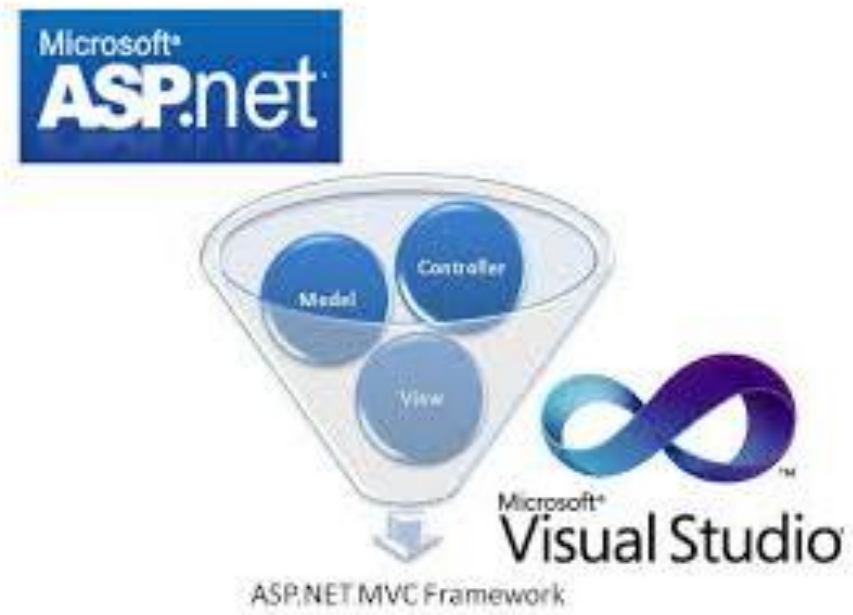
C#




# 3.RAZOR VIEWS IN ASP.NET MVC

# Overview

- Razor Syntax
  - Transition between C# code and HTML code
- Layout
- HTML Helpers
- XSS (Cross Site Scripting) and CSRF (cross site request forgeries)
- Partial Views

# Razor Templates


A templating engine built upon **Microsoft's Razor parsing technology**. The RazorEngine allows you to use Razor syntax to build robust templates

- Markup syntax for adding server-based code to web pages
- It has the power of traditional ASP.NET markup, but is easier to learn and easier to use.
- Razor supports C# and Visual Basic Programming Languages

# Razor Expression

- Razor code blocks are enclosed in @{...}
- Inline expressions (variables and functions) start with @
- Code statements end with semicolon
- Variables are declared with the var keyword
- Strings are enclosed with quotation marks
- C# code is case sensitive
- C# files have the extension **.cshtml**

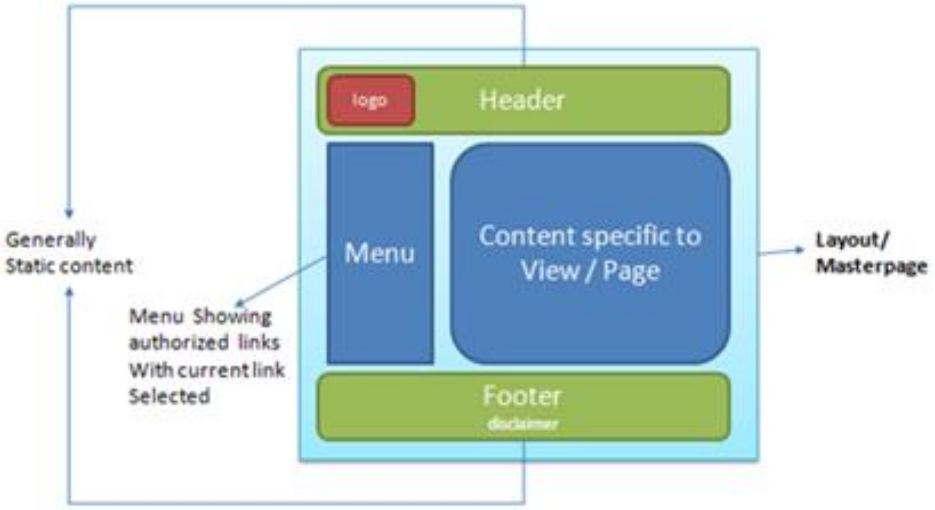
Razor code block

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

```
<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>
```

```
<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

# ContentBlocks




```
<html>
<body>
@RenderPage("header.cshtml")
<h1>Hello Web Pages</h1>
<p>This is a paragraph</p>
@RenderPage("footer.cshtml")
</body>
</html>
```

- With Web Pages you can use the **@RenderPage()** method to import content from separate files.
- Content block (from another file) can be imported anywhere in a web page, and can contain text, markup and code just like any regular web page
- Using common headers and footers as an example, this saves you a lot of work. You don't have to write the same content in every page, and when you change the header or footer files, the content is updated in all your pages.

# Razor Syntax

| Syntax/Sample | Razor | Web Forms Equivalent (or remarks) |
|---|---|---|
| Code Block | @{ int x = 123; string y = "because."; } | <% int x = 123; string y = "because."; %> |
| Expression (Html Encoded) | @model.Message | <%: model.Message %> |
| Expression (Unencoded) |  @Html.Raw(model.Message)  | <%= model.Message %> |
| Combining Text and markup | @foreach(var item in items) { @item.Prop } | <% foreach(var item in items) { %> <%: item.Prop %> <% } %> |

| Syntax/Sample | Razor | Web Forms Equivalent (or remarks) |
|---|---|---|
| Mixing code and Plain text | @if (foo) { <text>Plain Text</text> } | <% if (foo) { %> Plain Text <% } %> |
| Using block | @ using (Html.BeginForm()) { <input type="text" value="input here"> } | <% using (Html.BeginForm()) { %> <input type="text" value="input here"> <% } %> |
| Mixing code and plain text (alternate) | @if (foo) { @:Plain Text is @bar } | Same as above |
| Email Addresses | Hi philha@example.com | Razor recognizes basic email format and is smart enough not to treat the @ as a code delimiter |

| Syntax/Sample | Razor | Web Forms Equivalent (or remarks) |
|---|---|---|
| Explicit Expression | ISBN@(isbnNumber) | In this case, we need to be explicit about the expression by using parentheses. |
| Escaping the @ sign | In Razor, you use the @@foo to display the value of foo | @@ renders a single @ in the response. |
| Server side Comment | @* This is a server side multiline comment *@ | <%-- This is a server side multiline comment --%> |
| Calling generic method | @(MyClass.MyMethod<AType>()) | Use parentheses to be explicit about what the expression is. |
| Creating a Razor Delegate | @{ Func<dynamic, object> b = @@item; } @b("Bold this") | Generates a Func<T, HelperResult> that you can call from within Razor. See this blog post for more details. |
| Mixing expressions and text | Hello @title. @name. | Hello <%: title %>. <%: name %>. |

# NEW IN RAZOR v2.0/ASP.NET MVC 4

| Syntax/Sample | Razor | Web Forms Equivalent (or remarks) |
|---|---|---|
| Conditional attributes |  | When className = null When className = "" When className = "my-class" |
| Conditional attributes with other literal values |   | When className = null *Notice the leading space in front of* foo *is removed.*<br>When className = "my-class"  |
| Conditional data-* attributes.<br><br>*data-* attributes are always rendered.* |  | When xpos = null or "" When xpos = "42" |
| Boolean attributes | <input type="checkbox" checked="@isChecked" /> | When isChecked = true<input type="checkbox" checked="checked" />When isChecked = false<input type="checkbox" /> |
| URL Resolution with tilde | <script src="~/myscript.js"> </script> | When the app is at /<script src="/myscript.js"> </script>When running in a virtual application named MyApp<script src="/MyApp/myscript.js"> </script> |

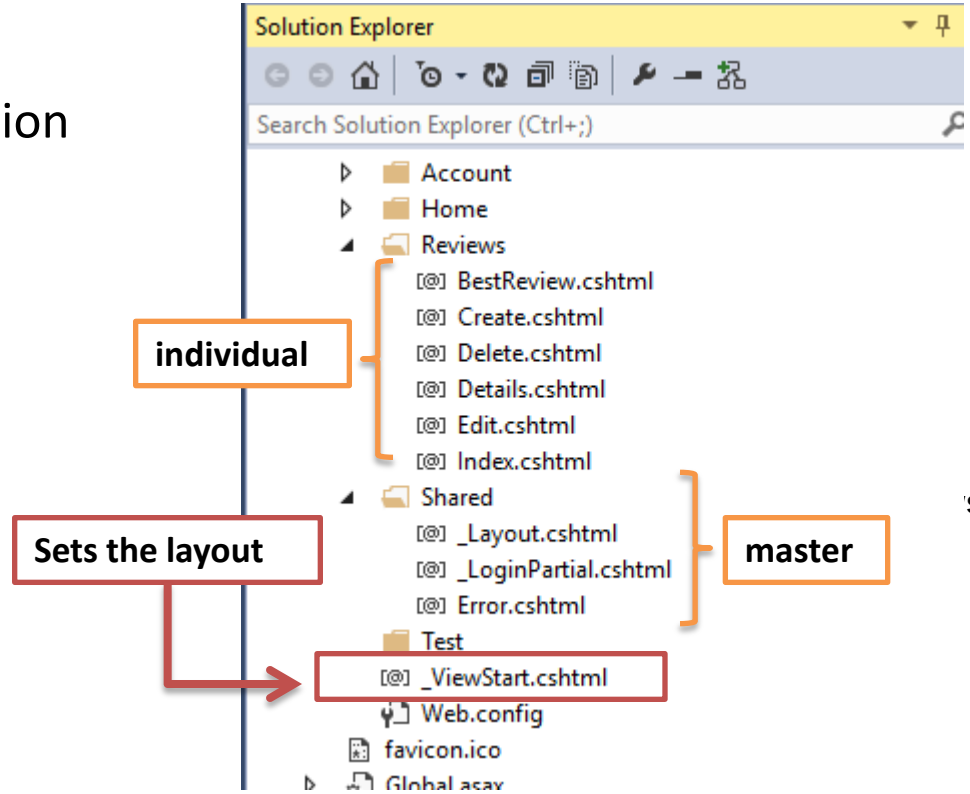
# Razor Layout

http://razorcheatsheet.com/

- Layout views are “master pages” for razor
- Use inherited method to specify content areas
  - RenderBody
  - RenderSection
- Common UI Structure for application

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")"
            type="text/javascript"></script>
</head>

<body>
    @RenderBody()
</body>
</html>
```



# Razor Layout

```
        <div id="logindisplay">
            @Html.Partial("_LogOnPartial")
        </div>

        <div id="menucontainer">

            <ul id="menu">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
            </ul>

        </div>
    </div>

    <div id="main">
        @RenderBody()
        <div id="footer">
        @RenderSection("Footer", true);
        </div>
    </div>
</div>
</body>
</html>
```

# HTML Helpers

- HTML Helpers are used to modify HTML output (easy to create small blocks of HTML)
- HTML is a property of the ViewPage base class
  - Create input
  - Create links
  - Create forms

HTML.ActionLink

```
Html.ActionLink(article.Title,
                "Item",   // <-- ActionMethod
                "Login",  // <-- Controller Name.
                new { article.ArticleID }, // <-- Route arguments.
                null  // <-- htmlArguments .. which are none. You need this value
                      //     otherwise you call the WRONG method ...
                      //     (refer to comments, below).
                )
```

```
routes.MapRoute(
    "Default",     // Route name
    "{controller}/{action}/{id}",                           // URL with parameters
    new { controller = "Home", action = "Index", id = "" }  // Parameter defaults
);
```

# HTML.ActionLink()

- The easiest way to render an HTML link in is to use the **HTML.ActionLink()** helper.
- With MVC, the **Html.ActionLink()** does not link to a view. It creates a link to a controller action.

| Property | Description | |
|---|---|---|
| @Html.ActionLink() | | |
| @Html.linkText | | |
| @Html.action() | | |
| @Html.AntiForgeryToken() | | |
| @Html.RouteCollection() | | |
| | | |

# HTML Helper : FORM ELEMENTS

- HTML helpers are available for every kind of form control
- HTML label elements use descriptive text to form control and provide usability improvements
- Each helper method provides a shorthand way to render valid HTML for the specific control

| Helper | HTML Element |
|---|---|
| Html.CheckBox | <input type="checkbox" /> |
| Html.DropDownList | <select></select> |
| Html.Hidden | <input type="hidden" /> |
| Html.Label | <label for="" /> |
| Html.ListBox | <select></select> or <select multiple></select> |
| Html.Password | <input type="password" /> |
| Html.Radio | <input type="radio" /> |
| Html.TextArea | <textarea></textarea> |
| Html.TextBox | <input type="text" /> |

```
@using ( Html.BeginForm() )
{
    <fieldset>
        <legend>Product</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.TextBoxFor(model => model.Name)
        </div>

        <div class="editor-label">
            <div style="float: left;">
                @Html.CheckBoxFor(model => model.Featured)
            </div>
            @Html.LabelFor(model => model.Featured)
        </div>

        <!-- Price, Featured, ... -->
    </fieldset>
}
```

C#

```
@using (Html.BeginForm()) {
        @Html.ValidationSummary(true)
            <div class="editor-label">
                @Html.LabelFor(model => model.FirstName)
            </div>
            <div class="editor-field">
                @Html.EditorFor(model => model.FirstName)
                @Html.ValidationMessageFor(model => model.FirstName)
            </div>
```

# TryUpdateModel()

- **TryUpdateModel()** allows you to bind parameters to your model inside your action.
- useful if you want to load your model from a database then update it based on user input rather than taking the entire model from user input.
- use this method to update the model that backs a particular view via the given controller.

```
public class Student {
  public string studentID { get; set; }
}

// ... in the controller
public ActionResult Save() {
  var myStudent = new Student();
  TryUpdateModel(myStudent);
}
```

# Custom Helpers

- can create custom HTML Helpers that you can use within your MVC views.
- By taking advantage of HTML Helpers, you can reduce the amount of tedious typing of HTML tags that you must perform to create a standard HTML page.

We need a custom helper for image tag as shown below

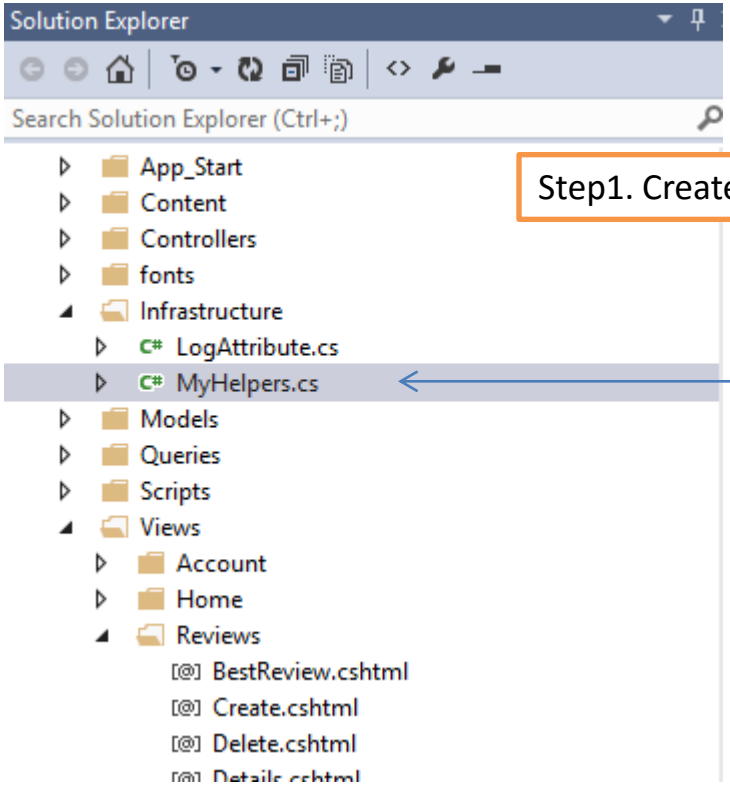
```
<img src="@item.Restaurant.ImageUrl" alt="@item.Restaurant.Name" />
```

Custom Html helper tag in the view that generates above html snippet:

```
@Html.Image(item.Restaurant.ImageUrl, item.Restaurant.Name)
```

**Introduced in C# 3.0 Extension Methods to create custom helpers**

# Steps to Create a Custom Helper




Step1. Create a Helpers Class

Step2: Write the builder function

```
namespace ZomatoReviewApp.Infrastructure
{
    0 references
    public class MyHelpers
    {
        //html required output
        //<img src="@item.Restaurant.ImageUrl" alt="@item.Restaurant.Name"/>
        //custom helper
        //@html.Image(@item.Restaurant.ImageUrl, @item.Restaurant.Name)
        0 references
        public static MvcHtmlString Image(this
          System.Web.WebPages.Html.HtmlHelper helper,
            string src, string altText)
        {
            var builder = new TagBuilder("img");
            builder.MergeAttribute("src", src);
            builder.MergeAttribute("alt", altText);
            return MvcHtmlString.Create(builder.ToString
              (TagRenderMode.SelfClosing));
        }
    }
}
```

Step3: include the namespace in the views

@ using zomatoreviewapp.Infrastructure **(locally)**

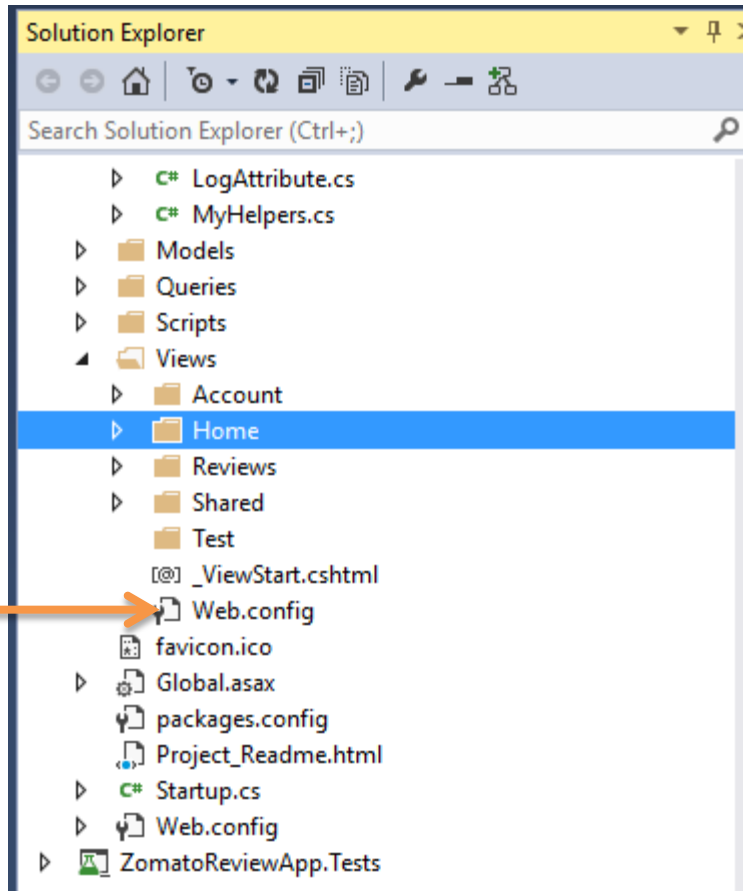
Step4: Use Html.Image() in View

@Html.Image(item.Restaurant.ImageUrl, item.Restaurant.Name)

# Registering Custom Helper in NameSpace Globally

**Now you can use the**
**@html.Image()**
**Custom helper globally in all the views**

1.Open Web.config



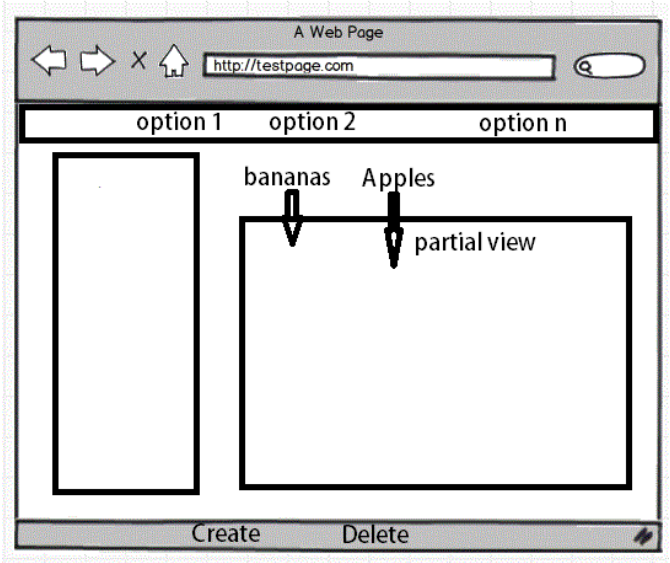

2.add

```
<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web.Mvc,
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Optimization"/>
      <add namespace="System.Web.Routing" />
      <add namespace="ZomatoReviewApp" />
      <add namespace="ZomatoReviewApp.Infrastructure" />
    </namespaces>
  </pages>
</system.web.webPages.razor>
```

http://www.asp.net/mvc/overview/older-versions-1/views/creating-custom-html-helpers-cs

# Partial Views

- Allows users to put html and C# code into file for code reusability.
- A partial view enables you to define a view that will be rendered inside a parent view.
- By using partial view we can render a view inside a parental view and to create reusable content in the project




```
<div id="main">
    @RenderBody()
    <div id="footer">
    @Html.Action("BestReview", "Reviews")
    @RenderSection("Footer", false)
    </div>
</div>
```

```
<h2>The Latest Reviews</h2>

@foreach (var item in Model)
{
    @Html.Partial("_Review", item)
}
<p>
    @Html.ActionLink("Create New", "Create")
</p>

@section Footer {

    <p>This is the footer</p>

}
```

# Security

- Encoding
  - Helps to avoid XSS(cross site scripting) attacks
  - Not encoding user input makes you particularly vulnerable
- **Html.AntiForgeryToken**
  - Helps to avoid CSRF attacks
  - Requires a ValidateAntiForegeryToken attribute on controller action
  - Valid only for POST operators
  - Cross Site Request forgery is a type of a hack where the hacker exploits the trust of a website on the user.
  - the site trusts the user (because they have authenticated themselves) and accepts data that turns out to be malicious.

# Summary

- Razor Syntax – Implicit and Explicit expressions
- HTML Helpers and Creating Custom HTML Helpers
- Razor Layout
- Partial Views
- Introduced the concepts of XSS and CSRF