ASP Razor

Razor is the name of the MVC Framework view engine

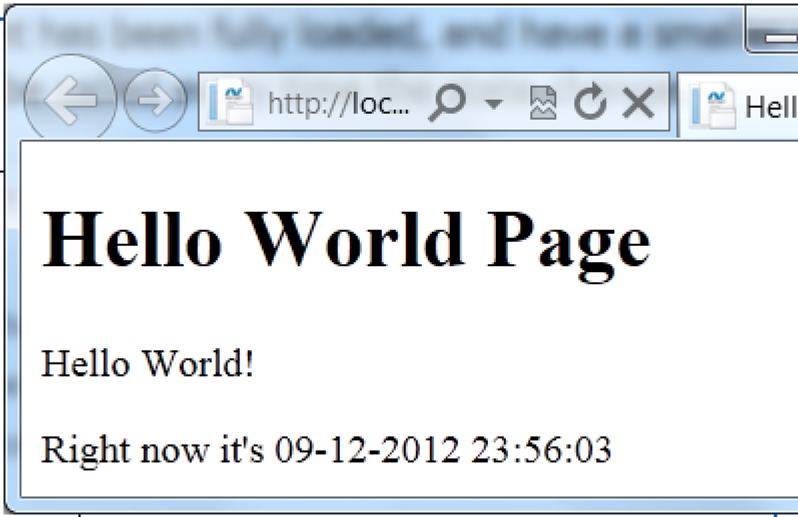
A ***view engine*** processes content and looks for instructions, typically to insert dynamic content into the output sent to the browser

Razor and C#

- Razor refers to the small set of conventions for how you embed C# code into a page
- For example, the convention of using **@** to mark code in the page and using **@{ }** to embed a code block is the Razor aspect of a page
  - Html Helpers are also considered to be part of Razor
- Razor syntax is used in both MVC view files and ASP.NET Web Pages
- But you should not use Razor to perform business logic or manipulate your domain model objects in any way!

ASP –Razor Code

```
@{
    var currentDateTime = DateTime.Now;
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Hello World Page</title>
</head>
<body>
    <h1>Hello World Page</h1>
    <p>Hello World!</p>
    <p>Right now it's @currentDateTime</p>
</body>
</html>
```



Hello World Page

Hello World!

Right now it's 09-12-2012 23:56:03

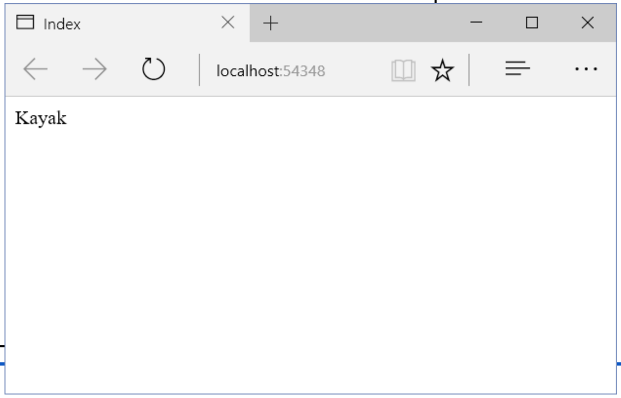
```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Hello World Page</title>
  </head>
  <body>
    <h1>Hello World Page</h1>
    <p>Hello World!</p>
    <p>Right now it's 09-12-2012 2
  </body>
</html>
```

@ marks code that ASP has to process before the pages is send to the client (browser)

The Model Object

```
public ActionResult Index()
{
    return View(myProduct);
}
```

```
@model Razor.Models.Product

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        @Model.Name
    </div>
</body>
</html>
```



Kayak

No Layout

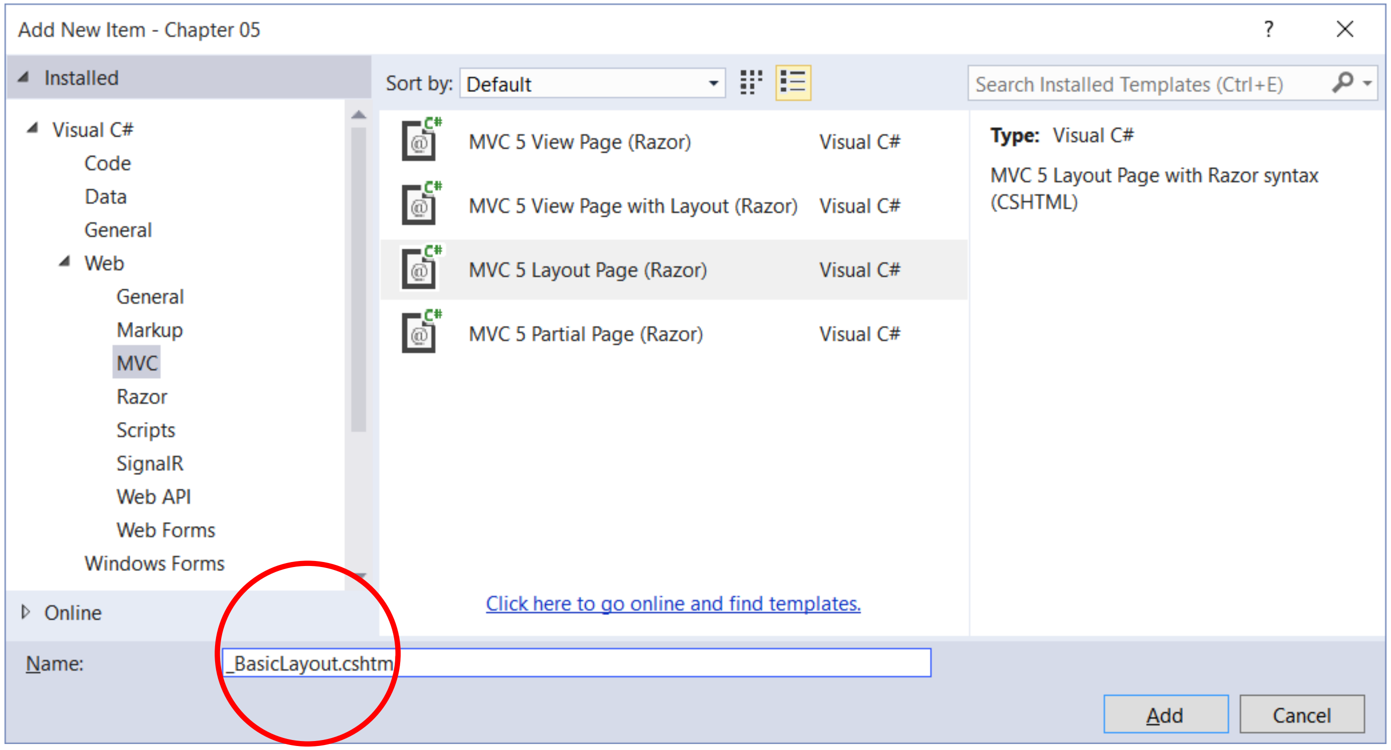
- Setting the Layout property to null tells the MVC framework that the view is self-contained and will render all of the content required for the client

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

A full html page

Creating a Layout Template

- Layouts are templates that contain markup that you use to create consistency across your app



Add New Item - Chapter 05

Installed

- Visual C#
  - Code
  - Data
  - General
  - Web
    - General
    - Markup
    - MVC
    - Razor
    - Scripts
    - SignalR
    - Web API
    - Web Forms
  - Windows Forms

Online

Sort by: Default

Search Installed Templates (Ctrl+E)

MVC 5 View Page (Razor)	Visual C#
MVC 5 View Page with Layout (Razor)	Visual C#
MVC 5 Layout Page (Razor)	Visual C#
MVC 5 Partial Page (Razor)	Visual C#

Type: Visual C#

MVC 5 Layout Page with Razor syntax (CSHTML)

Click here to go online and find templates.

Name: _BasicLayout.cshtml

Add Cancel

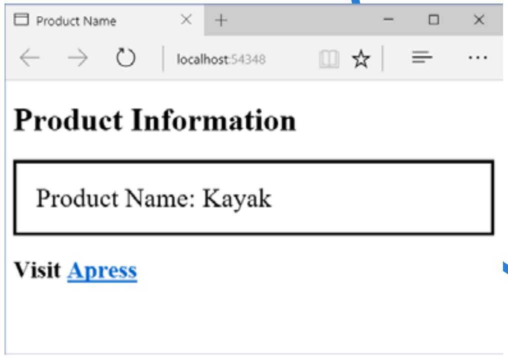
Files in the Views folder whose names begin with an underscore (_) are not returned to the user

Applying the Layout

_BasicLayout.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <h1>Product Information</h1>
    <div style="padding: 20px; border: solid medium black; font-size: 20pt">
        @RenderBody()
    </div>
    <h2>Visit <a href="http://apress.com">Apress</a></h2>
</body>
</html>
```

```
@model Razor.Models.Product

@{
    ViewBag.Title = "Product Name";
    Layout = "~/Views/_BasicLayout.cshtml";
}

Product Name: @Model.Name
```



Product Information

Product Name: Kayak

Visit Apress

Using a View Start File

- The MVC framework will look for a file called _ViewStart.cshtml
  - The contents of this file will be treated as though they were contained in the view file itself
  - It can be used to automatically set a value for the Layout property

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

- The ASP.MVC vizard generates this _ViewStart.cshtml file:

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

@RenderSection

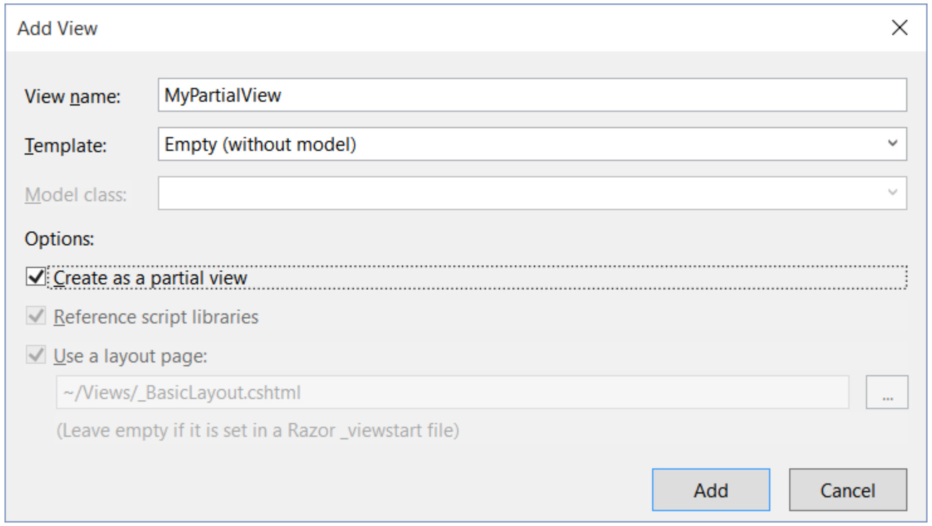
- With @RenderSection you can insert defined sections in a layout

```
<!DOCTYPE html>
<html>
<head>
  <title>@ViewBag.Title - My ASP.NET Application</title>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <button type="button">
    </div>
    <div class="container body-content">
        @RenderBody()
    </div>
    <div>
        @RenderSection("Footer",  false)
    </div>
    @RenderSection("scripts", required: false)
</body>
</html>
```

```
@section Footer {
    Copyright @DateTime.Now.Year ase.au.dk
}
```

Partial Views

- Use partial views when you need to use the same fragments of Razor tags and HTML markup in several different places in the application



Add View

View name: MyPartialView

Template: Empty (without model)

Model class:

Options:

[✓] Create as a partial view

[✓] Reference script libraries

[✓] Use a layout page:

~/Views/_BasicLayout.cshtml

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

- What makes a view a partial is:
  - its content
    - it only contains a fragment of HTML, and doesn't reference layouts
  - the way that it is used

Using a Partial View

```
@*DemoPartialView.cshtml*@

@{
    ViewBag.Title = "DemoPartialView";
    Layout = "~/Views/_BasicLayout.cshtml";
}

<h2>DemoPartialView</h2>
@Html.Partial("MyPartialView")
```

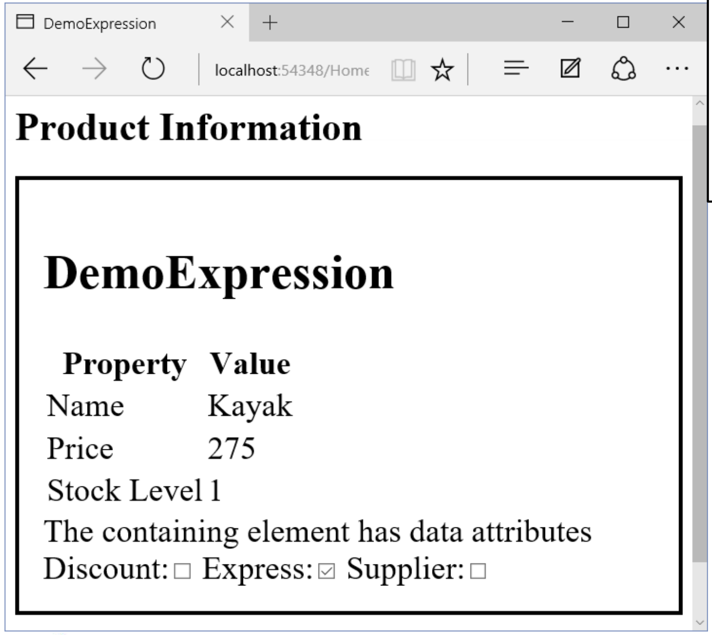
```
@*MyPartialView.cshtml*@

<div>
    This is the message from the partial view.<br />
    @Html.ActionLink("This is a link to the Index action", "Index")
</div>
```

Setting Attribute Values

```
public ActionResult DemoExpression()
{
    ViewBag.ProductCount = 1;
    ViewBag.ExpressShip = true;
    ViewBag.ApplyDiscount = false;
    ViewBag.Supplier = null;
    return View(myProduct);
}
```

```
<div data-discount="@ViewBag.ApplyDiscount"
     data-express="@ViewBag.ExpressShip"
     data-supplier="@ViewBag.Supplier">
    The containing element has data attributes
</div>
Discount:<input type="checkbox“
              checked="@ViewBag.ApplyDiscount" />
Express:<input type="checkbox"
              checked="@ViewBag.ExpressShip" />
Supplier:<input type="checkbox"
              checked="@ViewBag.Supplier" />
```

```
<div data-discount="False" data-express="True"
    data-supplier="">
   The containing element has data attributes
</div>
Discount:<input type="checkbox" />
Express:<input type="checkbox" checked="checked" />
Supplier:<input type="checkbox" />
```



Product Information

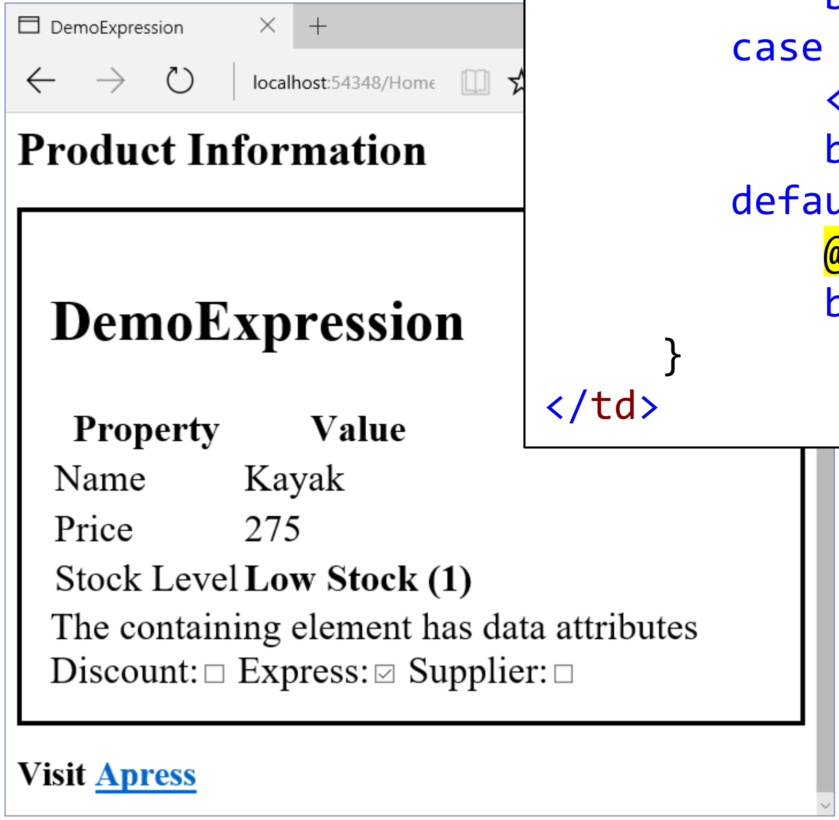
DemoExpression

Property	Value
Name	Kayak
Price	275
Stock Level	1

The containing element has data attributes
Discount: ☐ Express: ☑ Supplier: ☐

Conditional Statements

```
<td>Stock Level</td>
<td>
    @switch ((int)ViewBag.ProductCount)
    {
        case 0:
            @: Out of Stock
            break;
        case 1:
            <b>Low Stock (@ViewBag.ProductCount)</b>
            break;
        default:
            @ViewBag.ProductCount
            break;
     }
</td>
```


DemoExpression

localhost:54348/Home

Product Information

DemoExpression

Property	**Value**
Name	Kayak
Price	275
Stock Level	**Low Stock (1)**

The containing element has data attributes

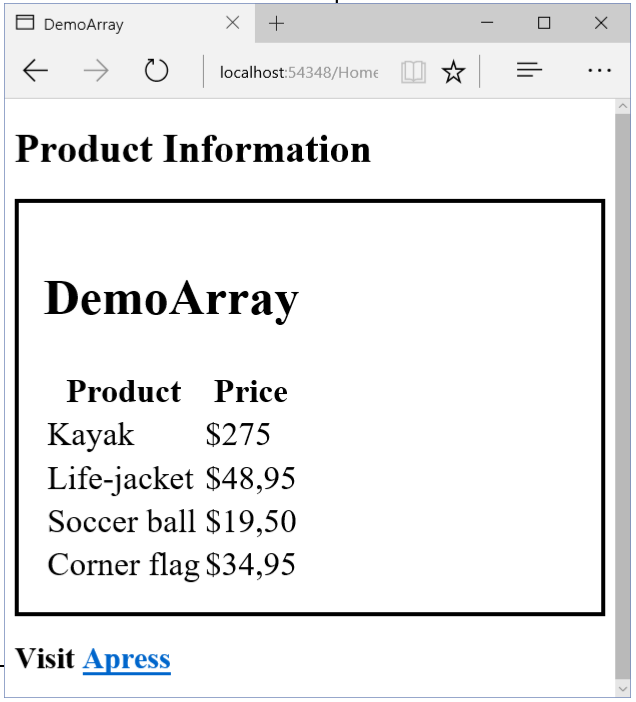
Discount: ☐ Express: ☑ Supplier: ☐

Visit Apress

AARHUS UNIVERSITY SCHOOL OF ENGINEERING

Arrays and Collections

```
@model Razor.Models.Product[]

@if (Model.Length > 0)
{
    <table>
        <thead><tr><th>Product</th><th>Price</th></tr></thead>
        <tbody>
            @foreach (var p in Model)
            {
                <tr>
                    <td>@p.Name</td>
                    <td>$@p.Price</td>
                </tr>
            }
        </tbody>
    </table>
}
else
{
    <h2>No product data</h2>
}
```



Product Information

DemoArray

Product	Price
Kayak	$275
Life-jacket	$48,95
Soccer ball	$19,50
Corner flag	$34,95

Visit Apress

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Using Namespaces

- You may occasionally need to reference namespaces in a Razor file
  - E.g. if you pass a model object (custom type) through the ViewBag
- To do so simply at @ in front of the using statement

```
@using Razor.Models
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

HTML Helpers

- Are extension methods to the HTMLHelper class
- Use them to produce clean markup

```
@Html.LabelFor(m => m.UserName)
@Html.TextBoxFor(m => m.UserName)
```

- If the property includes a DisplayAttribute, its value will be displayed
  - Otherwise, the name of the property will be displayed

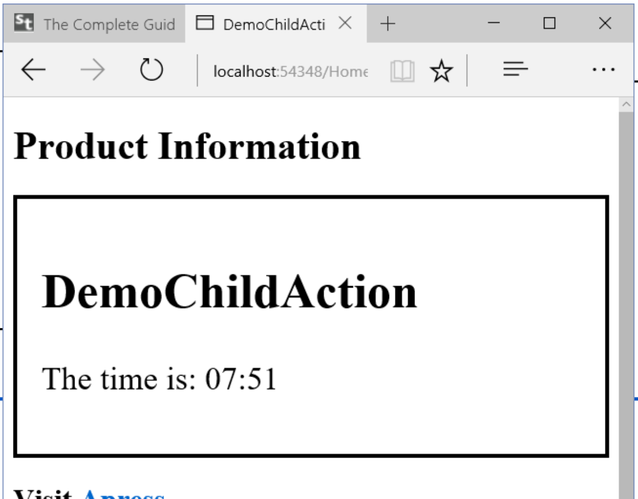
Child Actions

- Are action methods invoked from within a view
- You can use a child action whenever you want to display some data-driven widget that appears on multiple pages and contains data unrelated to the main action

```
@*DemoChildAction.cshtml*@
@{
    ViewBag.Title = "DemoChildAction"
    Layout = "~/Views/_BasicLayout.cs
}

<h2>DemoChildAction</h2>
<p>@Html.Action("Time")</p>
```

```
public class HomeController : Controller
{
    [ChildActionOnly]
    public ActionResult Time()
    {
        return PartialView(DateTime.Now);
    }
}
```

```
@*Time.cshtml*@

@model DateTime

<p>The time is: @Model.ToShortTimeString()</p>
```



Product Information

DemoChildAction

The time is: 07:51

Visit Apress

References & Links

- Pro ASP.Net MVC 5 chapter 5 and 20
- http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-(c)
- http://www.w3schools.com/aspnet/razor_intro.asp