.NET Framework: Developing Modern Web Apps with ASP.NET MVC – Workshop*PLUS*

Wael Kdouh

Senior Consultant

v1.0

Module 4: Views

Module Overview

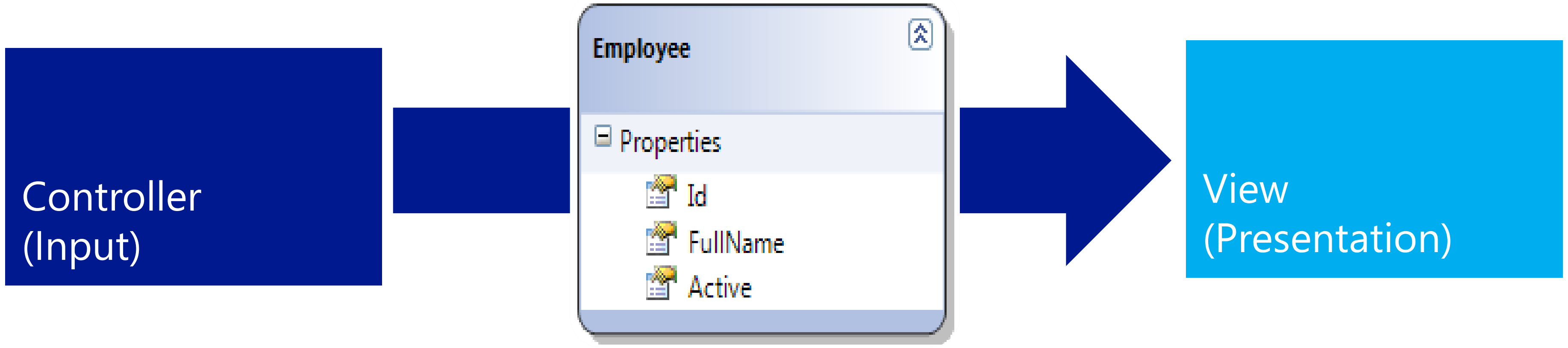
Module 4: Views

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



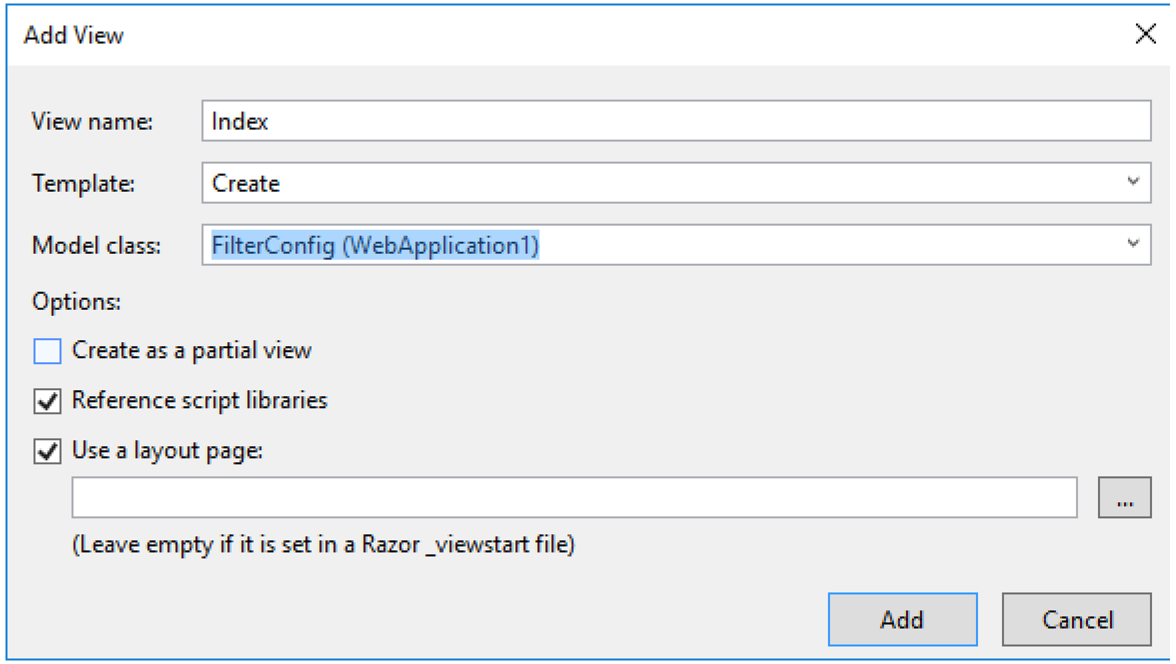
Role of a View

- 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, *.vbhtml
- View can be created through:
  - Solution Explorer
  - Action Method


Add View

View name: Index

Template: Create

Model class: FilterConfig (WebApplication1)

Options:

☐ Create as a partial view

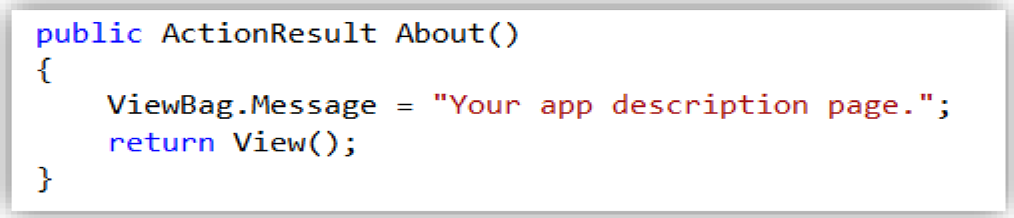
☑ Reference script libraries

☑ Use a layout page:

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

Specifying Views

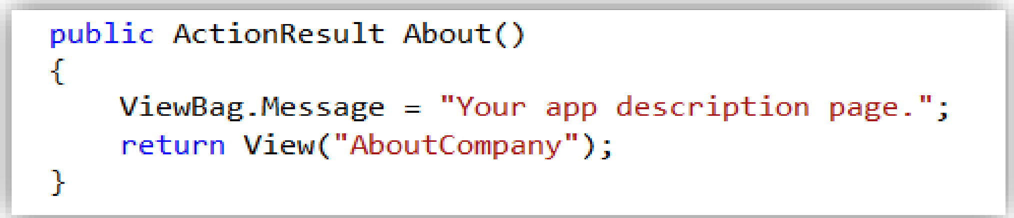
- Select View using default convention



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

Views > Home > About.cshtml

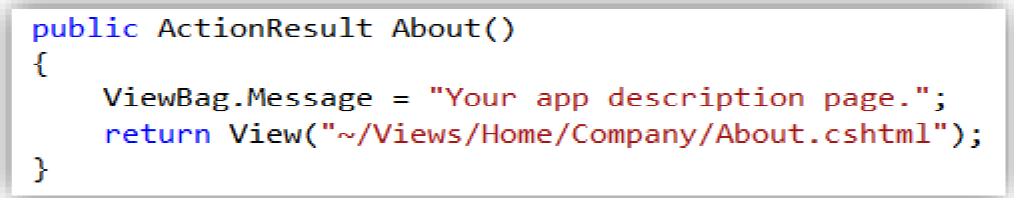
- Select a particular view



```
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

Demo: Views

Module 4: 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 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!

Strongly Typed Views

- Page that derives from System.Web.Mvc.ViewPage<TModel>
- 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

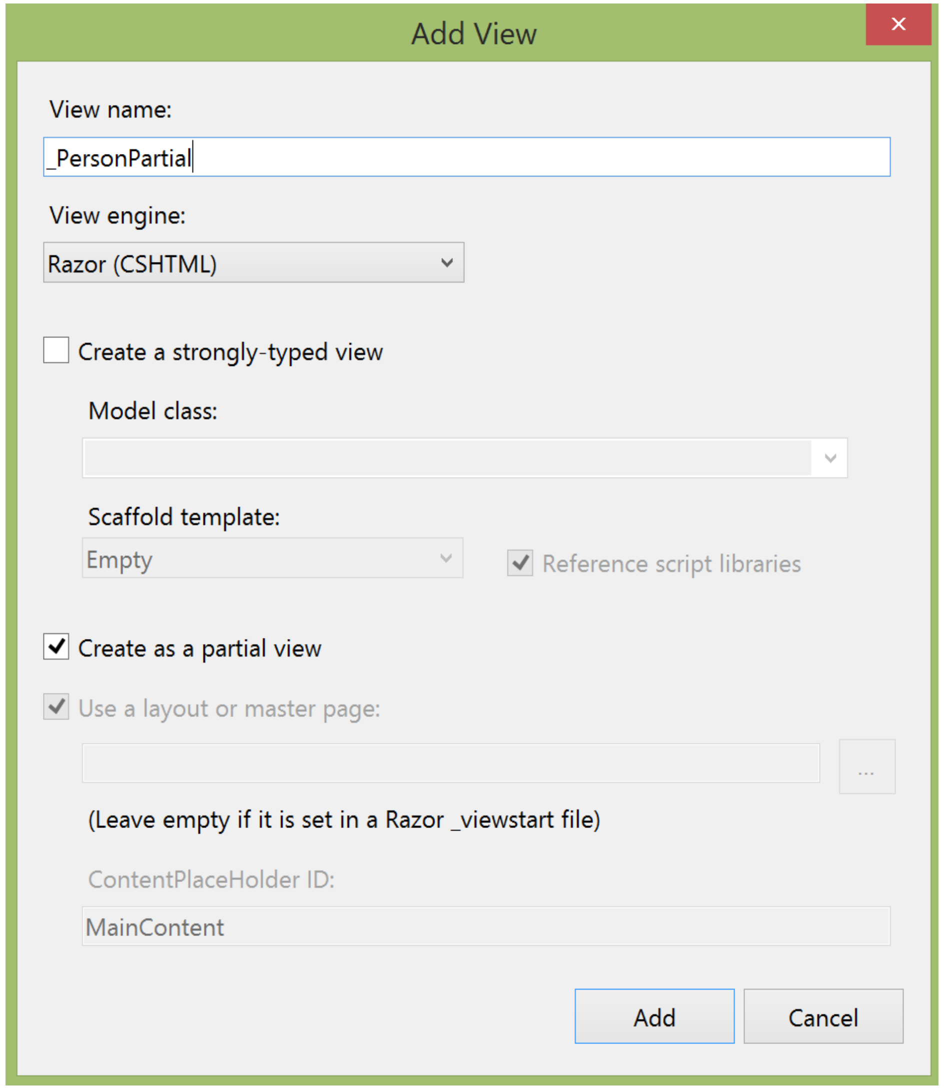
```
@ViewData["Name"]
@ViewData["Age"]
```

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: _PersonPartial

View engine: Razor (CSHTML)

- [ ] Create a strongly-typed view

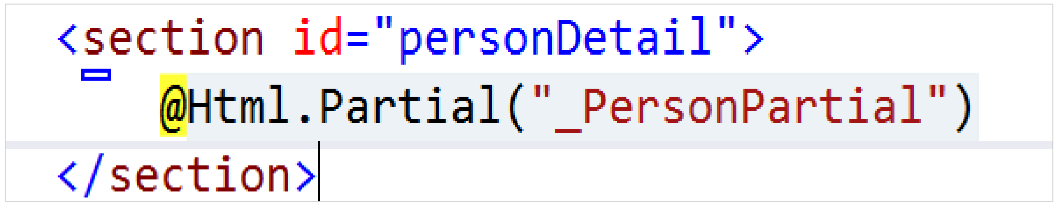
Model class:

Scaffold template: Empty

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

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

ContentPlaceHolder ID: MainContent


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

Demo: Partial & Strongly Typed Views

Module 4: Views

Section 2: Razor View Engine

Lesson: Razor View Engine

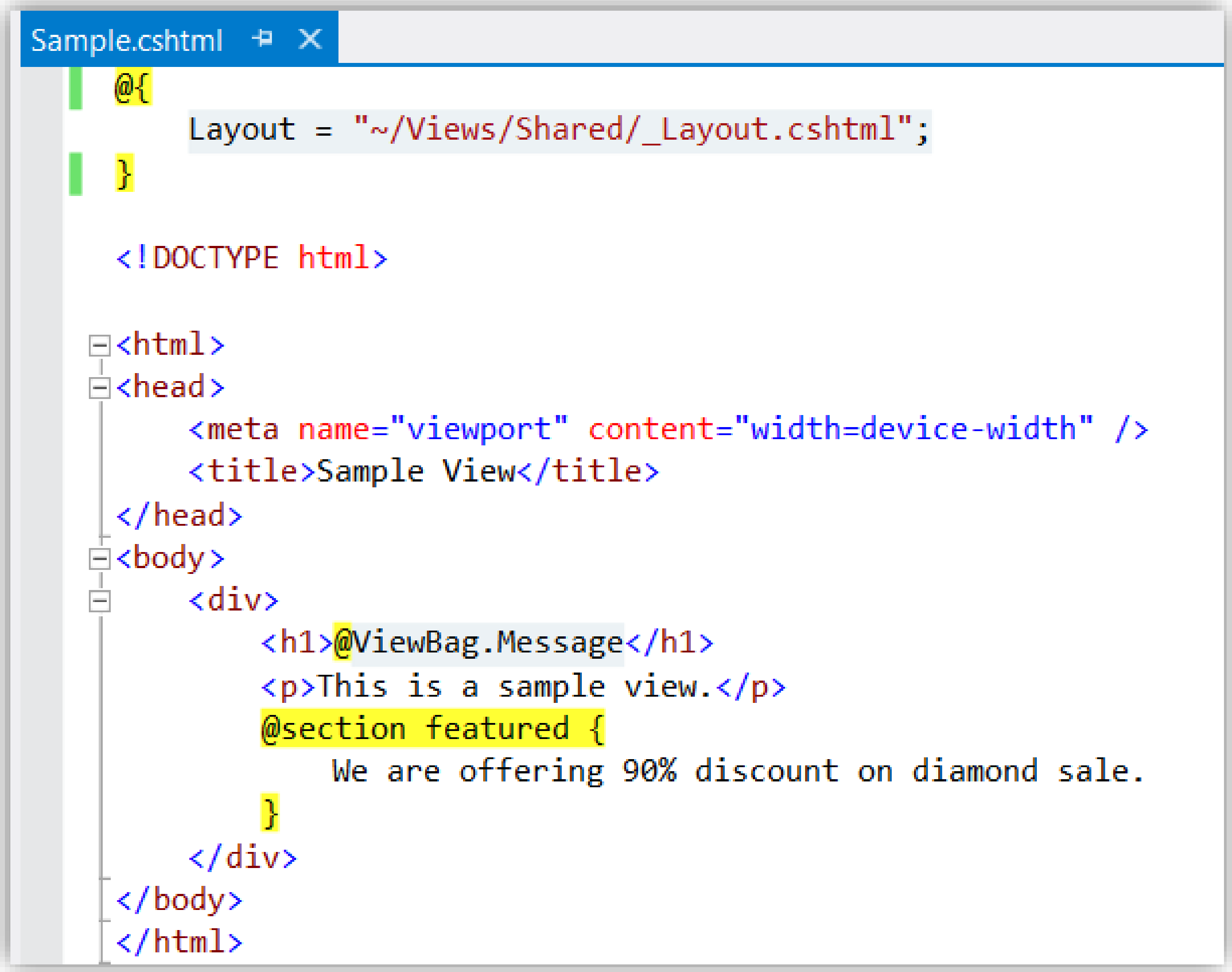
View Engines

- ASP.NET MVC comes with Razor view engine by default
- 3rd party 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



```
@{
    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>
```

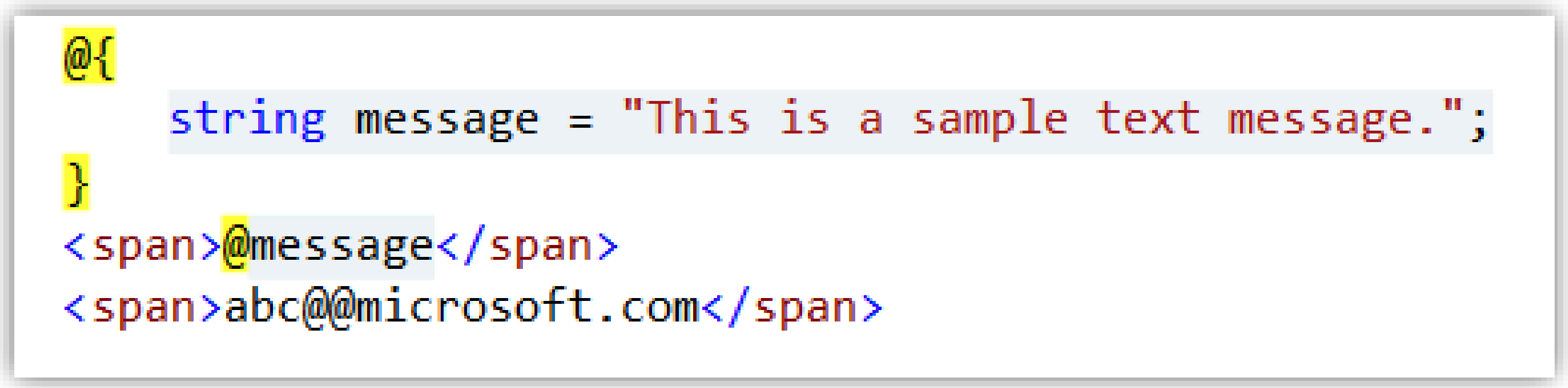
Module 4: Views

Section 2: Razor View Engine

Lesson: Razor View Syntax

Code Expressions

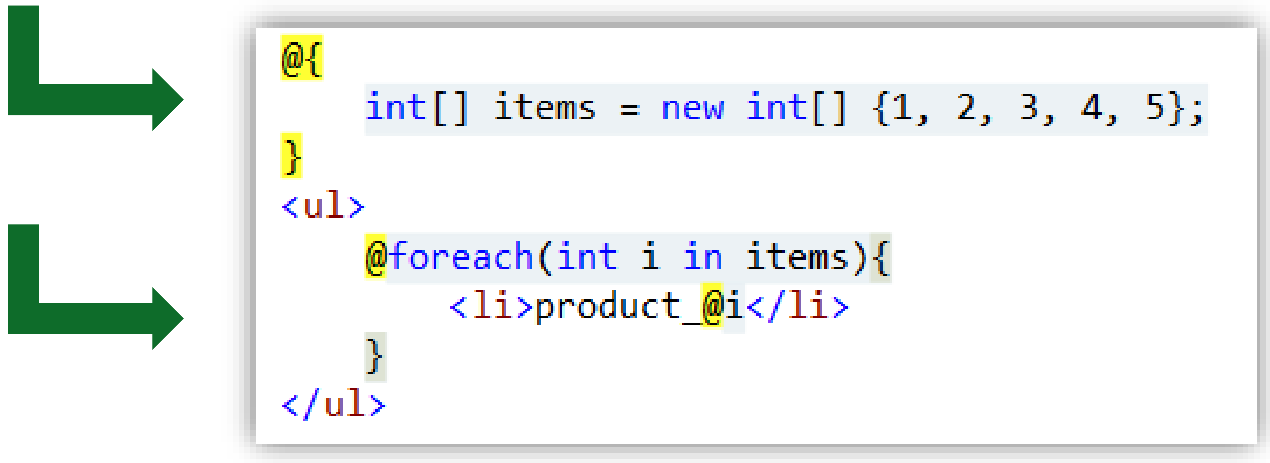
- ‘**@**’ sign used for transition from markup to code and back
- @@ used as an escape sequence



```
@{
    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 vs. Web Forms

Razor Syntax	Web Forms Syntax
Implicit code expression	
`<span>@model.Message</span>`	`<span> <%: model.Message %>`
	`</span>`
Explicit code expression	
`<span>ISBN@isbn</span>`	`<span>ISBN<%: isdn %></span>`
Not sanitized output	
`<span>`	`<span> <%:`
`    @Html.Raw(model.AlertMessage)`	`    Html.Raw( model.AlertMessage)`
`</span>`	`%></span>`
Code block	
`@{`	`<%`
`    int x = 567;`	`    int x = 567;`
`    string s = “Microsoft”;`	`    string s = “Microsoft”;`
`}`	`%>`

Razor vs. Web Forms (continued)

Code and markup

Razor Syntax

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

Web Forms Syntax

```
<% foreach(var item in items){
%>
   <span>
     Item <%: @item.Id %>
   </span>
<% } %>
```

Code and plain text

Razor Syntax

```
@if(showMessage) {
   <text>
      Text Message.
   </text>
}
```

Web Forms Syntax

```
<% if(showMessage) { %>
   Text Message.
<% } %>
```

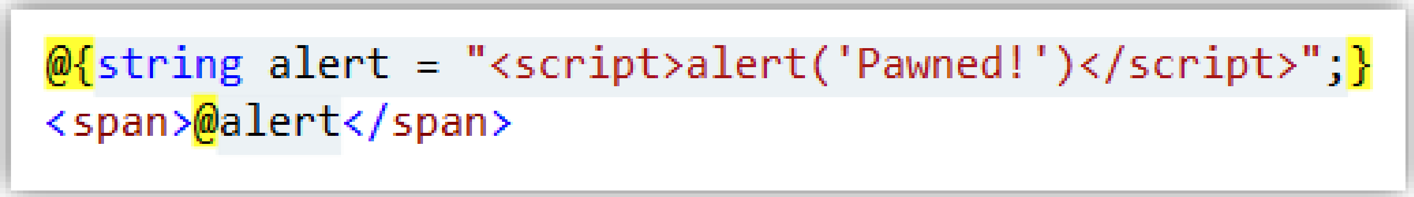
Razor vs. Web Forms (continued)

Razor Syntax	Web Forms Syntax
Comments @* Multi-line comment Product name: @ViewBag.Product *@	<!-- Multi-line comment Product name: @ViewBag.Product -->

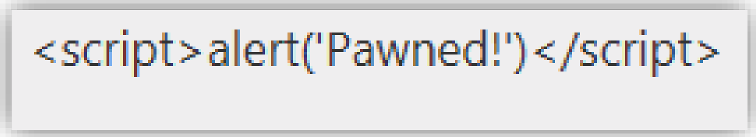
Demo: Razor View Engine

HTML Encoding

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

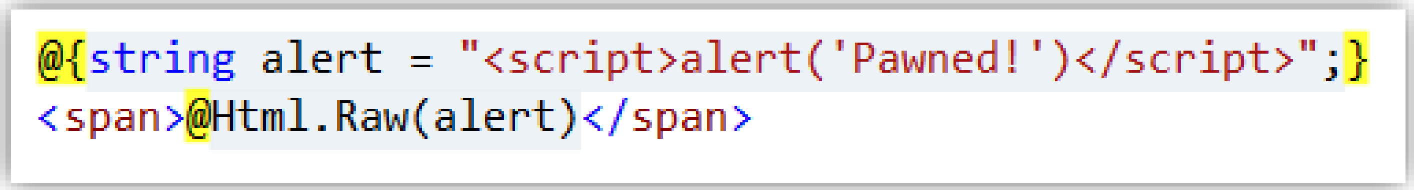


```
@{string alert = "<script>alert('Pawned!')</script>";}
<span>@alert</span>
```

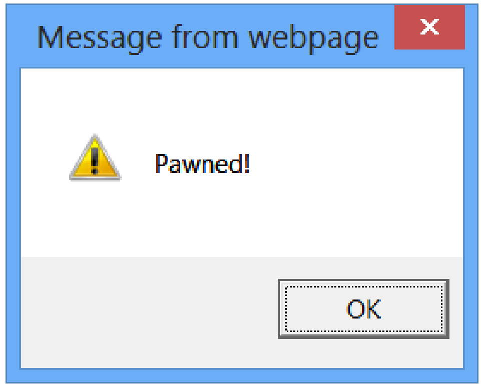


```
<script>alert('Pawned!')</script>
```

- Use Html.Raw( ) for showing HTML markup



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



Message from webpage

Pawned!

Module 4: Views

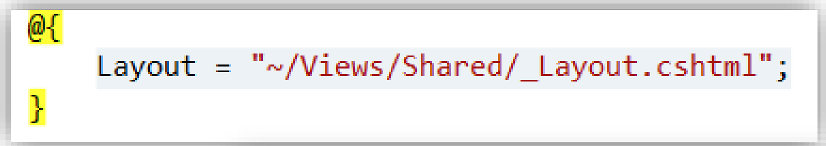
Section 2: Razor View Engine

Lesson: Layouts and Sections

Layouts

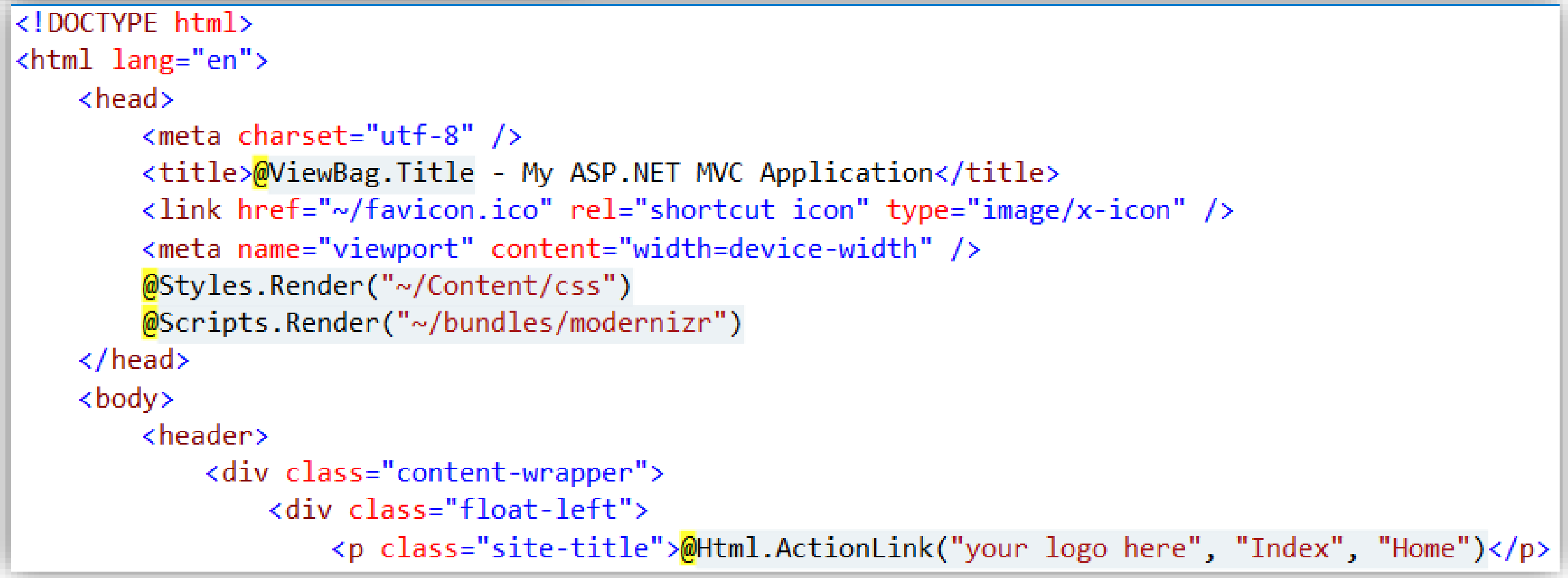
- Layouts are to views what Master Pages are to web pages in ASP.NET
- Layout defines a common template for ASP.NET MVC site
- @RenderBody( ) defines placeholder for view body

_ViewStart.cshtml



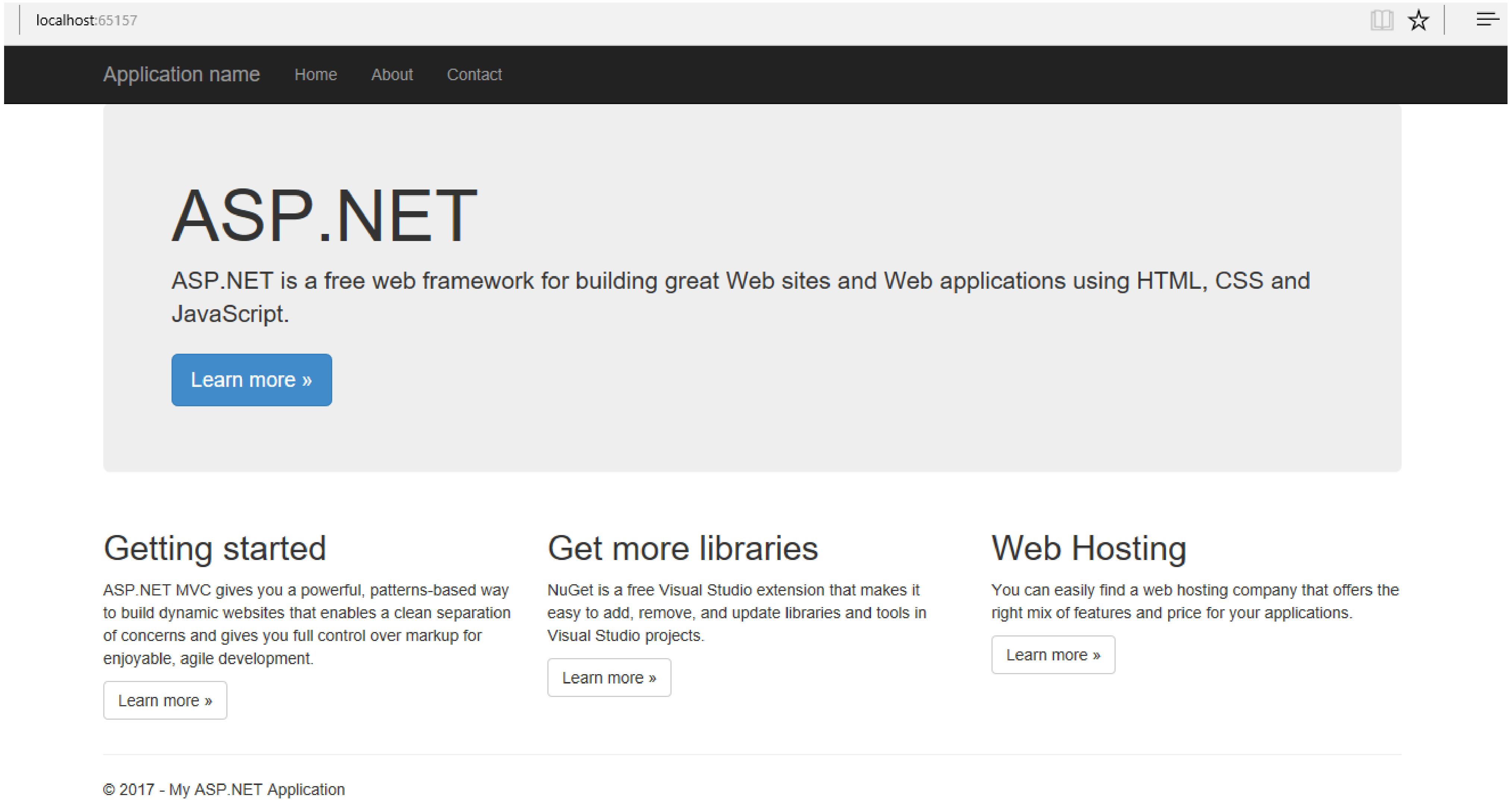
```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

_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>
```

Layouts – Default ASP.NET MVC Template



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

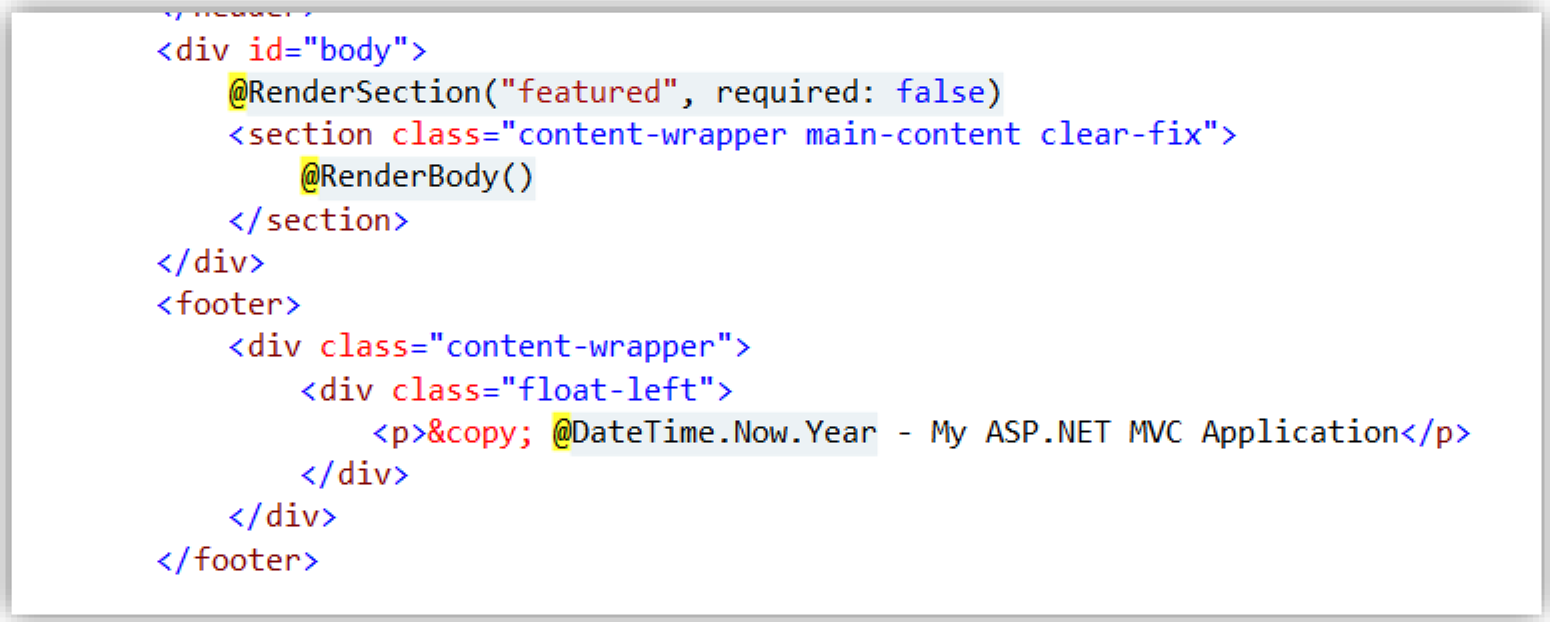
Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

© 2017 - My ASP.NET Application

Layout Sections

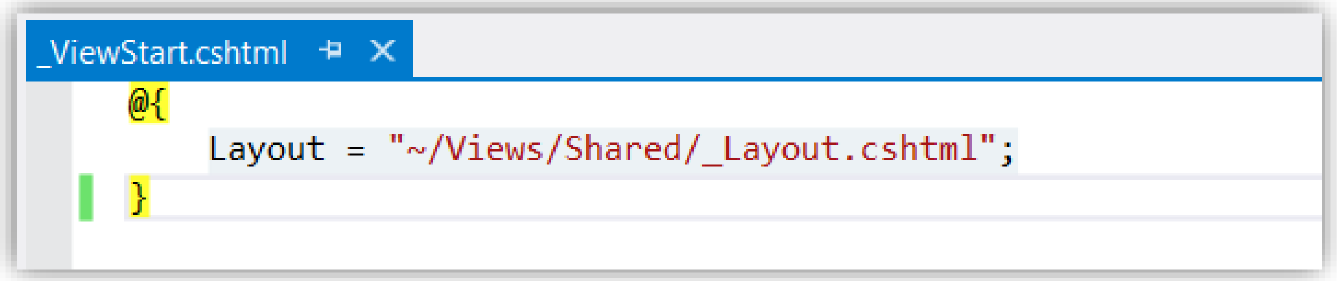
- Layout may have multiple sections
- View must provide content for all layout sections, unless explicitly made optional
- @RenderSection(…) defines placeholder for layout sections



```
<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>
```

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

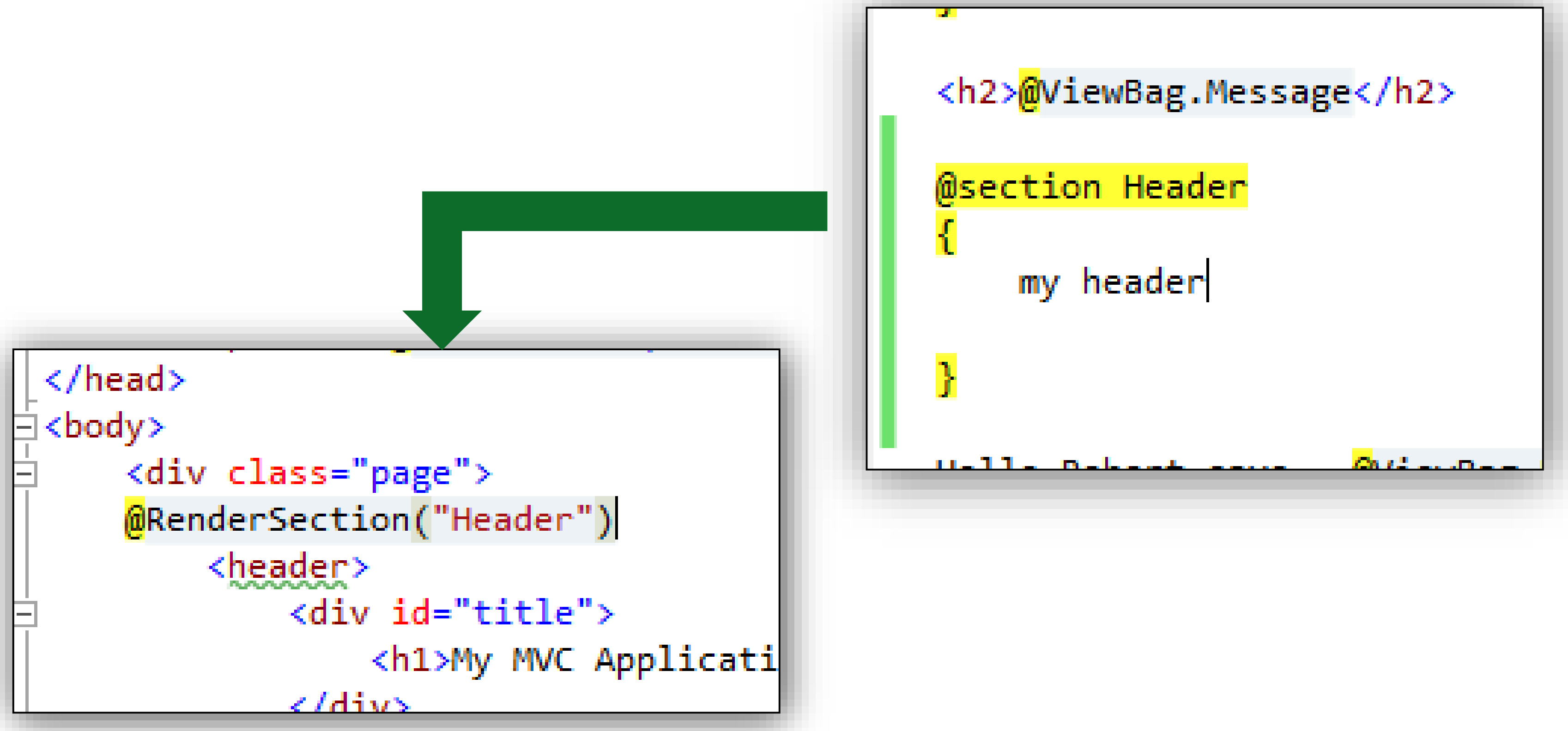


```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

_ViewStart.cshtml

Sections

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



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

@section Header
{
    my header

}
```

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

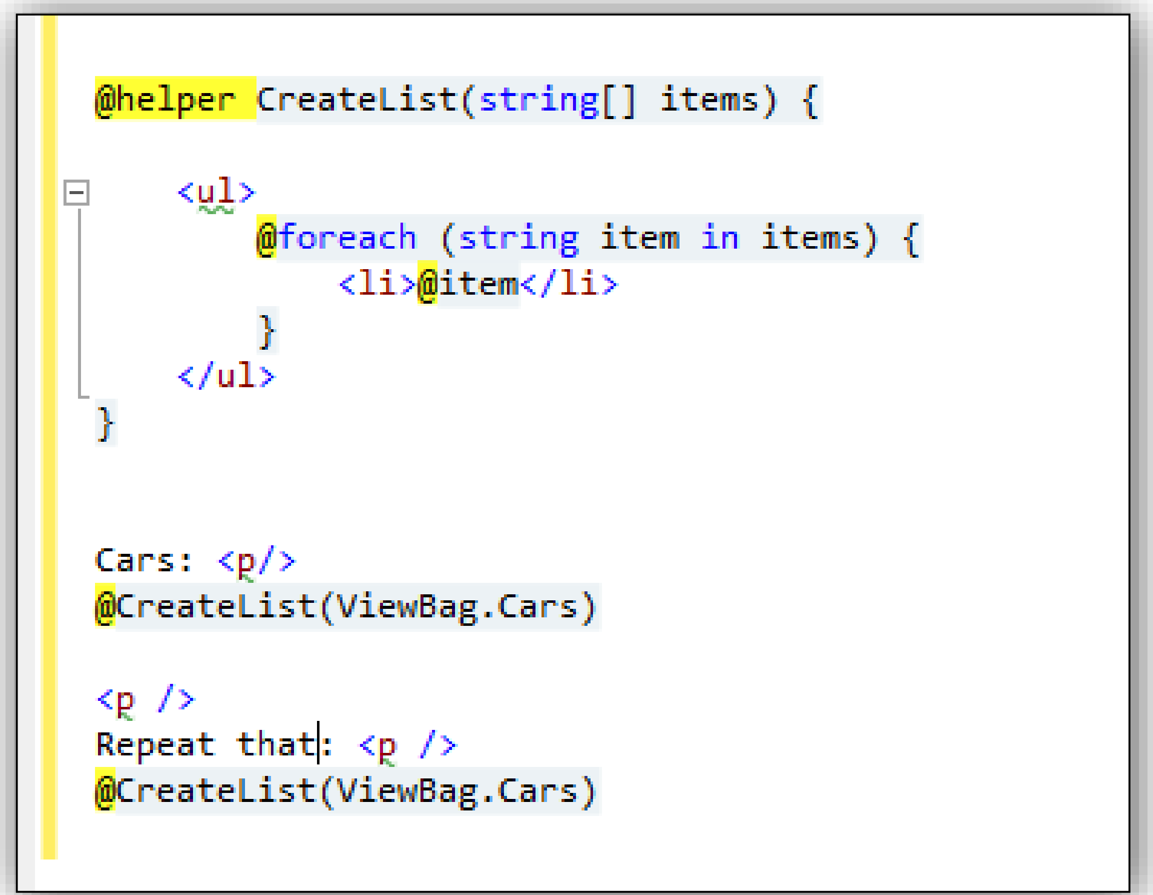
Module 4: Views

Section 2: Razor View Engine

Lesson: HTML Helpers, Display, and Editor Templates

HTML Helpers

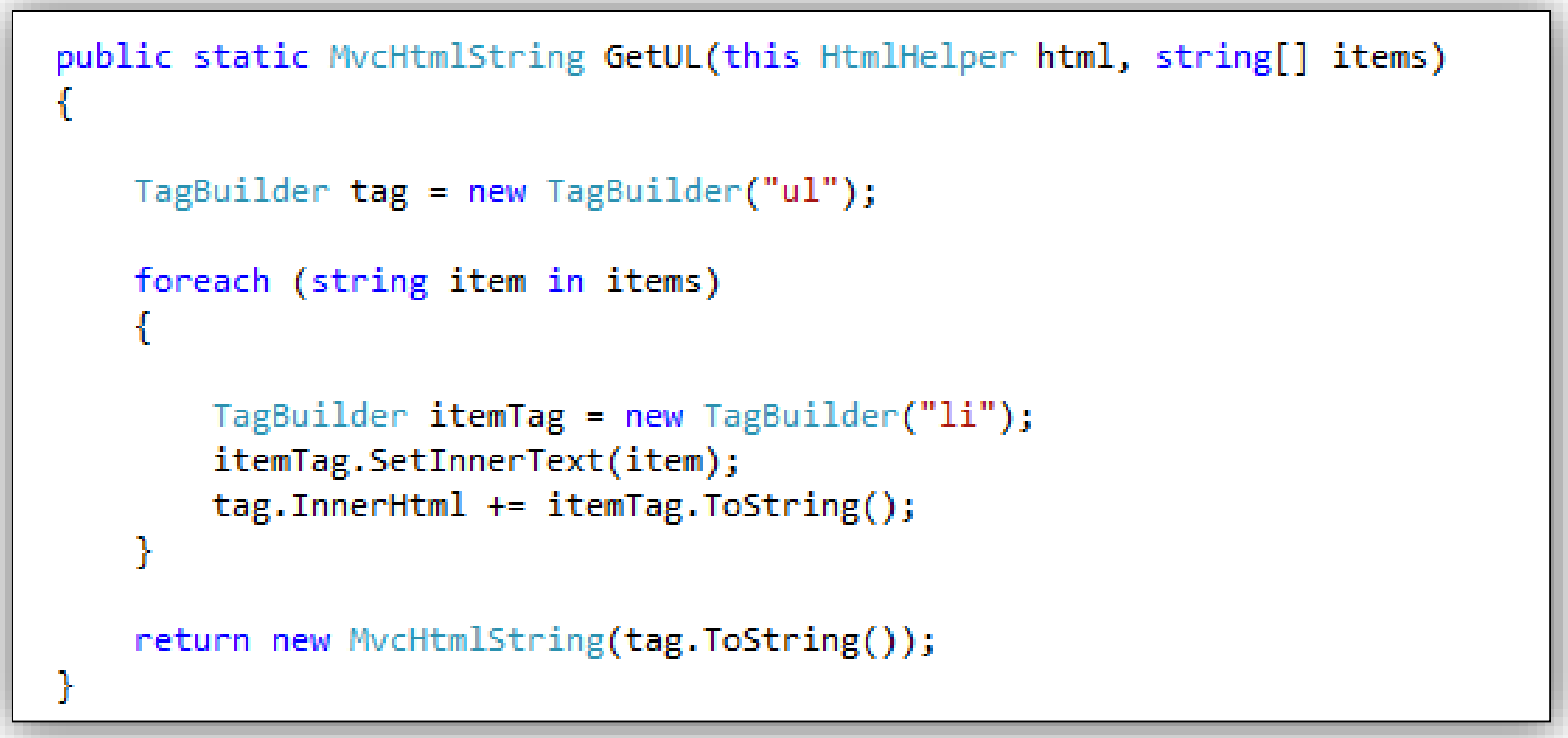
- Inline can be used only from the view in which they are declared



```
@helper CreateList(string[] items) {

    <ul>
        @foreach (string item in items) {
            <li>@item</li>
        }
    </ul>
}


Cars: <p/>
@CreateList(ViewBag.Cars)

<p />
Repeat that: <p />
@CreateList(ViewBag.Cars)
```

HTML Helpers (continued)

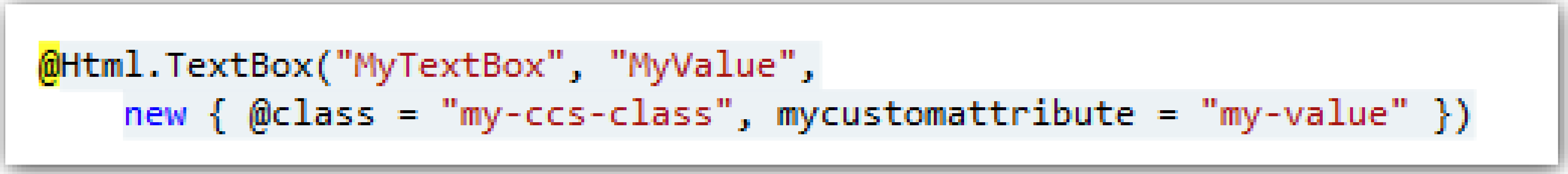
- External helpers are like regular extension methods and it takes the first parameter to HtmlHelper object



```
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 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")



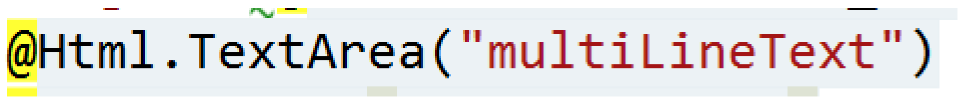
```
@Html.TextBox("MyTextBox", "MyValue",
    new { @class = "my-ccs-class", mycustomattribute = "my-value" })
```

Built-in Display Templates

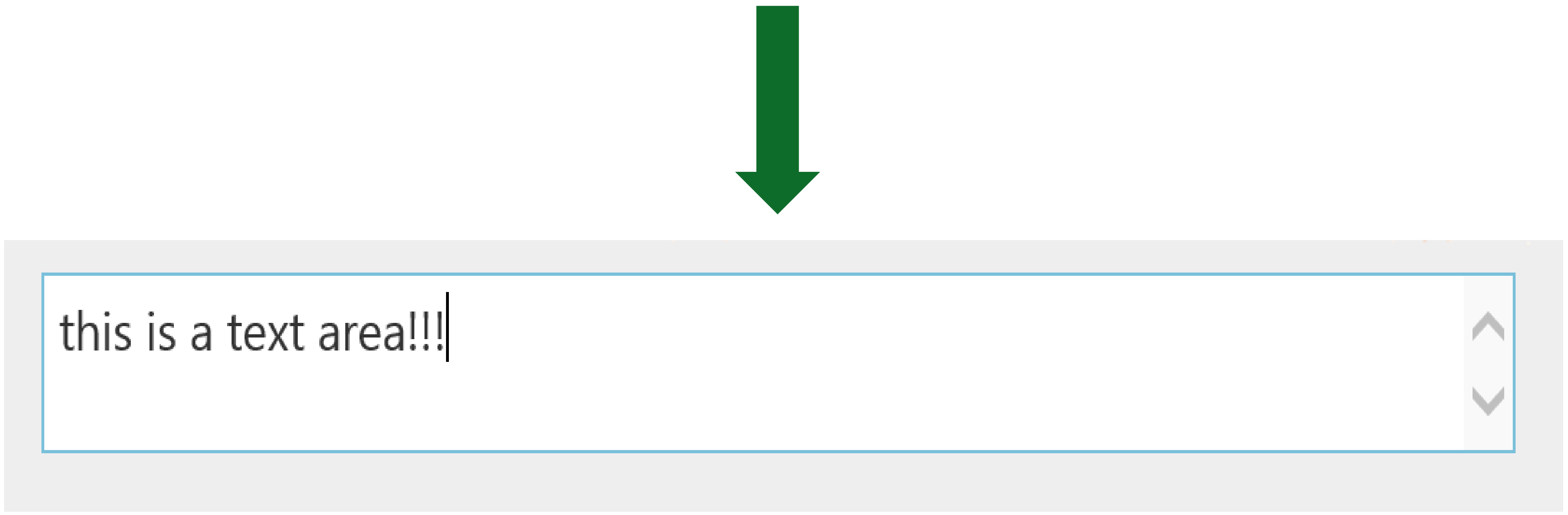
- EmailAddress
- HiddenInput
- HTML
- Text and Raw
- URL
- Collection
- Boolean
- Decimal
- String
- Object

Built-in Editor Templates

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

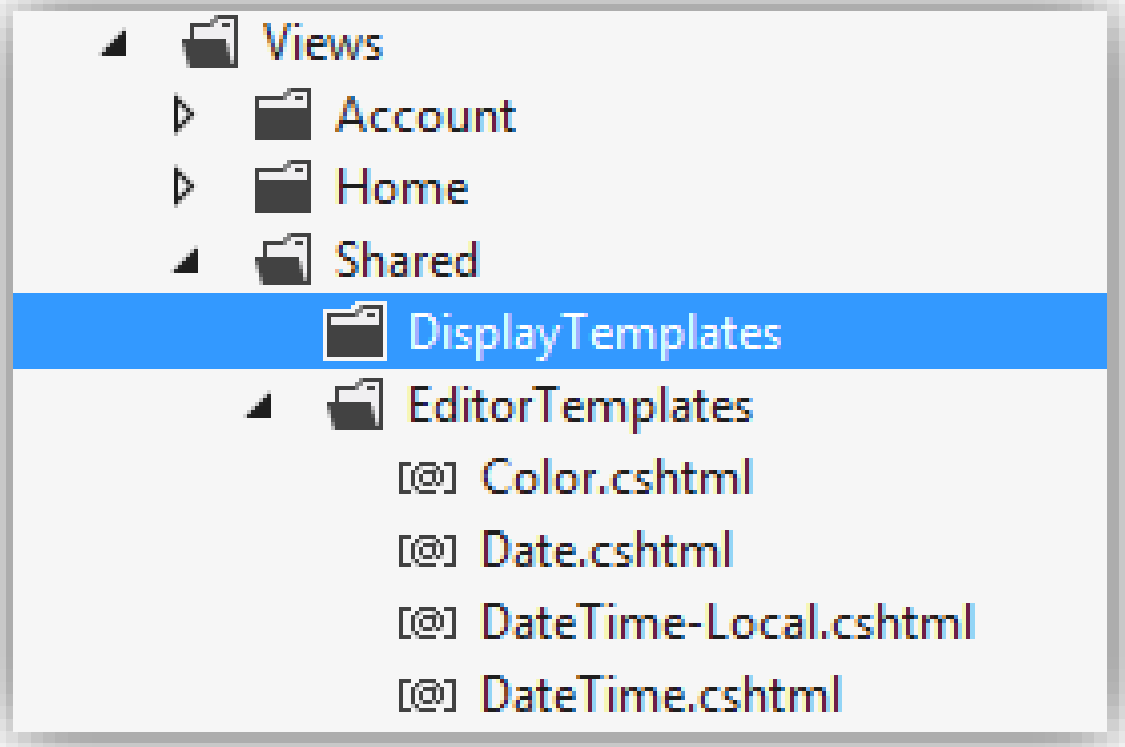


```
@Html.TextArea("multiLineText")
```



this is a text area!!!

Display and Editor Templates



Demo: Editor

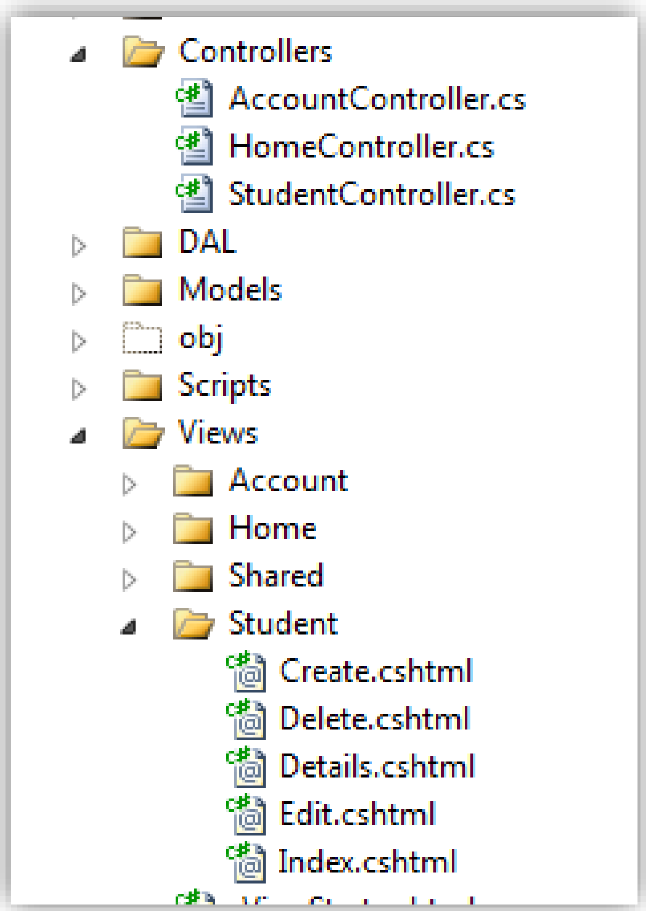
Module 4: Views

Section 3: Scaffolding

Lesson: Scaffolding Templates

Scaffolding

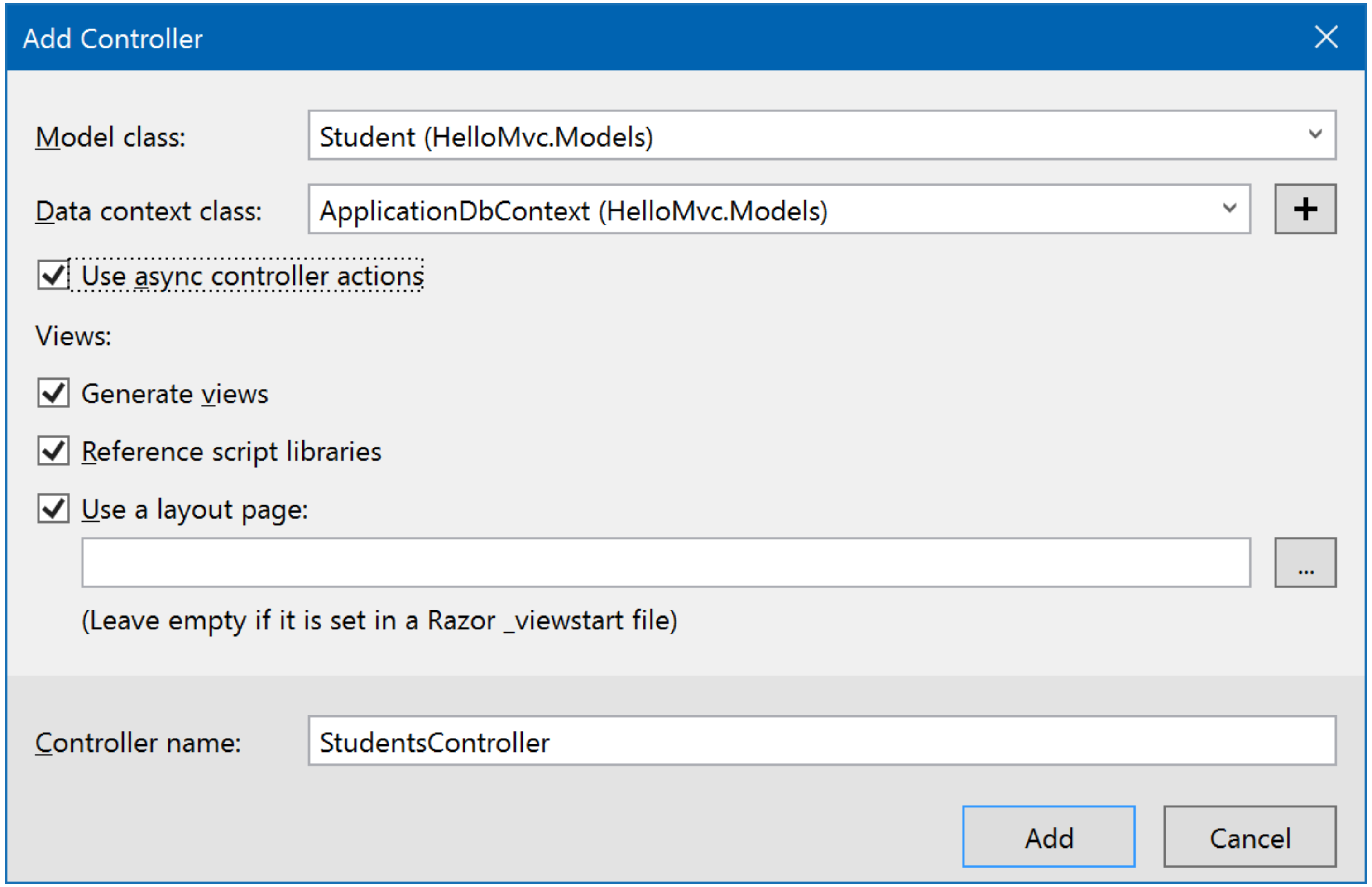
- It means generating code for Create, Read, Update, and Delete (CRUD) functionality against a model
- It examines the type definition of model(s) to:
  - Generate controller(s)
  - Generate Controller's associated views
- It automatically names controllers and views
- All the generated controllers and views are placed correctly in the project structure



```
Controllers
    AccountController.cs
    HomeController.cs
    StudentController.cs
DAL
Models
obj
Scripts
Views
    Account
    Home
    Shared
    Student
        Create.cshtml
        Delete.cshtml
        Details.cshtml
        Edit.cshtml
        Index.cshtml
```

For example, *StudentController* and 5 views (*Views* ➔ *Student directory*) are automatically generated through scaffolding.

ASP.NET MVC Scaffolding in Visual Studio



Model class: Student (HelloMvc.Models)

Data context class: ApplicationDbContext (HelloMvc.Models)

[✓] Use async controller actions

Views:

[✓] Generate views

[✓] Reference script libraries

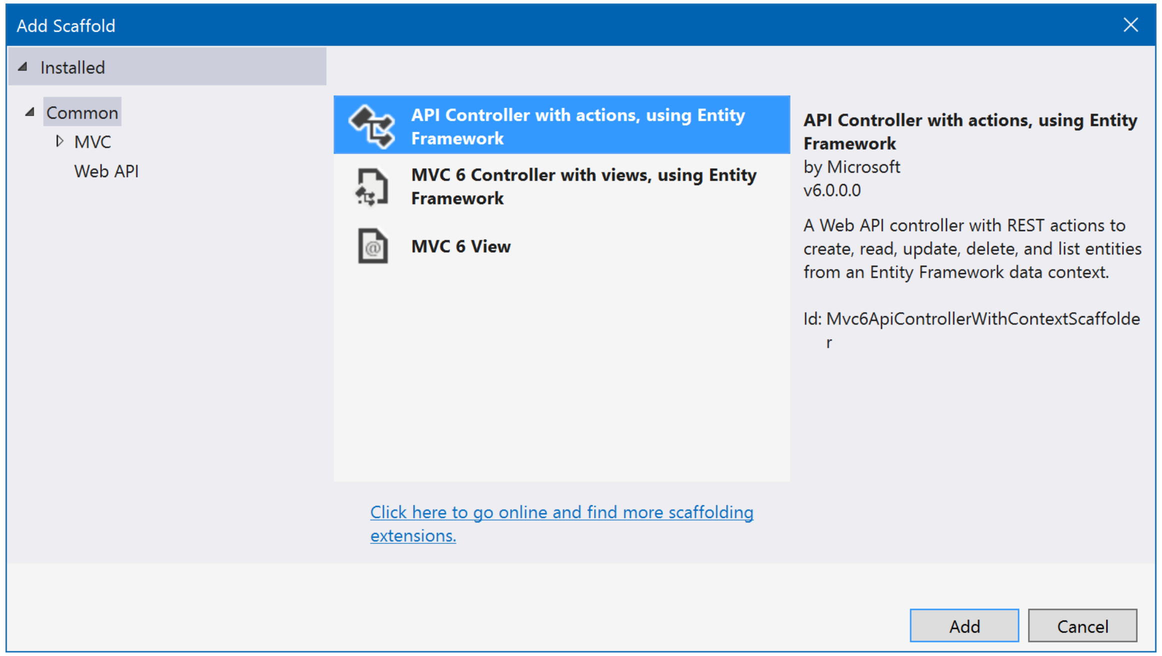
[✓] Use a layout page:

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

Controller name: StudentsController

Scaffolding Templates

- Scaffolding template determines how far would it go with code generation
- Default Scaffolding Templates:
  - API Controller with actions, using Entity Framework
  - MVC 6 Controllers with views, using Entity Framework
- Alternative scaffolding templates are available through **NuGet**



Add Scaffold

Installed

Common
MVC
Web API

API Controller with actions, using Entity Framework
MVC 6 Controller with views, using Entity Framework
MVC 6 View

API Controller with actions, using Entity Framework
by Microsoft
v6.0.0.0

A Web API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.

Id: Mvc6ApiControllerWithContextScaffolder

Click here to go online and find more scaffolding extensions.

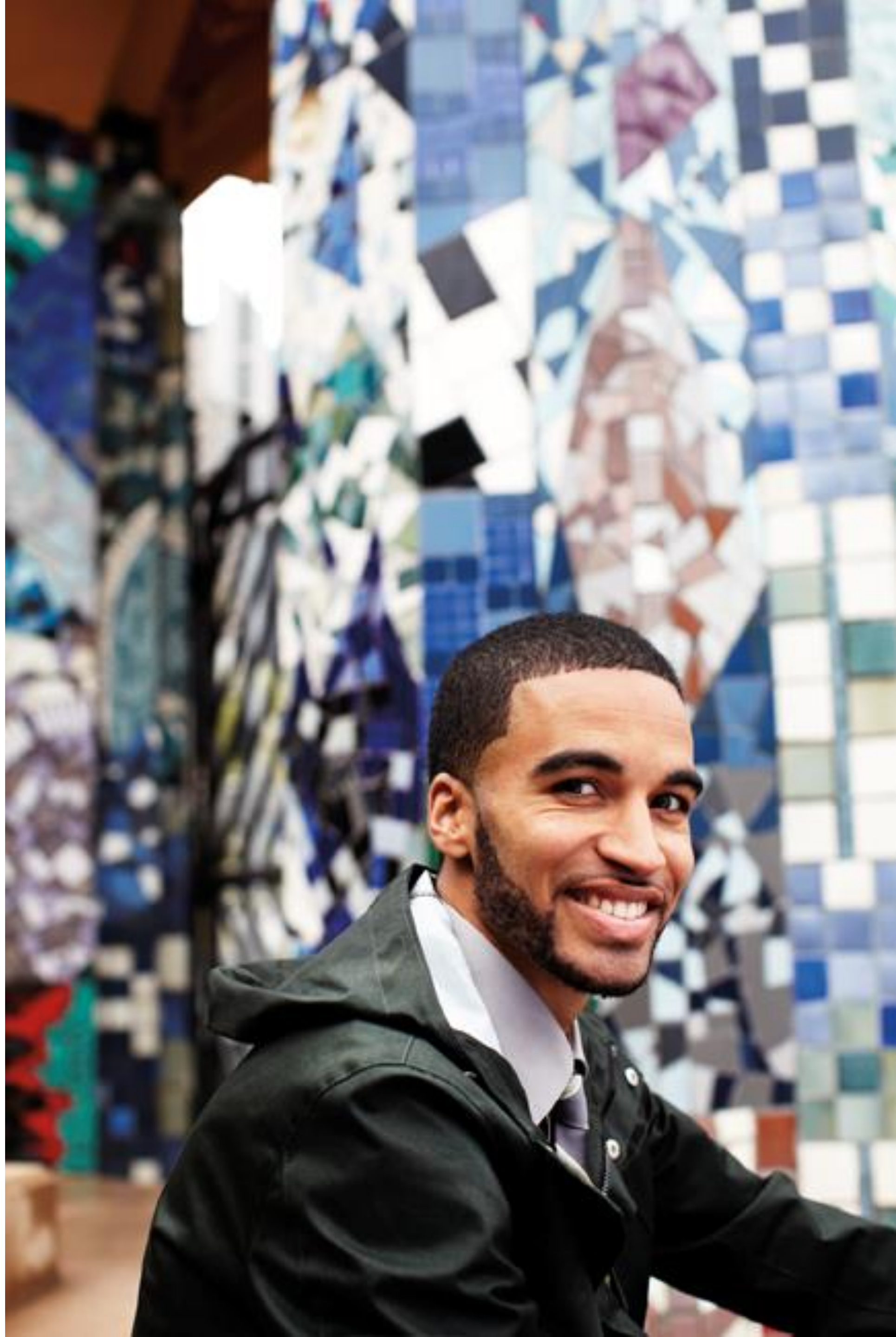
Add Cancel

Demo: Scaffolding

Demo: Binding

Module Summary

- In this module, you understand the following:
    - Views and their role in MVC pattern
    - Partial and strongly typed views
    - View engines and Razor view engine
    - Scaffolding



Lab: Views

Microsoft