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

Wael Kdouh - @waelkdouh

Senior Consultant

v3.0

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

http://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx

Active Directory, Azure, IntelliSense, Internet Explorer, Microsoft, Microsoft Corporate Logo, Silverlight, SharePoint, SQL Server, Visual Basic, Visual Studio, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Module 4: Views

Module Overview

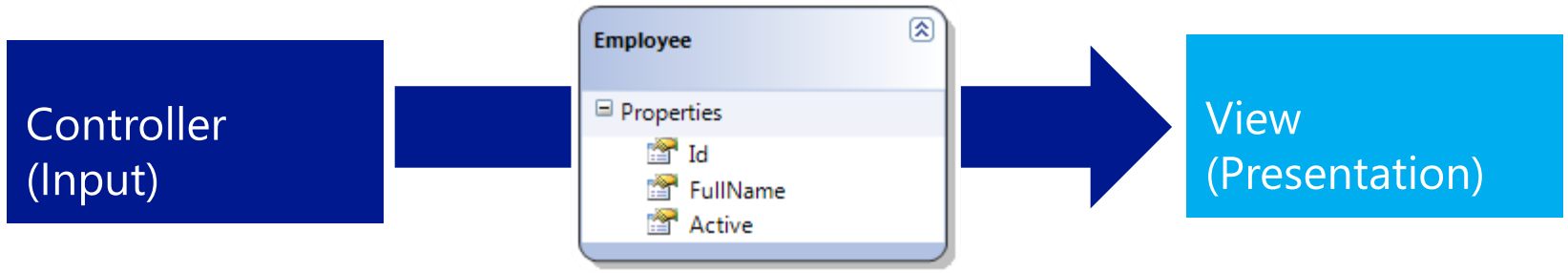
Module 4: Views

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



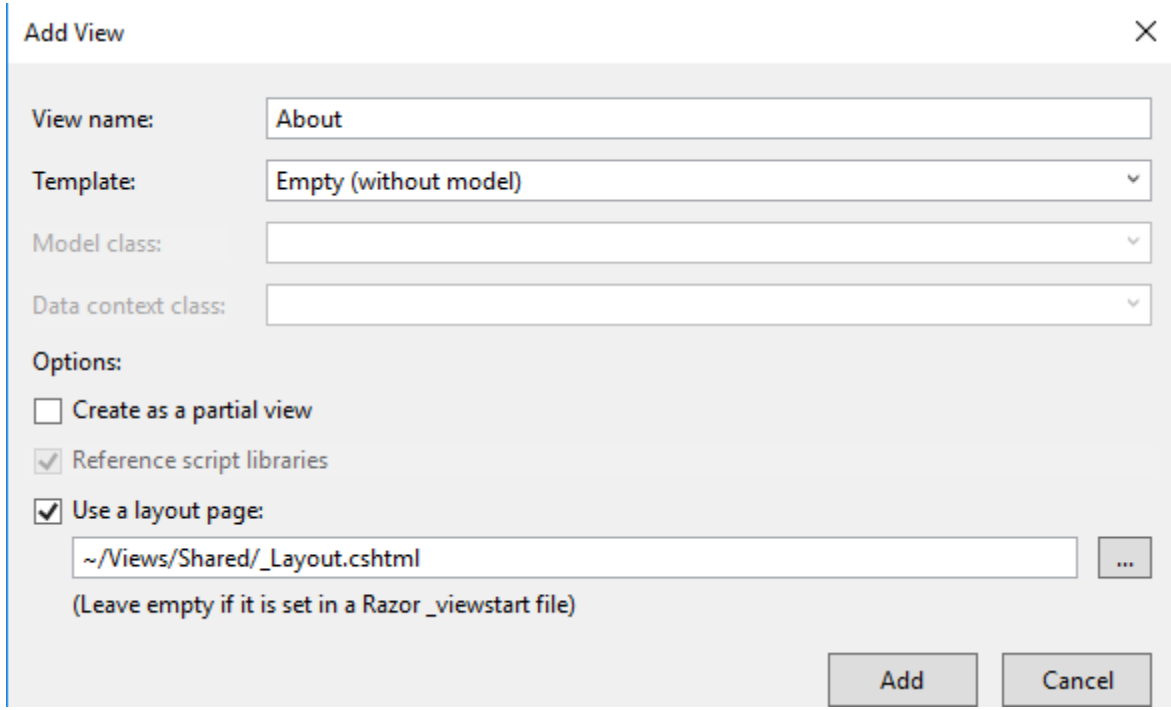
Role of a View

- 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 or *.vbhtml (for classic asp.net)
  - View can be created through:
  - Solution Explorer
  - Action Method



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 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

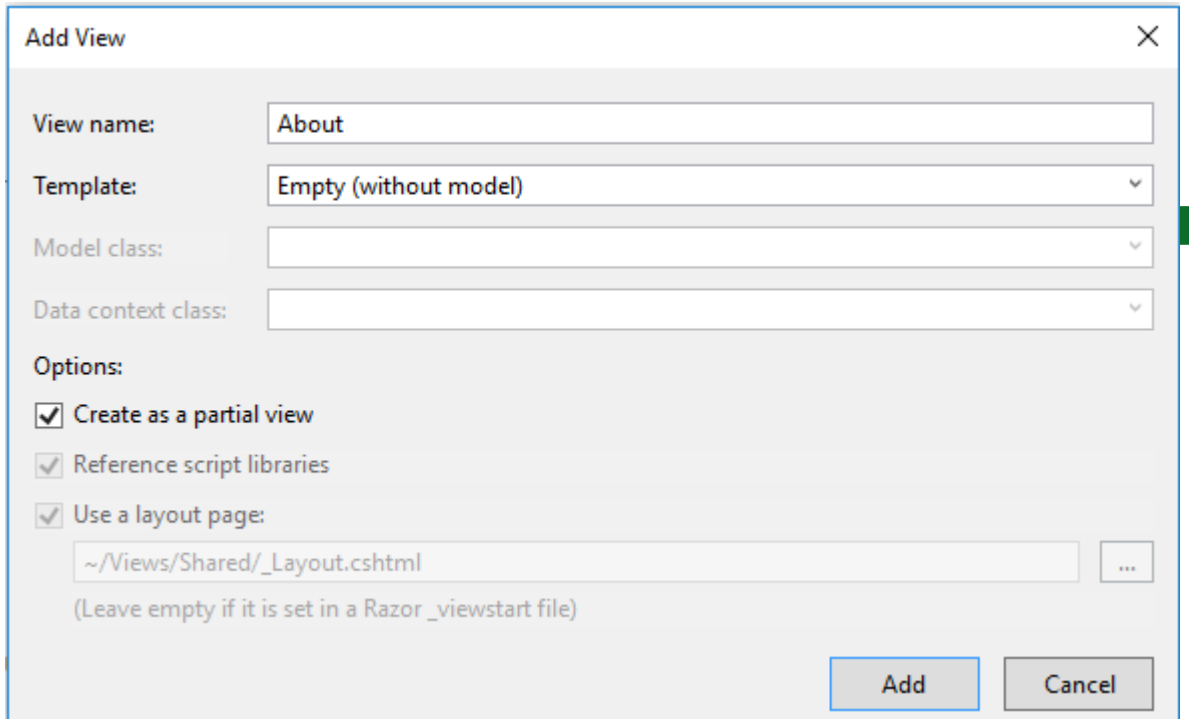
```
@ViewData["Name"]
@ViewData["Age"]
```

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)




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

Module 4: 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]

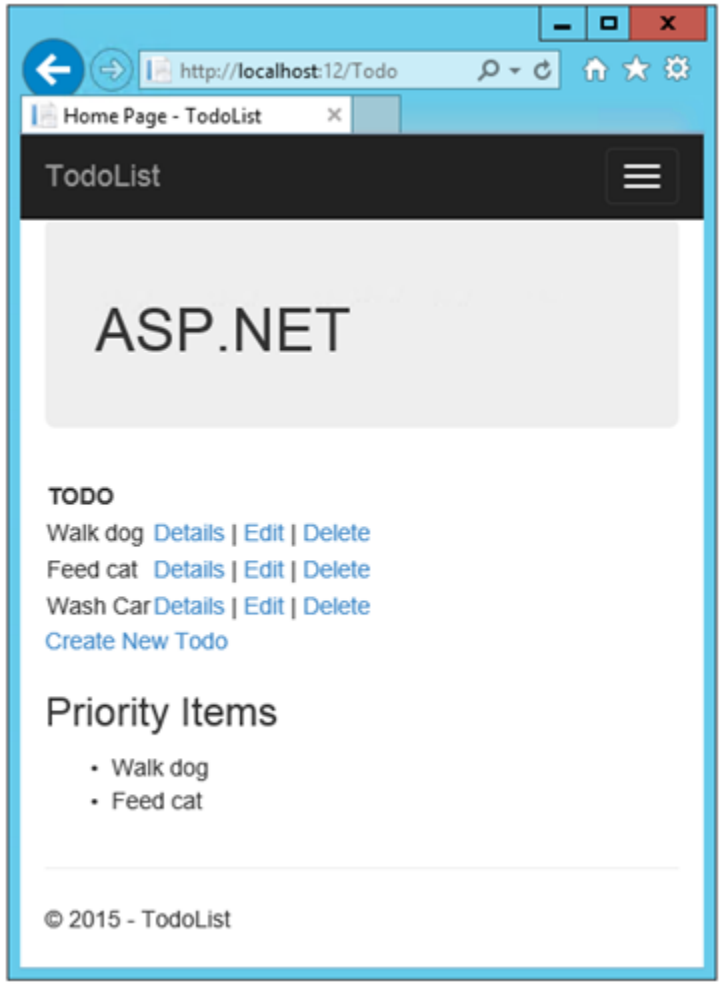
```
@model IEnumerable<TodoList.Models.TodoItem>

<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



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 4: 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>
```

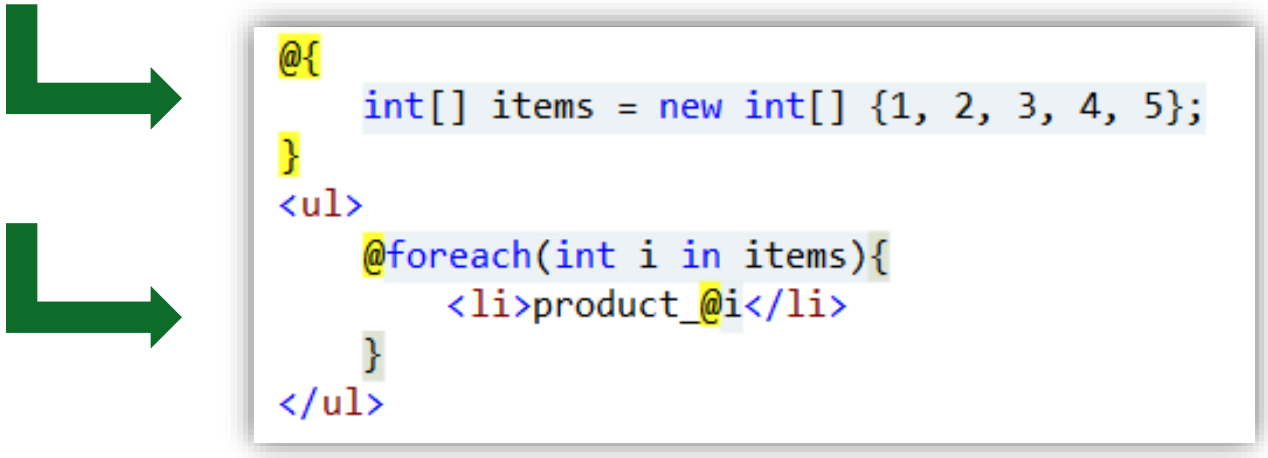
Code Expressions

- '**@**' 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 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>`

Unencoded code expression
`<span>`
`  @Html.Raw(model.AlertMessage)`
`</span>`
	`<span><%:`
	`  Html.Raw( model.AlertMessage)`
	`%></span>`
	
	`Or`
	
	`<span><%= Model.Message %></span>`

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

Razor vs. Web Forms (continued)

Razor Syntax	Web Forms Syntax

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

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

Razor vs. Web Forms (continued)

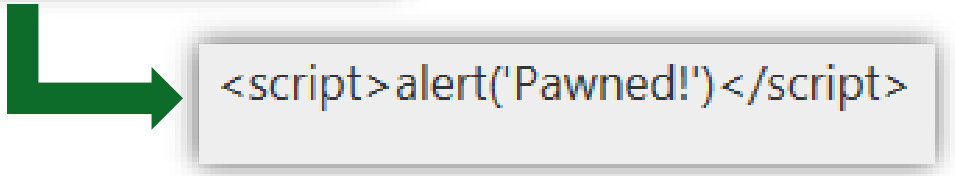
Razor Syntax	Web Forms Syntax
Comments @* Multi-line comment Product name: @ViewBag.Product *@	<!-- Multi-line comment Product name: @ViewBag.Product -->

Demo: Razor View Engine

HTML Encoding

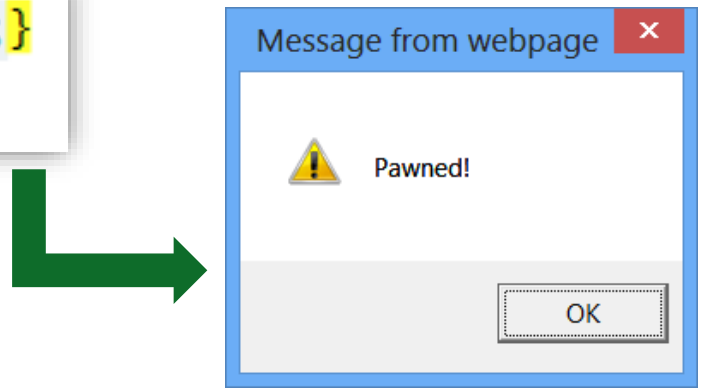
- Razor expressions are always HTML encoded!
  - Defense against Cross-Site Scripting (XSS) attack, etc.

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



- 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 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

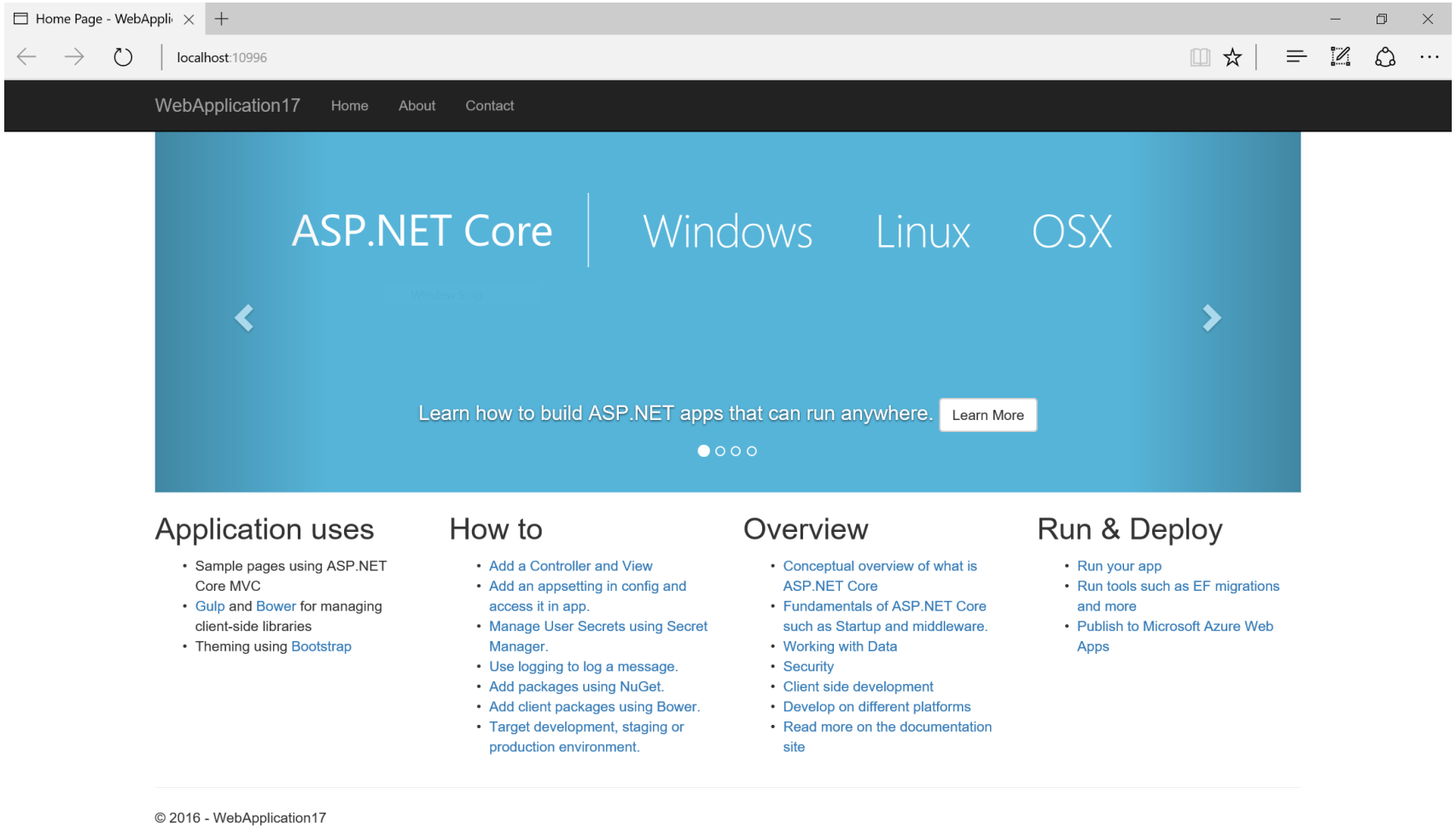
_ViewStart.cshtml

```
@{
    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



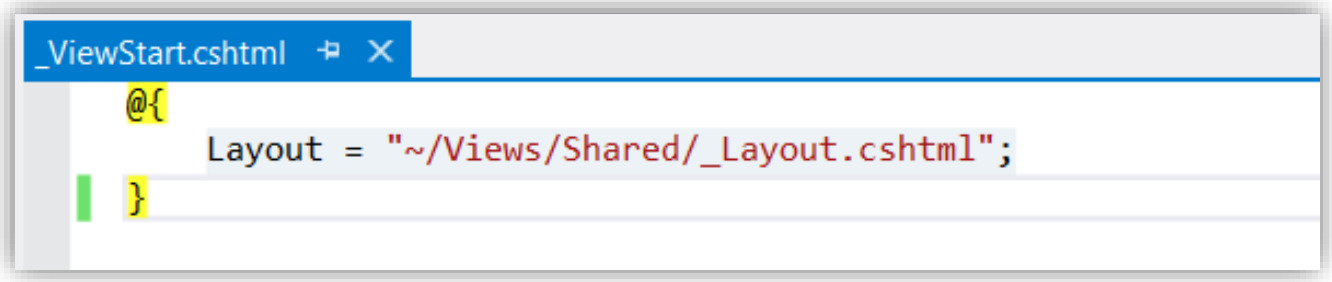
Layout Sections

- 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

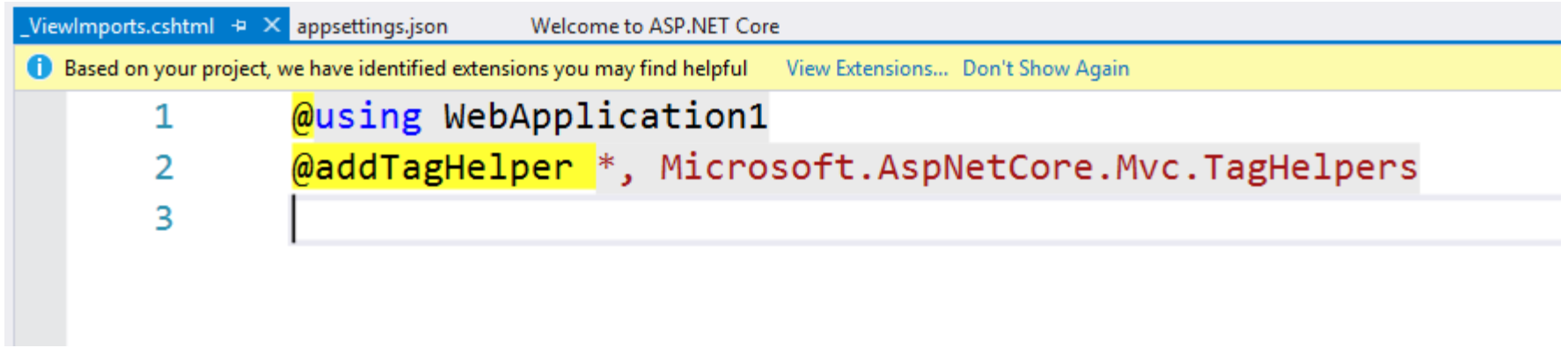
- _ViewStart.cshtml is used to include the same layout in all views by default
- Default layout can be overridden for specific views
  - Blank layout property means no layout has been defined



_ViewStart.cshtml

View Imports

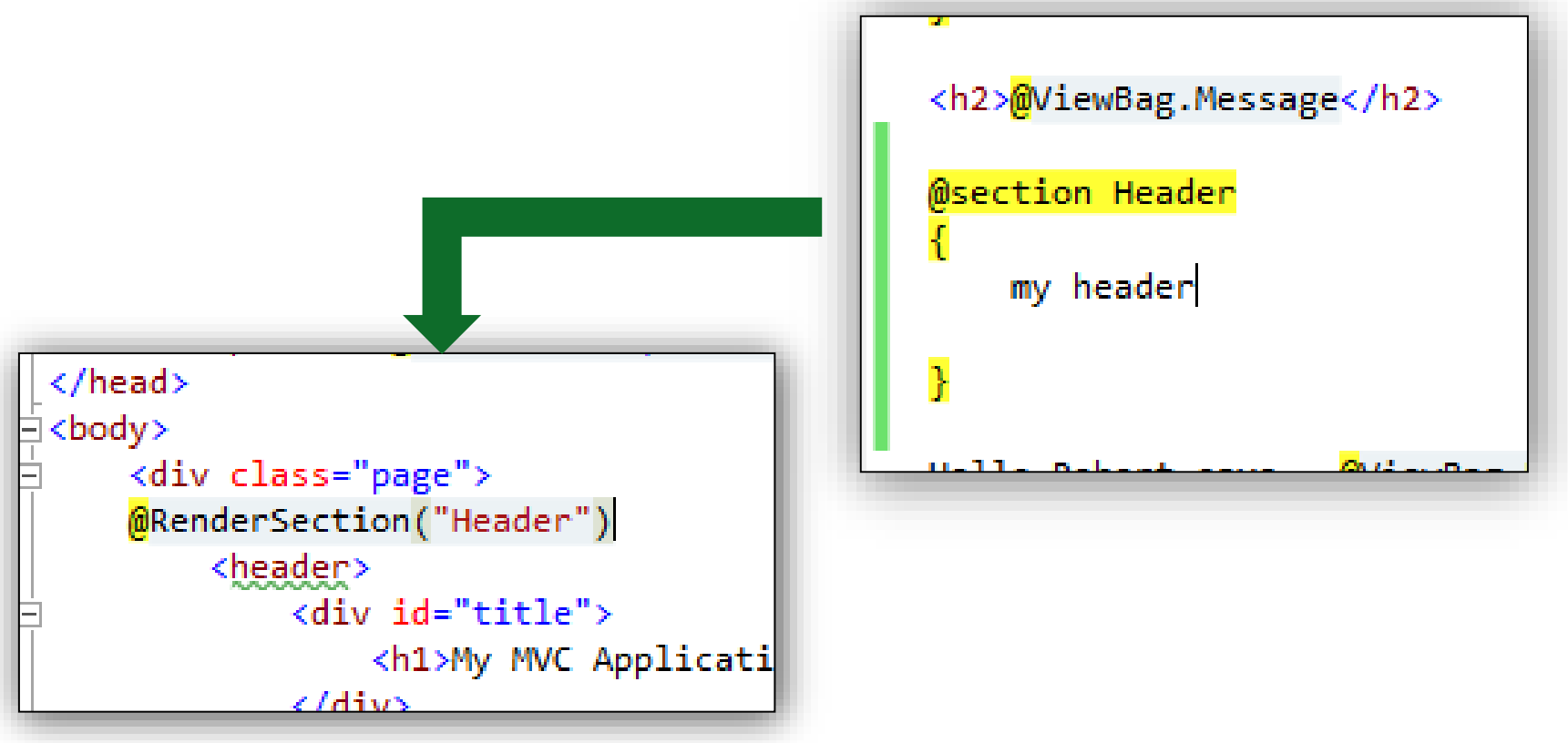
- _ViewImports.cshtml is used to import all the namespaces used by Views
- Views can add specific views in respective files
- Tag Helper global scope is set here



_ViewImports.cshtml

Sections

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



Demo: Razor View Engine

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

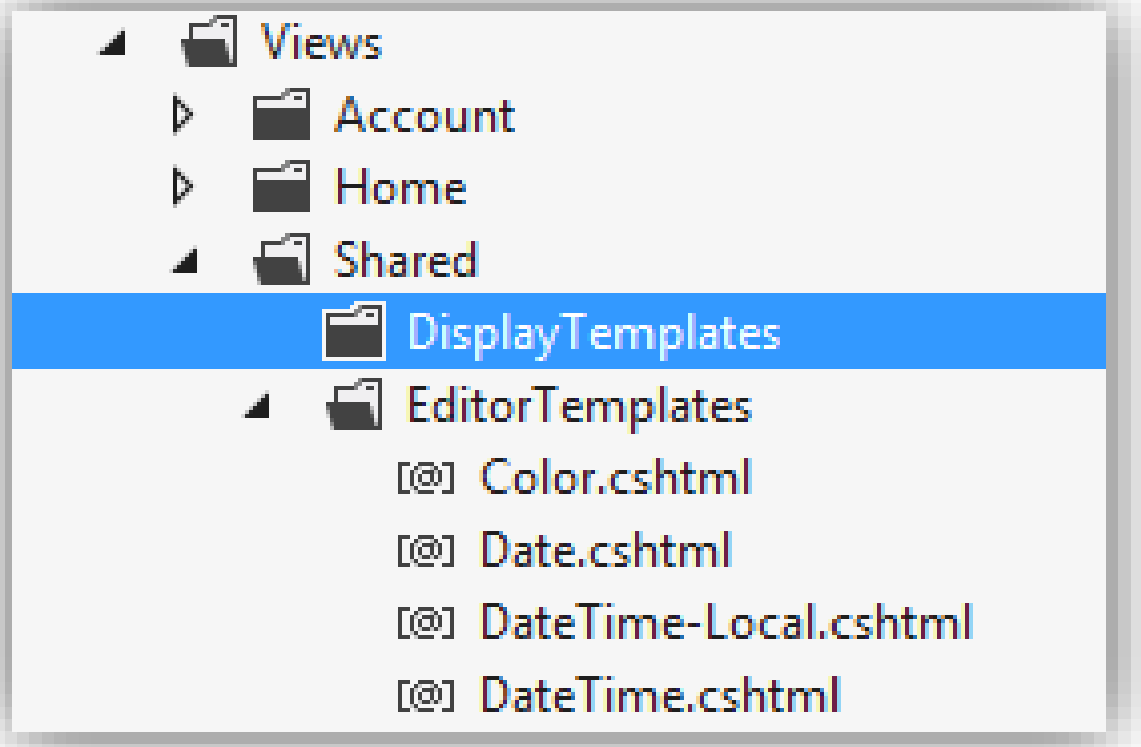
Built-in Editor Templates

- 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 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

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 Helper Scope

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

_ViewImports.cshtml | Program.cs

```
@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>
```

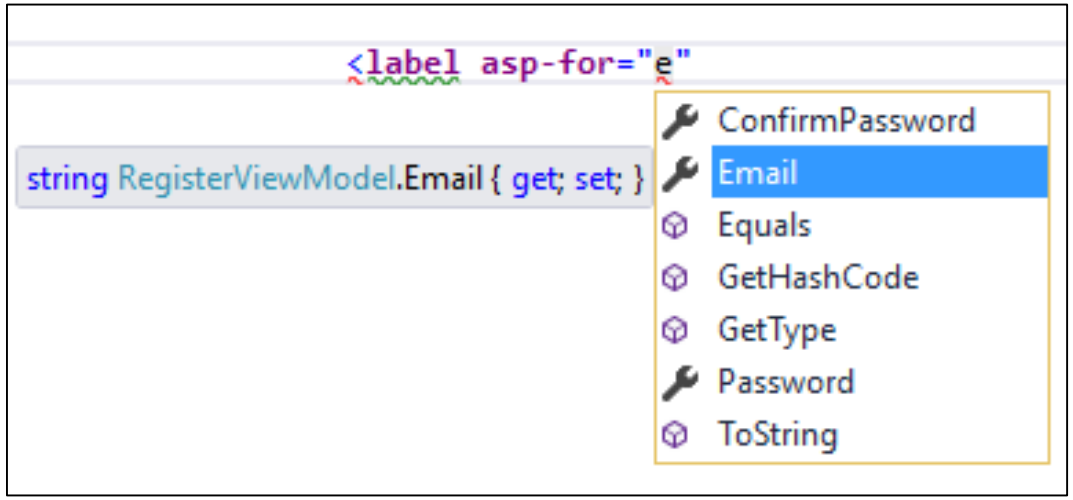
Microsoft.AspNet.Mvc.TagHelpers

- 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

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



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>
```

Tag Helper vs. Web Server Control

- **Tag Helper**
  - No knowledge of browser
  - Only participates in rendering of an element
  - Has no Document Object Model (DOM)
  - Can only modify content of HTML elements, it is scoped to
- **Web Server Control**
  - Automatic browser detection
  - Declared and invoked on a page
  - Adds functionality to DOM
  - Behavior may affect other parts of the page

Demo: Tag Helpers

Module 4: 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.AddMvc();
  services.AddTransient<TodoList.Services.StatisticsService>();
}
```

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

```
@inject TodoList.Services.StatisticsService Statistics
```

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

Module 4: Views

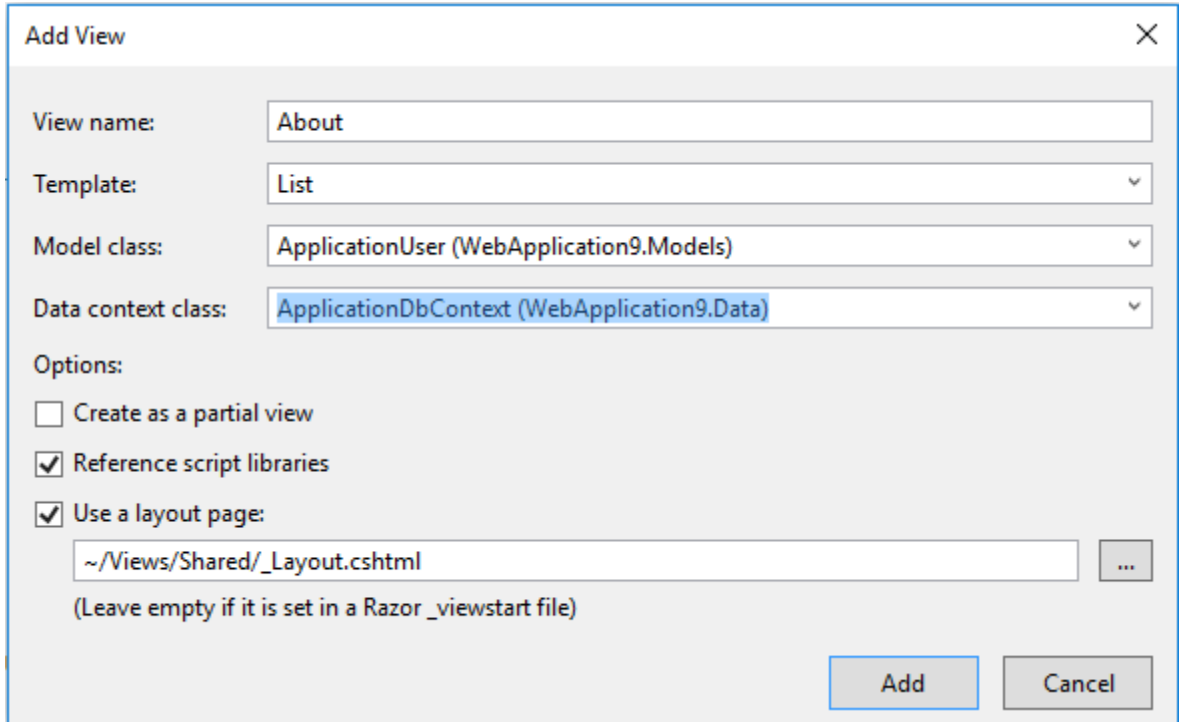
Section 3: Scaffolding

Lesson: Scaffold Templates

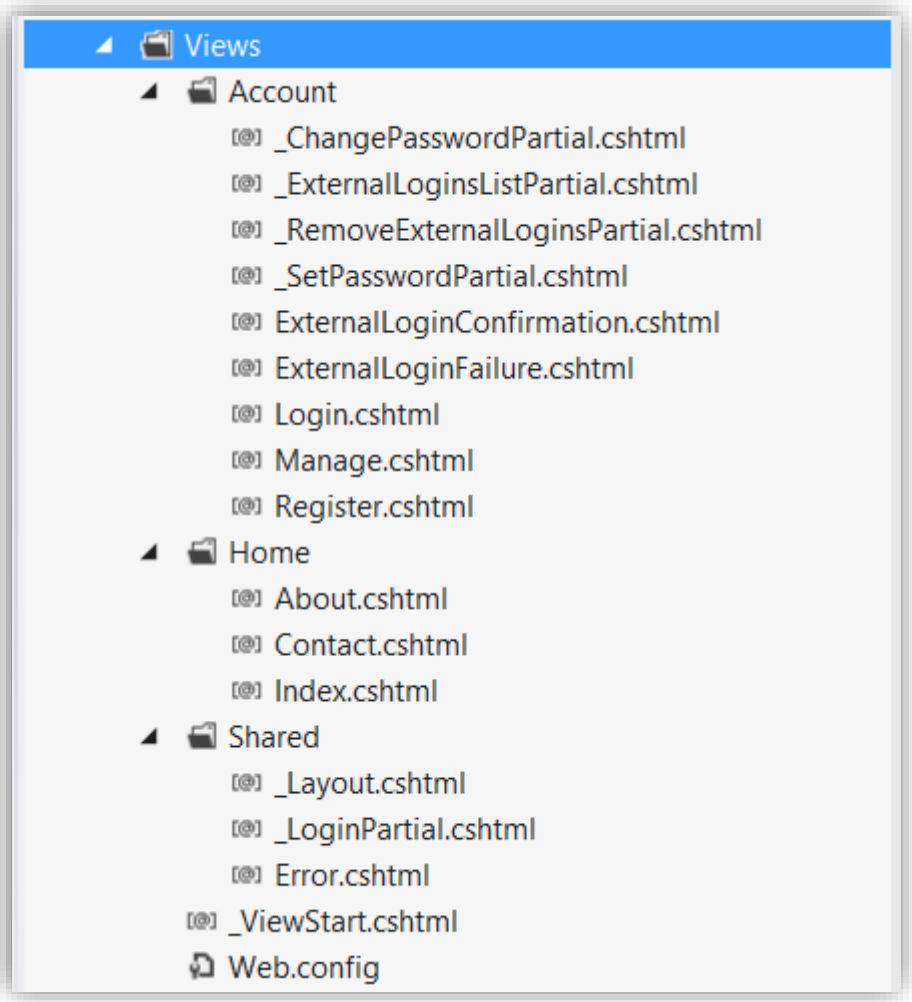
Scaffold Templates - I

- Allows view generation based on selected model type
  - *Empty:* Empty view
  - *Create:* View for creating new instance of a model
  - *Delete:* View for deleting a model instance
  - *Details:* View for showing model instance details
  - *Edit:* View for editing model instance details
  - *List:* View for listing model instances

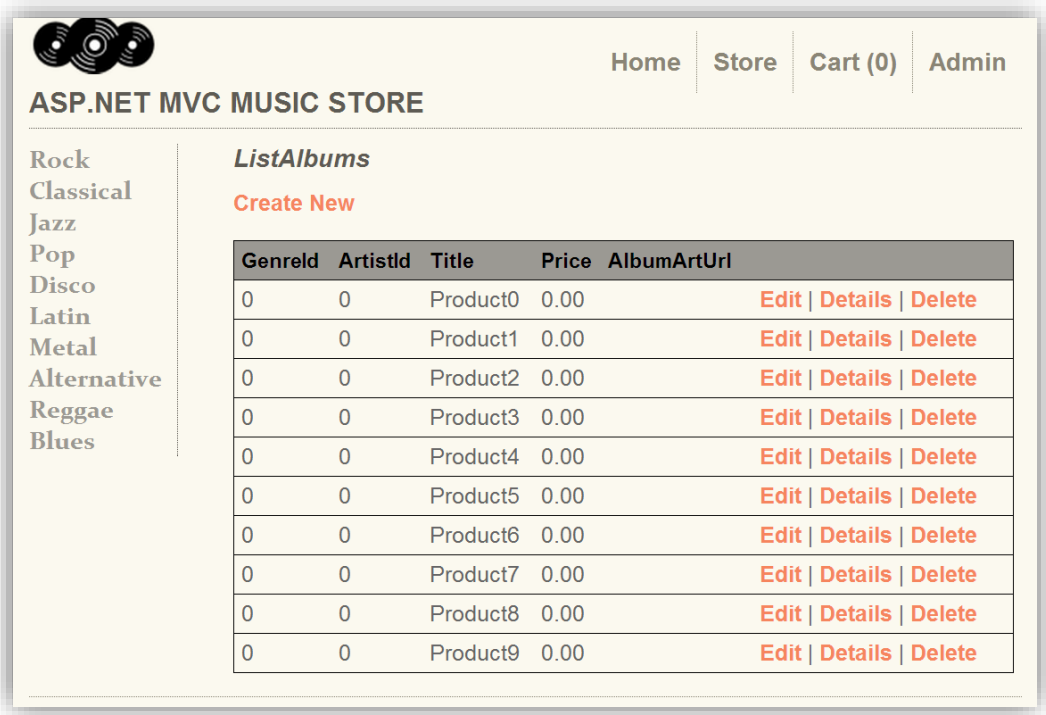
Scaffold Templates - II



Default View Organization



Scaffolding Templates - III



View shown in browser
http://localhost:26641/home/list

View code generated through scaffolding

```
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.GenreId)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ArtistId)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.AlbumArtUrl)
        </td>
```

Module 4: Views

Section 6: Localization

View Localization

- IViewLocalizer service provides localized strings for a view.

_Layout.cshtml

```
1   @using Microsoft.AspNetCore.Mvc.Localization
2   @inject IViewLocalizer Localizer
3
4   <!DOCTYPE html>
5   <html>
6   <head>...</head>
22  <body>
23      <div class="navbar navbar-inverse navbar-fixed-top">
24          <div class="container">
25              <div class="navbar-header">...</div>
34              <div class="navbar-collapse collapse">
35                  <ul class="nav navbar-nav">
36                      <li><a asp-area="" asp-controller="Home" asp-action="Index">@Localizer["Home"]</a></li>
37                      <li><a asp-area="" asp-controller="Home" asp-action="About">@Localizer["About"]</a></li>
38                      <li><a asp-area="" asp-controller="Home" asp-action="Contact">@Localizer["Contact"]</a></li>
39                      <li><a asp-area="" asp-controller="Home" asp-action="SetRussianLanguage">Set Russian lanauge in cookies</a></li>
40                  </ul>
41              </div>
42          </div>
43      </div>
```

View Localization configuration

- Configure service
  - services.AddLocalization(options => options.ResourcesPath = "Resources");
- Configure default culture
  - app.UseRequestLocalization(new RequestLocalizationOptions
  - {
  - DefaultRequestCulture = new RequestCulture("en-US"),
  - // Formatting numbers, dates, etc.
  - SupportedCultures = supportedCultures,
  - // UI strings that we have localized.
  - SupportedUICultures = supportedCultures
  - });
- Create Resources folder in a project and all necessary resource files for all supported languages

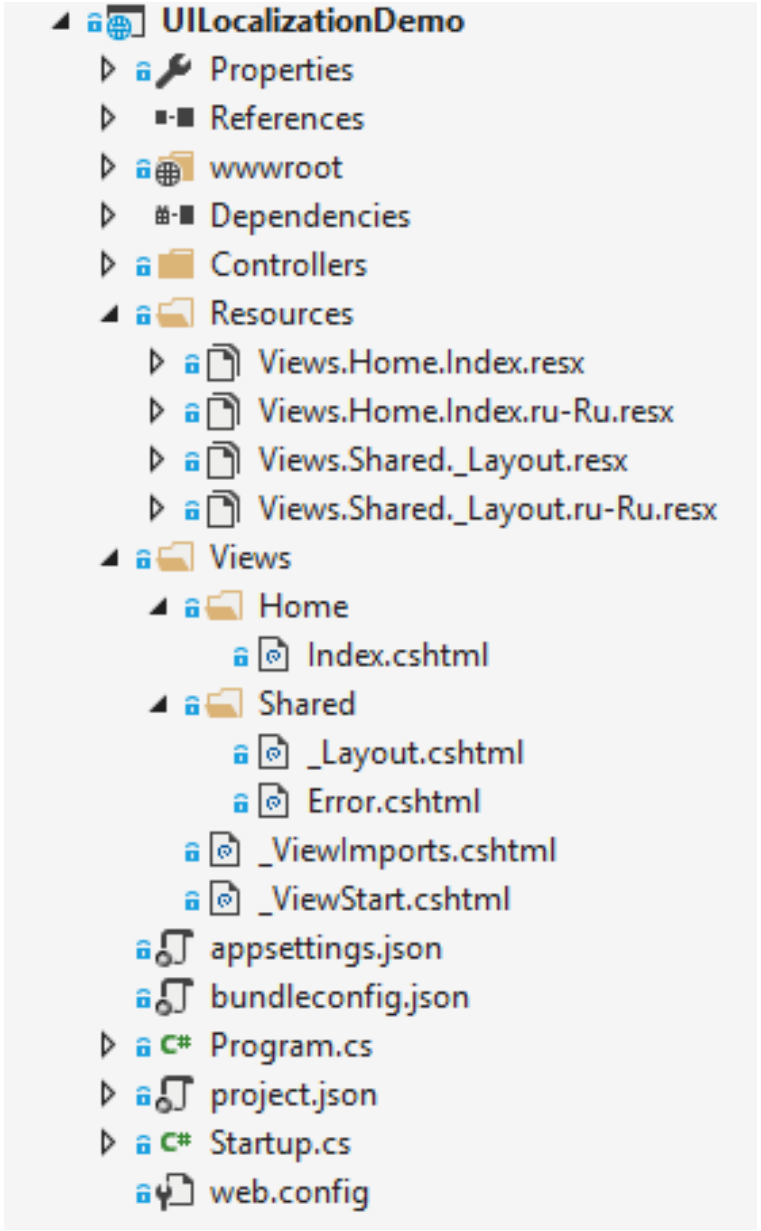
Culture

- What is it en-US ?
  - en is a language
  - US is a culture

- Alternative English cultures:
  - en-AU English - Australia
  - en-BZ English - Belize
  - en-CA English - Canada
  - en-CB English - Caribbean
  - en-IE English - Ireland
  - en-JM English - Jamaica
  - en-NZ English - New Zealand
  - en-PH English - Philippines
  - en-ZA English - South Africa
  - en-TT English - Trinidad and Tobago
  - en-GB English - United Kingdom
  - en-US English - United States
  - en-ZW English - Zimbabwe

View Localization mechanics

- Localization of Index view of home Controller.
  - Full path to a view by default: Views/Home/Index.cshtml
- Asp.Net Core will automatically map view to a resx file
  - Views/Home/Index.cshtml -> Views.Home.Index.resx
- Views.Home.Index.resx location for default language.
  - To additional language we need additiona resx files with similar name:
  - Fur Russian language - Views.Home.Index.ru-Ru.resx
- You can implement IViewLocalizer interface to realize another behavior.



Determining Request Culture

- Query String
  - Default key is culture (http://localhost:5000/?culture=en-CA&culture=fr-FR)
- Using a Cookie
  - Default Cookie name .AspNetCore.Culture
- Accept-Language Header
  - Header name : Accept-Language: de-DE, en-US;q=0.8, fr-FR;q=0.7
- Custom Request Culture Providers

  Implement IRequestCultureProvider interface

Demo: Localization

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
  - HTML5 Markup Elements and Controls
  - Bootstrap

Lab: Views


Microsoft