ASP.NET MVC 5

MD. MAHEDEE HASAN

MAHEDEE.NET

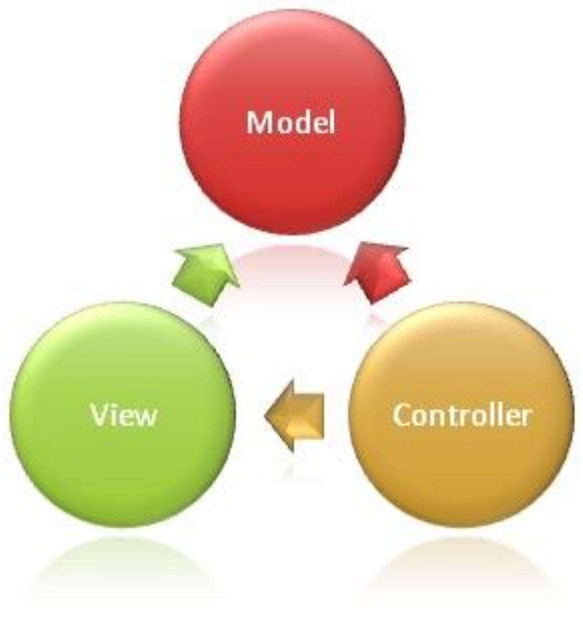
ASP.NET MVC 5

Md. Mahedee Hasan
Software Architect
Leadsoft Bangladesh Limited
Blog: ***http://mahedee.net/***
http:mahedee.blogspot.com
Email: ***mahedee.hasan@gmail.com***
Linkedin: ***http://www.linkedin.com/in/mahedee***
Facebook: ***http://www.facebook.com/mahedee19***

Chapter 1: Introduction to ASP.NET MVC

What is MVC?

- MVC Stands for **Model – View – Controller**
- It is **Software Architectural pattern**
    - An architectural pattern is a **general**, **reusable solution** of a **commonly occurring problem** in **software architecture** within a given context.
    - Architectural patterns are **similar to software design patterns** but have a **broader scope**.
- It **divides an application's implementation** into three components
    - Models, views, and controllers.
- The **Model** represents the application core (for instance a list of **database records**).
- The **View displays** the **data** (the database records).
- The **Controller** handles the input.



Routing Example

- If you request a page, first **controller handles the request** and **orders an action** which will serve the request.
- **Action** can **use model** to get database record.
- Then **action displays** results to the **view pages.**

MVC Logic Layers

- MVC provides **full control** over **HTML**, **CSS** and **JavaScript**.
- You can think MVC model defines web applications with **3 logic layers:**
    - The **business layer (Model logic)**
    - The **display layer (View logic)**
    - The **input control (Controller logic)**

- **The Model** is the part of the application that **handles** the **logic for the application data.**
    - **Often model objects retrieve data (and store data) from a database.**

- **The View** is the **parts of the application** that **displays data.**
    - **Most often** the **views are created** from the **model data**.

- The **Controller** is the part of the application that **handles user interaction**.
    - Typically **controllers read data from a view**, **control user input**, and send input data to the model.

- The **MVC separation** helps you manage **complex applications**, because you can **focus on one aspect** a time.
    - For example, you can focus on the view without depending on the business logic.
    - It also makes **easier to test** an application.

- **The MVC separation also simplifies group development**.
    - Different **developers** can work on the view, the controller logic, and the business logic in **parallel**.

MVC Request Response process

ASP.NET MVC request response process step by step

1. User sends the **request by the URL.**
2. **UrlRoutingModule** intercepts/cut offs the request and starts **parsing** it.
3. The **appropriate Controller** and handler **will be identified from the URL** by looking at the **RouteTable** collection.
    - Also any data coming along with the request is kept in **RouteData**.
4. The **appropriate** action in the identified controller will be executed.
5. This action will **call the Model class based on data.**
6. **The action will then pass** to some view and tell the view to proceed.
7. Now the **view** will execute and **create the markup based** on the logic and Model's data and then push **the HTML back to the browser**.

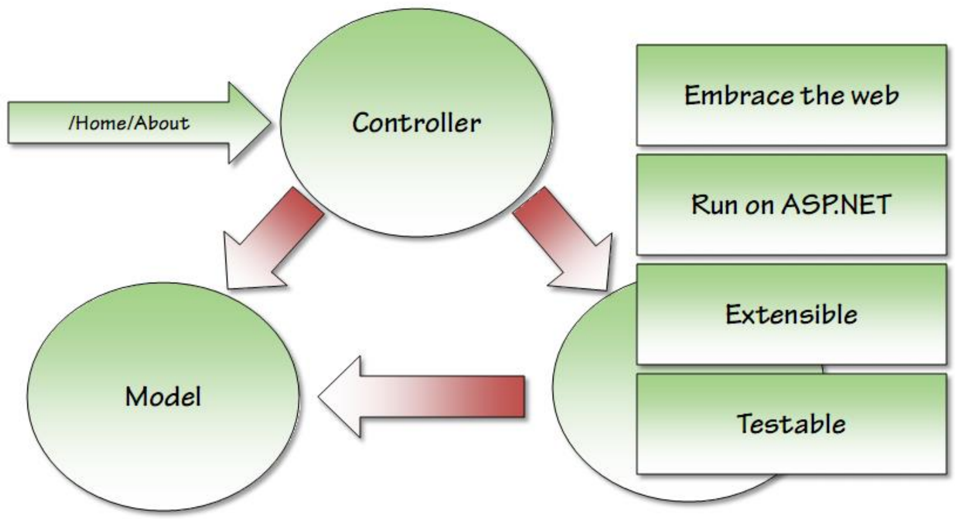
This life cycle above is defined for explanation and has omitted some technical details.

What is View Engine?

- **View Engines are responsible for rendering the HTML** from **views to the browser**.
- The view engine template will have different syntax for implementation.
- Currently there are few numbers of view engines available for ASP.NET MVC and the top four view engines are **Razor, traditional ASPX, Spark and NHaml.**

Chapter 2: Getting Started

ASP.NET MVC Design Goal

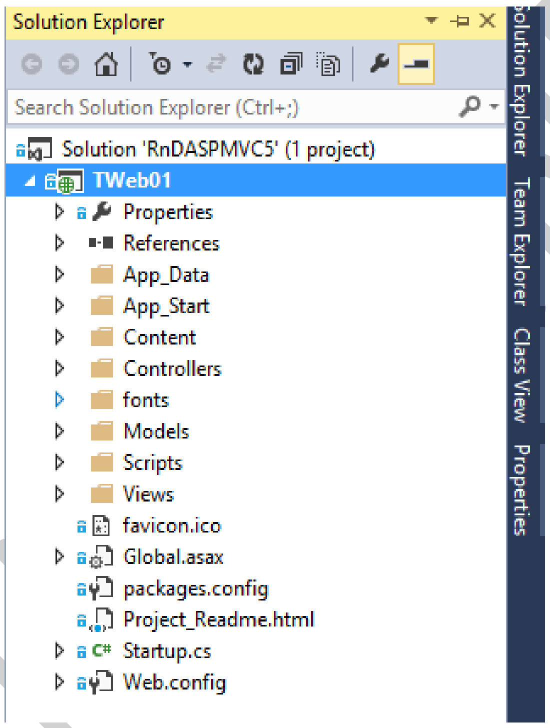


- **Separation of Concern**
  - *"The process of **breaking** a computer program into **distinct features** that **overlap** in **functionality** as **little as possible**." – Wikipedia*
  - Responsibility of Model, View and Controllers are separate

- **Embrace the web**
  - Able to **adopt standard web technology** HTML, **HTML5**, CSS, **CSS3**
- **Run on ASP.NET**
  - Run on top of **ASP.NET runtime**
- **Extensible**
  - Uses different **plugins** and change as you like
- **Testable**
  - MVC designed to allow **loosely couple.**
  - MVC designed using **solid object oriented concept.**
  - Any solution or code designed **in this fashion is testable**.

Demo: How to create first Application using ASP.NET MVC?

ASP.NET MVC Folder Structure



- The MVC framework is based on **default naming.**
  - It is called **convention over configuration**

- **Controllers** are in the **Controllers folder**, Views are in **the Views folder**, and Models are in the Models folder.

- A typical ASP.NET MVC web application has the following folder content

- Application information
  - Properties
    - Also called project properties
    - Consists **AssemblyInfo.cs** - It consists of all of the build options for the project, including version, company name, GUID, compilers options etc.

    - **References**
        - **Contains libraries**.

- **Application folders**
    - App_Data Folder
    - Content Folder
    - Controllers Folder
    - Models Folder
    - Scripts Folder
    - Views Folder

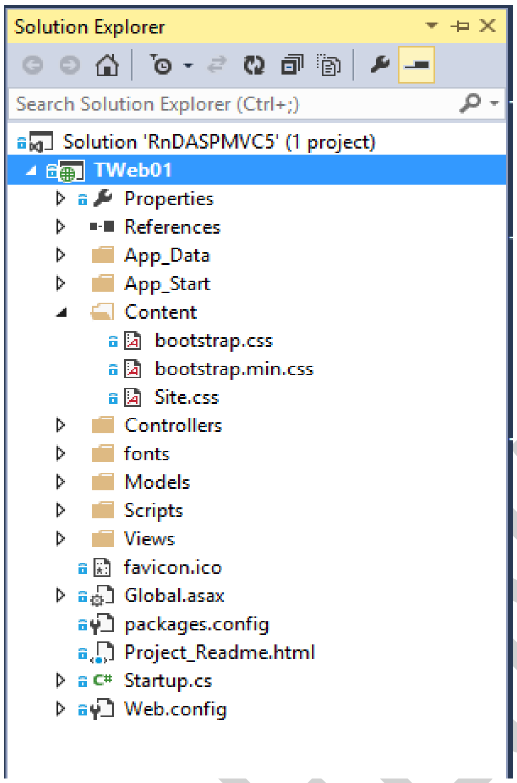
- **Configuration files**
    - Global.asax
    - packages.config
    - Web.config

The App_Data Folder

- The **App_Data** folder is for storing application data.
- We can add an **SQL database** to the App_Data folder
    - Can add SQL(Sequel) Server Compact Edition

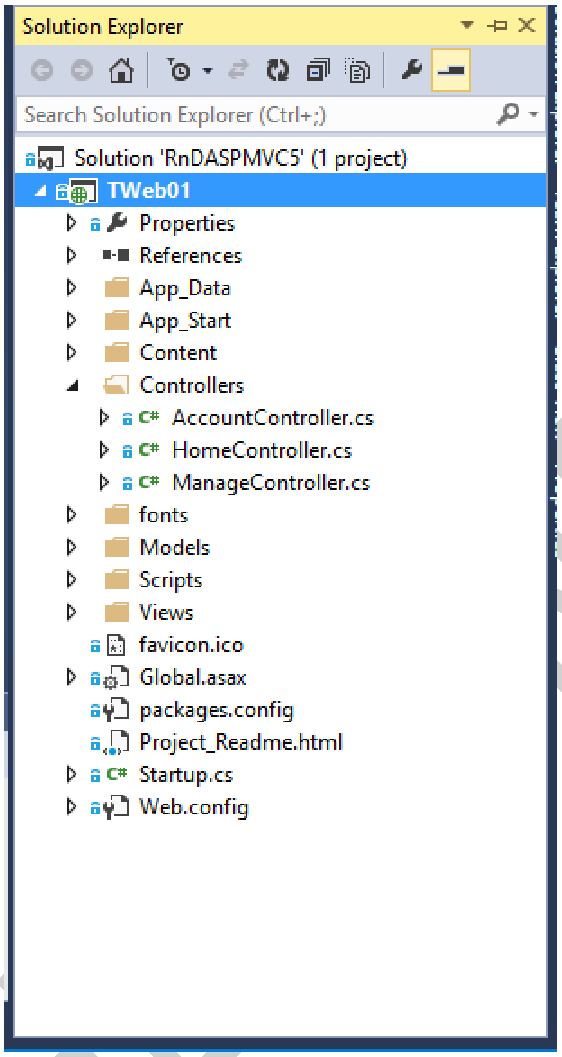
The Content Folder

- The **Content** folder is used for **static files like style sheets (css files), icons and images.**
- Visual Web Developer automatically adds some css files



The Controllers Folder

- The Controllers folder contains the **controller classes.**
- Controller responsible for **handling user input and responses.**
- MVC requires the name of all controller files to **end with "Controller".**

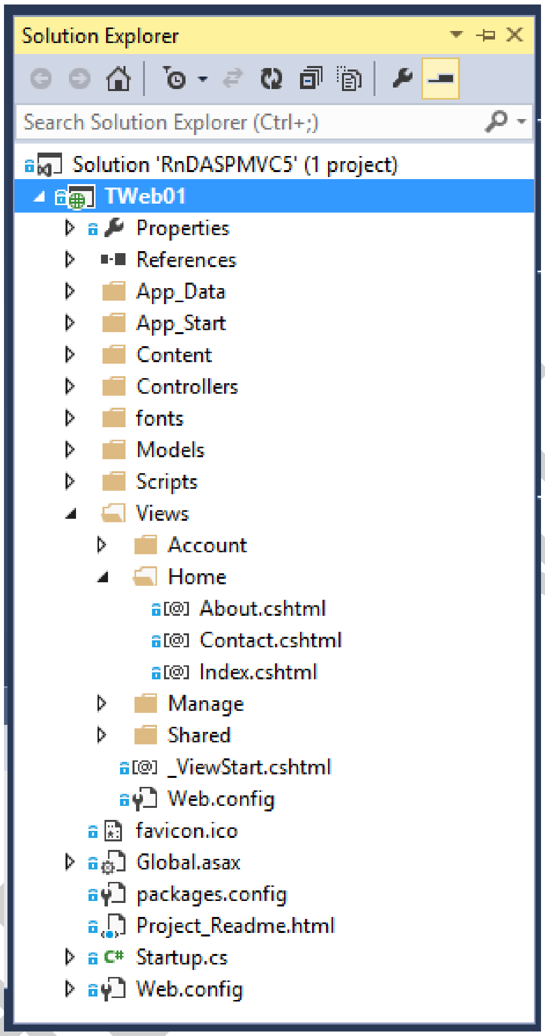


The Models Folder

- The **Models folder contains** the classes that represent the **application models.**
- Models hold and manipulate **application data**

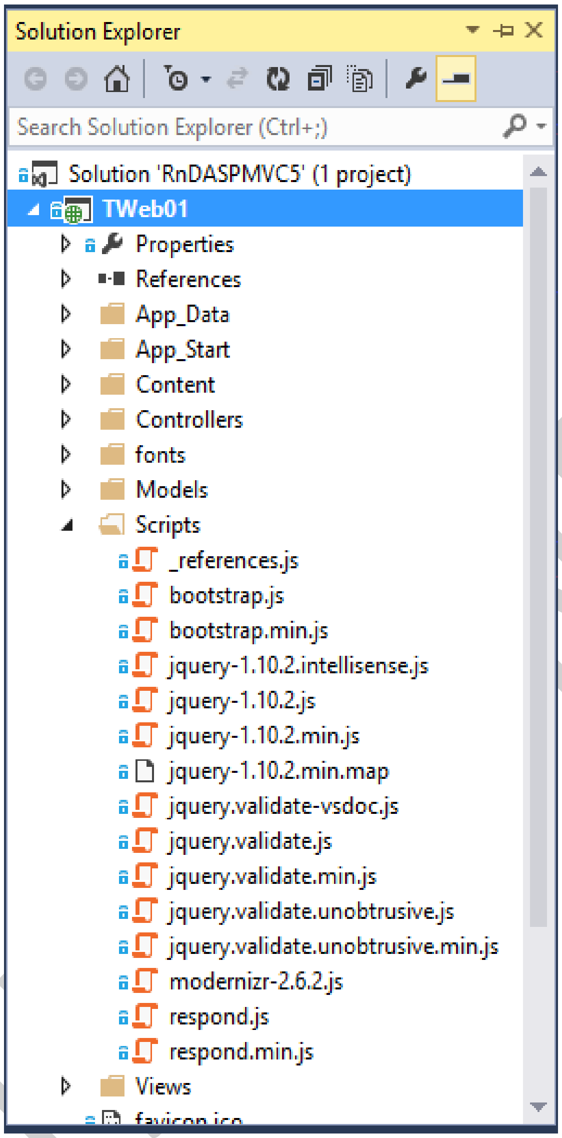
The Views Folder

- The Views folder contains **HTML files** related to the user interface called view.
- The Views folder contains **one folder for each controller**.
- Visual Web Developer has created an Account folder, a Home folder, and a Shared folder (inside the Views folder).
- The Account folder contains pages for **registering** and **logging** in to user accounts.
- The Home folder is used for storing application pages like the **home page and the about page**.
- The **Shared folder is used to store views shared between controllers** (**layout pages**)



The Scripts Folder

- The Scripts folder stores the **JavaScript files** of the application.
- By default Visual Web Developer **fills** this folder with standard **MVC, Ajax, and jQuery files**
- The files named "**modernizr**" are JavaScript files used for supporting **HTML5 and CSS3 features in the application.**


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'RnDASPMVC5' (1 project)
  TWeb01
    Properties
    References
    App_Data
    App_Start
    Content
    Controllers
    fonts
    Models
    Scripts
      _references.js
      bootstrap.js
      bootstrap.min.js
      jquery-1.10.2.intellisense.js
      jquery-1.10.2.js
      jquery-1.10.2.min.js
      jquery-1.10.2.min.map
      jquery.validate-vsdoc.js
      jquery.validate.js
      jquery.validate.min.js
      jquery.validate.unobtrusive.js
      jquery.validate.unobtrusive.min.js
      modernizr-2.6.2.js
      respond.js
      respond.min.js
    Views
    favicon.ico
```

Layout

- The file **_Layout.cshtml** represents the layout of each page in the application.
- It is located in the Shared folder inside the Views folder.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title - MAHEDEE.NET</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>
            </div>
            <div class="float-right">
                <section id="login">
                    @Html.Partial("_LoginPartial")
                </section>
                <nav>
                    <ul id="menu">
                        <li>@Html.ActionLink("Home", "Index", "Home")</li>
                        <li>@Html.ActionLink("About", "About", "Home")</li>
                        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                    </ul>
                </nav>
            </div>
        </div>
    </header>
    <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>

    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)
</body>
</html>
```

- **Url.Content()**: Content() method is a method of UrlHelper class. It **converts a virtual (relative) path to an application absolute path.**

- **Html.ActionLink()**: The easiest way to render an **HTML link** in is to use the HTML.ActionLink()

- **RenderSection()**: RenderSection() is a **method** of the WebPageBase class. The first parameter to the "RenderSection()" helper method specifies the **name of the section** we want to render.

- **RenderBody()**:
    - In layout pages, **renders the portion of a content page** that is not within a named section.
    - It returns the HTML content to render.
    - RenderBody is required, since it renders each view.

Chapter 3: Views

- The **View displays** the **data** or database records
- Provide user **interface (UI)** for the user
- View stored in **Views folder**
- There is a folder in Views for **each controller**
- **Typically,** each view **maps corresponding actions**

Uses of ViewBag

- The **dynamic** view **data dictionary**.
- **Dynamic Type object**
- ASP.NET MVC offers us ViewBag for **passing data from controller to view**

```
public object ViewBag { get; }
```

Example:

- Create a Model name – AboutModel

```
public class AboutModel
{
    public string Name { get; set; }
    public string Location { get; set; }
}
```

- Edit **HomeController** and update the **About** action as follows
    - In this action, you will see, ViewBag.Message is used to hold a string
    - AboutModel object name aboutModel with its two property which will also use in the view.

```
public ActionResult About()
{
    ViewBag.Message = "Company Information";

    var aboutModel = new AboutModel();
    aboutModel.Name = "Leadsoft Bangladesh Limited";
    aboutModel.Location = "41/6, Purana Platan, Sawdesh Tower, Dhaka - 1000";

    return View(aboutModel);
```

```
        }
```

- Edit About view (About.cshtml) as follows
    - Add @model TMS.Models.AboutModel at the top of the view.
    - To view the ViewBag data use like @ViewBag.Message .

```
@model TMS.Models.AboutModel

@{
    ViewBag.Title = "About";
}

<hgroup class="title">
    <h1>@ViewBag.Title.</h1>

</hgroup>

<div>
    <h2>@ViewBag.Message</h2>
    <h2>@Model.Name</h2>
   Location : @Model.Location
</div>
```

What is ViewData, ViewBag and TempData?

- ASP.NET MVC offers us three options **ViewData, ViewBag and TempData** for **passing data** from controller to view.
- **ViewData and ViewBag** are almost **similar** and **TempData performs additional** responsibility.

Similarities between ViewBag & ViewData

- Helps to **maintain data** when you move from **controller to view.**
- Used to **pass data** from controller to corresponding view.
- Short life means value becomes null when **redirection occurs.**
- This is because their goal is to provide a way to communicate between controllers and views.
- It's a communication mechanism **within the server** call.

Difference between ViewBag & ViewData

- **ViewData is a dictionary of objects** that is derived from **ViewDataDictionary** class and accessible using **strings as keys**.
- **ViewBag** is a **dynamic property** that takes advantage of the new dynamic features in C# 4.0.
- **ViewData** requires **typecasting** for complex data type and check for null values to avoid error.
- ViewBag doesn't require typecasting for complex data type.

<u>ViewBag & ViewData Example:</u>

```
        public ActionResult Index()
        {
            ViewBag.Name = "Mahedee Hasan";
            return View();

        }

        public ActionResult Index()
        {
            ViewData["Name"] = "Mahedee Hasan";
            return View();

        }
```

In View:

```
@ViewBag.Name
@ViewData["Name"]
```

TempData

- TempData is also a **dictionary** derived from **TempDataDictionary** class
- Stored in **short lives session** and it is a string key and object value.
- The difference is that the life cycle of the object.
    - **TempData keep** the information for the time of **an HTTP Request**.
    - This mean only from one page to another.
- This also work with a 302/303 redirection because it's in the same HTTP Request.
- **Helps to maintain data when you move from one controller** to other controller or from one action to other action.
- In other words when you redirect, "Tempdata" helps to maintain data between those redirects. **It internally uses session variables.**
- Temp data use during the **current and subsequent request** only means it is use when you are sure that next request will be redirecting to next view.
- It **requires typecasting for complex data** type and check for null values to avoid error. generally used to store only one time messages like error messages, validation messages.

```
        public ActionResult Index()
        {
            var model = new Review()
                        {
                            Body = "Start",
                            Rating = 5
                        };
            TempData["ModelName"] = model;
            return RedirectToAction("About");
        }
        public ActionResult About()
        {
            var model = TempData["ModelName"];
            return View(model);
        }
```

Layout with Razor

- Use **inherited methods** to specify **content areas**
    - **RenderBody**
    - **RenderSection**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")

</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new { area = "" },
new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
                @Html.Partial("_LoginPartial")
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

Configure Layout

In _ViewStart.cshtml

- The _ViewStart.cshtml file will **execute at the start of each view's rendering**.
- Any code contained within the **code block in this file will execute before** any code in the view.
- Typically**, this file** will **set the layout template** to be used by the views in the application

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

- An MVC application **can have multiple _ViewStart.cshtml files.**
- The **order in which these files execute** is dependent upon the **location of the files in the folder hierarchy** and the **particular view being** rendered.
- The MVC Runtime will first execute the **code in the _ViewStart.cshtml** file located in the root of the **Views folder.**
- It will then **work its way up the folder hierarchy**, executing the code in each _ViewStart.cshtml file it finds along the way.
- For **example**, we **may have _ViewStart.cshtml** files in the **following locations**

  1) View_ViewStart.cshtml
  2) View\Home_ViewStart.cshtml
  3) View\Products_ViewStart.cshtml

In Pages

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

Chapter 4: Introduction to Razor

What is Razor?

- Razor is **not a programming language.**
- It's a **server side markup language**.
  - Markup language includes a set of tags
- It lets you **embed server-based code** (Visual Basic and **C#)** into web pages.
- Razor enables **mixing server-side code with HTML mark up** to generate an HTML response
- **Server-based code** can **create dynamic web content on the fly.**
- When a web page is called, the **server executes the server-based code inside the page** before it returns the page to the browser.

- By **running on the server**, the code can perform complex tasks, like **accessing databases**.
- **Razor is based on ASP.NET**, and **designed** for creating web **applications**.
- It has the **power of traditional ASP.NET** markup, but it is **easier to use, and easier to learn.**

Razor Syntax

- Razor uses syntax very similar to **PHP** and **Classic ASP.**

Razor:

```
<ul>
@for (int i = 0; i < 10; i++) {
<li>@i</li>
}
</ul>
```

PHP :

```
<ul>
<?php
for ($i = 0; $i < 10; $i++) {
echo("<li>$i</li>");
}
?>
</ul>
```

Web Forms (and Classic ASP):

```
<ul>
<% for (int i = 0; i < 10; i++) { %>
<li><% =i %></li>
<% } %>
</ul>
```

Razor's Programming Languages

- Razor supports both **C# (C sharp) and VB (Visual Basic).**

Main Razor Syntax Rules for C#

- **Razor code blocks** are enclosed in **@{ ... }**
- **Inline expressions** (variables and functions) start **with @**
- Code statements end with **semicolon**
- Variables are typically declared with the **var keyword**
- Strings are enclosed with quotation marks

- C# code is case sensitive
- Razor files have the **extension .cshtml or .vbhtml etc**
- **HTML Commenting**
    - `<!-- Single statement block -->`
- Razor commenting
```
@*@{
    This is a razor commenting
}*@
```

C# Examples

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- 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 + " Here in Dhaka it is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

How does it Work?

- Razor is a **simple programming syntax** for **embedding server code** in **web pages.**
- **Razor syntax** is **based on the ASP.NET framework**
    - **Part of the Microsoft.NET Framework** that's specifically designed **for creating web applications.**
- The Razor syntax gives you **all the power of ASP.NET,** but is using a **simplified syntax that's easier to learn**, and makes you **more productive.**
- **Razor web pages** is a **HTML pages** with **two kinds of content: HTML content and Razor code.**
- **When the server reads the page**, it runs the **Razor code first, finally sends the HTML page** to the browser.
- The **code that is executed on the server** can perform tasks that **cannot be done in the browser**.
    - Example: Accessing a server database.

Uses of @ Sign

- **To open a code block**

  Example:
```
@{
    ViewBag.Title = "Edit";
    Layout = "~/Views/Shared/_EditLayout.cshtml";
}
```

- **To denote an inline expression or statement**
  Example:

```
<ul>
    @foreach (var item in rows)
    {
        // do something
    }
</ul>
```

  - **Nested expressions or statements do not start with an @ sign**
    Example:

```
<ul>
    @foreach (var item in rows)
    {
        if (item.Equals(x))
        {
            // do something
        }
    }
</ul>
```

  - **Nested expressions are separated from the outer expression or statement by unmatched tags**
    Example:

```
<ul>
    @foreach (var item in rows)
    {
        <li>
            @if (item.Equals(x))
            {
                // do something
            }
        </li>
    }
</ul>
```

- To render the value of variables
  Example:

```
@DateTime.Now <!-- renders the current time to the browser -->

@(someCondition ? x : y)
<!-- renders the value of x or y to the browser ->

@Html.ActionLink("Back to List", "Index") <!-- renders a hyperlink -->
```

- To render single lines of content that contain plain text or unmatched HTML tags

Example:

```
@if (item == x) // plain text
{
    @:The time is @DateTime.Now
}
@if (item == x) // unmatched tags
{
    @:<ul>
}
else
{
    @:<ol>
}
```

Working with Objects

- Server coding **often involves with objects.**
- The "**Date**" object is a typical **built-in ASP.NET object**.
    - Objects can also be **self-defined, a web page, a text box, a file, a database record**, etc.
- Objects **may have methods** they can perform.
    - A **database object** might have a "**Save**" method, an image object might have a "**Rotate**" method
    - An **email object** might have a **"Send"** method, and so on.
- **Objects** also have **properties** that describe their **characteristics**.
    - A database record might have a **FirstName and a LastName property** (amongst others).
- The ASP.NET **Date object has a Now property (written as Date.Now),** and the **Now property has a Day property (written as Date.Now.Day).**

Example:

```
<table border="1">
       <tr>
              <th width="100px">Name</th>
              <th width="100px">Value</th>
       </tr>
       <tr>
              <td>Day</td><td>@DateTime.Now.Day</td>
       </tr>
       <tr>
              <td>Hour</td><td>@DateTime.Now.Hour</td>
       </tr>
       <tr>
              <td>Minute</td><td>@DateTime.Now.Minute</td>
       </tr>
       <tr>
              <td>Second</td><td>@DateTime.Now.Second</td>
       </tr>
       </td>
</table>
```

If and Else Conditions

- Determine **what to do based on conditions.**
- The common way to do this is with the if ... else statements:

```
@{
    var txt = "";
    if (DateTime.Now.Hour > 12)
    { txt = "Good Evening"; }
    else
    { txt = "Good Morning"; }
}
<p>The message is @txt</p>
```

Reading User Input

- Another important **feature** of **dynamic web pages** is that you can **read user input**.
- **Input is read by the Request[]** function, and **posting (input**) is **tested** by the **IsPost** condition.
- The **<label>** tag defines a label for an **<input>** element.

Example:

```
@{
var totalMessage = "";
if(IsPost)
    {
    var num1 = Request["text1"];
    var num2 = Request["text2"];
    var total = num1.AsInt() + num2.AsInt();
    totalMessage = "Total = " + total;
    }
}

<form action="" method="post">
<p><label for="text1">First Number:</label><br>
<input type="text" name="text1" /></p>
<p><label for="text2">Second Number:</label><br>
<input type="text" name="text2" /></p>
<p><input type="submit" value=" Add " /></p>
</form>
<p>@totalMessage</p>
```

Variables

- **Variables** are used to **store data.**
- The name of a variable must **begin with an alphabetic character** and cannot contain **whitespace** or **reserved characters**.
- Variables are declared using the **var keyword**, or by **using the type**

Examples

```
        // Using the var keyword:
        var greeting = "Welcome to mahedee.net";
        var counter = 103;
        var today = DateTime.Today;

        // Using data types:
        string greeting = "Welcome to mahedee.net";
        int counter = 103;
        DateTime today = DateTime.Today;
```

Data Types

- Below is a list of common data types:

Type	Description	Examples
int	Integer (whole numbers)	103, 12, 5168
float	Floating-point number	3.14, 3.4e38
decimal	Decimal number (higher precision)	1037.196543
bool	Boolean	true, false
string	String	"Welcome to mahedee.net", "Arif"

Operators

- An operator tells ASP.NET what kind of **command** to perform in an expression.
- The **C# language supports many operators**. Below is a list of common operators.

Operator	Description	Example
=	Assigns a value to a variable.	i=6

+
-
*
/
	Adds a value or variable.
	Subtracts a value or variable.
	Multiplies a value or variable.
	Divides a value or variable.
		i=5+5
		i=5-5
		i=5*5
		i=5/5

+=
-=
	Increments a variable.
	Decrements a variable.
		i += 1
		i -= 1

==	Equality. Returns true if values are equal.	if (i==10)
!=	Inequality. Returns true if values are not equal.	if (i!=10)

<
>
<=
>=
	Less than.
	Greater than.
	Less than or equal.
	Greater than or equal.
		if (i<10)
		if (i>10)
		if (i<=10)
		if (i>=10)

+	Adding strings (concatenation).	"mahedee" + " hasan"
.	Dot. Separate objects and methods.	DateTime.Hour
()	Parenthesis. Groups values.	(i+5)
()	Parenthesis. Passes parameters.	x=Add(i,5)
[]	Brackets. Accesses values in arrays or collections.	name[3]
!	Not. Reverses true or false.	if (!ready)

&&
||
	Logical AND.
	Logical OR.
		if (ready && clear)
		if (ready || clear)

Converting Data Types

- Converting from one data type to another is sometimes useful.
- The most common example is to **convert string input to another type, such as an integer** or a date.
- **As a rule, user input comes as strings, even if the user entered a number**.
  - Therefore, numeric input values must be converted to numbers before they can be used in **calculations**.

Below is a list of common conversion methods:

Method	Description	Example

AsInt()
IsInt()
	Converts a string to an integer.	if (myString.IsInt())
		{myInt=myString.AsInt();}

AsFloat()
IsFloat()
	Converts a string to a floating-point number.	if (myString.IsFloat())
		{myFloat=myString.AsFloat();}

AsDecimal()
IsDecimal()
	Converts a string to a decimal number.	if (myString.IsDecimal())
		{myDec=myString.AsDecimal();}

AsDateTime()	Converts a string to an ASP.NET DateTime	myString="10/10/2012";

IsDateTime()	type.	myDate=myString.AsDateTime();

AsBool()
IsBool()
	Converts a string to a Boolean.	myString="True";
		myBool=myString.AsBool();

ToString()	Converts any data type to a string.	myInt=1234;
		myString=myInt.ToString();

Chapter 5: Conditional and control statement

C# Loops and Arrays

For Loops

- If you need to run the **same statements repeatedly**, you can program a **loop**.
- If you **know how many times** you want to loop, you can use a **for loop**.
- This kind of loop is especially useful for **counting up** or **counting down.**

Example

```
<html>
<body>

    @for (var i = 10; i < 21; i++)
    {<p>Line @i</p>}


</body>
</html>
```

foreach Loops

- If you work with a **collection or an arra**y, you often use a **foreach loop**.
- A collection is a group of **similar objects**, and the **foreach** loop lets you carry out a task on **each item.**
- The foreach loop **walks through a collection until it is finished.**
- The example below walks through the ASP.NET **Request.ServerVariables** collection. Such as ALL_HTTP, ALL_RAW

Example

```
<html>
<body>

<ul>
@foreach (var x in Request.ServerVariables)
    {<li>@x</li>}
</ul>
</body>
</html>
```

While Loops

- The **while loop** is a general purpose loop.
- **A while loop begins with the while keyword**, followed by parentheses, **where you specify how long the loop** continues, then a block to repeat.
- While loops typically add to, or subtract from, a variable used for counting.
- In the example below, the += operator adds 1 to the variable i, each time the loop runs.

Example

```
<html>
<body>
@{
var i = 0;
while (i < 5)
    {
    i += 1;
    <p>Line @i</p>
    }
}
</body>
</html>
```

Arrays

- An array is useful when you want to **store similar variables**.

Example

```
@{
       string[] members = {"Mahedee", "Saiful", "Hasan", "Arif"};
       int i = Array.IndexOf(members, "Saiful") + 1;
       int len = members.Length;
       string x = members[2-1];
}
<html>
<body>
<h3>Members</h3>
@foreach (var person in members)
```

```
{
       <p>@person</p>
}
<p>The number of names in Members are @len</p>
<p>The person at position 2 is @x</p>
<p>Saiful is now in position @i</p>
</body>
</html>
```

C# Logic Conditions

The If Condition

- C# **lets you execute code based on conditions.**
- To test a condition you use an **if statement**.
- The **if statement returns true or false**, based on your test
- The if statement starts a code block
- The condition is written inside **parenthesis ()**
- The **code inside the braces** is executed if the test is true {}

Example

```
@{var price=50;}
<html>
<body>
@if (price>30)
    {
    <p>The price is too high.</p>
    }
</body>
</html>
```

The Else Condition

- An if statement can include an **else condition**.
- The else condition defines **the code to be executed if the condition is false.**

Example

```
@{var price=20;}
@if (price>30)
  {
  <p>The price is too high.</p>
  }
else
  {
  <p>The price is OK.</p>
  }
```

The Else If Condition

- **Multiple conditions** can be tested with an **else if condition**:

Example

```
@{var price = 25;}
<html>
<body>
    @if (price >= 30)
    {
        <p>The price is high.</p>
    }
    else if (price > 20 && price < 30)
    {
        <p>The price is OK.</p>
    }
    else
    {
        <p>The price is low.</p>
    }
</body>
</html>
```

Switch Conditions

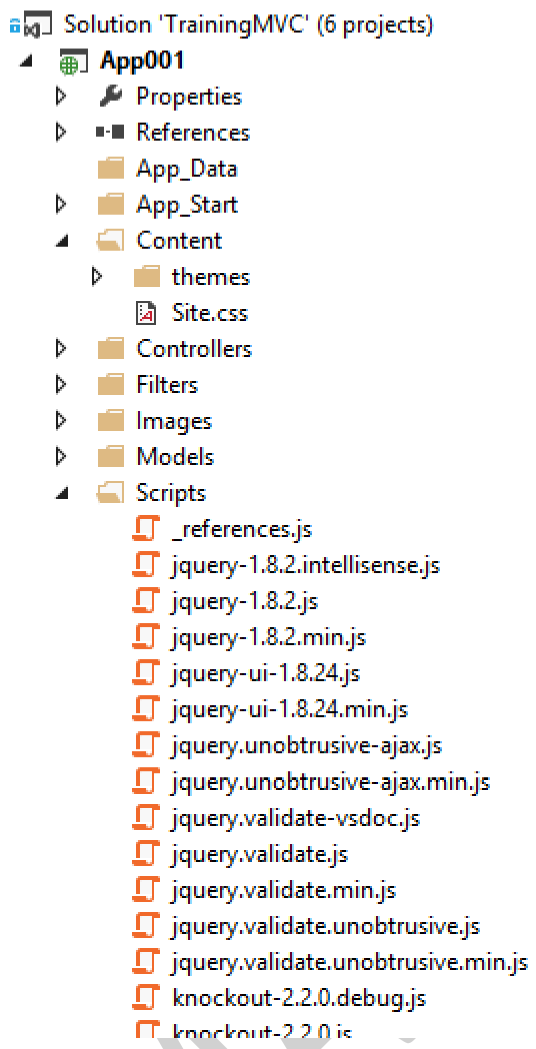
- A **switch block** used to test **a number of individual conditions**

Example

```
@{
    var weekday = DateTime.Now.DayOfWeek;
    var day = weekday.ToString();
    var message = "";
}
<html>
<body>
    @switch (day)
    {
        case "Monday":
            message = "This is the first weekday.";
            break;
        case "Thursday":
            message = "Only one day before weekend.";
            break;
        case "Friday":
            message = "Tomorrow is weekend!";
            break;
        default:
            message = "Today is " + day;
            break;
    }
    <p>@message</p>
</body>
</html>
```

Chapter 6: JavaScript and CSS

Folder Structure JavaScript and CSS



Solution 'TrainingMVC' (6 projects)
- App001
  - Properties
  - References
  - App_Data
  - App_Start
  - Content
    - themes
    - Site.css
  - Controllers
  - Filters
  - Images
  - Models
  - Scripts
    - _references.js
    - jquery-1.8.2.intellisense.js
    - jquery-1.8.2.js
    - jquery-1.8.2.min.js
    - jquery-ui-1.8.24.js
    - jquery-ui-1.8.24.min.js
    - jquery.unobtrusive-ajax.js
    - jquery.unobtrusive-ajax.min.js
    - jquery.validate-vsdoc.js
    - jquery.validate.js
    - jquery.validate.min.js
    - jquery.validate.unobtrusive.js
    - jquery.validate.unobtrusive.min.js
    - knockout-2.2.0.debug.js
    - knockout-2.2.0.js

JavaScript

- JavaScript is the **programming language** for the Web.
- JavaScript is **client side** scripting languages
  - **Scripting languages** are **special type** of programming language
  - Run on different environment
  - It **interpreted** rather compile
- All **modern HTML pages** are using JavaScript.
- JavaScript is **easy to learn.**

Example:

```
<script type="text/javascript">
    function myMessage()
    {
        alert("You have clicked a button.");
    }
</script>

<input type = "button" value="Click Me" onclick="myMessage()" />
```

CSS

- CSS stands for **Cascading Style Sheets**
- CSS defines **how HTML elements are to be displayed**
- Styles were added to **HTML 4.0** to solve a problem
- CSS saves a lot of work
- **External Style Sheets** are stored in CSS files

Example: Change in **site.css**

```
body {
    background-color: #333;
    border-top: solid 10px #000;
    color: #fff;
    font-size: .85em;
    font-family: "Segoe UI", Verdana, Helvetica, Sans-Serif;
    margin: 0;
    padding: 0;
}
```

Chapter 7: Controllers and Routes

Controllers

What is Controllers?

- **Controllers are classes** that handle **incoming browser requests, retrieve data, and then specify view templates** that return a **response to the browser.**
- It's responsible for **handling user input and responses**.
- MVC requires the name of all controllers to **end with "Controller".**
- Controller classes **inherited** from Controller base class.
- The **Controllers Folder** contains the controller classes
- In a sentence, controllers are **responsible** for processing **incoming requests, handling input, saving data, and sending a response to send back to the client.**

Example:

- Create a Controller name HomeController as follows

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Hi, this is my first application.";

        return View();
    }

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

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

- Modify Index action as follows.
- **RouteData** - Gets the route data for the **current request.**

```
public ActionResult Index()
{
    var controller = RouteData.Values["controller"];
    var action = RouteData.Values["action"];
    var id = RouteData.Values["id"];

    var message = string.Format("{0}::{1} {2}",controller, action, id);

    ViewBag.Message = message;

    return View();
}
```

- Now if you run the application you will see in the home page – Home::Index
- If you slightly modify the url like - http://localhost:25379/Home/Index/2323.
  - You will see home page - Home::Index 2323

Actions and Parameters

- Action is a **public method**

- **The controller defines action methods.**
- **Controllers** can include as many action methods as needed.
- Action methods **typically have a one-to-one mapping with user interactions**
- Can take **one or more parameter.**
- The method **cannot be a static method**.
- The method cannot be an **extension method**.
- The method **cannot be a constructor, getter, or setter**.
- The method cannot **have open generic types.**
- The method cannot contain **ref or out parameters**.

Example:

- Modify About action of Home controller as follows

```
public ActionResult About(int id = 0)
{
    if(id == 1)
    {
        ViewBag.CompanyName = "Leadsoft Bangladesh Limited";
        ViewBag.Founder = "Sheikh Abdul Aziz";
        ViewBag.Address = "41/6, Purana Paltan. Dhaka - 1000";
    }
    else
    {
        ViewBag.CompanyName = "Leadsoft Bangladesh Limited";
        ViewBag.Address = "41/6, Purana Paltan. Dhaka - 1000";
    }
    //ViewBag.Message = "Your application description page.";

    return View();
       }
```

- Modify About view as follows

```
@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
@*<h3>@ViewBag.Message</h3>

    <p>Use this area to provide additional information.</p>*@
<p>
    <h2>Company Information</h2>
    <h3>Company Name: @ViewBag.CompanyName</h3>
    <h4>Founder:      @ViewBag.Founder</h4>
    <h4>Address:      @ViewBag.Address</h4>

</p>
```

Input 1: http://localhost:14714/Home/About/

Input 2: http://localhost:14714/Home/About/1

What is Global.asax file?

- Global.asax file also known as **ASP.NET application file**
- Its **contains code for responding to application-level events** raised by ASP.NET or by HttpModules
- The **Global.asax file resides in the root directory** of an ASP.NET-based application derived from the HttpApplication base class.
- The Global.asax file is **optional**. If you do not define the file, the **ASP.NET page framework assumes** that you have **not defined any application or session event handlers**.

Methods corresponding to events that fire on each request

- **Application_BeginRequest()** – fired when a request for the web application comes in.
- **Application_AuthenticateRequest** –fired just before the user credentials are authenticated. You can specify your own authentication logic over here.
- **Application_AuthorizeRequest()** – fired on successful authentication of user's credentials. You can use this method to give authorization rights to user.
- **Application_ResolveRequestCache()** – fired on successful completion of an authorization request.
- **Application_AcquireRequestState()** – fired just before the session state is retrieved for the current request.
- **Application_PreRequestHandlerExecute()** - fired before the page framework begins **before executing an event handler** to handle the request.
- **Application_PostRequestHandlerExecute()** – fired after HTTP handler has executed the request.
- **Application_ReleaseRequestState()** – fired before current state data kept in the session collection is serialized.
- **Application_UpdateRequestCache()** – fired before information is added to output cache of the page.
- **Application_EndRequest()** – fired at the end of each request

Methods corresponding to events that do not fire on each request

- **Application_Start()** – fired when the first resource is requested from the web server and the web application starts.
- **Session_Start()** – fired when session starts on each new user requesting a page.
- **Application_Error()** – fired when an error occurs.
- **Session_End()** – fired when the session of a user ends.
- **Application_End()** – fired when the web application ends.
- **Application_Disposed() -** fired when the web application is destroyed.

Example: Global.asax.cs

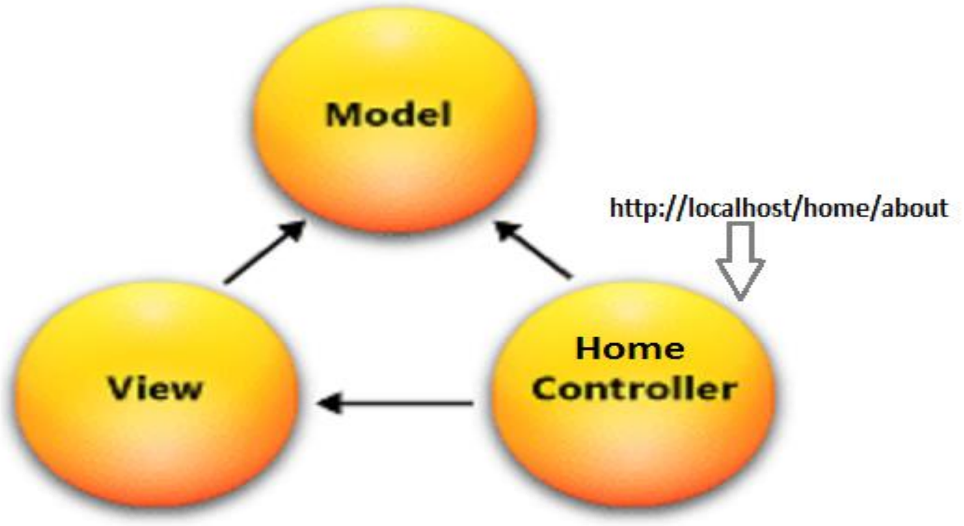
```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes); //Responsible for routing
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
  }
```

Routing

- **Routing** plays an important role in an ASP.NET MVC **Application execution flow.**
- It **maps request URL** to a specific **controller action** using a **Routing Table**.

Routing Engine

- Let's consider a url http://localhost/home/about .
- **How does it work** in an MVC application?
- How does it **deliver a request?**
    - Yes, **routing engine** is responsible for this task.
- So, what is **routing engine?**
    - In MVC routing engine **direct requests** to controllers.
    - It is a **core part** of asp.net.
- The **goal** of the **routing engine**
    - Its job is to **examine a URL** and **figure out where to send it for processing**.



Route in ASP.NET MVC

- In **Application_Start()** method of Global.asax, the following statement is **responsible for routing.**
  ```
  RouteConfig.RegisterRoutes(RouteTable.Routes);
  //An object that contains all the routes in the collection.
  ```

- If you click right button on "**RegisterRoutes**" and go to definition it will go to the RegisterRoutes method of the RouteConfig class which is App_Start folder. Here is the RouteConfig class.

```
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = 
UrlParameter.Optional }
            );
        }
    }
```

- Following statement means **ignore route** which is **in this pattern.**
  ```
  routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
  ```

- **Sample Route**

```
routes.MapRoute(

         name: "Default",                    //Route Name
         url: "{controller}/{action}/{id}",  //URL with parameters (Pattern for the
route)
         defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional } //Default route

            );
```

- **It means add a new route name "Default" and routing pattern must be** `"{controller}/{action}/{id}"`
- **Defaults route** means, when you **don't type any controller or action in the URL**, its goes to the index action of home controller.

Example 1:

RouteConfig

```
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

           routes.MapRoute("Cuisine",
                "cuisine/{name}",
                new {controller = "cuisine", action = "search", name = "" });

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
            );
        }
    }
```

- **Create a Cuisine Controller as follows**

```
    public class CuisineController : Controller
    {
        //
        // GET: /Cuisine/

        public ActionResult Search()
        {
            return Content("Hello");
        }
```

```
}
```

- **Input URL:** http://localhost:50945/Cuisine

Example 2:

- **Modify RouteConfig**

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute("Cuisine",
            "cuisine/{name}",
             new { controller = "cuisine", action = "search", name =
UrlParameter.Optional });
             //new { controller = "cuisine", action = "search", name = "" });
            //new {controller = "cuisine", action = "search"});

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
        );
    }
}
```

- **Modify Cuisine Controller as follows**

```
public class CuisineController : Controller
{
    public ActionResult Search(string name)
    {
        var message = Server.HtmlEncode(name);
        return Content(message);
    }

}
```

- Input URL: http://localhost:50945/Cuisine/mahedee

Example 3:

- **Modify RouteConfig**

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

```
            routes.MapRoute("Cuisine",
                "cuisine/{name}",
                 new { controller = "cuisine", action = "search", name =
UrlParameter.Optional });
                 //new { controller = "cuisine", action = "search", name = "" });
                //new {controller = "cuisine", action = "search"});

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
            );
        }
    }
```

- **Modify Cuisine Controller**

```
    public class CuisineController : Controller
    {
        public ActionResult Search(string name = "Default parameter")
        {
            var message = Server.HtmlEncode(name);
            return Content(message);
        }
    }
```

- **Test URL:**

URL: http://localhost:50945/Cuisine
URL: http://localhost:50945/Cuisine/Mahedee%20hasan
URL: http://localhost:50945/Cuisine/?name=I%20am%20Mahedee

Some Valid Route Pattern

Route Pattern	**URL Example**
mysite/{username}/{action}	~/mysite/jatten/login
public/blog/{controller}-{action}/{postId}	~/public/blog/posts-show/123
{country}-{lang}/{controller}/{action}/{id}	~/us-en/products/show/123
products/buy/{productId}-{productName}	~/products/but/2145-widgets

Action Results Return Type

- **Action** typically **returns** an **ActionResult**
- Most action methods return an **instance of a class** that **derives** from **ActionResult**
- The **ActionResult** class is the **base** for all action results.
- However, there are **different action result types**, depending on the task
    - The most **common action** is to call the **View method**.
    - The **View method returns** an instance of the **ViewResult** class, which is derived from **ActionResult**.
- You can create action methods that **return an object of any type**, such as a string, an integer, or a Boolean value. These return types are wrapped in an appropriate ActionResult type before they are rendered to the response stream.
- The following table shows the built-in action result types and the action helper methods that return them.

Action Result	Helper Method	Description
ViewResult	View	Renders a view as a Web page.
PartialViewResult	PartialView	**Renders a partial view**, which defines a section of a view that can be rendered inside another view.
RedirectResult	Redirect	**Redirects to another action method** by using its URL.
RedirectToRouteResult	RedirectToAction RedirectToRoute	**Redirects to another action method**.
ContentResult	Content	Returns a user-defined content type.
JsonResult	Json	Returns a serialized JSON object.
JavaScriptResult	JavaScript	Returns a script that can be executed on the client.
FileResult	File	**Returns binary output** to write to the response.
EmptyResult	(None)	Represents a return value that is used if the action method must return a **null** result (void).

Explore to Action Result and Return Type

RedirectPermanent

Example 1:

```
    public class CuisineController: Controller
    {
        public ActionResult Search(string title = "Default parameter")
        {
            return RedirectPermanent("http://mahedee.net");
        }
    }
```

Input URL: http://localhost:50945/Cuisine

RedirectToAction

Example 1:

```
        public ActionResult Search(string title = "Default parameter")
        {
            return RedirectToAction("Index", "Home");
        }
```

Input URL: http://localhost:50945/Cuisine

Example 2:

```
    public class CuisineController : Controller
    {
        public ActionResult Search(string name = "Default parameter")
        {
            return RedirectToAction("Index", "Home", new { title = title });
        }
    }
```

```
Test URL: http://localhost:50945/?title=Decorator Pattern
```

RedirectToRoute

Example 1:

```
    public class CuisineController : Controller
    {
        public RedirectToRouteResult Search(string title = "About me")
        {
            return RedirectToRoute("Default", new { controller = "Home", action = "About"
});
        }

    }
```

Test URL: http://localhost:50945/Article/Search

File

Example 1:

```
    public class CuisineController : Controller
    {
        public FileResult Search(string title = "About CSS")
        {
            return File(Server.MapPath("~/Content/site.css"), "text/css");
        }

    }
```

Test URL: http://localhost:50945/Article/Search

JSON

Example 1:

```
        public JsonResult Search(string title = "Object Oriented Programming with C#")
        {
            //string msg = Url.Encode(title);
            return Json(new { Id = 1, Title = title, Body = "List of Topics" },
JsonRequestBehavior.AllowGet);
        }

}
```

Test URL: http://localhost:50945/Article/

Chapter 8: Action Selectors and Action Filters

Action Selectors

- **Action selectors** are **attributes**
    - Can be **applied to action methods**
    - Are used to influence **which action method gets invoked in response to a request**.

- **AcceptVerbs**
  - This attribute is used when we want to execute some action when a **particular HTTP operation** is performed like **POST, GET, DELETE**, etc. E.g.:
  - **HttpPost, HttpGet**

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create(Employee employee)
{
      // To Do Code Here
}
```

- **AcceptName**
  - This attribute is used when you **expose an action name with a different name than** its method name
  - Or you can use an action name attribute to expose two methods with the same name as the action with different names. E.g.:

```
[ActionName("Modify")]
[HttpPost]
public ActionResult Edit(string departmentName)
{
    // ...
}
```

Example 1:

```
public class SelectorController : Controller
{
    [HttpPost]
    public ActionResult Search(string name = "mahedee")
    {
        string msg = Server.HtmlEncode(name);
        return Content(msg);
    }

    [HttpGet]
    public ActionResult Search()
    {
       return Content("This without parameter!");
    }
}
```

Test URL: http://localhost:27233/Selector/Search

In this occasion second method will call. If we keep only first method and type above URL it will give server error with resource cannot be found.

Example 2:

```
        [ActionName("SearchResult")]
        [HttpGet]
        public ActionResult Search()
        {
           return Content("This without parameter!");
        }
```

Test URL: http://localhost:27233/Selector/SearchResult

Action Filters

- An action filter is an **attribute**
    - You can apply to a controller **action or an entire controller**
    - It **modifies the way in which the action is executed.**

Name	Description
OutputCache	Cache the output of a controller
ValidateInput	**Turn off** request validation and allow dangerous input
Authorize	Restrict an action to authorized users or roles
ValidateAntiForgeryToken	Helps prevent **cross site request forgeries (CSRF)**
HandleError	Can specify a view to render in the event of an unhandled exception

Example:

```
        [OutputCache(Duration = 10, VaryByParam = "none")] //10 seconds
        public ActionResult Index()
        {
            ViewBag.Message = DateTime.Now.ToString();
            return View();
        }
```

Example:

```
    //[Authorize]
    public class SelectorController : Controller
    {
        [HttpPost]
        public ActionResult Search(string name = "mahedee")
        {
            // HtmlEncode is only meant to encode characters for display in HTML
            string msg = Server.HtmlEncode(name);
            return Content(msg);
        }

        [Authorize]
        //[Authorize(Roles = "Admin")]
        public ActionResult Search()
        {
            return Content("This without parameter!");
        }
    }
```

Test URL: http://localhost:27233/Selector/Search

Example:

```
    public class SelectorController : Controller
    {
        public ActionResult Search()
        {
            throw new Exception("Something terrible has happend!");
            return Content("This without parameter!");
        }
    }
```

```
Web.config

  <system.web>
    <customErrors mode="On"></customErrors>

    <!--<customErrors mode="RemoteOnly"></customErrors>-->

 </system.web>
```

Test URL: http://localhost:27233/Selector/Search

Error Page
Views -> Shared->Error.cshtml

Responsible:
FilterConfig.cs

Custom Action Filters

- **Derive** from `ActionFilterAttribute` base class
- **Create a Custome Atrribute class in Filters folder**

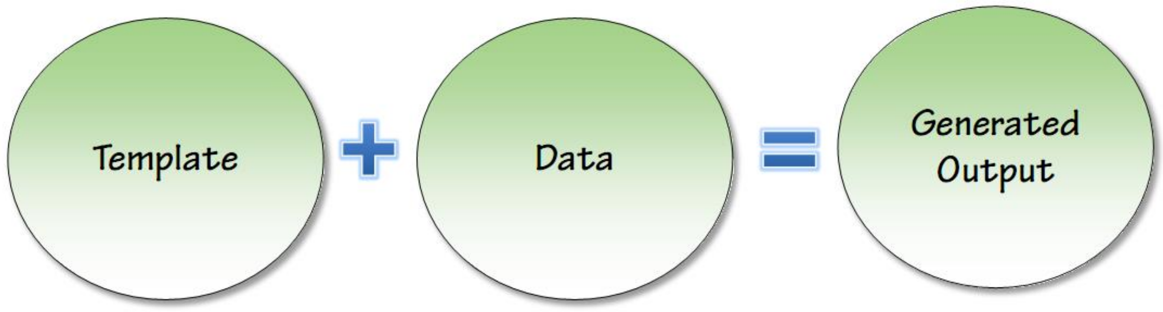
```
public class LogAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        base.OnActionExecuting(filterContext);
    }

    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        base.OnActionExecuted(filterContext);
    }

    //Before the view render
    public override void OnResultExecuting(ResultExecutingContext filterContext)
    {
        base.OnResultExecuting(filterContext);
    }

   //After the view render
    public override void OnResultExecuted(ResultExecutedContext filterContext)
    {
        base.OnResultExecuted(filterContext);
    }

}
```

- **Use custom filter in Controller's action**

```
public class SelectorController : Controller
{
    [Log]
    public ActionResult Search()
    {
        throw new Exception("Something terrible has happend!");
        return Content("This without parameter!");
    }
}
```

Test URL: http://localhost:27233/Selector/Search

Chapter 9: Razor Template



Display List view

Example

- Create a ASP.NET MVC Project. Example. **RestaurantOnline**

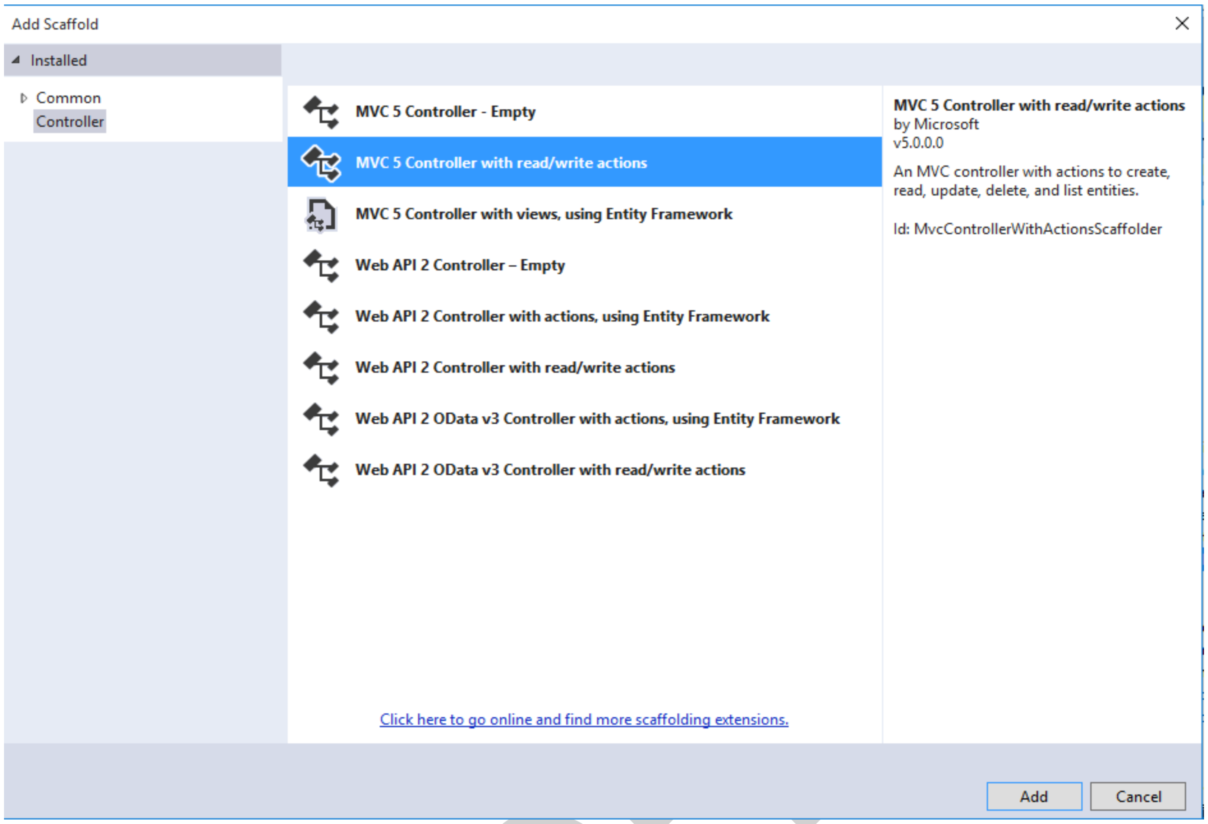
- Create a Model Name **RestaurantReview**

```
public class RestaurantReview
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string City { get; set; }
    public string Country { get; set; }
    public int Rating { get; set; }
}
```

- Create a Repository Name **RestaurantReviewRepository**

```
public class RestaurantReviewRepository
{
    public List<RestaurantReview> GetReviews()
    {
        List<RestaurantReview> lstRestaurantReview = new List<RestaurantReview>()
        {
            new RestaurantReview
            {
                Id = 1,
                Name = "Pan pacific Sonaragaon",
                City = "Dhaka",
                Country = "Bangladesh",
                Rating = 9
            },
            new RestaurantReview
            {
                Id = 2,
                Name = "Hotel Ruposhi Bangla",
                City = "Dhaka",
                Country = "Bangladesh",
                Rating = 8
            },
            new RestaurantReview
            {
                Id = 3,
                Name = "Radison",
                City = "Dhaka",
                Country = "Bangladesh",
                Rating = 10
            },

            new RestaurantReview
            {
                Id = 4,
                Name = "The House of Elliot",
                City = "Ghent",
                Country = "Belgium",
                Rating = 10
            }
        };

        return lstRestaurantReview;
    }
}
```

- Create a Controller Name **ReviewsController**
    - Select a template

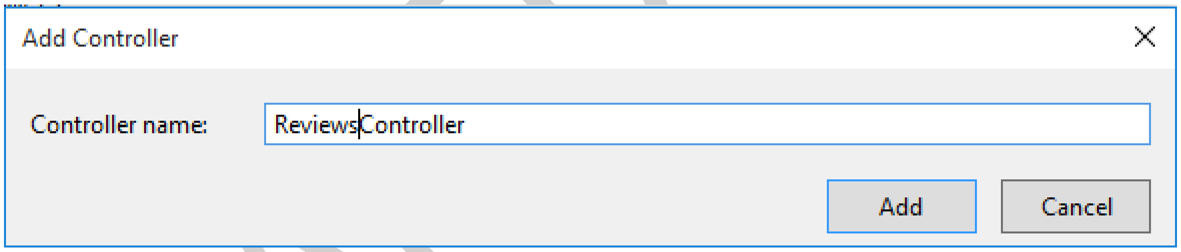

Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
Click here to go online and find more scaffolding extensions.
Add
Cancel

MVC 5 Controller with read/write actions
by Microsoft
v5.0.0.0

An MVC controller with actions to create, read, update, delete, and list entities.

Id: MvcControllerWithActionsScaffolder

- **Add Controller**



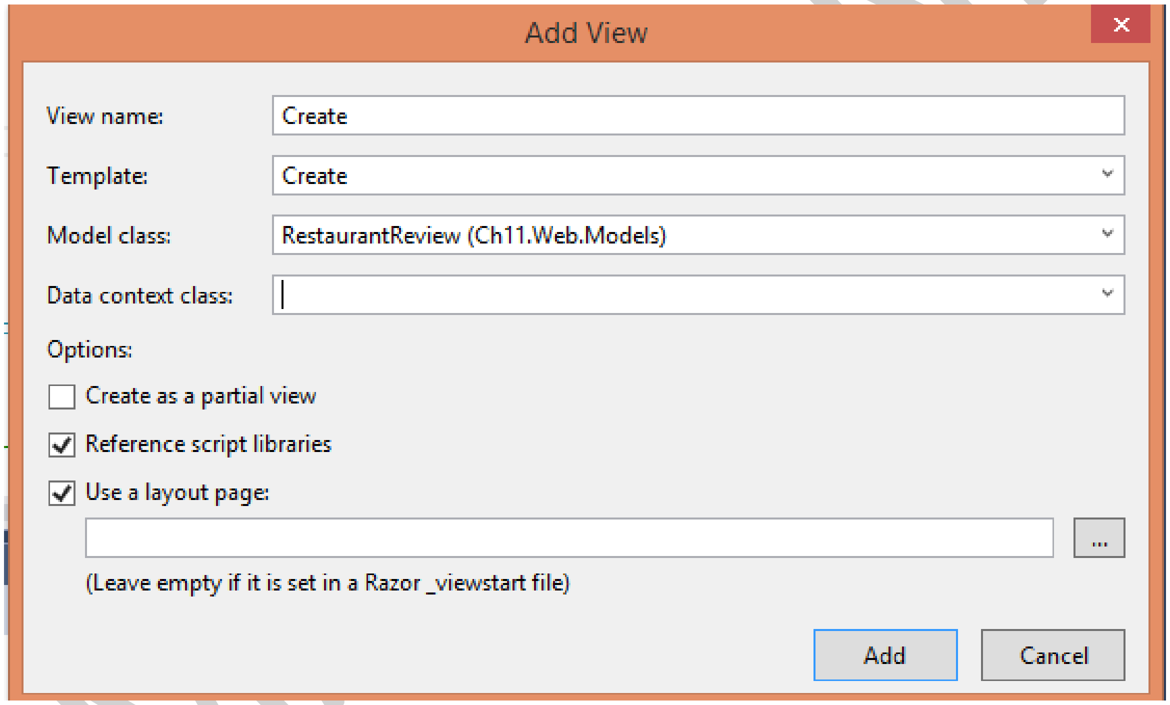
Controller name: ReviewsController

```
    public class ReviewsController : Controller
    {
        //
        // GET: /Reviews/

        public ActionResult Index()
        {
            RestaurantReviewRepository objRestaurantReviewRepository = new
RestaurantReviewRepository();

            var model =
                from r in objRestaurantReviewRepository.GetReviews()
                orderby r.Country
```

```
                    select r;

                return View(model);
            }

    }
```

Create a View

- Click right button on Action Index of RestaurantReview Controller
- Add new view and select as follows.



View name: Create
Template: Create
Model class: RestaurantReview (Ch11.Web.Models)
Data context class:
Options:
- [ ] Create as a partial view
- [✓] Reference script libraries
- [✓] Use a layout page:

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

A closer look on Index.cshtml

```
@model IEnumerable<MvcApplication1.Models.RestaurantReview>

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

<h2>Latest Reviews</h2>

We are showing the latest @Model.Count() reviews
<br />
```

```
@VirtualPath


<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.City)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Country)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Rating)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.City)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Country)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Rating)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}

</table>
```

Test URL: http://localhost:50945/Reviews

Details View

- Demo Details View
- Action for Details View

```
       //
        // GET: /RestaurantReviews/Details/5
        public ViewResult Details(int id)
        {
            RestaurantReview restaurantreview = context.RestaurantReviews.Single(x =>
x.Id == id);
            return View(restaurantreview);
        }
```

Create view

- Demo of Create View
- Actions for Create View

```
       //
        // GET: /RestaurantReviews/Create

        public ActionResult Create()
        {
            return View();
        }

        //
        // POST: /RestaurantReviews/Create

        [HttpPost]
        public ActionResult Create(RestaurantReview restaurantreview)
        {
            if (ModelState.IsValid)
            {
                context.RestaurantReviews.Add(restaurantreview);
                context.SaveChanges();
                return RedirectToAction("Index");
            }

            return View(restaurantreview);
        }
```

Edit View

- Demo of Edit View
- Actions for Edit View

```
        //
        // GET: /RestaurantReviews/Edit/5

        public ActionResult Edit(int id)
        {
                        RestaurantReview restaurantreview = new
RestaurantReviewRepository().GetReviews().Single(x => x.Id == id);
            return View(restaurantreview);

        }
        //
        // POST: /RestaurantReviews/Edit/5

        [HttpPost]
        public ActionResult Edit(RestaurantReview restaurantreview)
        {
            if (ModelState.IsValid)
            {
                context.Entry(restaurantreview).State = EntityState.Modified;
                context.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(restaurantreview);
        }
```

Delete View

- Demo of Delete View
- Actions for Delete View

```
        //
        // GET: /RestaurantReviews/Delete/5

        public ActionResult Delete(int id)
        {
            RestaurantReview restaurantreview = context.RestaurantReviews.Single(x =>
x.Id == id);
            return View(restaurantreview);
        }

        //
        // POST: /RestaurantReviews/Delete/5

        [HttpPost, ActionName("Delete")]
        public ActionResult DeleteConfirmed(int id)
        {
            RestaurantReview restaurantreview = context.RestaurantReviews.Single(x =>
x.Id == id);
            context.RestaurantReviews.Remove(restaurantreview);
            context.SaveChanges();
            return RedirectToAction("Index");
        }
```

Chapter 10: Code Expressions

Malicious scripts

- Also known as **cross site scripts (XSS)**
- Scripting attack

Change the previous repository as below.

```
        public List<RestaurantReview> GetReviews()
        {
            List<RestaurantReview> lstRestaurantReview = new List<RestaurantReview>()
            {
                new RestaurantReview
                {
                    Id = 1,
                    Name = "Pan pacific Sonaragaon",
                    City = "Dhaka",
                    Country = "Bangladesh",
                    Rating = 9
                },
                new RestaurantReview
                {
                    Id = 2,
                    Name = "Hotel Ruposhi Bangla",


                    City = "<script>alert('Your password is: abcd');</script>",


                    Country = "Bangladesh",
                    Rating = 8

                },
                new RestaurantReview
                {
                    Id = 3,
                    Name = "Radison",
                    City = "Dhaka",
                    Country = "Bangladesh",
                    Rating = 10
                },

                new RestaurantReview
                {
                    Id = 4,
                    Name = "The House of Elliot",
                    City = "Ghent",
                    Country = "Belgium",
                    Rating = 10
                }
            };

            return lstRestaurantReview;
        }
```

Test URL: http://localhost:50945/Reviews

And View source.

- What if , if you use

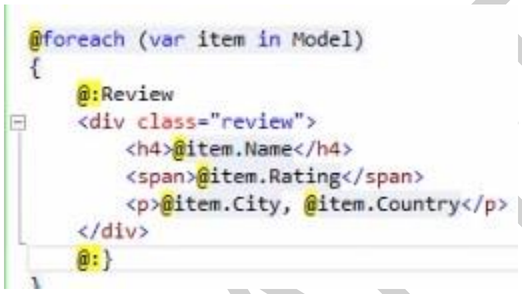
```
    @* @Html.DisplayFor(modelItem => item.City)*@
     @Html.Raw(item.City)
```

Test URL: http://localhost:50945/Reviews

Syntax

```
@item.Rating/10   @* 9/10 *@

@(item.Rating/10) @* 1 1 1 0 *@

m@item.Rating @* m@item.Rating 1 1 1 0 *@

@@item.Rating @* @item.Rating   *@
```

Code Blocks



```
@foreach (var item in Model)
{
    @:Review
    <div class="review">
        <h4>@item.Name</h4>
        <span>@item.Rating</span>
        <p>@item.City, @item.Country</p>
    </div>
    @:}
}
```

Output:

Pan pacific Sonargaon
} Review
Hotel Ruposhi Bangla
} Review
Radison
} Review
The House of Elliot
}

Chapter 11: HTML Helpers

What is HTML Helpers

- HTML Helpers are **used to modify HTML output**
- With MVC, HTML helpers are much **like traditional ASP.NET Web Form controls**.

- Just like web form controls in ASP.NET, HTML helpers are used to modify HTML.
    - But HTML helpers are more **lightweight**.
    - **Unlike** Web Form controls, an HTML helper **does not have an event model and a view state.**

- In **most cases**, an **HTML helper** is just a method that **returns a string**.

- With MVC, **you can create your own helpers**, or use the built in HTML helpers.

Standard HTML Helpers

- MVC includes standard helpers for the most **common types of HTML elements**, like **HTML links** and **HTML form** elements.

HTML Links

- The easiest way to render an HTML link 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**.

Razor Syntax:
```
@Html.ActionLink("About this Website", "About")
```

The `Html.ActionLink()` helper above, outputs the following HTML:

<a href="/Home/About">About this Website</a>

```
@Html.ActionLink("Edit Record", "Edit", new { Id = 3 })
```

The Html.ActionLink() helper above, outputs the following HTML:

<a href="/Home/Edit/3">Edit Record</a>

HTML Form Elements

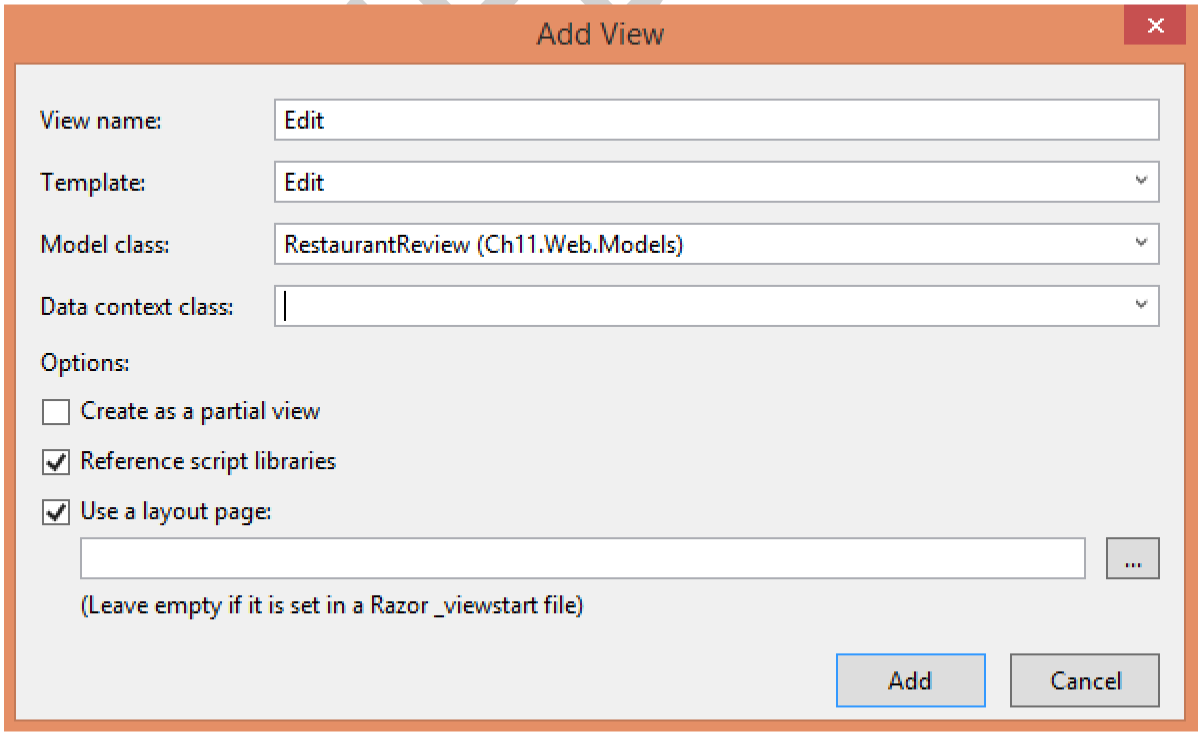
There following HTML helpers can be used to render (modify and output) HTML form elements:

- BeginForm()
- EndForm()
- TextArea()
- TextBox()
- CheckBox()
- RadioButton()
- ListBox()
- DropDownList()
- Hidden()
- Password()

- Html is a property of the ViewPage base class
    - Create inputs
    - Create links
    - Create forms

Example: `@Html.ActionLink("Edit", "Edit", new { id=item.Id })`

```
        public ActionResult Edit(int id)
        {
            RestaurantReviewRepository objRestaurantReviewRepository = new
RestaurantReviewRepository();
            var singleReview = objRestaurantReviewRepository.GetReviews().Single(r =>
r.Id == id);
            return View(singleReview);
        }
```



View name: Edit

Template: Edit

Model class: RestaurantReview (Ch11.Web.Models)

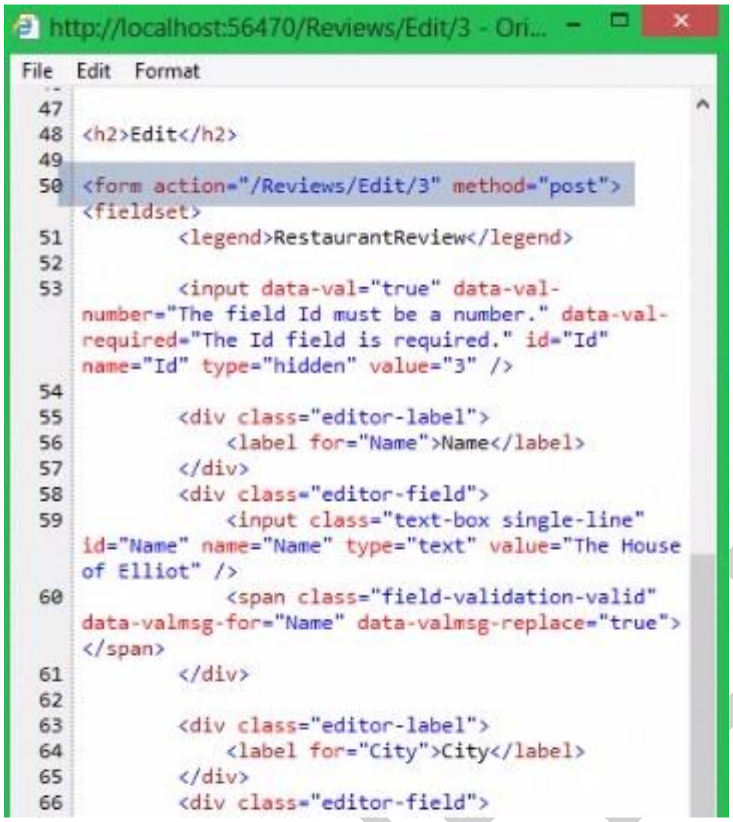
Data context class:

Options:

- [ ] Create as a partial view
- [✓] Reference script libraries
- [✓] Use a layout page:

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

- **Source file for HTLM Helper**

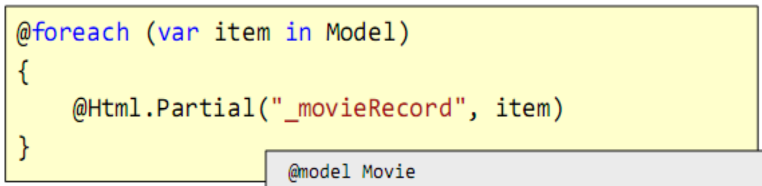


```

<h2>Edit</h2>

<form action="/Reviews/Edit/3" method="post">
<fieldset>
        <legend>RestaurantReview</legend>

        <input data-val="true" data-val-
number="The field Id must be a number." data-val-
required="The Id field is required." id="Id"
name="Id" type="hidden" value="3" />

        <div class="editor-label">
            <label for="Name">Name</label>
        </div>
        <div class="editor-field">
            <input class="text-box single-line"
id="Name" name="Name" type="text" value="The House
of Elliot" />
            <span class="field-validation-valid"
data-valmsg-for="Name" data-valmsg-replace="true">
</span>
        </div>

        <div class="editor-label">
            <label for="City">City</label>
        </div>
        <div class="editor-field">
```

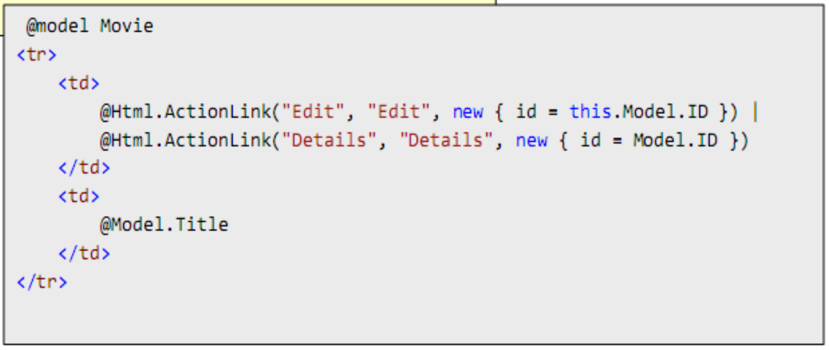
```
        [HttpPost]
        public ActionResult Edit(int id, FormCollection collection)
        {
              //string firstVal = collection[1].ToString();

            RestaurantReviewRepository objRestaurantReviewRepository = new
RestaurantReviewRepository();
            var singleReview = objRestaurantReviewRepository.GetReviews().Single(r =>
r.Id == id);
            if (TryUpdateModel(singleReview))
            {
                try
                {
                    // TODO: Add update logic here

                    return RedirectToAction("Index");
                }
                catch
                {
                    return View();
                }
            }
            return View();
        }
```

Partial Views

- Partial views **render portions of a page**
    - **Reuse** pieces of a view
    - Html helpers – **Partial** and Action
    - Razor partial views are still .cshtml files

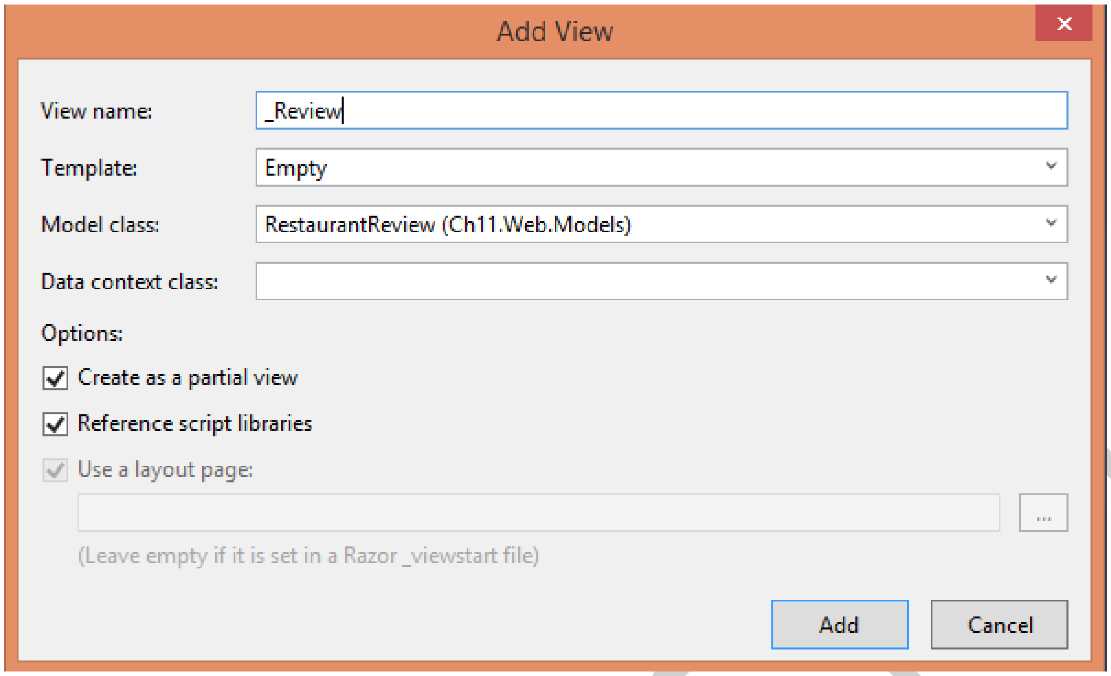


```
@foreach (var item in Model)
{
    @Html.Partial("_movieRecord", item)
}
```



```
 @model Movie
<tr>
    <td>
        @Html.ActionLink("Edit", "Edit", new { id = this.Model.ID }) |
        @Html.ActionLink("Details", "Details", new { id = Model.ID })
    </td>
    <td>
        @Model.Title
    </td>
</tr>
```

Create a Partial View



View name: _Review

Template: Empty

Model class: RestaurantReview (Ch11.Web.Models)

Data context class:

Options:

[✓] Create as a partial view

[✓] Reference script libraries

[✓] Use a layout page:

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

Create Shared->_Review.cshtml

```
@model MvcApplication1.Models.RestaurantReview

<tr>
    <td>
        @Model.Name
    </td>

    <td>
        @Model.Rating
    </td>

    <td>
        @Model.City
    </td>

    <td>
        @Model.Country
    </td>

</tr>
```

ReviewsController

```
    public class ReviewsController : Controller
    {
        public ActionResult BestReview()
        {
            RestaurantReviewRepository objRestaurantReviewRepository = new
RestaurantReviewRepository();

            var bestReview =
                from r in objRestaurantReviewRepository.GetReviews()
                orderby r.Rating descending
                select r;

            return PartialView("_Review", bestReview.First());

        }

}
```

Example:

Test URL: http://localhost:50945/

http://localhost:50945/Home/About etc

Test URL : http://localhost:50945/Reviews/BestReview

```
        [ChildActionOnly]
        public ActionResult BestReview()
        {

         }
```

Test URL: http://localhost:50945/Reviews/BestReview

http://localhost:50945/Home/About

- **Modify _Review Partial view a bit**

_Review.cshtml

```
@model Ch11.Web.Models.RestaurantReview

<tr>
    <td>
        @Model.Name
    </td>

    <td>
        @Model.Rating
    </td>

    <td>
        @Model.City
    </td>
    <td>
        @Model.Country
    </td>

    <td>
        @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
        @Html.ActionLink("Details", "Details", new { id = Model.Id }) |
        @Html.ActionLink("Delete", "Delete", new { id = Model.Id })
    </td>

</tr>
```

- **Now Modify Index View as follows**

```
@model IEnumerable<Ch11.Web.Models.RestaurantReview>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.City)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Country)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Rating)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    @Html.Partial("_Review", item);
}

</table>
```

Test URL: http://localhost:50945/Reviews/

Chapter 12: Introduction to Entity Framework

What is ORM?

- ORM stands for **Object-Relational-Mapping**.
- Is **mechanism** to **access database** object **without considering data source.**
- There are both **free and commercial packages available** of ORM, although some programmers opt to create their **own ORM tools**.
- Compared to traditional techniques of exchange between an object-oriented language and a relational database, **ORM often reduces the amount of code that needs to be written**
- **Disadvantages** of ORM tools generally stem from the high level of abstraction obscuring what is actually happening in the implementation code. Also, heavy reliance on **ORM software has been cited as a major factor in producing poorly designed databases**
- **Lately, alternatives to ORMs** such as **Slazure** (http://www.slazure.com/) have become available
    - **Slazure** is a revolutionary .NET database client library

ADO.NET Entity Framework

- Entity Framework is an ORM – Object Relational Mapper
- Performing basic CRUD (Create, Read, Update, and Delete) operations.
- Easily managing "1 to 1", "1 to many", and "many to many" relationships.
- Ability to have **inheritance relationships** between entities.
- **Access** a relational **database** with strongly-typed **LINQ** queries

Benefits of Entity Framework

- We can have all **data access logic** written in **higher level languages.**
- The conceptual **model can be represented** in a better way by using **relationships among entities.**
- The **underlying data store can be replaced without much overhead** since all data access logic is present at a higher level.

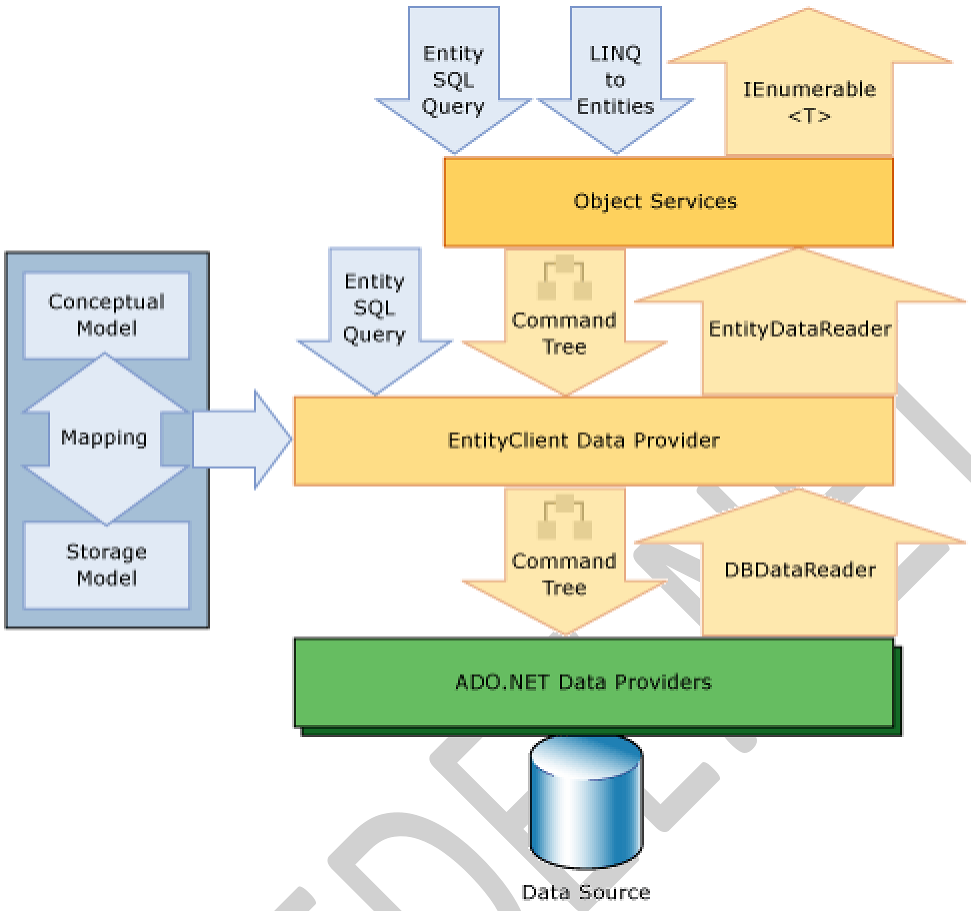
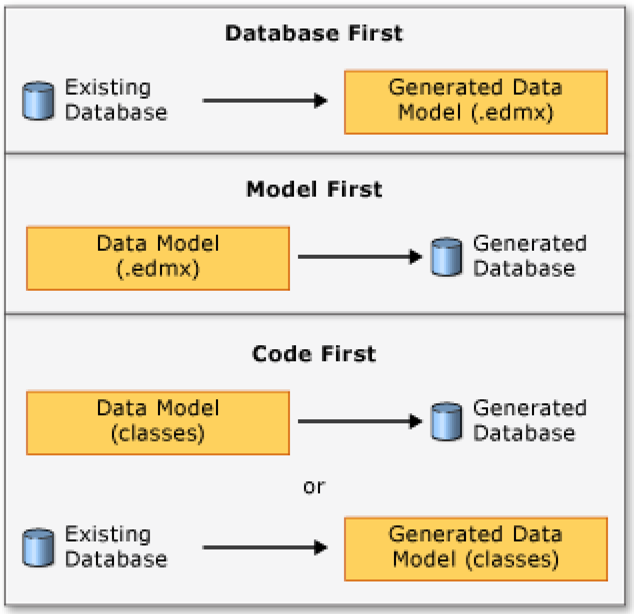


Fig: Entity Framework architecture for accessing data

Entity Framework Development Approach

- There are three ways you can work with data models and databases in the Entity Framework: **Database First, Model First, and Code First**.



- **Database First**

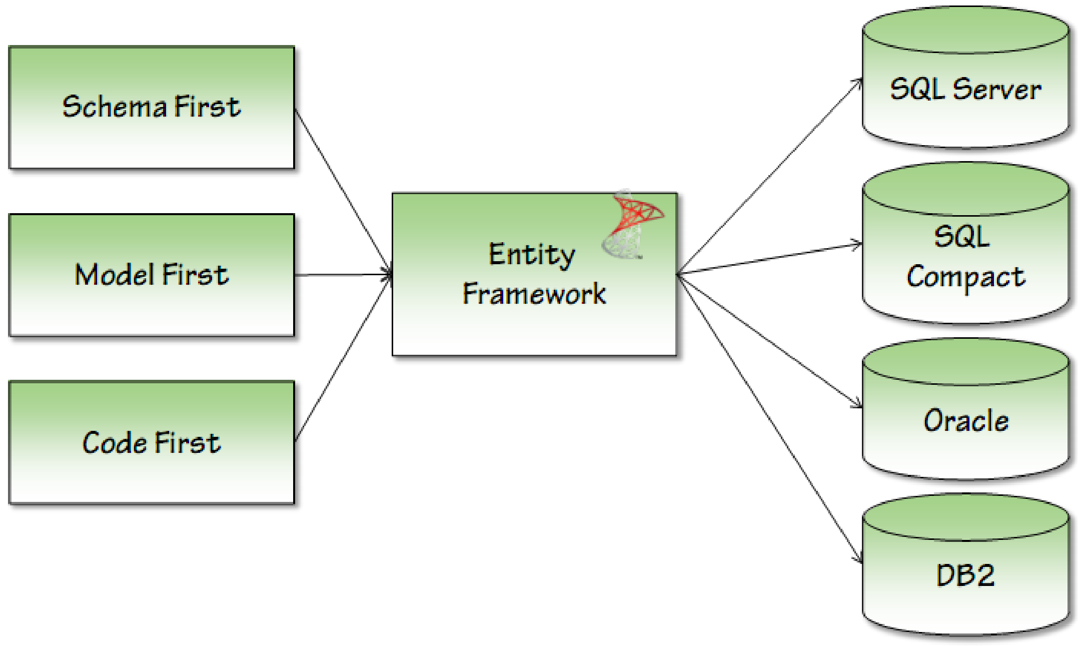
If you already have a database, the Entity Framework designer built into Visual Studio can automatically generate a data model that consists of classes and properties that correspond to existing database objects such as tables and columns. The information about your database structure (store schema), your data model (conceptual model), and the mapping between them is stored in XML in an .edmx file. The Entity Framework designer provides a graphical UI that you can use to display and edit the .edmx file.

- **Model First**

If you don't have a database yet, you can begin by creating a model in an .edmx file by using the Entity Framework graphical designer in Visual Studio. When the model is finished, the Entity Framework designer can generate DDL (data definition language) statements to create the database. As in Database First, the .edmx file stores model and mapping information.

- **Code First**

Whether you have an existing database or not, you can use the Entity Framework without using the designer or an .edmx file. If you don't have a database, you can code your own classes and properties that correspond to tables and columns. If you do have a database, Entity Framework tools can generate the classes and properties that correspond to existing tables and columns. The mapping between the store schema and the conceptual model represented by your code is handled by convention and by a special mapping API. If you let Code First create the database, you can use Code First Migrations to automate the process of deploying the database to production. Migrations can also automate the deployment of database schema changes to production when your data model changes.



Building Entity

- Install package – Entity Framework (if required)
    - `PM> Install-Package EntityFramework`
- Enable Migration
    - `PM> Enable-Migrations -ContextTypeName RestaurantContext`
- Add Migration
    - `PM> add-migration initalMigration`
- Update database
    - `PM> Update-Database –Verbose`

Creating application using ASP.NET MVC and EF

Create Models

```
public class RestaurantReview
{
    public int Id { get; set; }
    public int Rating { get; set; }
    public string Body { get; set; }
    public int RestautranId { get; set; }
}
```

```
    public class Restaurant
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public ICollection<RestaurantReview> Reviews { get; set; }

    }
```

Add ConnectionString

```
<add name="DefaultConnection" connectionString="Data Source=PCSW83;Initial
Catalog=RestaurantDB;User ID=sa; Password=leads@123" providerName="System.Data.SqlClient"
/>
```

Create Context

```
    public class RestaurantContext : DbContext
    {
        public RestaurantContext()
            : base("name=DefaultConnection")
        {

        }
        public DbSet<Restaurant> Restaurants {get; set;}
        public DbSet<RestaurantReview> Reviews{get; set;}
    }
```

Enable Migration

- PM> Enable-Migrations -ContextTypeName RestaurantContext

Update Configuration

```
    internal sealed class Configuration :
DbMigrationsConfiguration<MvcApplication3.Models.RestaurantContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
        }

        protected override void Seed(MvcApplication3.Models.RestaurantContext context)
        {
            context.Restaurants.AddOrUpdate(r => r.Name,
                new Restaurant { Name = "Hotel Westing", City = "Dhaka", Country =
"Bangladesh" },
                new Restaurant { Name = "Hotel Sonargaon", City = "Dhaka", Country =
"Bangladesh" },
                new Restaurant
                {
                    Name = "Hotel Radison",
                    City = "Dhaka",
                    Country = "Bangladesh",
                    Reviews = new List<RestaurantReview>
                    {
                        new RestaurantReview{Rating = 9, Body = "Very good Food!"}
                    }
                });
            }
        }
```

Update Home Controller

```
    public class HomeController : Controller
    {
        RestaurantContext _context = new RestaurantContext();

        public ActionResult Index()
        {
            var model = _context.Restaurants.ToList();
            return View(model);
        }
```

Update Index View

```
@model IEnumerable<MvcApplication3.Models.Restaurant>
@{
    ViewBag.Title = "Home Page";
}

@foreach (var item in Model)
{
    <div>
        <h4>@item.Name</h4>
        <div>@item.City, @item.Country</div>
        <br />
    </div>
}
```

Using LINQ

- LINQ stands for Language Integrated Query
- **Comprehension** Query Syntax

```
var query = from r in _db.Restaurants
            where r.Country == "USA"
            orderby r.Name
            select r;
```

- **Extension** Method Syntax

```
var query = _db.Restaurants
               .Where(r => r.Country == "USA")
               .OrderBy(r => r.Name)
               .Skip(10)
               .Take(10);
```

- References for LINQ:
  - https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b
  - http://www.linqpad.net/

Sample Code 1

```
RestaurantContext _context = new RestaurantContext();

public ActionResult Index()
{
    var model = from r in _context.Restaurants //r is range variable
                orderby r.Name ascending
                select r;
    return View(model);
}
```

Sample Code 2

```
RestaurantContext _context = new RestaurantContext();

public ActionResult Index()
{
    var model = from r in _context.Restaurants //r is range variable
                orderby r.Reviews.Count() ascending
                select r;
    return View(model);
}
```

Sample Code 3

```
RestaurantContext _context = new RestaurantContext();
```

```
        public ActionResult Index()
        {
            var model = from r in _context.Restaurants //r is range variable
                        orderby r.Reviews.Average(review => review.Rating) ascending
                        select r;
            return View(model);
        }
```

Sample Code 4

```
        RestaurantContext _context = new RestaurantContext();

        public ActionResult Index()
        {
            var model = from r in _context.Restaurants //r is range variable
                        orderby r.Reviews.Average(review => review.Rating) ascending
                        select new
                        {
                            r.Id,
                            r.Name,
                            r.City,
                            r.Country,
                            NoOfReviews = r.Reviews.Count()
                        };

            return View(model);
        }
```

Modified Project

Add Model RestaurantReviewViewModel

```
    public class RestaurantViewModel
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public int CountOfReviews { get; set; }
    }
```

Update Index Action of Home Controller

```
        RestaurantContext _context = new RestaurantContext();

        public ActionResult Index()
        {
            var model = from r in _context.Restaurants //r is range variable
                        orderby r.Reviews.Average(review => review.Rating) ascending
                        select new RestaurantViewModel
                        {
                            Id = r.Id,
                            Name = r.Name,
                            City = r.City,
                            Country = r.Country,
                            CountOfReviews = r.Reviews.Count()
                        };

            return View(model);
        }
```

Modify Index View

```
@model IEnumerable<MvcApplication3.Models.RestaurantViewModel>
@{
    ViewBag.Title = "Home Page";
}

@foreach (var item in Model)
{
    <div>
        <h4>@item.Name</h4>
        <div>@item.City, @item.Country</div>
        <div>Reviews: @item.CountOfReviews</div>
        <br />
    </div>
}
```

Modify the Index action using extended method

```
        RestaurantContext _context = new RestaurantContext();

        public ActionResult Index()
        {
            var model = _context.Restaurants
                .OrderByDescending(r => r.Reviews.Average(review => review.Rating))
                .Take(10)
                .Select(r => new RestaurantViewModel
                        {
                            Id = r.Id,
                            Name = r.Name,
                            City = r.City,
                            Country = r.Country,
                            CountOfReviews = r.Reviews.Count()
                        });

            return View(model);
        }
```

Filtering using LINQ

Modify Index action of Home Controller

```
        RestaurantContext _context = new RestaurantContext();

        public ActionResult Index(string searchTerm = null)
        {
            var model = _context.Restaurants
                .OrderByDescending(r => r.Reviews.Average(review => review.Rating)) //r
is range variable
                .Where(r => searchTerm == null || r.Name.StartsWith(searchTerm))
                .Take(10)
                .Select(r => new RestaurantViewModel
                        {
                            Id = r.Id,
                            Name = r.Name,
                            City = r.City,
                            Country = r.Country,
                            CountOfReviews = r.Reviews.Count()
                        });

            return View(model);
        }
```

Test URL: localhost:34165

Test URL : http://localhost:34165/?searchTerm=Hotel%20R

Sample Example

- Modify the index.cshtml and see the output

```
@model IEnumerable<MvcApplication3.Models.RestaurantViewModel>
@{
    ViewBag.Title = "Home Page";
}

<form method="get">
    <input type="search" name="searchTerm" />
    <input type="submit" value="Search by Name" />
</form>

@foreach (var item in Model)
{
    <div>
        <h4>@item.Name</h4>
        <div>@item.City, @item.Country</div>
        <div>Reviews: @item.CountOfReviews</div>
        <br />
    </div>
}
```

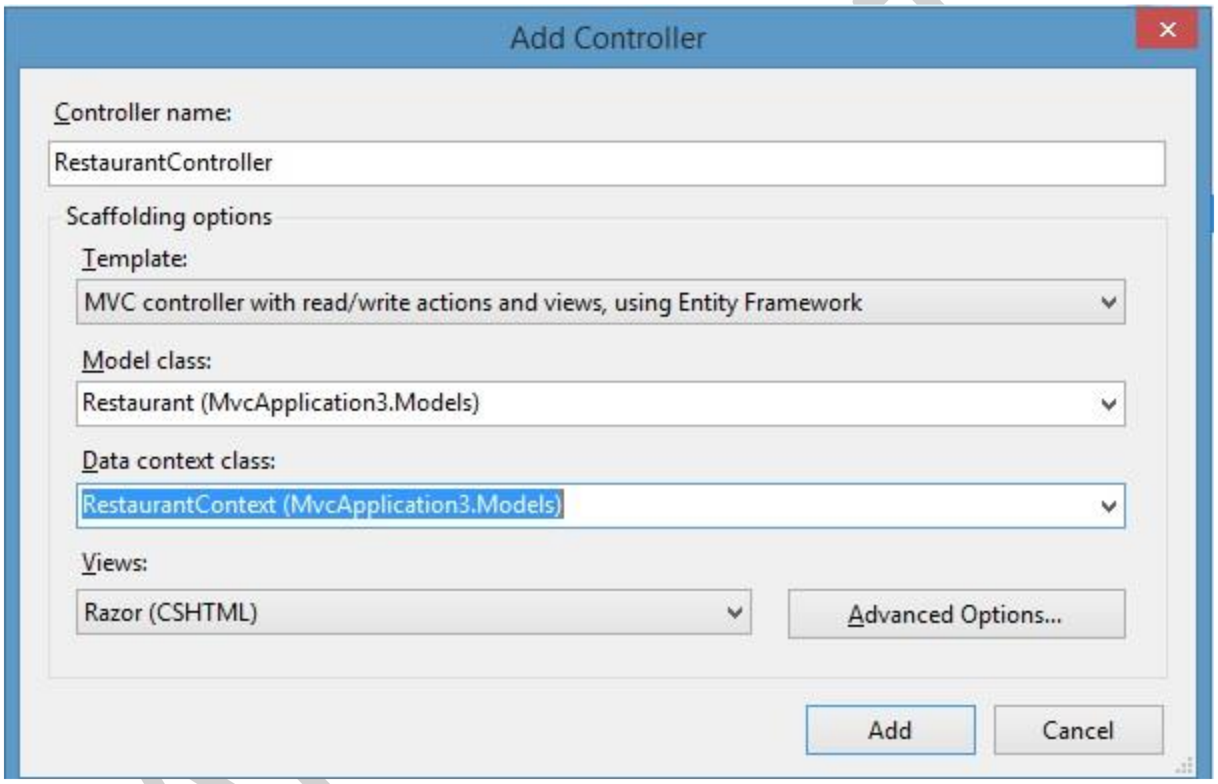
Chapter 13: Working with Data Using Entity Framework

Listing using Scaffolding

1. Scaffolding uses for CRUD Operation
2. Add a link in Layout

```
<li>@Html.ActionLink("Restaurants","Index","Restaurant")</li>
```

3. Create a Restaurant Controller



Controller name: RestaurantController

Scaffolding options

Template: MVC controller with read/write actions and views, using Entity Framework

Model class: Restaurant (MvcApplication3.Models)

Data context class: RestaurantContext (MvcApplication3.Models)

Views: Razor (CSHTML)

4. Run application and Click Restaurant Links – and Add, Modify, Update & Delete

Example 2

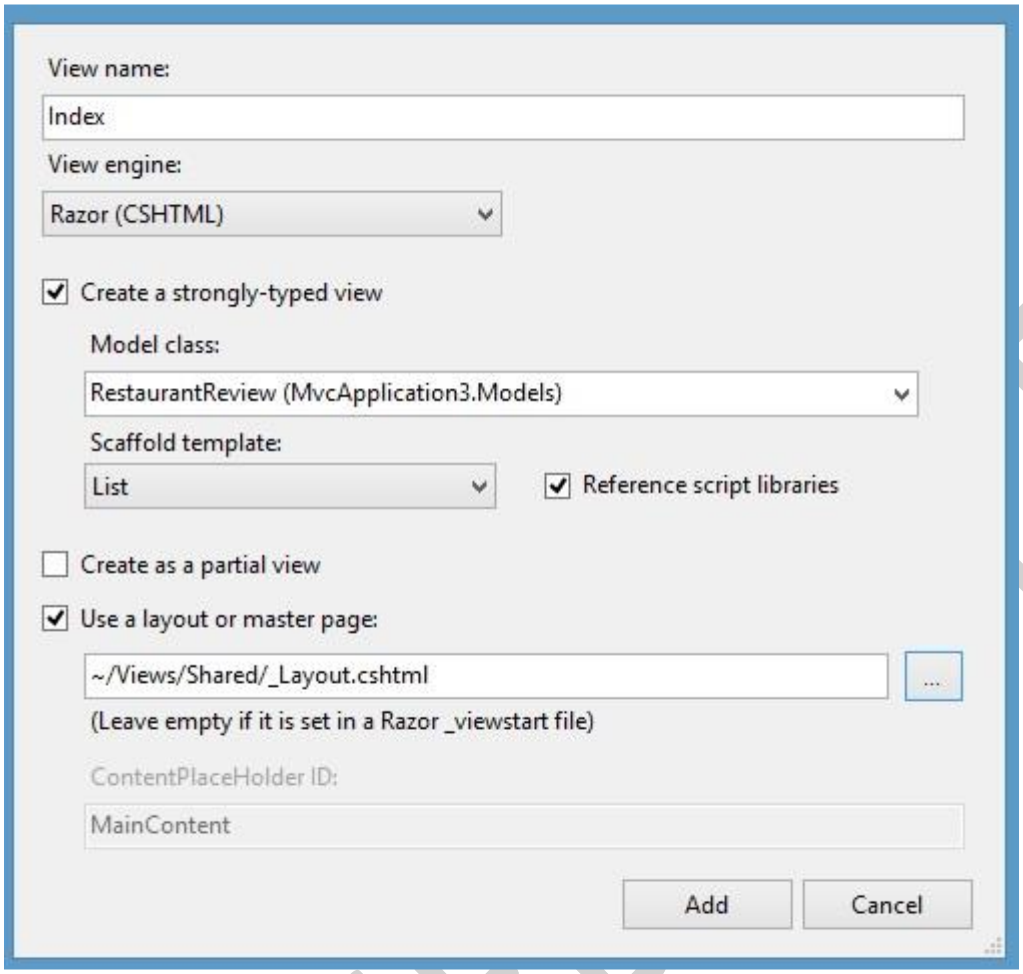
Update Index.cshtml in Restaurant

```
@Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
@Html.ActionLink("Reviews", "Index","Reviews", new { id=item.Id },null) |
@Html.ActionLink("Delete", "Delete", new { id=item.Id })
```

Create a ReviewController

```
public class ReviewsController : Controller
{
    RestaurantContext context = new RestaurantContext();

    public ActionResult Index([Bind(Prefix="id")]int restaurantId)
    {
        var restaurant = context.Restaurants.Find(restaurantId);
        if (restaurant != null)
        {
            return View(restaurant);
        }
        return HttpNotFound();
    }

    protected override void Dispose(bool disposing)
    {
        context.Dispose();
        base.Dispose(disposing);
    }

}
```

- Right click on index and create a view like below


View name: Index
View engine: Razor (CSHTML)
[✓] Create a strongly-typed view
Model class: RestaurantReview (MvcApplication3.Models)
Scaffold template: List
[✓] Reference script libraries
[ ] Create as a partial view
[✓] Use a layout or master page:
~/Views/Shared/_Layout.cshtml
(Leave empty if it is set in a Razor _viewstart file)
ContentPlaceHolder ID: MainContent

Modify the index view

```
@model MvcApplication3.Models.Restaurant

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

<h2>Reviews for @Model.Name</h2>

@Html.Partial("_Reviews", @Model.Reviews);

<p>
    @Html.ActionLink("Create New", "Create")
</p>
```

Create a partial view

```
@model IEnumerable<MvcApplication3.Models.RestaurantReview>
```

```
<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Rating)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Body)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Rating)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Body)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id })
        </td>
    </tr>
}

</table>
```

Modify Resturant Model

```
    public class Restaurant
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public virtual ICollection<RestaurantReview> Reviews { get; set; }

    }
```

Run application and test

Create or Insert using Scaffolding

- Have a look of the model

```
    public class Restaurant
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public virtual ICollection<RestaurantReview> Reviews { get; set; }
    }
```

```
public class RestaurantReview
{
    public int Id { get; set; }
    public int Rating { get; set; }
    public string Body { get; set; }
    public int RestaurantId { get; set; }
}
```

- Add Action in review controller

```
    [HttpGet]
    public ActionResult Create(int restaurantId)
    {
        return View();
    }

    [HttpPost]
    public ActionResult Create(RestaurantReview review)
    {
        if (ModelState.IsValid)
        {
            context.Reviews.Add(review);
            context.SaveChanges();
            return RedirectToAction("Index", new { id = review.RestaurantId });
        }
        return View(review);
    }
```

- Right click on the Create action and scaffold to create a form

Reviews->create.cshtml

```
@model MvcApplication3.Models.RestaurantReview

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

<h2>Create</h2>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>New Review</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Rating)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Rating)
            @Html.ValidationMessageFor(model => model.Rating)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Body)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Body)
            @Html.ValidationMessageFor(model => model.Body)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

- Now run the application add review of corresponding restaurant.

Performing edit using scaffolding

- Modify the ReviewsController

```
        public ActionResult Edit(int id)
        {
            var model = context.Reviews.Find(id);
            return View(model);
        }

        [HttpPost]
        public ActionResult Edit(RestaurantReview review)
        {
            if (ModelState.IsValid)
            {
                context.Entry(review).State = System.Data.EntityState.Modified;
                context.SaveChanges();
                return RedirectToAction("Index", new { id = review.RestaurantId});
            }
            return View(review);
        }
```

- Create Edit form using scaffolding
- Modify Edit form

```
    @model MvcApplication3.Models.RestaurantReview

    @{
```

```
    ViewBag.Title = "Edit";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Edit</h2>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>RestaurantReview</legend>

        @Html.HiddenFor(model => model.Id)

        <div class="editor-label">
            @Html.LabelFor(model => model.Rating)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Rating)
            @Html.ValidationMessageFor(model => model.Rating)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Body)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Body)
            @Html.ValidationMessageFor(model => model.Body)
        </div>

            @Html.HiddenFor(model => model.RestaurantId)

        <p>
            <input type="submit" value="Save" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

- Now edit reviews and look at the changes

Delete using scaffolding

- Modify Reviews->_Reviews.cshtml

```
 <td>
     @Html.ActionLink("Edit", "Edit", new { id=item.Id })
     |@Html.ActionLink("Delete", "Delete", new { id = item.Id })
 </td>
```

- Modify ReviewsController
  - Add Following actions to ReviewsController

```
public ActionResult Delete(int id = 0)
{
    RestaurantReview review = context.Reviews.Find(id);
    if (review == null)
    {
        return HttpNotFound();
    }
    return View(review);
}

//
// POST: /Reviews/Delete/5

[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
{
    RestaurantReview review = context.Reviews.Find(id);
    context.Reviews.Remove(review);
    context.SaveChanges();
    return RedirectToAction("Index", new { id = review.RestaurantId });
}
```

- Create Delete Template
  - Create delete template – Just click right button on Delete Action and Scaffold and then modify the Delete.cshtml like below

```
@model MvcApplication3.Models.RestaurantReview

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

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<fieldset>
    <legend>RestaurantReview</legend>

    <div class="display-label">
         @Html.DisplayNameFor(model => model.Rating)
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Rating)
    </div>

    <div class="display-label">
         @Html.DisplayNameFor(model => model.Body)
    </div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Body)
    </div>
```

```
</fieldset>
@using (Html.BeginForm()) {
    <p>
        <input type="submit" value="Delete" /> |
        @Html.ActionLink("Back to List", "Index")
    </p>
}
```

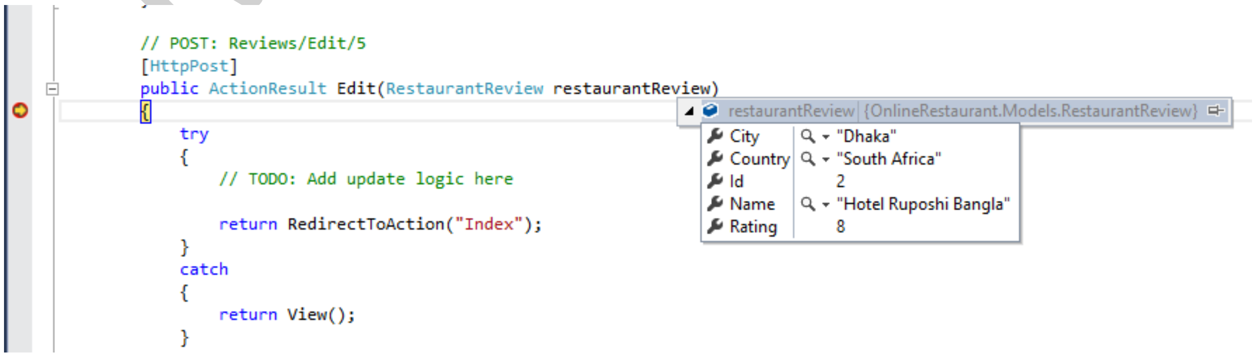
- Now run application and look at the output.

Mass Assignment or Over posting

- Remove Country field in edit Form. Edit form like below

```
@model OnlineRestaurant.Models.RestaurantReview

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>RestaurantReview</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.Id)

        <div class="form-group">
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class
= "form-control" } })
                @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-
danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.City, htmlAttributes: new { @class = "control-
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.City, new { htmlAttributes = new { @class
= "form-control" } })
                @Html.ValidationMessageFor(model => model.City, "", new { @class = "text-
danger" })
            </div>
        </div>

        @*<div class="form-group">
            @Html.LabelFor(model => model.Country, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Country, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Country, "", new { @class =
"text-danger" })
            </div>
        </div>*@

        <div class="form-group">
            @Html.LabelFor(model => model.Rating, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Rating, new { htmlAttributes = new {
@class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Rating, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

- Change URL like http://localhost:5562/Reviews/Edit/2?Country=South%20Africa
- And click Save and set debugger in Edit Action



```
        // POST: Reviews/Edit/5
        [HttpPost]
        public ActionResult Edit(RestaurantReview restaurantReview)
        {
            try
            {
                // TODO: Add update logic here

                return RedirectToAction("Index");
            }
            catch
            {
                return View();
            }
```

```
restaurantReview {OnlineRestaurant.Models.RestaurantReview}
    City     "Dhaka"
    Country  "South Africa"
    Id       2
    Name     "Hotel Ruposhi Bangla"
    Rating   8
```

- Prevent Mass Assignment or Overposting
  - Use Exclude or Black list property like ([Bind(Exclude="Country,City")]

```
[HttpPost]
public ActionResult Edit([Bind(Exclude="Country")] RestaurantReview review)
{
    if (ModelState.IsValid)
    {
        context.Entry(review).State = System.Data.EntityState.Modified;
        context.SaveChanges();
        return RedirectToAction("Index", new { id = review.RestaurantId});
    }
    return View(review);
}
```

Chapter 14: Annotations

Validation Annotations

- Modify the model

```
public class RestaurantReview
{
    public int Id { get; set; }

    [Range(1,10)]
    [Required]
    public int Rating { get; set; }

    [StringLength(1000)]
    [Display(Name="Comments")]
    [DisplayFormat(NullDisplayText = "No Comments!")]
    public string Body { get; set; }


    public int RestaurantId { get; set; }
}
```

- Run the following command in Package Manager console
  - **PM> Update-Database -Verbose –Force**

Custom Validation

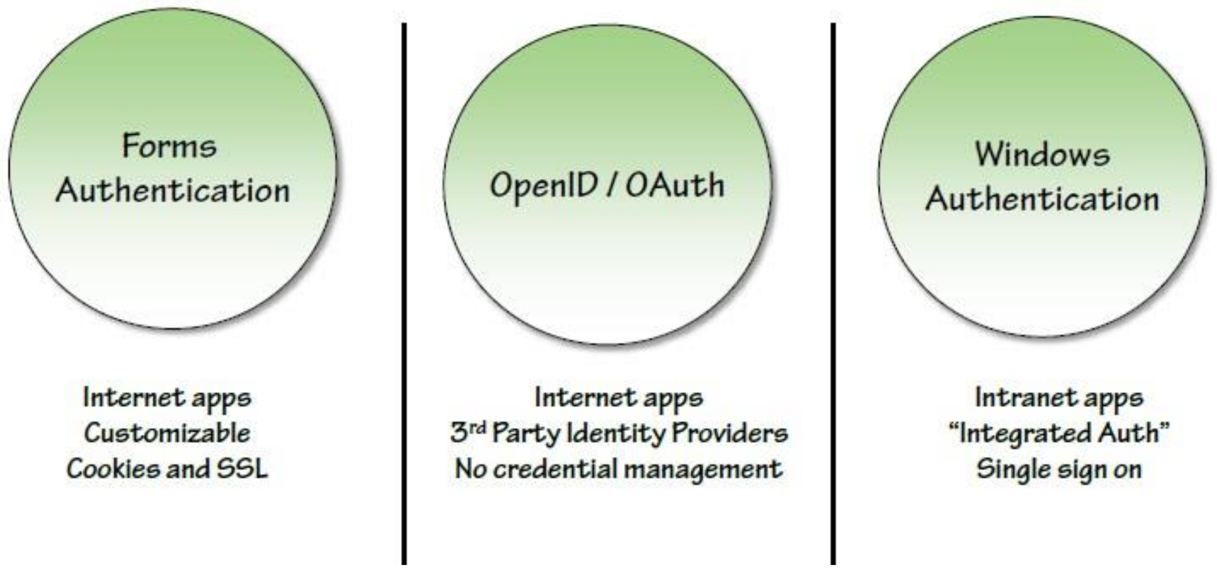
```
public class RestaurantReview : IValidatableObject
{
    public int Id { get; set; }

    [Range(1,10)]
    [Required]
    public int Rating { get; set; }

    [Required]
    [StringLength(1024,ErrorMessage="You cannot exceed 1024 character")]
```

```
        public string Body { get; set; }

        [Display(Name="User Name")]
        [DisplayFormat(NullDisplayText="anonymous")]
        public string ReviewerName { get; set; }
        public int RestaurantId { get; set; }

        public IEnumerable<ValidationResult> Validate(ValidationContext
validationContext)
        {
            if (Rating < 2 && ReviewerName.ToLower().StartsWith("Mahedee"))
            {
                yield return new ValidationResult("Sorry, Mahedee, you can't do this");
            }
        }
    }
```

Chapter 15: Security in ASP.NET MVC

Authentication

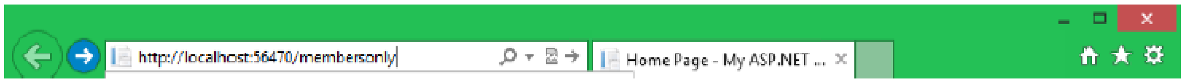
- When you authenticate a user, you are verifying the identity of the user.
- You might need to know a user's identity because you're building an application that only specific users should access, like a payroll system.
- You cannot let just anyone poke around in the salary information.
- So the first step would be identifying the user and making sure you know who they are.



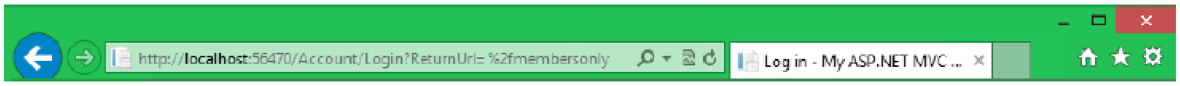
Windows Authentication

Forms Authentication

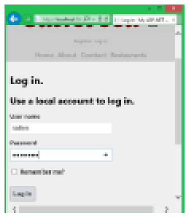
1. User tries to access "members only" page.


http://localhost:56470/membersonly

2. ASP.NET redirects user to login page ("~/Account/Login")

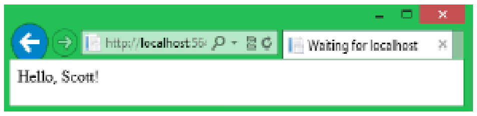

http://localhost:56470/Account/Login?ReturnUrl=%2fmembersonly

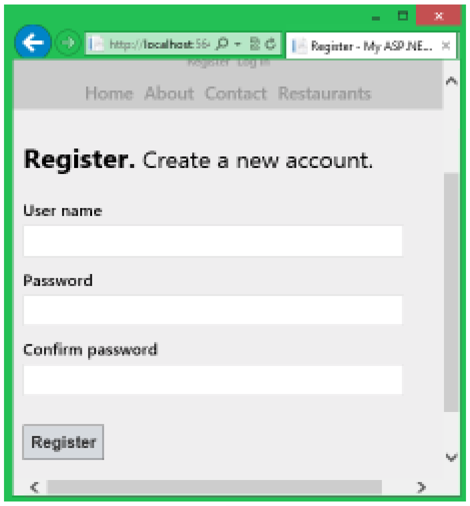
3. User submits username and password


Log in.

Use a local account to log in.

4. ASP.NET sets authentication cookie, redirects back to "members only" page.


Hello, Scott!



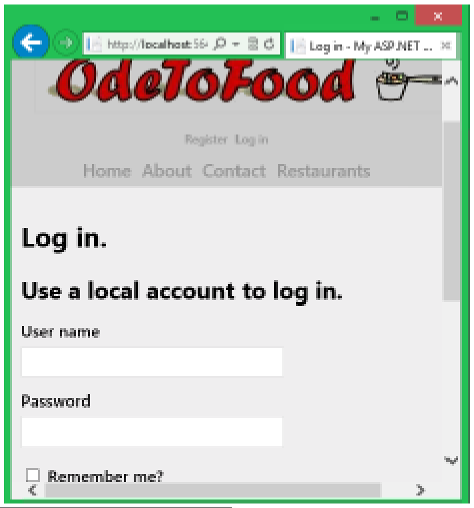
Register. Create a new account.

User name

Password

Confirm password

Register



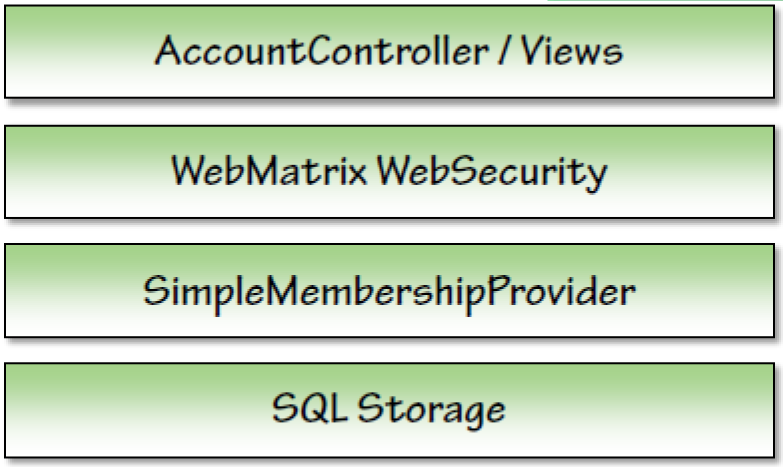
Log in.

Use a local account to log in.

User name

Password

☐ Remember me?



Taking Control of Membership

- Create an ASP.NET MVC Internet application
- In filter folder copy the following line from InitializeSimpleMembershipAttribute class

```
WebSecurity.InitializeDatabaseConnection("DefaultConnection", "UserProfile",
"UserId", "UserName", autoCreateTables: true);
```

- Add the above code to Global.asax

```
        protected void Application_Start()
        {
            WebSecurity.InitializeDatabaseConnection("DefaultConnection", "UserProfile",
"UserId", "UserName", autoCreateTables: true);

            AreaRegistration.RegisterAllAreas();
            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            AuthConfig.RegisterAuth();
        }
```

- Change AccountModel as follows

```
 //public class UsersContext : DbContext
 //{
 //    public UsersContext()
 //        : base("DefaultConnection")
 //    {
 //    }

 //    public DbSet<UserProfile> UserProfiles { get; set; }
 //}

 [Table("UserProfile")]
 public class UserProfile
 {
     [Key]
     [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
     public int UserId { get; set; }
     public string UserName { get; set; }
     public string FavoriteResturant { get; set; }
 }
```

- Add User profile to RestaurantContext

```
 public class RestaurantContext : DbContext
 {
     public RestaurantContext()
         : base("name=DefaultConnection")
     {

     }
     public DbSet<Restaurant> Restaurants {get; set;}
     public DbSet<RestaurantReview> Reviews{get; set;}
     public DbSet<UserProfile> UserProfiles { get; set; }
 }
```

- Change Account Controller as follows

```
[Authorize]
 //[InitializeSimpleMembership]
 public class AccountController : Controller
 {

 //////////////

 }
```

```
if (ModelState.IsValid)
{
// Insert a new user into the database
using (var db = new RestaurantContext())
{

//……………

}

}
```

- Now register and login to the application

Authorization

-
  ```
  [Authorize]
  //[InitializeSimpleMembership]
  public class AccountController : Controller
  {
      //
      // GET: /Account/Login

      [AllowAnonymous]
      public ActionResult Login(string returnUrl)
      {
          ViewBag.ReturnUrl = returnUrl;
          return View();
      }

  }
  ```

-
  ```
  <authentication mode="Forms">
    <forms loginUrl="~/Account/Login" timeout="2880" />
  </authentication>
  ```

- `[Authorize(Users="mahedee,hasan")]`
- `[Authorize(Roles="Admin,Contributor")]`

Seeding Membership

- Add the following code snippet in Configuration Class

```
protected override void Seed(MvcApplication3.Models.RestaurantContext context)
        {
            context.Restaurants.AddOrUpdate(r => r.Name,
                new Restaurant { Name = "Hotel Westing", City = "Dhaka", Country =
"Bangladesh" },
```

```
                new Restaurant { Name = "Hotel Sonargaon", City = "Dhaka", Country =
"Bangladesh" },
                new Restaurant
                {
                    Name = "Hotel Radison",
                    City = "Dhaka",
                    Country = "Bangladesh",
                    Reviews = new List<RestaurantReview>
                    {
                        new RestaurantReview{Rating = 9, Body = "Very good Food!"}
                    }
                });

            SeedMembership();
        }

        private void SeedMembership()
        {
            WebSecurity.InitializeDatabaseConnection("DefaultConnection", "UserProfile",
"UserId", "UserName", autoCreateTables: true);
            var roles = (SimpleRoleProvider)Roles.Provider;
            var membership = (SimpleMembershipProvider)Membership.Provider;

            if (!roles.RoleExists("Admin"))
            {
                roles.CreateRole("Admin");
            }
            if (membership.GetUser("ehsan", false) == null)
            {
                membership.CreateUserAndAccount("ehsan", "leads@123");
            }
            if (!roles.GetRolesForUser("ehsan").Contains("Admin"))
            {
                roles.AddUsersToRoles(new []{"ehsan"}, new []{"Admin"});
            }
               }
```

- Add the following in web.config

```
    <membership defaultProvider="simple">
      <providers>
        <clear/>
        <add name="simple" type="WebMatrix.WebData.SimpleMembershipProvider,
WebMatrix.WebData" />
      </providers>
    </membership>

    <roleManager enabled="true" defaultProvider="simple">
      <providers>
        <clear/>
        <add name="simple" type="WebMatrix.WebData.SimpleRoleProvider, WebMatrix.WebData"
/>
      </providers>
    </roleManager>


  </system.web>
```

- Run following command in Package Manager Console

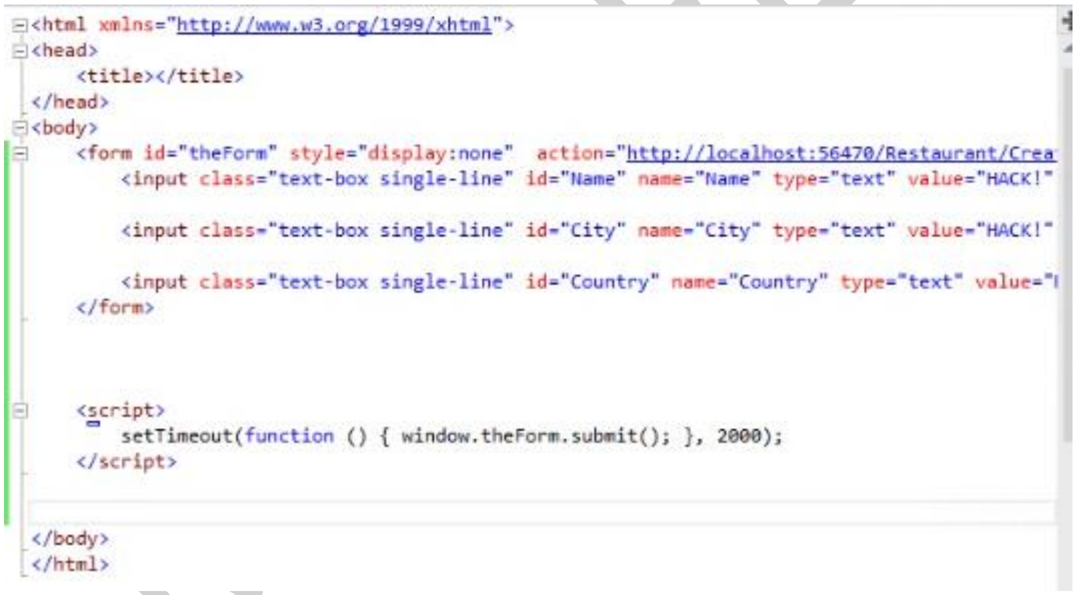
```
PM> Update-Database -Verbose
```

- Authorize like following

```
@if(User.IsInRole("admin"))
{
    @Html.ActionLink("Create New", "Create")
}


[Authorize(Roles="Admin")]
```

Cross site request forgery

- Cross site request forgery known as CSRF
- Demo



```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <form id="theForm" style="display:none"  action="http://localhost:56470/Restaurant/Crea
        <input class="text-box single-line" id="Name" name="Name" type="text" value="HACK!"

        <input class="text-box single-line" id="City" name="City" type="text" value="HACK!"

        <input class="text-box single-line" id="Country" name="Country" type="text" value="
    </form>



    <script>
        setTimeout(function () { window.theForm.submit(); }, 2000);
    </script>

</body>
</html>
```

OpenID and OAuth

- Uncomment the following line in AuthConfig class
  `OAuthWebSecurity.RegisterGoogleClient();`

Go to the login page and Use another service (Google) to log in.

- For more information visit http://go.microsoft.com/fwlink/?LinkID=252166

    - Which is mentioned in AuthConfig class

- References:
    - http://openid.net/
    - http://oauth.net/
    - http://dotnetopenauth.net/
    - https://github.com/dotnetopenauth

Chapter 16: Deployment

Deploying to IIS

- Demo: ASP.NET MVC Application Deployment in IIS

Deploying to Windows Azure

- Demo: ASP.NET MVC Application Deployment in Windows Azure

Introduction to Unit Testing

- One of the goal of ASP.NET MVC framework is testable
- Specifically your controller should be testable
- Demo