("PriorityList", 4, true)
    <h3>Stats</h3>
    <ul>
      <li>Items: @await Statistics.GetCount()</li>
      <li>Completed:@await Statistics.GetCompletedCount()</li>
      <li>Average Priority:@await Statistics.GetAveragePriority()</li>
    </ul>
  </div>
</div>
```

Views\ToDo\Index.cshtml

Demo: Service Injection in Views

Demo: Razor Runtime Compile

Module Summary

- In this module, you learned:
  - Views and their role in MVC pattern
  - Partial and strongly typed views
  - View engines and Razor view engine
  - Tag Helpers
  - View Components
  - Service Injection in Views
  - Scaffolding
  - Razor Pages

Lab: Views

Microsoft