ASP.NET CORE MVC

Module 3

ASP.NET Core

- ASP.NET Core is the new version of the ASP.NET Web Framework mainly targeted to run on .NET Core Platform.
- There are three general approaches to building modern web UI with ASP.NET Core:
  - Apps that render UI from the server.(MVC and Razor Pages)
  - Apps that render UI on the client in the browser.

What is Asp.Net MVC?

- Framework for building web applications
- Based on Model-View-Controller pattern
  - Model manages the applications data and enforces constraints on that model.
    - Often accessed through persistent objects
  - Views: Views are the components that display the app's user interface (UI). Generally, this UI displays the model data
  - Controllers: Classes that:
    - Handle browser requests.
    - Retrieve model data.
    - Call view templates that return a response.

MVC Life Cycle

- Clients request a named action on a specified controller, e.g.:
  - http://localhost/aController/anAction
- The request is routed to aController's anAction method.
  - That method decides how to handle the request, perhaps by accessing a model's state and returning some information in a view.

What is a Model?

- A model is a file of C# code and an associated data store, e.g., an SQL database or XML file.
    - The file of C# code manages all access to the application’s data through objects.
    - Linq to SQL and Linq to XML create queries into these data stores
        - This can be direct
        - More often it is done through objects that wrap db tables or XML files and have one public property for each attribute column of the table.

What is a View?

- Views are usually aspx files with only HTML and inline code, e.g., <% … C# code here … %>.
  - Code is used just to support presentation and does no application processing.
  - The HTML is augmented by HTML Helpers, provided by Asp.Net MVC that provide shortcuts for commonly used HTML constructs.
  - Asp.Net MVC comes with jQuery (Javascript) libraries to support reacting to client actions and doing AJAX communication with the server.

What is a Controller?

- A controller is a C# file with controller classes that derive from the class Controller.
  - A controller defines some category of processing for the application.
  - Its methods define the processing details.
  - Routing to a controller is defined in the Global.Asax.cs file. Its default processing is usually what you need.

What is a Controller?

- https://localhost:5001/Home/Privacy: specifies the Home controller and the Privacy action.
- https://localhost:5001/Movies/Edit/5: is a request to edit the movie with ID=5 using the Movies controller and the Edit action, which are detailed later in the tutorial

Web Application Development

- Create a new Asp.Net MVC project
  - Delete any part of that you don’t need
- Add a controller for each category of processing in your application:
  - A category is usually a few pages and db tables that focus on some particular application area
- Add methods to each controller for each request you wish to handle.
- Add views as needed for each controller action
- Add Model classes to support the application area:
  - Each model class has public properties that are synchronized with data in the model db or XML file.

Example-Controller

- Look at code
- Every public method in a controller is callable as an HTTP endpoint.
- In the sample above, both methods return a string.
- The first comment states this is an HTTP GET method that's invoked by appending /HelloWorld/ to the base URL.
- The second comment specifies an HTTP GET method that's invoked by appending /HelloWorld/Welcome/ to the URL

Example-Controller

- /[Controller]/[ActionName]/[Parameters]
- The routing format is set in the Program.cs file.

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Views

- Change index to

```
public ViewResult Index()
  {
      return View();
  }
```

- To create a view
  - Right-click on the Views folder, and then Add > New Folder and name the folder HelloWorld.
  - Right-click on the Views/HelloWorld folder, and then Add > New Item.
  - In the Add New Item In the search box in the upper-right, enter view

  - Select Razor View - Empty
  - Keep the Name box value, Index.cshtml.
  - Select Add

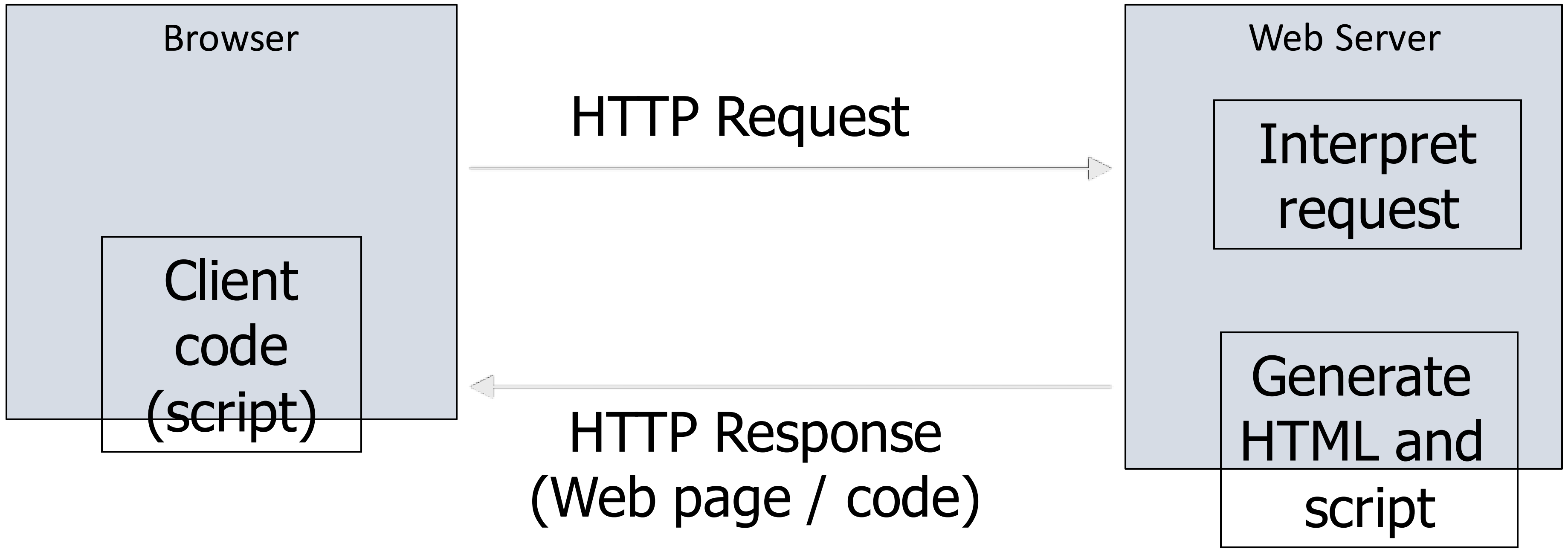
Layout File

- Open the Views/Shared/_Layout.cshtml file.
  - Layout templates allow:
  - Specifying the HTML container layout of a site in one place.
  - Applying the HTML pagecontainer layout across multiple s in the site.

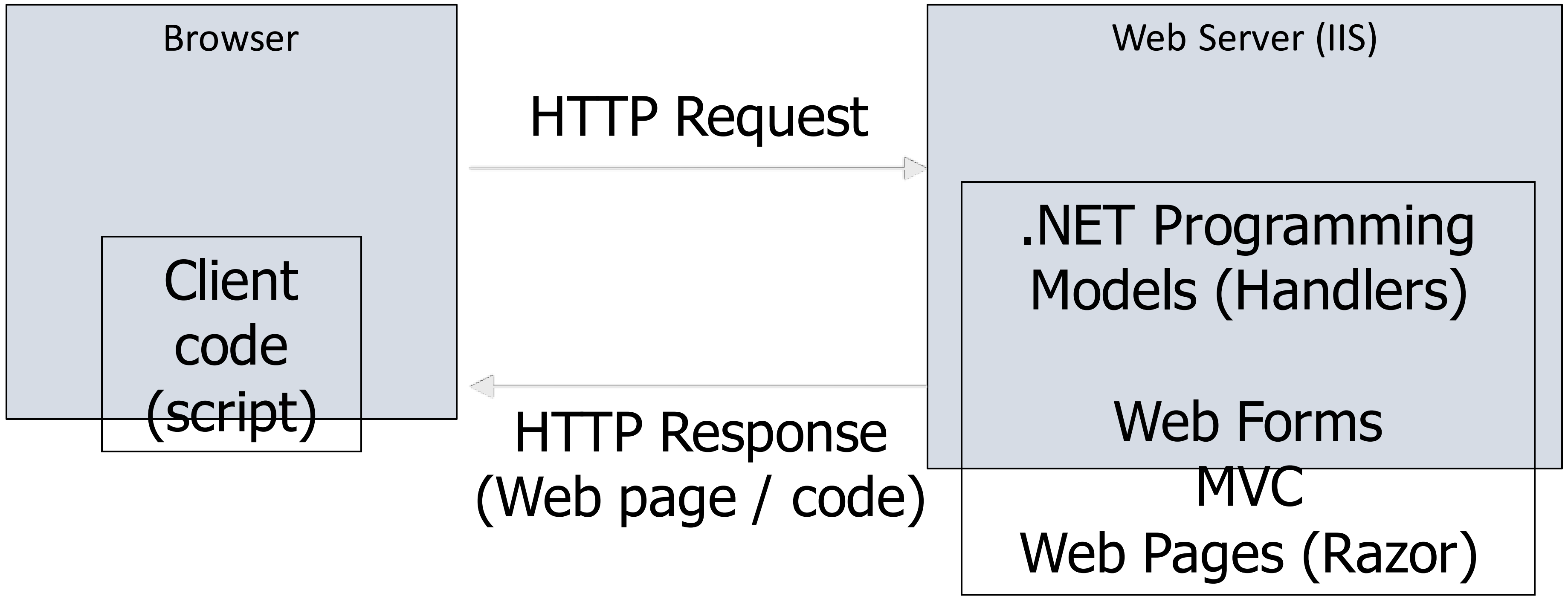
- Action Views are going to be created inside the **folder whose name is the same as the Controller name.**
- We want to create a view for the Index action method of Home Controller.

ASP MVP Web applications and Razor

Remember this?



A Revised Model



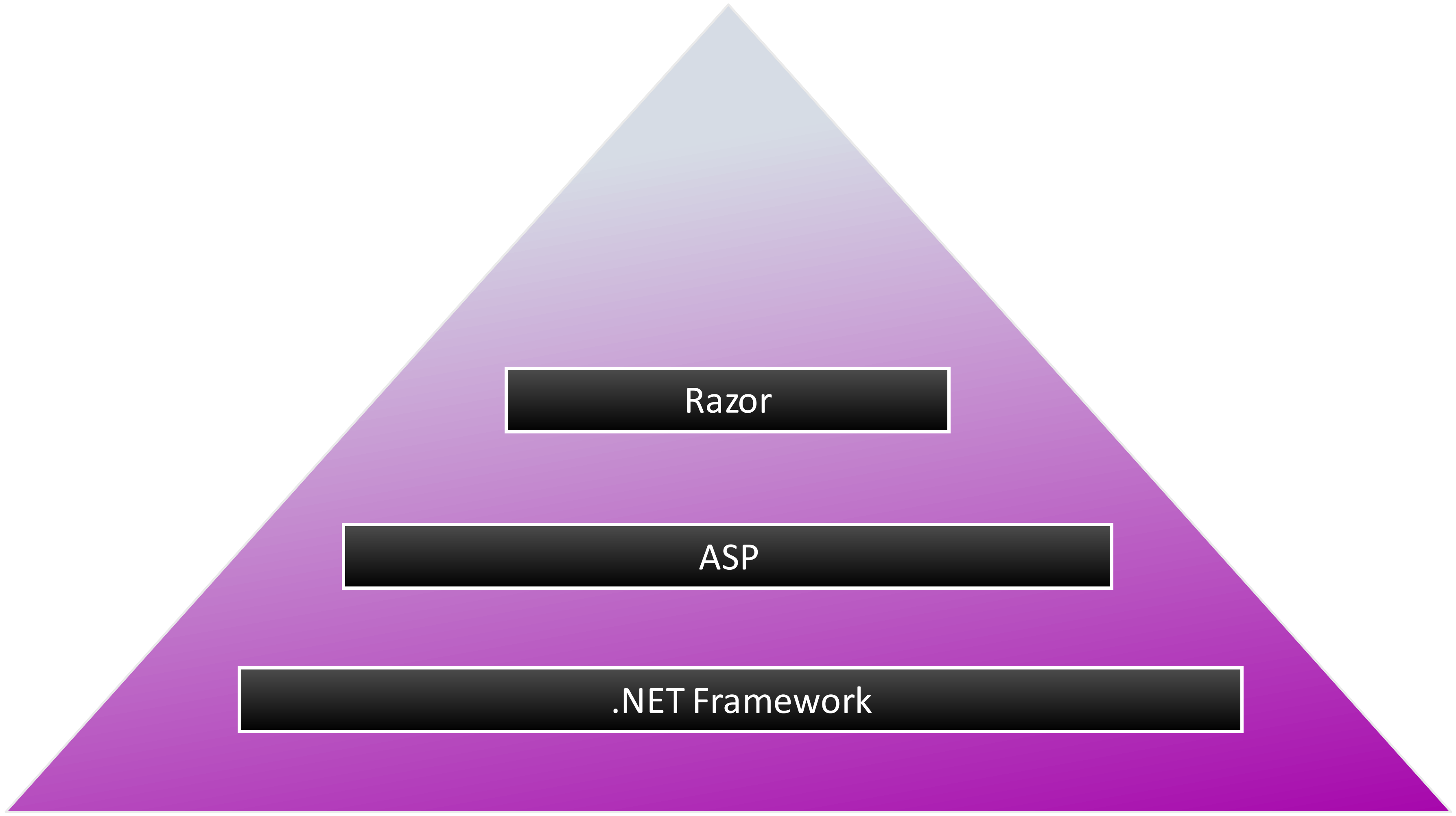
The .NET Models (Pillars)

- Web Forms (These are the traditional ASP.NET applications we have been creating)
- Web Pages (Razor)
  - Looks much like PHP
- Model View Controller (MVC)
  - A framework-based environment
  - It uses Razor as the rendering engine
  - Same MVC discussed last time

Comparing MVC / Razor and Traditional Web Forms

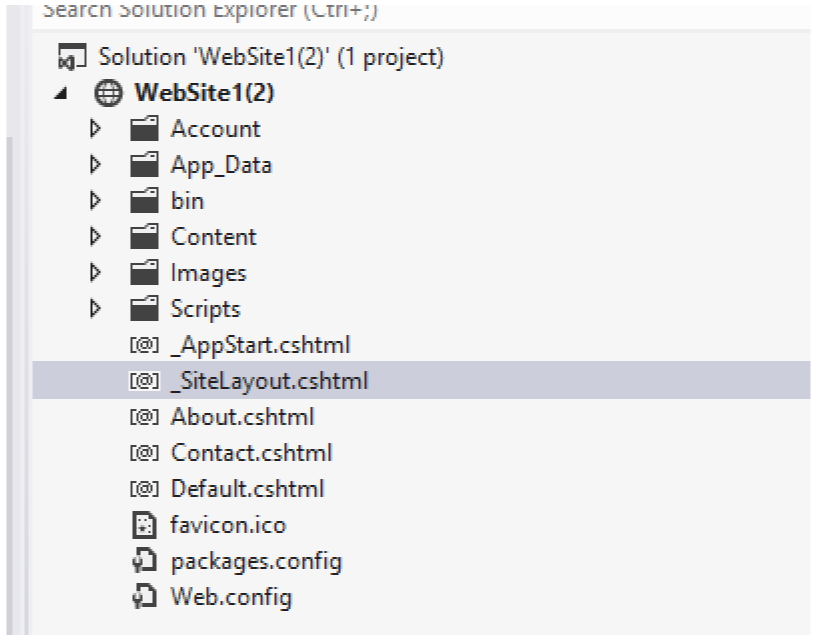
- Traditional Web forms are rich with 100s of server-side events
  - Without these, MVC applications are more work to develop
- Razor and MVC are lightweight
- MVC does not use view state or session state
  - It’s up to you to handle this task
- It’s designed to be RESTful

The Razor Model



MVC Razor Project Structure

- MVC and Razor projects are expected to have a well-defined structure
  - Adhere to it

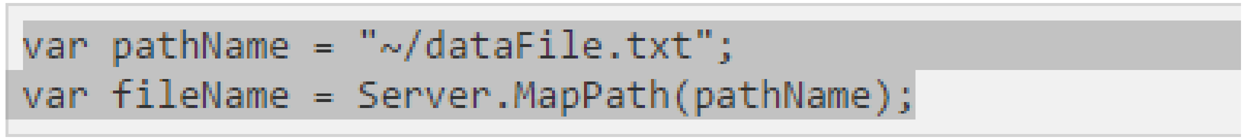


Search Solution Explorer (Ctrl+;)

```
Solution 'WebSite1(2)' (1 project)
◢ WebSite1(2)
    ▷ Account
    ▷ App_Data
    ▷ bin
    ▷ Content
    ▷ Images
    ▷ Scripts
      _AppStart.cshtml
      _SiteLayout.cshtml
      About.cshtml
      Contact.cshtml
      Default.cshtml
      favicon.ico
      packages.config
      Web.config
```

MVC Razor Page References (1)

- Use relative references
- To specify the virtual root in programming code, use the ~ operator (same as what you are used to)
- Use **`Server.MapPath`** to make absolute references

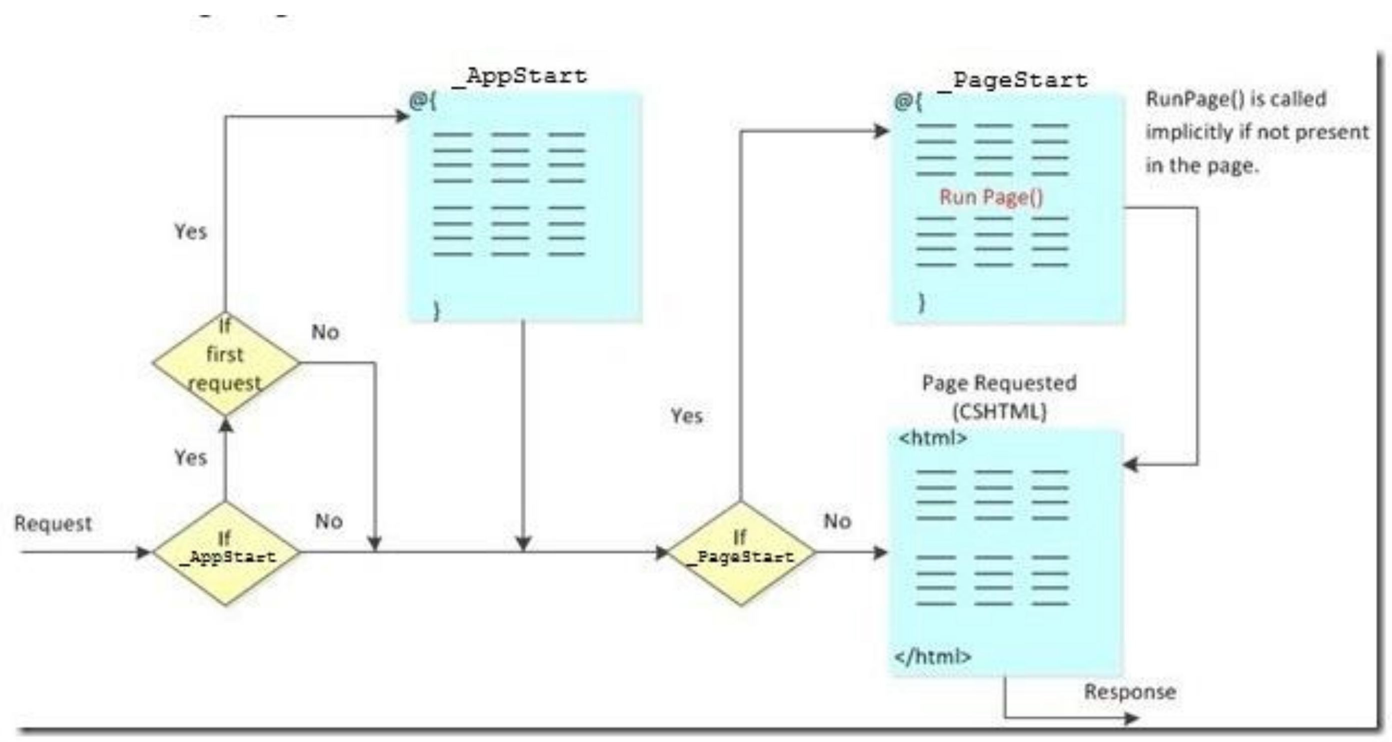


```
var pathName = "~/dataFile.txt";
var fileName = Server.MapPath(pathName);
```

Razor Application Flow

- _AppStart runs first and executes startup code
  - The underscore character keeps files from being browsed directly
  - Only runs on the first site request
- _PageStart runs before every page in a particular folder (again this environment is folder-based)

Razor Application Flow



Razor (Introduction)

- It’s the newer view engine for ASP
- It’s code looks much like PHP
    - Code is embedded into an HTML document
        - These are .cshtml files in Razor
- It relies on the .NET framework, as you would expect
- It works with databases
- We still have web.config

Razor (Introduction)

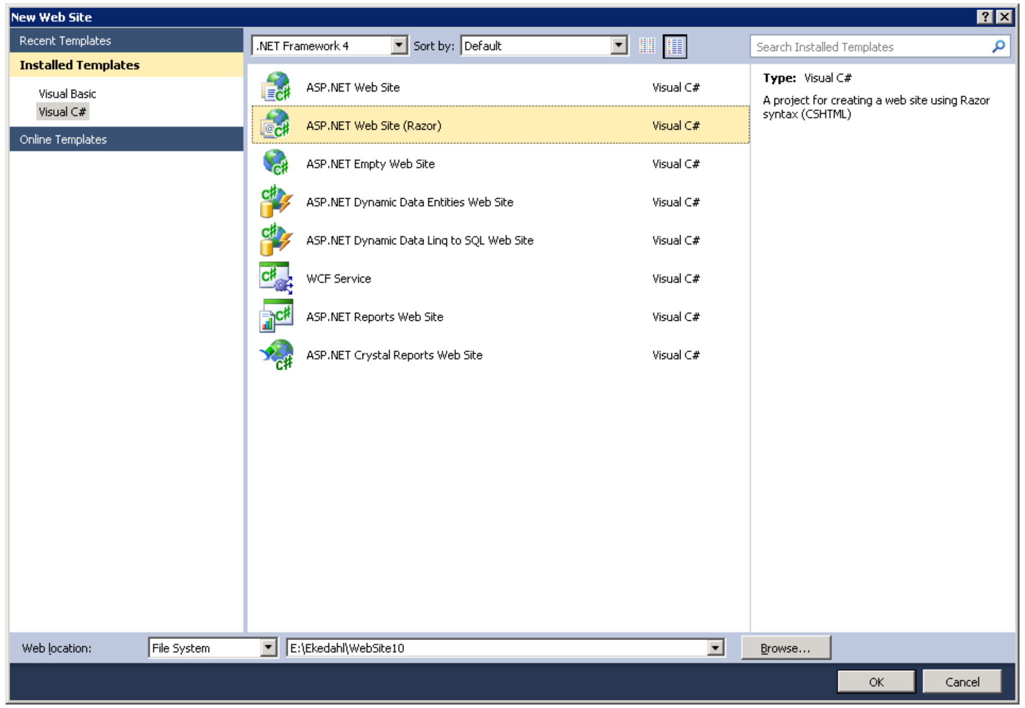
- ASP.NET control’s don’t work. So you can only use the intrinsic HTML controls
- Razor is HTML 5 compliant
- You can also use jQuery and JavaScript
- It has both C# and VB versions
  - C# is case sensitive as always
- There is no code-behind file

Razor (Characteristics)

- The `<%  %>` syntax is replaced with a leading `@`
- Code blocks appear in `@{ }` blocks in C#
  - VB has a similar construct
- There are a bunch of new objects

Creating a Razor Project

- It’s a project with a template like anything else
- Just create a new Web project



New Web Site

.NET Framework 4 | Sort by: Default | Search Installed Templates

ASP.NET Web Site	Visual C#
ASP.NET Web Site (Razor)	Visual C#
ASP.NET Empty Web Site	Visual C#
ASP.NET Dynamic Data Entities Web Site	Visual C#
ASP.NET Dynamic Data Linq to SQL Web Site	Visual C#
WCF Service	Visual C#
ASP.NET Reports Web Site	Visual C#
ASP.NET Crystal Reports Web Site	Visual C#

Type: Visual C#

A project for creating a web site using Razor syntax (CSHTML)

Web location: File System | E:\Ekedahl\WebSite10

Types of Razor Code Blocks

- Inline
  - The `@` character calls a function or evaluates a character within some HTML code
    - It says the code is Razor code, rather than HTML
- Single Statement
  - Statement appears in the `@{}` block as in
    `@{ var x = 10;}`
- Multi Statement
  - Use the single statement syntax with multiple statements (separated by a semi-colon);

Inline (Example)

- Code appears embedded inside some HTML
- Call the **`Today`** method
  - Note that the .NET classes are the same as always

```
@System.DateTime.Today.ToString()
```

Single Statement Example

- As the name implies, it’s a single statement that appears in a `@{}` block

```
@{ var x = 10;}
```

Multi-statement Example

- Multiple statements appear in a @{} block
- A semi-colon separates each statement

```
@{
    Layout = "~/_SiteLayout.cshtml";
    Page.Title = "Welcome to my Site!";
}
```

Layouts (Introduction)

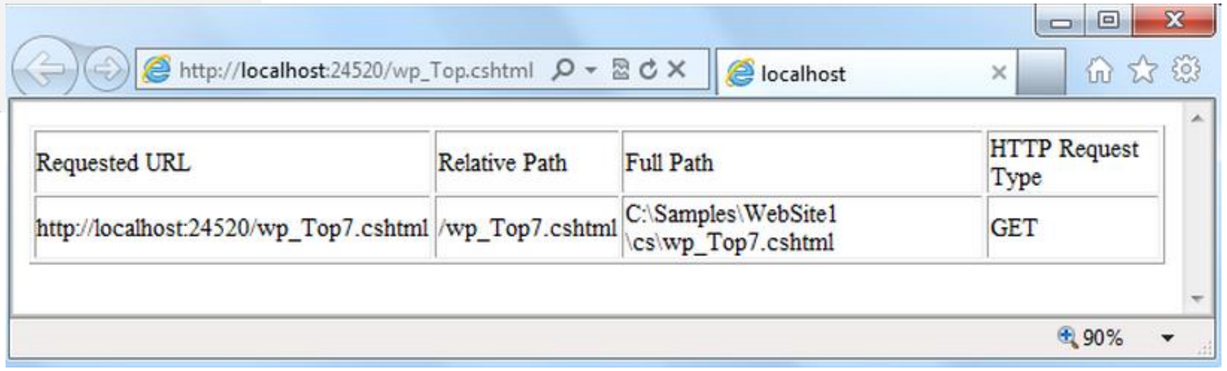
- We have been using Master pages to give consistent look-and-feel to pages
- Layouts are the Razor method of doing this

- Create a layout page
- From the other pages, we reference the Layout page with the **`@Layout`** method

Razor Syntax (2)

- There are many redefined objects Request, Response… just as in asp.net

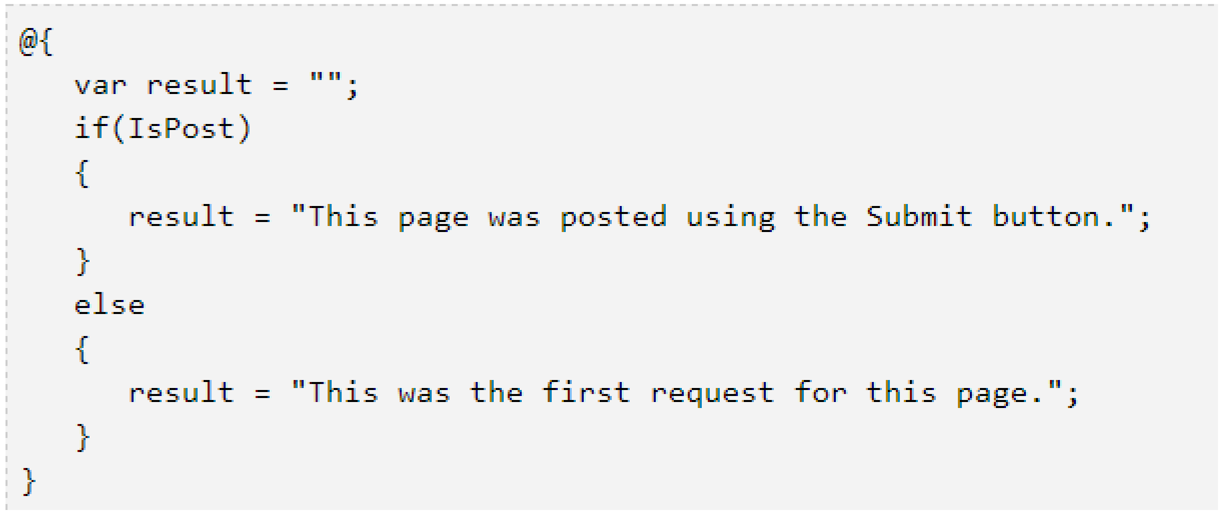
```
<table border="1">
<tr>
    <td>Requested URL</td>
    <td>Relative Path</td>
    <td>Full Path</td>
    <td>HTTP Request Type</td>
</tr>
<tr>
    <td>@Request.Url</td>
    <td>@Request.FilePath</td>
    <td>@Request.MapPath(Request.FilePath)</td>
    <td>@Request.RequestType</td>
</tr>
</table>
```



Requested URL	Relative Path	Full Path	HTTP Request Type
http://localhost:24520/wp_Top7.cshtml	/wp_Top7.cshtml	C:\Samples\WebSite1 \cs\wp_Top7.cshtml	GET

Razor Syntax (3)

- Decision making and loops are supported



```
@{
   var result = "";
   if(IsPost)
   {
      result = "This page was posted using the Submit button.";
   }
   else
   {
      result = "This was the first request for this page.";
   }
}
```

User Input (Forms)

- Unlike Traditional Web forms, we use traditional HTML input controls
- We reference those through the **`Request`** object
- Example (Get the contents if the input control named text1
  - `var num1 = Request["text1"];`

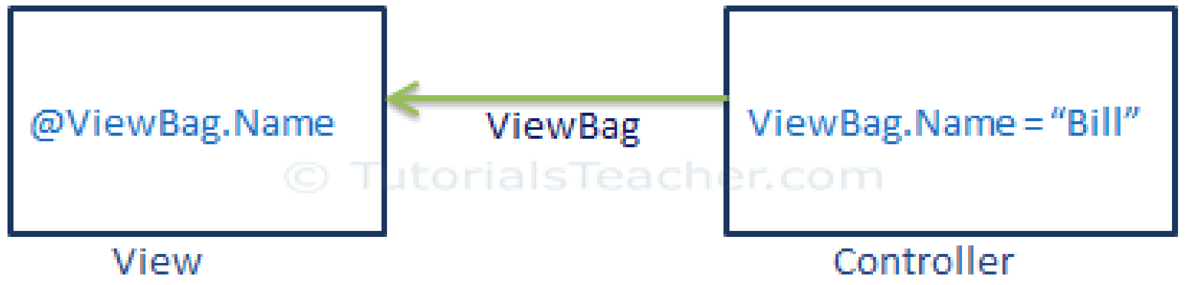
http://www.w3schools.com/aspnet/webpages_forms.asp

Helpers

- Helpers are conceptually similar to ASP.NET controls. They simplify the process of doing something
- http://www.w3schools.com/aspnet/webpages_helpers.asp

The **`ViewBag`**

- The model is used to send data to a Razor view
- Temporary data (not included in the model) can also be transferred from the controller to the view
  - Data is not transferred from the view to the controller



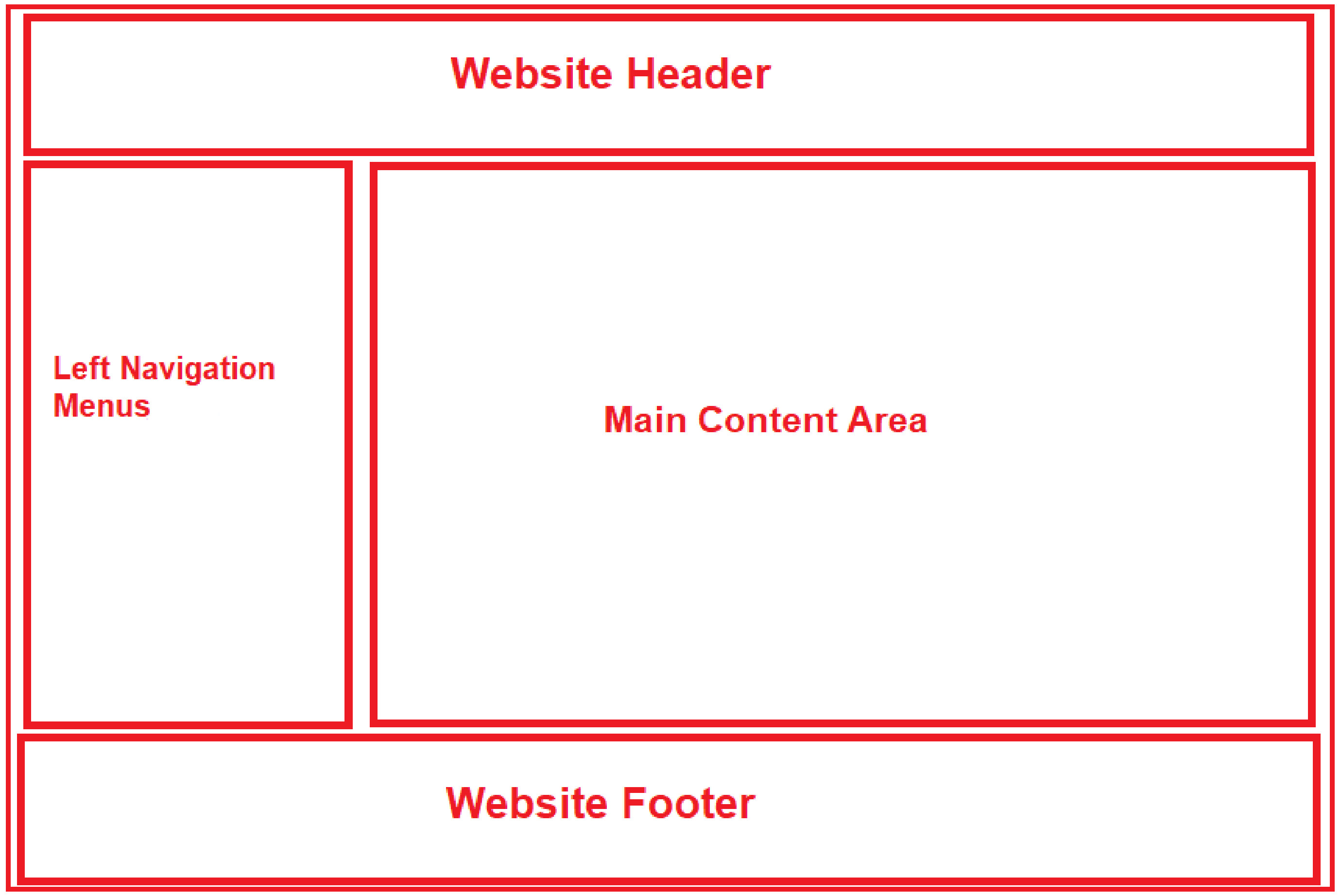
Other Razor Capabilities

- We can work with text files
- We can work with database

Type Conversion

- Like PHP and JavaScript, Razor is loosely typed
  - Call the “As” functions to convert types
    - AsInt, AsFloat, As*xxx*…

Layout


Website Header
Left Navigation Menus
Main Content Area
Website Footer

Layout View

- Instead of putting all the sections (i.e. the HTML) in each and every view pages, it is always better and advisable to put them in a layout view and then inherit that layout view in each and every view
- As the layout views are not specific to any controller, so, we usually place the layout views in a subfolder called “**Shared**” within the “**Views**” folder.

Creating

1. Right-click on the “**Views**” folder and then add a new folder with the name “**Shared**“.
2. Next, Right-click on the “**Shared**” folder and then select the “**Add**” – “**New Item**” option from the context menu which will open the Add New Item window.
3. From the “**Add New Item**” window search for **Layout** and then select “**Razor Layou**t”, give a meaning full name (_Layout.cshtml) to your layout view and finally click on the “**Add**” button as shown below which should add **_Layout.cshtml** file within the Shared folder.

Adding Layout

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

State Management

- HTTP/HTTPs doesn't remember what website or URL we visited, or in other words we can say it doesn't hold the state of a previous website that we visited before closing our browser, that is called stateless.
- In ASP.NET there are the following 2 State Management methodologies:
  - Client-Side State Management:Whenever we use Client-Side State Management, the state related information will directly get stored on the client-side.
  - In Server-Side State Management all the information is stored in the server memory

Client Side

- Client-Side State Management techniques are,
  - Cookies

Cookies

- Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser. It does not use server memory. Generally a cookie is used to identify users.
  - **Persistence Cookie**: Cookies which you can set an expiry date time are called persistence cookies. Persistence cookies are permanently stored till the time you set.
  - Response.Cookies.Append("Username", "Student1");
  - CookieOptions option = new CookieOptions();
  - option.Expires = DateTime.Now.AddMinutes(10); //for expiry date

- **Non-Persistence Cookie**: Non persistence cookies are not permanently stored on the user client hard disk folder.
- `Response.Cookies.Append("Username", "Student1"); //No expiry date`

Server Side

Session

- Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser).
  - HttpContext.Session.SetString("Name", "John");
- In Program.cs file you will also need to add
  - builder.Services.AddSession();
  - app.UseSession();

Validation

- Validation is an important aspect in ASP.NET MVC applications.
- It is used to check whether the user input is valid. ASP.NET MVC provides a set of validation that is easy-to-use and at the same time, it is also a powerful way to check for errors and, if necessary, display messages to the user.

Attribute	Usage
Required	Specifies that a property value is required.
StringLength	Specifies the minimum and maximum length of characters that are allowed in a string type property.
Range	Specifies the numeric range constraints for the value of a property.
RegularExpression	Specifies that a property value must match the specified regular expression.
CreditCard	Specifies that a property value is a credit card number.
CustomValidation	Specifies a custom validation method that is used to validate a property.
EmailAddress	Validates an email address.
FileExtension	Validates file name extensions.
MaxLength	Specifies the maximum length of array or string data allowed in a property.
MinLength	Specifies the minimum length of array or string data allowed in a property.
Phone	Specifies that a property value is a well-formed phone number.