Controllers in an mvc application

- A controller is responsible for controlling the way that a user interacts with an MVC application.
- A controller contains the flow control logic for an ASP.NET MVC application.
- A controller determines what response to send back to a user when a user makes a browser request.
- A controller is just a class (for example, a Visual Basic or C# class). The sample ASP.NET MVC application includes a controller named HomeController.cs located in the Controllers folder.

Views in an mvc application

- A **view** is an HTML template with embedded Razor markup.
- Razor markup is code that interacts with HTML markup to produce a web page that's sent to the client.
- In ASP.NET Core MVC, views are .cshtml files that use the **C#** programming language in Razor markup.
- View contains the HTML markup and content that is sent to the browser. A view is the equivalent of a page when working with an ASP.NET MVC application.
- ViewBag & ViewData is a mechanism to pass data from the controller to the view.
- We use **"@"** symbol to switch between html and c# code.

ViewData and ViewBag in mvc

- Both **ViewData** and **ViewBag** are used to pass data from a controller to a view.
- ViewData is a dictionary of objects that are stored and retrieved using strings as keys.
- The syntax of ViewData is very similar to that of ViewState, SessionState and ApplicationState.

```
// Storing data in ViewData
ViewData["YourData"] = "SomeData";

// Retrieving data from ViewData
string strData = ViewData["YourData"].ToString();
```

- ViewData does not provide compile time error checking. For example, if you mis-spell the key names you wouldn't get any compile time error. You get to know about the error only at runtime.
- **ViewBag** uses the dynamic feature that was introduced in to C# 4.0. It allows an object to have properties dynamically added to it.
- Using ViewBag the above code can be rewritten as below.

```
// Storing data in ViewBag
ViewBag.YourData = "SomeData";

// Retrieving data from ViewBag
string strData = ViewBag.YourData;
```

- Just like ViewData, ViewBag does not provide compile time error checking. For example, if you misspell the property name, you wouldn't get any compile time error. You get to know about the error only at runtime.
- Internally ViewBag properties are stored as name/value pairs in the ViewData dictionary.
- **Note:** To pass data from controller to a view, It's always a good practice to use strongly typed view models instead of using ViewBag & ViewData. Strongly typed view models provide compile time error checking.

Models in an mvc application

- The **model** classes represents domain-specific data and business logic in the MVC application.
- It represents the shape of the data as public properties and business logic as methods.
- In the ASP.NET MVC Application, all the **Model** classes must be created in the **Model** folder.
- Models mainly contain properties.
- An MVC model contains all of your application logic that is not contained in a view or a controller.
- The model should contain all of your application business logic, validation logic, and database access logic. For example, if you are using the Microsoft Entity Framework to access your database, then you would create your Entity Framework classes (your .edmx file) in the Models folder.
- A view should contain only logic related to generating the user interface.
- A controller should only contain the bare minimum of logic required to return the right view or redirect the user to another action (flow control). Everything else should be contained in the model.
- Data access in mvc using entity framework

https://csharp-video-tutorials.blogspot.com/2013/05/part-8-data-access-in-mvc-using-entity.html

Generate hyperlinks using actionlink html helper

- @model is set to IEnumerable<MVCDemo.Models.Employee>
- We are using Html.ActionLink html helper to generate links

- For more details:

https://csharp-video-tutorials.blogspot.com/2013/05/part-9-generate-hyperlinks-using.html

Using business objects as model in mvc

- In MVC there are several conventions that needs to be followed.
- For example, controllers need to have the word controller in them and should implement IController interface either directly or indirectly.
- Views should be placed in a specific location that MVC can find them.
- But with models, there are no strict rules**.** Infact **"Models"** folder is optional and they can live anywhere. They can even be present in a separate project.
- For more info

https://csharp-video-tutorials.blogspot.com/2013/05/part-11-using-business-objects-as-model.html

FormCollection in mvc

- **FormCollection** class will automatically receive the posted form values in the controller action method, in **key/value** pairs. **Keys** & **values** can be accessed using key names or index.
- Do we really have to write all the dirty code of retrieving data from **FormCollection** and assign it to the properties of **"employee"** object?

The answer is no. This is the job of the **modelbinder** in MVC.

Mapping asp.net request data to controller action simple parameter types

- **The order of the parameters does not matter**. What matters is the name of the parameter. If the parameter name is different from the form control name, then the form data will not be mapped as expected.
- For more info https://csharp-video-tutorials.blogspot.com/2013/05/part-14-mapping-aspnet-request-data-to.html

Updatemodel function in mvc

- **"ActionName"** is specified as **"Create"** for both of these methods. So, if a **"GET"** request is made to the **"URL - http://localhost/MVCDemo/Employee/Create"** then **"Create_Get()"** controller action method is invoked. On the other hand if a **"POST"** request is made to the same URL, then **"Create_Post()"** controller action method is invoked.
  3. Instead of passing **"Employee"** object as a parameter to **"Create_Post()"** action method, we are creating an instance of an "Employee" object with in the function, and updating it using **"UpdateModel()"** function. **"UpdateModel()"** function inspects all the **HttpRequest** inputs such as posted Form data, QueryString, Cookies and Server variables and populate the employee object.

Difference between updatemodel and tryupdatemodel

- The difference is **UpdateModel**() throws an exception if validation fails, where as **TryUpdateModel**() will never throw an exception.
- The similarity is, both the functions are used to update the Model with the Form values and perform the validations.
- Is it mandatory to use "UpdateModel()" or "Try"UpdateModel()" function to update the Model?
  The answer is NO.
- **Why do we need to explicitly invoke model binding?**
  If you want to limit on what can be bound, explicitly invoking model binding can be very useful.

Including and excluding properties from model binding using bind attribute

- Notice that, we are using **"BIND"** attribute and specifying the properties that we want to include in model binding. Since, **"Name"** property is not specified in the **INCLUDE** list, it will be excluded from model binding.
  ```
  public ActionResult Edit_Post([Bind(Include = "Id, Gender, City, DateOfBirth")] Employee employee)
  ```
- Alternatively, **to exclude** properties from binding, we can specify the **exclude list** using **"Bind"** attribute as shown below.
  ```
  [HttpPost]
  [ActionName("Edit")]
  public ActionResult Edit_Post([Bind(Exclude = "Name")] Employee employee){
      // Rest of the method implementation remains the same
  }
  ```

Including and excluding properties from model binding using interfaces

To include and exclude properties from model binding using interfaces
Step 1: Create an interface **"IEmployee"** as shown below. Notice that this interface, has got only the properties that we want to include in model binding. **"Name"** property is not present. This means, **"Name"** property will be excluded from model binding. Copy and paste this code in **"Employee.cs"** class file in **"BusinessLayer"** project

```
public interface IEmployee
{
    int ID { get; set; }
    string Gender { get; set; }
    string City { get; set; }
    DateTime? DateOfBirth { get; set; }
}
```

Step 2: Make **"Employee"** class inherit from **IEmployee** interface

```
public class Employee : IEmployee
{
    public int ID { get; set; }
    public string Name { get; set; }
    [Required]
    public string Gender { get; set; }
    [Required]
    public string City { get; set; }
    [Required]
    public DateTime? DateOfBirth { get; set; }
}
```

Step 3: Modify **"Edit_Post()"** controller action method that is present in **"EmployeeController.cs"** file, as shown below.

```
[HttpPost]
[ActionName("Edit")]
public ActionResult Edit_Post(int id)
{
    EmployeeBusinessLayer employeeBusinessLayer
= new EmployeeBusinessLayer();
    Employee employee = employeeBusinessLayer.Employees.Single(x => x.ID == id);
    UpdateModel<IEmployee>(employee);

    if (ModelState.IsValid)
    {
        employeeBusinessLayer.SaveEmmployee(employee);
        return RedirectToAction("Index");
    }

    return View(employee);
}
```

Notice that we are explicitly calling the model binder, by calling **UpdateModel**() function passing in our interface **IEmployee**. The model binder will update only the properties that are present in the interface.

So, in short, there are several ways to **include** and **exclude** properties from **Model Binding**. Depending on the architecture and requirements of your project, you may choose the approach that best fit your needs.

Why deleting database records using get request is bad

- Deleting database records using GET request opens a security hole and is not recommended by Microsoft.
- Just imagine what can happen if there is an image tag in a malicious email as shown below. The moment we open the email, the image tries to load and issues a GET request, which would delete the data.

  Also, when search engines index your page, they issue a **GET** request which would delete the data.
- In general **GET** request should be free of any side-effects, meaning it should not change the state.
- Deletes should always be performed using a **POST** request.

Using data transfer object as the model in mvc

- For more info
  https://csharp-video-tutorials.blogspot.com/2013/06/part-29-using-data-transfer-object-as.html

View engines in asp.net mvc

- **Out of the box asp.net offers the following 2 view engines.**
  1. ASPX
  2. Razor
- **What is the difference between RAZOR and ASPX view engines?**
  It mostly, boils down to the syntax. Otherwise there are no major differences between the two. In ASPX view engine, the server side script is wrapped between <% %>, where as in RAZOR we use @. Personally, I prefer using RAZOR views, as it is very easy to switch between HTML and Code.

  Depending on the programming language you have chosen, RAZOR views have the extension of **.CSHTML** or **.VBHTML**, where as ASPX views has the extension of **.ASPX**
- **Is it possible, to have both RAZOR and ASPX views in one application?**
  Yes, when you right click on any controller action method, and select **"Add View"** from the context menu, you will have the option to choose the view engine of your choice from the "Add View" dialog box.
- **Is it possible, to use a third party view engine with asp.net mvc?**
  ASP.NET MVC is designed with **extensibility** in mind. So, it's very easy to include third party view engine as well.
- In addition to the above 2 view engines, there are several **custom view engines** that can be used with asp.net mvc. The following are a few of these custom view engines.
  1. Spark
  2. NHaml
  3. SharpDOM
  4. Brail etc.…

How does a controller find a view in mvc

- **1.** How does asp.net mvc know, which view to use
  2. What locations does asp.net mvc search
- MVC looks for a view with the same name as that of the controller action method in the following locations
  1. Views/Shared
  2. Views/FolderNameMatchingControllerName
- Please note that, the view extension can be any of the following
  a).cshtml
  b).vbhtml
  c).aspx
  d).ascx
- If I have all of the following files in "Views/Employee" folder, then MVC picks up "Index.aspx"
  a) Index.aspx
  b) Index.cshtml
  c) Index.vbhtml
  d) Index.ascx
- If you want to use "Index.cshtml" instead, then specify the full path as shown below.

```
public ActionResult Index()
{
    var employees = db.Employees.Include("Department");
    return View("~/Views/Employee/Index.cshtml", employees.ToList());
}
```

- If you specify only the name of the view along with it's extension as shown below, you will get an error.

```
return View("Index.cshtml", employees.ToList());
```

- If you want to use a view name which is not inside the views folder of the current controller, then specify the full path as shown below.

```
public ActionResult Index()
{
    var employees = db.Employees.Include("Department");
    return View("~/Views/Home/Index.aspx", employees.ToList());
}
```

- **Please note that asp.net mvc**
  1. Is all about convention over configuration
  2. Views folder contains one folder for each controller and a **"Shared"** folder
  3. Shared folder is used to store views shared between controllers. Master and Layout pages are stored in "Shared" folder.

Html helpers in mvc

- What is an HTML helper?
  An HTML helper is a method that is used to render html content in a view. HTML helpers are implemented as extension methods.
- **Example**, to produce the HTML for a textbox with id="firstname" and name="firstname", we can type all the html in the view as shown below

```
<input type="text" name="firtsname" id="firstname" />
```

  OR

We can use the "TextBox" html helper.

```
@Html.TextBox("firstname")
```

- There are several overloaded versions. To set a value, along with the name, use the following overloaded version.

```
@Html.TextBox("firstname", "John")
```

The above html helper, generates the following HTML

```
<input id="firstname" name="firstname" type="text" value="John" />
```

To set **HTML attributes**, use the following overloaded version. Notice that, we are passing HTML attributes (**style** & **title**) as an **anonymous type**.

```
@Html.TextBox("firstname", "John", new { style = "background-color:Red; color:White; font-weight:bold", title="Please enter your first name" })
```

Some of the **html attributes**, are reserved keywords. Examples include **class, readonly** etc. To use these attributes, use **"@"** symbol as shown below.

```
@Html.TextBox("firstname", "John", new { @class = "redtextbox", @readonly="true" })
```

- **To generate a label for "First Name"**

```
@Html.Label("fisrtname", "First Name")
```

- **To generate a textbox to enter password, so that the input is masked**

```
@Html.Password("Password")
```

- **To generate a multi-line textbox with 5 rows and 20 columns**

```
@Html.TextArea("Comments", "", 5, 20, null)
```

- **To generate a hidden textbox**

```
@Html.Hidden("id")
```

- **Hidden textbox** is used to store id values. Id values are not displayed on the page to the end user, but we need them to update data when the form is posted to the server.
- **Is it possible to create our own custom html helpers?**
  Yes.
- **Is it mandatory to use HTML helpers?**
  No, you can type the required HTML, but using HTML helpers will greatly reduce the amount of HTML that we have to write in a view. Views should be as simple as possible. All the complicated logic to generate a control can be encapsulated into the helper, to keep views simple.

Generating a dropdownlist control in mvc using HTML helpers

- To generate a dropdownlist, use **DropDownList** html helper.
- A **dropdownlist in MVC** is a collection of **SelectListItem** objects. Depending on your project requirement you may either **hard code the values in code** or **retrieve them from a database table**.
- **Generating a dropdownlist using hard coded values**: We will use the following overloaded version of "DropDownList" html helper.

```
DropDownList(string name, IEnumerable<SelectListItem>
selectList, string optionLabel)
```

The following code will generate a departments dropdown list. The first item in the list will be **"Select Department"**.

```
@Html.DropDownList("Departments", new List<SelectListItem>
{
   new SelectListItem { Text = "IT", Value = "1", Selected=true},
   new SelectListItem { Text = "HR", Value = "2"},
```

```
    new SelectListItem { Text = "Payroll", Value = "3"}
}, "Select Department")
```

- The downside of hard coding dropdownlist values with-in code is that, if we have to add or remove departments from the dropdownlist, the code needs to be modified.
- In most cases, we get values from the database table. For this example, let's use entity framework to retrieve data, Using **ADO.NET** entity data model.
- **To pass list of Departments from the controller, store them in "ViewBag"**

```
public ActionResult Index()
{
    // Connect to the database
    SampleDBContext db = new SampleDBContext();
    // Retrieve departments, and build SelectList
    ViewBag.Departments = new SelectList(db.Departments, "Id", "Name");

    return View();
}
```

Now in the "Index" view, access Departments list from "ViewBag"

```
@Html.DropDownList("Departments", "Select Department")
```

Difference between Html.TextBox and Html.TextBoxFor

- Html.TextBox amd Html.DropDownList are not strongly typed and hence they doesn't require a strongly typed view. This means that we can hardcode whatever name we want.
- On the other hand, Html.TextBoxFor and Html.DropDownListFor are strongly typed and requires a strongly typed view, and the name is inferred from the lambda expression.
- Strongly typed HTML helpers also provide compile time checking.
- Since, in real time, we mostly use strongly typed views, prefer to use Html.TextBoxFor and Html.DropDownListFor over their counterparts.
- Whether, we use Html.TextBox & Html.DropDownList OR Html.TextBoxFor & Html.DropDownListFor, the end result is the same, that is they produce the same HTML.
- Strongly typed HTML helpers are added in MVC2.

Display Name, Display Format, Scaffold Column attributes

- We can control the display of data in a view using display attributes that are found in System.ComponentModel.DataAnnotations namespace. It is not a good idea, to add display attributes to the properties of auto-generated "Employee" class, as our changes will be lost, if the class is auto-generated again.

Using datatype and displaycolumn attributes in asp.net mvc application

- To understand **DataType** attributeCopy

```
public class EmployeeMetaData
{
    // Display mailto hyperlink
    [DataType(DataType.EmailAddress)]
    public string EmailAddress { get; set; }
```

```
    // Display currency symbol. For country specific currency, set
    // culture using globalization element in web.config.
    // For Great Britain Pound symbol
    // <globalization culture="en-gb"/>
    [DataType(DataType.Currency)]
    public int? Salary { get; set; }

    // Generate a hyperlink
    [DataType(DataType.Url)]
    public string PersonalWebSite { get; set; }

    // Display only Time Part
    // [DataType(DataType.Time)]
    // Display only Date Part
    [DataType(DataType.Date)]
    public DateTime? HireDate { get; set; }
}
```

- **DisplayColumn** attribute is useful, when a class has a property of complex type, and you want to pick one property of this complex object for display purpose. Let's understand this with an example.

  Right click on the **"Models"** folder and add **Company.cs** class file. Copy and paste the following code.

```
public class Company
{
    public Employee CompanyDirector
    {
        get
        {
            SampleDBContext db = new SampleDBContext();
            return db.Employees.Single(x => x.Id == 1);
        }
    }
}
```

  Notice that, this class has **CompanyDirector** property which returns an **Employee** object. Employee is a complex type. Employee object has got several properties. If you want FullName to be used for display purpose, then make the following changes.

  Decorate **"Employee"** partial class in **"Models"** folder, with **DisplayColumn** attribute.

```
[MetadataType(typeof(EmployeeMetaData))]
[DisplayColumn("FullName")]
public partial class Employee
{
}
```

  Change "Details" action method in "Home" controller as shown below.

```
public ActionResult Details(int id)
{
    Company company = new Company();
    return View(company);
}
```

Copy and paste the following code in Details.cshtml view

```
@model MVCDemo.Models.Company

@{
    ViewBag.Title = "Details";
}

@Html.DisplayTextFor(x => x.CompanyDirector)
```

Navigate to localhost/MVCDemo/Home/Details/1 and you should see the FullName of the employee.

Opening a page in new browser window in asp.net mvc application

- **If you want the page to open in a new window,**
  1. Right click on **"Views"** folder, and add **"Shared"** folder.
  2. Right click on **"Shared"** folder, and add **"DisplayTemplates"** folder.
  3. Right click on **DisplayTemplates** folder, and add a view. Set **"Url"** as the name and use Razor view engine.
  4. Copy and paste the following code in Url.cshtml view

```
<a href="@ViewData.Model" target="_blank">@ViewData.Model</a>
```

  That's it. Build the application and click on the link. Notice that, the page, now opens in a new window.
- The downside of this approach is that, from now on all the links, will open in a new window. To overcome this, follow these steps.
  1. Rename **Url.cshtml** to **OpenInNewWindow.cshtml**
  2. Decorate **"PersonalWebSite"** property in **EmployeeMetaData** class with **UIHint** attribute and specify the name of the template to use. In our case, the name of the template is **"OpenInNewWindow"**.

```
public class EmployeeMetaData
{
    [DataType(DataType.Url)]
    [UIHint("OpenInNewWindow")]
    public string PersonalWebSite { get; set; }
}
```

- So, **UIHint** attribute is used to specify the name of the template to use to display the data field.

Hiddeninput and readonly attributes in mvc

- **HiddenInput attribute** is useful when you want to render a property using **input type=hidden**. This attribute is extremely useful, when you don't want the user to see or edit the property, but you need to post the property value to the server when the form is submitted, so the correct record can be updated.
- **HiddenInput** is present in **System.Web.Mvc** namespace.
-
```
// Id property is hidden and cannot be changed
    [HiddenInput(DisplayValue=false)]
    public int Id { get; set; }
```

- **ReadOnly** attribute is present in **System.ComponentModel** namespace. As the name suggests, this attribute is used to make a property readonly. Please note that, we will still be able to change the property value on the view, but once we

post the form the model binder will respect the readonly attribute and will not move the value to the property. You can also, make property of a class **readonly** simply, by removing the SET accessor.

```
// EmailAddress is read only
[ReadOnly(true)]
[DataType(DataType.EmailAddress)]
public string EmailAddress { get; set; }
```

Display and edit templated helpers in asp.net mvc

- **Templated helpers are introduced in mvc 2**. These built in templated helpers can be broadly classified into 2 categories.
  1. Display Templates
  2. Editor Templates
- **There are 3 DISPLAY templated helpers**
  @Html.Display("EmployeeData") - Used with a view that is not strongly typed. For example, if you have stored data in ViewData, then we can use this templated helper using the key that was used to store data in ViewData.
  @Html.DisplayFor(model => model) - Used with strongly typed views. If your model has properties that return complex objects, then this templated helper is very useful.
  @Html.DisplayForModel() - Used with strongly typed views. Walks thru each property, in the model to display the object.
- **Along the same lines, there are 3 EDIT templated helpers**
  @Html.Editor("EmployeeData")
  @Html.EditorFor(model => model)
  @Html.EditorForModel()
- To associate metadata with model class properties, we use attributes. These templated helpers use metadata associated with the model to render the user interface.
- The built-in display and edit templated helpers can be very easily customised.

Customize display and edit templates in asp.net mvc

- **The following is the convention used by MVC to find the customized templates**
  1. The customized display templates must be in a sub-folder that is named - **DisplayTemplates**. Editor templates must be in a sub-folder that is named - **EditorTemplates**.
  2. These sub-folders can live in **"Shared"** folder, or a specific views folder. If these folders are present in the Shared folder, then the templates are available for all the views. If they are in a specific views folder, then, they are available only for that set of views.
  3. The **name of the template** must match the **name of the type**. For example, as we are customizing DateTime template, the name of the template in this case has to be DateTime.ascx or DateTime.cshtml.

Accessing model metadata from custom templated helpers

- To access model metadata in templates, use @ViewData.ModelMetadata. For, example to access the DisplayFormatString, use @ViewData.ModelMetadata.DisplayFormatString. Along the same lines, if you want to know, the name of the containing class(i.e the class that contains

the HireDate property) then
use @ViewData.ModelMetadata.ContainerType.ToString().

Html encoding in asp.net mvc

- **What is HTML encoding?**
  HTML encoding is the process of replacing ASCII characters with their 'HTML Entity' equivalents. For example replacing

ASCII characters	HTML Encoded
<	&lt;
>	&gt;
&	&amp;
Single Quote (')	&#039;
Double Quote (")	&quot;

- **Why would you html encode?**
  To avoid cross site scripting attacks, all output is automatically html encoded in mvc.
- **Avoiding html encoding in razor views:**
  Sometimes, we have to avoid HTML encoding. There are 2 ways to disable html encoding
  1. @Html.Raw("YourHTMLString")
  2. Strings of type IHtmlString are not encoded
- **@Html.Raw()** method can also be used to avoid automatic html encoding.
  Notice that, the string that is returned by Image() method is passed as the input for Raw() method, which renders the image as expected.
  @Html.Raw(Html.Image(@Model.Photo, @Model.AlternateText))
- **Avoiding html encoding in ASPX views:**
  <%: %> syntax will automatically encode html in aspx views.
- So, the following will encode and display the html, instead of rendering the image. At the moment, the custom Image() html helper method is returning string of type system.string.
- If you make this method return IHtmlString, then the following code will render the image instead of html encoding it.
  <%: Html.Image(Model.Photo, Model.AlternateText) %>

  To avoid automatic html encoding, you can use
  1. <%= %>
  2. Html.Raw()
  3. Strings of type IHtmlString are not encoded

  Both the following code blocks will render the image
  <%= Html.Image(Model.Photo, Model.AlternateText) %>
  OR
  <%: Html.Raw(Html.Image(Model.Photo, Model.AlternateText)) %>

Different techniques to avoid automatic html encoding in MVC

Technique	Razor Views	ASPX Views
IHTML Strings	✔	✔
HTML.Raw()	✔	✔
<% = %>	✘	✔

Detect errors in views at compile time

The following code will display employee's **FullName** and **Gender**. Here we are working with a strongly typed view. **Employee** is the model class for this view. This class has got **"FullName"** and **"Gender"** properties.

```
@model MVCDemo.Models.Employee
<fieldset>
    <legend>Employee</legend>

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

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

For example, if you **mis-spell FullName property** as shown below, and when you compile the project, you wouldn't get any compile time errors.

```
@Html.DisplayNameFor(model => model.FullName1)
```

You will only come to know, about the error when the page crashes at run-time. If you want to enable compile time error checking for views in MVC

1. Open MVC project file using a notepad. Project files have the extension of **.csproj** or **.vbproj**

2. Search for **MvcBuildViews** under **PropertyGroup**. **MvcBuildViews** is **false** by default. Turn this to **true** as shown below.

```
<MvcBuildViews>true</MvcBuildViews>
```

3. Save the changes.

If you now build the project, you should get compile time error.

Please Note: Pre-compiling views is different from compile-time error checking.

Advantages of using strongly typed views

- There are several ways available to pass data **from a controller to a view** in an mvc application.
  1. ViewBag or ViewData
  2. Dynamic type
  3. Strongly typed view
- **The following are the advantages of using strongly typed views**. We get
  1. Intellisense and
  2. Compile-time error checking
- With ViewBag and Dynamic type, we don't have these advantages.

Partial views in mvc

- If you are an asp.net web-forms developer, then you will realize that partial views in mvc are similar to user controls in asp.net webforms.
- Partial views are used to **encapsulate re-usable view logic** and are a great means to simplify the complexity of views. These partial views can then be used on multiple views, where we need similar view logic.
- If you are using web forms view engine, then the partial views have the extension of **.ascx**. If the view engine is razor and programming language is c#, then partial views have the extension of **.cshtml**. On the other hand if the programming language is visual basic, then the extension is .vbhtml.

Language	Razor Views	ASPX Views
C#	.cshtml	.ascx
Visual Basic	.vbhtml	.ascx

- Please note that, partial views can be added to **"Shared"** folder or to a **specific views folder**. Partial views that are in the "Shared" folder are available for all the views in the entire project, where as partial views in a specific folder are available only for the views with-in that folder.
- There are several overloaded versions of this method. We are using a version that expects **2 parameters**, i.e the name of the partial view and the model object.

```
@model IEnumerable<MVCDemo.Models.Employee>
@foreach (var item in Model)
{
    @Html.Partial("_Employee", item)
}
```

Difference between html.partial and html.renderpartial

- **Differences:**
  1. The return type of **"RenderPartial"** is **void**, where as **"Partial"** returns **"MvcHtmlString"**

  2. Syntax for invoking **Partial**() and **RenderPartial**() methods in Razor views

```
@Html.Partial("PartialViewName")
{ Html.RenderPartial("PartialViewName");  }
```

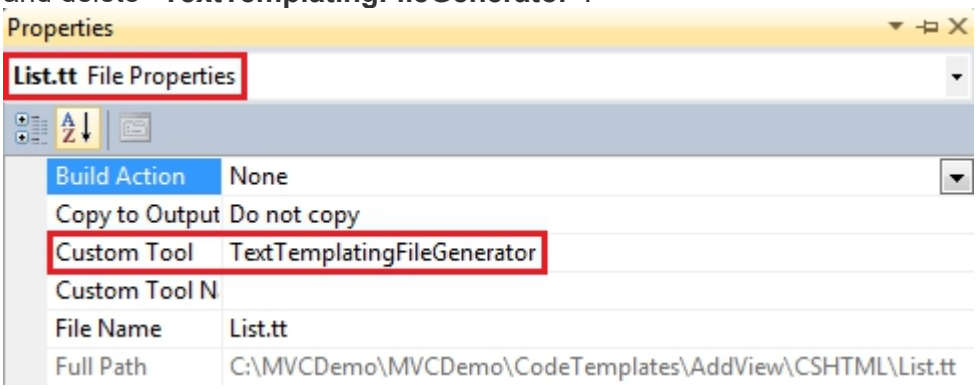
  3. Syntax for invoking **Partial**() and **RenderPartial**() methods in webform views

```
<%: Html.Partial("PartialViewName") %>
<% Html.RenderPartial("PartialViewName"); %>
```

- **When would you use Partial() over RenderPartial() and vice versa?**
  The main difference is that **"RenderPartial()"** returns void and the output will be written directly to the output stream, where as the **"Partial()"** method returns **MvcHtmlString**, which can be assigned to a variable and manipulate it if required. So, when there is a need to assign the output to a variable for manipulating it, then use **Partial**(), else use **RenderPartial**().
- **Which one is better for performance?**
  From a performance perspective, rendering directly to the output stream is better. **RenderPartial**() does exactly the same thing and **is better for performance** over **Partial**().

T4 templates in asp.net mvc

- **What are T4 templates and their purpose?**
  T4 stands for **Text Template Transformation Toolkit** and are used by visual studio to generate code when you add a view or a controller.
- **Where does these T4 templates live?**
  C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplates\[CSharp | FSharp | VisualBasic]\Web\[MVC 2 | MVC 3 | MVC 4]\CodeTemplates
- **What is the file extension for T4 templates?**
  .tt
- **Is it possible to customize T4 templates?**
  Yes, T4 templates can be customized in place, if you want the customized T4 templates available for all MVC projects on that machine.
- If you want to customize T4 templates, only for a specific project, then copy **"CodeTemplates"** folder and paste it in the root directory of your MVC project. Right click on the template files(with .tt extension), and select Properties and delete **"TextTemplatingFileGenerator"**.



  By deleting this from **CustomTool** property, we are telling visual studio not to run them during the build. They will be manually called when we add a view or a controller using **"Add View"** and **"Add Controller"** dialog box.
- Is it possible to add your own T4 template to the existing list of templates?
  Absolutely, simply create a file with **".tt"** file extension in **"AddController"** folder in **"CodeTemplates"**. If it is for adding a view, then put it in **"AspxCSharp"**(if view engine is aspx) or **"CSHTML"**(if view engine is razor) folder.

Cross site scripting attack

- **Cross-site scripting attack** also called as **XSS attack**, is a security vulnerability found in Web applications. **XSS** allows hackers to inject client-side script into

Web pages, and later, if that web page is viewed by others, the stored script gets executed.

- The consequences of XSS may range from a petty nuisance like displaying an alert() box to a significant security risk, like stealing session cookies.
- The following 2 changes that we have done has opened the doors for XSS.
  1. By allowing HTML to be submitted on Create.cshtml view
  2. By disabling HTML encoding on Index.cshtml
- **To initiate a cross site scripting attack**
  1. Navigate to "Create" view
  2. Type the following javascript code in "Comments" textbox
  <script type="text/javascript">
  alert('Your site is hacked');
  </script>
  3. Click "Create"
- Refresh Index view and notice that javascript alert() box is displayed, stating that the site is hacked. By injecting script, it is also very easy to steal session cookies. These stolen session cookies can then be used to log into the site and do destructive things.
- So, in short, by allowing HTML to be submitted and disabling HTML encoding we are opening doors for XSS attack.

How to prevent cross site scripting attack

- To achieve this let's filter the user input, and accept only and <u></u> tags. The following code,
  1. Disables input validation
  2. Encodes all the input that is coming from the user
  3. Finally we selectively replace, the encoded html with the HTML elements that we want to allow.

```
[HttpPost]
// Input validation is disabled, so the users can submit HTML
[ValidateInput(false)]
public ActionResult Create(Comment comment)
{
    StringBuilder sbComments = new StringBuilder();

    // Encode the text that is coming from comments textbox
    sbComments.Append(HttpUtility.HtmlEncode(comment.Comments));

    // Only decode bold and underline tags
    sbComments.Replace("&lt;b&gt;", "<b>");
    sbComments.Replace("&lt;/b&gt;", "</b>");
    sbComments.Replace("&lt;u&gt;", "<u>");
    sbComments.Replace("&lt;/u&gt;", "</u>");
    comment.Comments = sbComments.ToString();

    // HTML encode the text that is coming from name textbox
    string strEncodedName = HttpUtility.HtmlEncode(comment.Name);
    comment.Name = strEncodedName;

    if (ModelState.IsValid)
```

```
    {
        db.Comments.AddObject(comment);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(comment);
}
```

Warning: Relying on just filtering the user input, cannot guarantee XSS elimination. XSS can happen in different ways and forms. This is just one example.

Razor views in mvc

- Use @ symbol to switch between c# code and html.

```
@for (int i = 1; i <= 10; i++)
{
    <b>@i</b>
}
```

Output:
1 2 3 4 5 6 7 8 9 10

- Use @{ } to define a code block. If we want to define some variables and perform calculations, then use code block. The following code block defines 2 variables and computes the sum of first 10 even and odd numbers.

```
@{
    int SumOfEvenNumbers = 0;
    int SumOfOddNumbers = 0;

    for(int i =1; i<=10; i++)
    {
        if(i %2 == 0)
        {
            SumOfEvenNumbers = SumOfEvenNumbers + i;
        }
        else
        {
            SumOfOddNumbers = SumOfOddNumbers + i;
        }
    }
}

<h3>Sum of Even Numbers = @SumOfEvenNumbers</h3>
<h3>Sum of Odd Numbers = @SumOfOddNumbers</h3>
```

Output:
Sum of Even Numbers = 30
Sum of Odd Numbers = 25

- Use <text> element or @: to switch between c# code and literal text

```
@for (int i = 1; i <= 10; i++)
{
    <b>@i</b>
    if (i % 2 == 0)
```

```
    {
        <text> - Even </text>
    }
    else
    {
        <text> - Odd </text>
    }
    <br />
}
```

The above program can be re-written using @: as shown below.

```
@for (int i = 1; i <= 10; i++)
{
    <b>@i</b>
    if (i % 2 == 0)
    {
        @: - Even
    }
    else
    {
        @: - Odd
    }
    <br />
}
```

Output:

1 - Odd
2 - Even
3 - Odd
4 - Even
5 - Odd
6 - Even
7 - Odd
8 - Even
9 - Odd
10 - Even

- **Use @* *@ to comment in razor views**

```
@*This is a comment
in razor views*@
```

- **Transition between c# expressions and literal text**

```
@{
    int day = 31;
    int month = 12;
    int year = 2013;
}

Date is @day-@month-@year
```

Output:

Date is 31-12-2013

- **Using explicit code nugget**

```
@for (int i = 1; i <= 5; i++)
{
    <img src="~/Images/@(i).png" />
}
```

The above code generates the following HTML

```
<img src="/MVCDemo/Images/1.png" />
<img src="/MVCDemo/Images/2.png" />
<img src="/MVCDemo/Images/3.png" />
<img src="/MVCDemo/Images/4.png" />
<img src="/MVCDemo/Images/5.png" />
```

Output:

- @ symbol is used as code delimiter in razor views. However, razor is smart enough to recognize the format of internet email address and not to treat the @ symbol as a code delimiter.

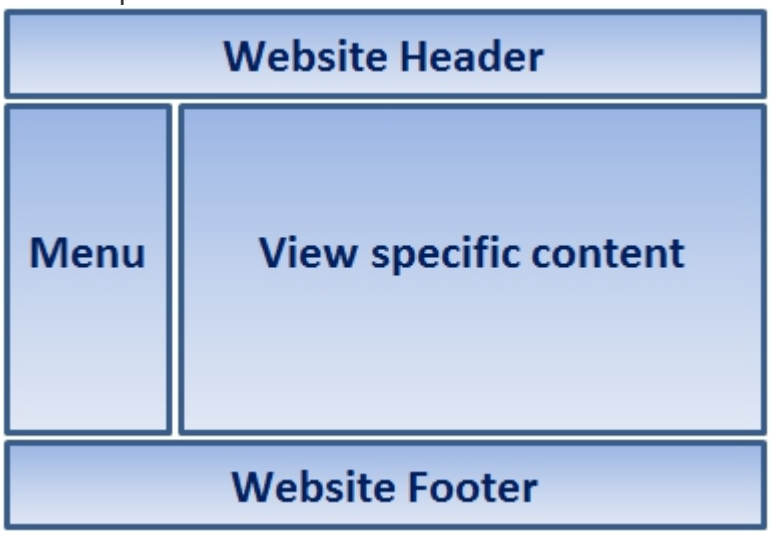
```
This is my email address<br />
<b>kudvenkat@gmail.com</b>

Use @ symbol to escape @
I will meet you @@ office
```

Output:

I will meet you @ office

Layout view in mvc

- **What is the advantage of using _Layout.cshtml view?**
  Layout views provide the advantage of maintaining consistent look and feel across all the views in an mvc application. A typical layout view consists of
  1. Header
  2. Footer
  3. Navigation menu
  4. View specific content



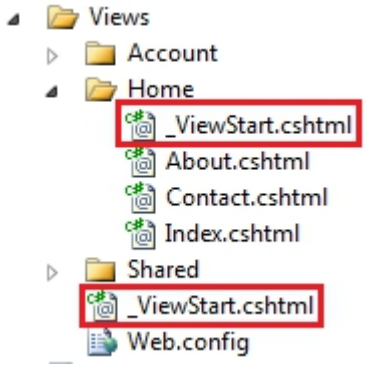
- Rather than having all of these sections, in each and every view, we can define them in a layout view and then inherit that look and feel in all the views. With layout views, maintaining the consistent look and feel across all the views becomes much easier, as we have only one layout file to modify, should there be

any change. The change will then be immediately reflected across all the views in entire application

- **Points to note:**
  1. View specific title is retrieved using @ViewBag.Title.
  2. View specific content will be plugged-in at the location, where RenderBody() function is called.

ViewStart in asp.net mvc

- to associate a view with a layout file, we have to set Layout property on each and every view. This violates **DRY** (Don't Repeat Yourself) principle, and has the following disadvantages
  1. Redundant code.
  2. Maintenance overhead. To use a different layout file, all the views need to be updated.
- **What is _ViewStart.cshtml file?**
  ASP.NET MVC 3, has introduced _ViewStart.cshtml. Specify the Layout property in this file and place it in the Views folder. All the views will then use the layout file that is specified in _ViewStart.cshtml. This eliminates the need to specify Layout property on each and every view, and making them more cleaner and maintainable.



  If you want a set of views in a specific folder, to use a different layout file, then you can include another _ViewStart.cshtml file in that specific folder.
- **When I use _ViewStart.cshtml file, can I still set Layout property on individual views?**
  Yes, if you want to use a layout file that is different from what is specified in _ViewStart.cshtml
- **Where else can I specify a layout file?**
  Layout file can also be specified in a controller action method or in an action filter.
- **In Controller Action Method:**
  Specify which layout to use when returning a view inside a controller action

```
public ActionResult Create()
{
    return View("Create", "_Layout");
}
```

- **Can we write some logic in "_ViewStart.cshtml" to dynamically specify which layout file to use?**
  Yes, the following code will change the layout file to use based on the browser type.
  If the browser is google chrome,
  then "_Layout.cshtml" layout file is used
  Else
  "_DifferentLayout.cshtml" layout file is used

Code in "_ViewStart.cshtml" file

```
@{
    Layout =
Request.Browser.IsBrowser("Chrome") ? "~/Views/Shared/_Layout.cshtml" : "~/
Views/Shared/_DifferentLayout.cshtml" ;
}
```

- **All partial views in my application are now using the layout file specified in "_ViewStart.cshtml". How do I prevent these partial views from using a layout file?**

  Details action method below, returns "_Employee" partial view, and is using the layout file specified in "_ViewStart.cshtml"

```
public ActionResult Details(int id)
{
    Employee employee = db.Employees.Single(e => e.Id == id);
    return View("_Employee", employee);
}
```

  To prevent this partial view from using the layout file, specified in **"_ViewStart.cshtml"**, return **"PartialViewResult"** from the controller action method as shown below.

```
public PartialViewResult Details(int id)
{
    Employee employee = db.Employees.Single(e => e.Id == id);
    return PartialView("_Employee", employee);
}
```

- **What will be the layout file extension, if VB.NET is my programming language?**

  .vbhtml

Action selectors in mvc

- Actions are public methods in an mvc controller that responds to an URL request. You can control or influence which action method gets invoked using action selectors in mvc. Action selectors are attributes that can be applied to an action method in a controller.
- **ActionName selector:** This action selector is used when you want to invoke an action method with a different name, than what is already given to the action method.
- Then decorate the action method with ActionName attribute as shown below.

```
public class HomeController : Controller
{
    [ActionName("List")]
    public string Index()
    {
        return "Index action method invoked";
    }
}
```

- **AcceptVerbs selector:** Use this selector, when you want to control, the invocation of an action method based on the request type. In the example below, the "Edit" method that is decorated with GET acceptverb responds to the GET request, where as the other "Edit" method responds to POST request. The default is GET. So, if you don't decorate an action method with any accept verb, then, by default, the method responds to GET request.

```
public class HomeController : Controller
{
    [AcceptVerbs(HttpVerbs.Get)]
    public ActionResult Edit(int id)
    {
        Employee employee = GetEmployeeFromDB(id);
        return View(employee);
    }

    [AcceptVerbs(HttpVerbs.Post)]
    public ActionResult Save(Employee employee)
    {
        if (ModelState.IsValid)
        {
            // Save employee to the database
            return RedirectToAction("Index");
        }
        return View(employee);
    }
}
```

- **HttpGet** and **HttpPost** attributes can be used as shown below. This is an alternative to using AcceptVerbs attribute.

```
public class HomeController : Controller
{
    [HttpGet]
    public ActionResult Edit(int id)
    {
        Employee employee = GetEmployeeFromDB(id);
        return View(employee);
    }

    [HttpPost]
    public ActionResult Save(Employee employee)
    {
        if (ModelState.IsValid)
        {
            // Save employee to the database
            return RedirectToAction("Index");
        }
        return View(employee);
    }
}
```

- What is the use of NonAction attribute in MVC?
  OR
  How do you restrict access to public methods in a controller?

An action method is a public method in a controller that can be invoked using a URL. So, by default, if you have any public method in a controller then it can be invoked using a URL request. To restrict access to public methods in a controller, NonAction attribute can be used. Let's understand this with an example.

We have 2 public methods in HomeController, Method1() and Method2(). Method1 can be invoked using URL **/Home/Method1**

Method2 can be invoked using URL **/Home/Method2**

```
public class HomeController : Controller
{
    public string Method1()
    {
        return "<h1>Method 1 Invoked</h1>";
    }

    public string Method2()
    {
        return "<h1>Method 2 Invoked</h1>";
    }
}
```

Let's say **Method2**() is for doing some internal work, and we don't want it to be invoked using a URL request. To achieve this, decorate **Method2**() with **NonAction** attribute.

```
[NonAction]
public string Method2()
{
    return "<h1>Method 2 Invoked</h1>";
}
```

Now, if you naviage to URL **/Home/Method2**, you will get an error - The resource cannot be found.

- **Another way to restrict access to methods in a controller, is by making them private.**

  ```
  private string Method2()
  {
      return "<h1>Method 2 Invoked</h1>";
  }
  ```

  In general, it's a bad design to have a public method in a controller that is not an action method. If you have any such method for performing business calculations, it should be somewhere in the model and not in the controller.

  However, if for some reason, if you want to have public methods in a controller and you don't want to treat them as actions, then use NonAction attribute.

Action filters in mvc

- **What are action filters in asp.net mvc?**
  Action filters are attributes that can be applied either on a controller action method or on a controller. When applied at the controller level, they are applicable for all actions within that controller. Action filters allow us to add, pre and post processing logic to an action method. This means, they allow us to modify the way in which an action is executed.
- **Name a few action filters in mvc?**
  Authorize
  ChildActionOnly
  HandleError
  OutputCache
  RequireHttps

ValidateInput
ValidateAntiForgeryToken

- **Can you create a custom action filter in mvc?**
  Yes

Authorize and AllowAnonymous action filters in mvc

- In ASP.NET MVC, by default, all the controller action methods are accessible to both **anonymous** and **authenticated** users. If you want action methods, to be available only for authenticated and authorised users, then use Authorize attribute.

childactiononly attribute in mvc

- **1.** Any action method that is decorated with [ChildActionOnly] attribute is a child action method.

  2. Child action methods will not respond to URL requests. If an attempt is made, a runtime error will be thrown stating - Child action is accessible only by a child request.

  3. Child action methods can be invoked by making child request from a view using "Action()" and "RenderAction()" html helpers.

  4. An action method doesn't need to have [ChildActionOnly] attribute to be used as a child action, but use this attribute to prevent if you want to prevent the action method from being invoked as a result of a user request.

  5. Child actions are typically associated with partial views, although this is not compulsory.

  6. Child action methods are different from NonAction methods, in that NonAction methods cannot be invoked using Action() or RenderAction() helpers. We discussed NonAction methods in Part 70 of ASP.NET MVC tutorial series.

  7. Using child action methods, it is possible to cache portions of a view. This is the main advantage of child action methods. We will cover this when we discuss [OutputCache] attribute.

HandleError attribute in mvc

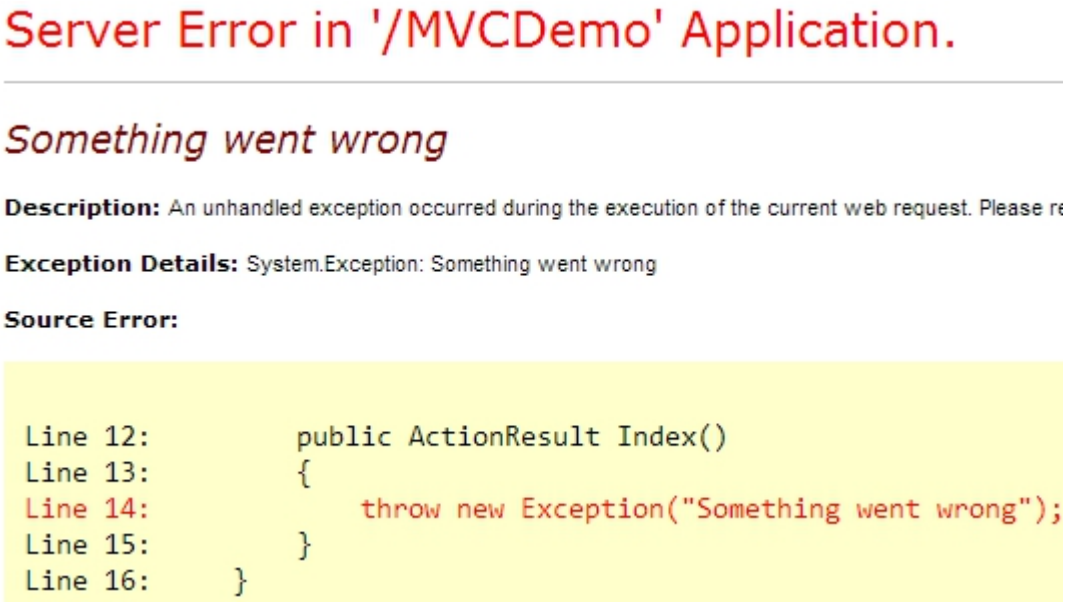
- HandleErrorAttribute is used to display friendly error pages to end user when there is an unhandled exception. Let us understand this with an example.

  Step 1: Create a blank asp.net mvc 4 application.

  Step 2: Add a HomeController. Copy and paste the following code.

```
public ActionResult Index()
{
    throw new Exception("Something went wrong");
}
```

Notice that, the **Index**() action method throws an exception. As this exception is not handled, when you run the application, you will get the default **"yellow screen of death"** which does not make sense to the end user.



Now, let us understand replacing this yellow screen of death, with a friendly error page.

Step 3: Enable custom errors in web.config file, that is present in the root directory of your mvc application. "customErrors" element must be nested under "<system.web>". For detailed explanation on MODE attribute, please watch Part 71 of ASP.NET Tutorial.

```
<customErrors mode="On">
</customErrors>
```

Step 4: Add **"Shared"** folder under **"Views"** folder. Add **Error.cshtml** view inside this folder. Paste the following HTML in Error.cdhtml view.

```
<h2>An unknown problem has occured, please contact Admin</h2>
```

Run the application, and notice that, you are redirected to the friendly **"Error"** view, instead of the generic **"Yellow screen of death"**.

We did not apply HandleError attribute anywhere. So how did all this work?
HandleErrorAttribute is added to the GlobalFilters collection in global.asax. When a filter is added to the GlobalFilters collection, then it is applicable for all controllers and their action methods in the entire application.

Right click on **"RegisterGlobalFilters()"** method in Global.asax, and select **"Go To Definition"** and you can find the code that adds **"HandleErrorAttribute"** to GlobalFilterCollection.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
}
```

Is the friendly error page displayed for HTTP status code 404?
No, but there is a way to display the friendly error page.

In the HomeController, we do not have List() action method. So, if a user navigates to **/Home/List**, we get an error - The resource cannot be found. HTTP 404.

To display a friendly error page in this case
Step 1: Add **"ErrorController"** to controllers folder. Copy and paste the following code.

```
public class ErrorController : Controller
{
    public ActionResult NotFound()
    {
        return View();
    }
}
```

Step 2: Right click on **"Shared"** folder and add **"NotFound.cshtml"** view. Copy and paste the following code.

```
<h2>Please check the URL. The page you are looking for cannot be found</h2>
```

Step 3: Change **"customErrors"** element in web.config as shown below.

```
<customErrors mode="On">
  <error statusCode="404" redirect="~/Error/NotFound"/>
</customErrors>
```

OutputCache attribute in mvc

- **OutputCache** attribute and partial caching in asp.net mvc. **OutputCacheAttribute** is used to cache the content returned by a controller action method, so that, the same content does not need to be generated each and every time the same controller action is invoked.

CacheProfiles in mvc

To cache the data returned by **Index**() action method, for 60 seconds, we would use **[OutputCache]** attribute as shown below.

```
[OutputCache(Duration=60)]
public ActionResult Index()
{
    return View(db.Employees.ToList());
}
```

In the example above, the **OutputCache** settings are specified in code. The disadvantage of this approcah is that
1. If you have to apply the same cache settings, to several methods, then the code needs to be duplicated.
2. Later, if we have to change these cache settings, then we need to change them at several places. Maintaining the code becomes complicated. Also, changing the application code requires build and re-deployment.

To overcome these disadvantages, cache settings can be specified in web.config file using cache profiles.
Step 1: Specify cache settings in web.config using cache profiles

```
<system.web>
  <caching>
    <outputCacheSettings>
      <outputCacheProfiles>
        <clear/>
        <add name="1MinuteCache" duration="60" varyByParam="none"/>
      </outputCacheProfiles>
    </outputCacheSettings>
  </caching>
</system.web>
```

Step 2: Reference the cache profile in application code

```
[OutputCache(CacheProfile = "1MinuteCache")]
public ActionResult Index()
{
    return View(db.Employees.ToList());
}
```

The cache settings are now read from one central location i.e from the web.config file. The advantage of using cache profiles is that
1. You have one place to change the cache settings. Mantainability is much easier.
2. Since the changes are done in web.config, we need not build and redeploy the application.

Using Cache profiles with child action methods

```
[ChildActionOnly]
[OutputCache(CacheProfile = "1MinuteCache")]
public string GetEmployeeCount()
{
    return "Employee Count = " + db.Employees.Count().ToString()
        + "@ " + DateTime.Now.ToString();
}
```

When Cache profiles are used with child action methods, you will get an error - Duration must be a positive number.

There colud be several ways to make cache profiles work with child action methods. The following is the approach, that I am aware of. Please feel free to leave a comment, if you know of a better way of doing this.

Create a **custom OutputCache attribute**, that loads the cache settings from the cache profile in web.config.
Step 1: Right click on the project name in solution explorer, and add a folder with name = Common

Setp 2: Right click on **"Common"** folder and add a class file, with name = PartialCacheAttribute.cs

Step 3: Copy and paste the following code. Notice that, I have named the custom **OutputCache** attribute as **PartialCacheAttribute**.

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Configuration;

namespace MVCDemo.Common
{
   public class PartialCacheAttribute : OutputCacheAttribute
   {
      public PartialCacheAttribute(string cacheProfileName)
      {
         OutputCacheSettingsSection cacheSettings
=              (OutputCacheSettingsSection)WebConfigurationManager.GetSection("sy
stem.web/caching/outputCacheSettings");
         OutputCacheProfile cacheProfile =
cacheSettings.OutputCacheProfiles[cacheProfileName];
         Duration = cacheProfile.Duration;
         VaryByParam = cacheProfile.VaryByParam;
         VaryByCustom = cacheProfile.VaryByCustom;
      }
   }
}
```

Step 4: Use PartialCacheAttribute on the child action method, and pass it the name of the cache profile in web.config. Please note that, PartialCacheAttribute is in MVCDemo.Common namespace.

```
[ChildActionOnly]
[PartialCache("1MinuteCache")]
public string GetEmployeeCount()
{
   return "Employee Count = " + db.Employees.Count().ToString()
      + "@ " + DateTime.Now.ToString();
}
```

RequireHttps attribute in mvc

- **[RequireHttps]** attribute forces an unsecured HTTP request to be re-sent over HTTPS. Let's understand [RequireHttps] attribute with an example.

  Step 1: Create an asp.net mvc4 application using "Empty" template

  Step 2: Add a HomeController. Copy and paste the Login() action method in the HomeController.

  ```
  [RequireHttps]
  public string Login()
  {
     return "This method should be accessed only using HTTPS protocol";
  }
  ```

- **RequireHttps** attribute can be applied on a controller as well. In this case, it is applicable for all action methods with in that controller.

- Sensitive data such as login credentials, credit card information etc, must always be transmitted using HTTPS. Information transmitted over https is encrypted.

ValidateInput attribute in mvc

- This attribute is used to enable or disable request validation. By default, request validation is enabled in asp.net mvc.

Custom action filters in asp.net mvc

- **Actions are public methods in a controller.**

- Action filters are attributes, that can be applied either on a controller or on a controller action method, which allow us to add pre and post processing logic to the action methods.

- So, in simple terms an action filter allow us to execute some custom code, either, just before an action method is executed or immediately after an action method completes execution.

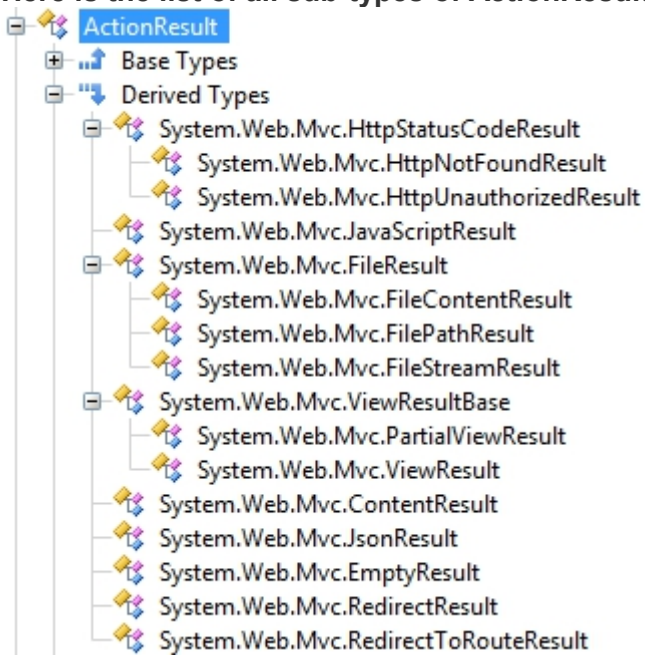
- **There are 4 types of filters in asp.net mvc.**
  1. Authorization filters - Implements IAuthorizationFilter. Examples include AuthorizeAttribute and RequireHttpsAttribute. These filters run before any other filter.
  2. Action filters - Implement IActionFilter
  3. Result filters - Implement IResultFilter. Examples include OutputCacheAttribute.
  4. Exception filters - Implement IExceptionFilter. Examples include HandleErrorAttribute.

Different types of ActionResult in asp.net mvc

-
  The following is the signature of a typical action method in a controller. Notice that, the return type is **ActionResult**. ActionResult is an abstract class and has several sub types.

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

Here is the list of all sub-types of ActionResult.



- **Why do we have so many sub-types?**
  An action method in a controller, can return a wide range of objects. For example, an action method can return
  1. ViewResult
  2. PartialViewResult
  3. JsonResult
  4. RedirectResult etc..

- **What should be the return type of an action method - ActionResult or specific derived type?**
  It's a good practise to return specific sub-types, but, if different paths of the action method returns different subtypes, then I would return an ActionResult object. An example is shown below.

```
public ActionResult Index()
{
   if (Your_Condition)
      return View();          // returns ViewResult object
   else
      return Json("Data");  // returns JsonResult object
}
```

- **The following table lists**
  1. Action Result Sub Types
  2. The purpose of each sub-type

3. The helper methods used to retrun the specific sub-type

Action Result Sub Type	Description	Helper Method
HttpNotFoundResult	Retruns an object to indicate that the requested resource cannot be not found.	**HttpNotFound**
HttpUnauthorizedResult	Represents the result of an unauthorized HTTP request	**None. Return new HttpUnauthorizedResult()**
JavaScriptResult	Returns a piece of JavaScript code that can be executed on the client	**JavaScript**
FileContentResult	Returns a file to the client	**File**
FilePathResult	Returns a file to the client, which is provided by the given path	**File**
FileStreamResult	Returns a file to the client, which is provided by a Stream	**File**
PartialViewResult	Returns a specified partial view	**PartialView**
ViewResult	Returns a specifed view	**View**
ContentResult	Writes content to the response stream without requiring a view	**Content**
JsonResult	Returns a JsonResult which serializes an object in JSON format.	**Json**
EmptyResult	An empty response is returned. Used when the action method returns void.	**None.**
RedirectResult	Performs an HTTP redirection to a specifed new URL	**Redirect**
RedirectToRouteResult	Performs an HTTP redirection to another action method that is determined by the routing engine, based on given route data	**RedirectToAction, RedirectToRoute, RedirectToActionPermanent, RedirectToRoutePermanent**

Areas in asp.net mvc

- When you create a new asp.net mvc application, the following folders are created with in the root directory.
  1. Models
  2. Views
  3. Controllers

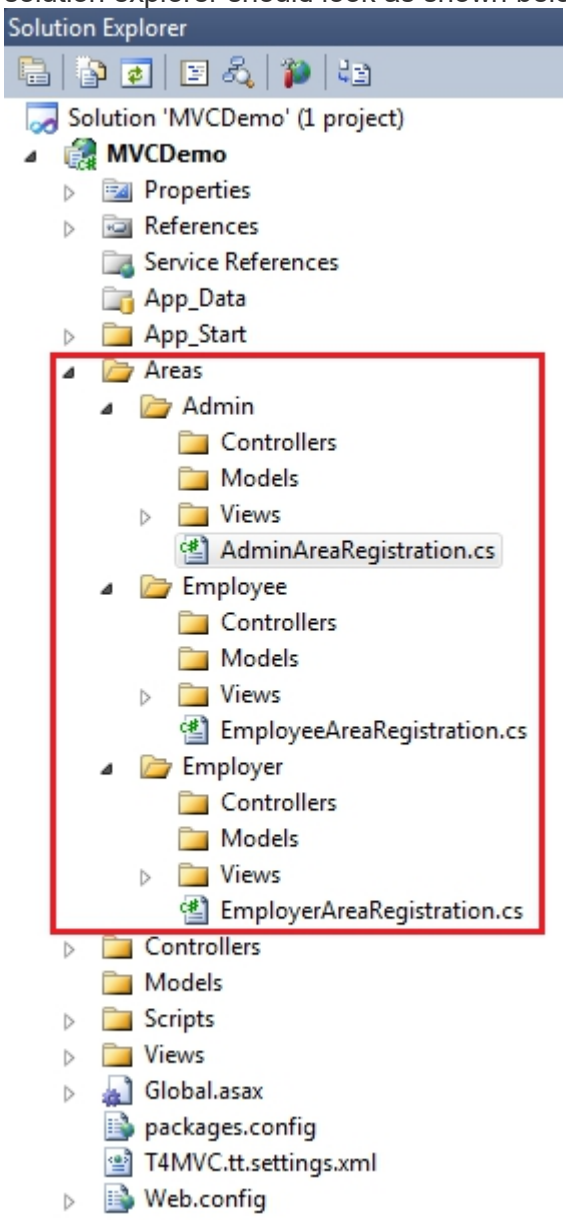
- This structure is fine for simple applications, but when your application gets big and complex, maintaining your Models, Views and Controllers can get complicated.

- The structure of a complex asp.net mvc application can be very easily maintained using areas. So, in short areas are introduced in asp.net mvc 2, that allow us to breakdown a large complex application into a several small sections called areas. These areas can have their own set of
  1. Models
  2. Views
  3. Controllers
  4. Routes

- **To create an area in an MVC application**
  1. Right click on the project name in Solution Explorer and select Add => Area
  2. Provide a meaningful name. For example "Employee" and click Add

  At this point, "Areas" folder will be created, and with in this, a folder for Employee area will be added. You can add as many areas as you want.

In a similar fashion, add areas for Employer and Admin. At this point, your solution explorer should look as shown below. Notice the Areas folder.



Notice that in each area folder (Employee, Employer and Admin), you have a set of Models, Views and Controllers folders. Also, there is **"AreaRegistration.cs"** file, which contains the code to register a route for the area.

StringLength attribute in asp.net mvc

- This attribute is present in **System.ComponentModel.DataAnnotations** namespace and is used to enforce minimum and maximum length of characters that are allowed in a data field.

- **Points to remember:**
  1. [StringLength] attribute is present in System.ComponentModel.DataAnnotations namespace.
  2. [StringLength] attribute verifies that a string is of certain length, but does not enforce that the property is **REQUIRED**. If you want to enforce that the property is required use **[Required]** attribute.

Range attribute in asp.net mvc

- RangeAttribute checks if the value of a data field is within a specified range of values.

- Example

```
public class EmployeeMetaData
{
    [StringLength(10, MinimumLength = 5)]
    [Required]
    public string Name { get; set; }

    [Range(1, 100)]
    public int Age { get; set; }
}
```

- **Range** attribute can also be used to validate **DateTime** fields.

- Example

```
 [Range(typeof(DateTime), "01/01/2000", "01/01/2010")]
    [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]
    public DateTime HireDate { get; set; }
```

RegularExpression attribute in asp.net mvc

- **Regular expression attribute** is great for pattern matching validation.

- **Here is the requirement for validating Name property**
  1. Name can contain first and last name with a single space.
  2. Last name is optional. If last name is not present, then there shouldn't be any space after the first name.
  3. Only upper and lower case alphabets are allowed.

- This requirement can be very easily met using **RegularExpression** attribute. In Employee.cs class file, decorate Name property with RegularExpression attribute.

```
[RegularExpression(@"^(([A-za-z]+[\s]{1}[A-za-z]+)|([A-Za-z]+))$")]
public string Name { get; set; }
```

  Notice that, we are passing regular expression string to the attribute constructor. Regular expressions are great for pattern matching and ensures that, the value for name property is in the format that we want. Also, notice that we are using a verbatim literal(@ symbol) string, as we don't want escape sequences to be processed.

- Understanding and writing regular expressions is beyond the scope of this video. If you are interested in learning to write regular expressions, here is a link from MSDN
  http://msdn.microsoft.com/en-us/library/az24scfc.aspx

  The following website is very helpful, for writing and testing regular expressions. This website also contains commonly used regular expressions. Infact, I have picked up the regular expression for validating Name property from here.
  http://gskinner.com/RegExr/

  Let's discuss another example of using validation attribute. A valid internet email address should have an @ and a DOT symbol in it. To match this pattern, use the following regular expression.
  ^[\w-\._\+%]+@(?:[\w-]+\.)+[\w]{2,6}$

  In **Employee.cs** class file, decorate Email property with **RegularExpression** attribute as shown below.

```
[RegularExpression(@"^[\w-\._\+%]+@(?:[\w-]+\.)+[\w]{2,6}$", ErrorMessage = "Please enter a valid email address")]
public string Email { get; set; }
```

Compare attribute in asp.net mvc

- **Compare attribute is used to compare 2 properties of a model.** Comparing email addresses and passwords is the common use case of Compare attribute.

- Example:

```
public partial class Employee
{
   [Compare("Email")]
   public string ConfirmEmail { get; set; }
}
```

What is Unobtrusive JavaScript

- **Unobtrusive JavaScript**, is a JavaScript that is separated from the web site's html markup. There are several benefits of using Unobtrusive JavaScript. Separation of concerns i.e the HTML markup is now clean without any traces of javascript. Page load time is better. It is also easy to update the code as all the Javascript logic is present in a separate file. We also get, better cache support, as all our JavaScript is now present in a separate file, it can be cached and accessed much faster.

- **Example:**
  We want to change the backgroundColor of **"Save"** button on **"Edit"** view to **"Red"** on MouseOver and to **"Grey"** on MouseOut.

  First let's look at achieving this using obtrusive javascript.
  Step 1: Implement MouseOver() and MouseOut() functions

```
<script type="text/javascript" language="javascript">
    function MouseOver(controlId) {
```

```
        var control = document.getElementById(controlId);
        control.style.backgroundColor = 'red'
    }

    function MouseOut(controlId) {
        var control = document.getElementById(controlId);
        control.style.backgroundColor = '#d3dce0'
    }
</script>
```

Step 2: Associate the javascript functions with the respective events.

```
<input id="btnSubmit" type="submit" value="Save"
    onmouseover="MouseOver('btnSubmit')" onmouseout="MouseOut('btnSubmit')
" />
```

Now let's look at making this javascript unobtrusive, using jQuery

Step 1: Right click on the "Scripts" folder in "Soultion Explorer", and add a jScript file with name = "CustomJavascript.js"

Step 2: Copy and paste the following code in CustomJavascript.js file.

```
$(function () {
    $("#btnSubmit").mouseover(function () {
        $("#btnSubmit").css("background-color", "red");
    });

    $("#btnSubmit").mouseout(function () {
        $("#btnSubmit").css("background-color", "#d3dce0");
    });
});
```

Step 3: Add a reference to CustomJavascript.js file in Edit view.

```
<script src="~/Scripts/CustomJavascript.js" type="text/javascript"></script>
```

Step 4: Remove the following obtrusive Javascript from "Edit" view

```
<script type="text/javascript" language="javascript">
    function MouseOver(controlId) {
        var control = document.getElementById(controlId);
        control.style.backgroundColor = 'red'
    }

    function MouseOut(controlId) {
        var control = document.getElementById(controlId);
        control.style.backgroundColor = '#d3dce0'
    }
</script>
```

Also, remove "onmouseover" and "onmouseout" events from the button control.

```
<input id="btnSubmit" type="submit" value="Save"
    onmouseover="MouseOver('btnSubmit')" onmouseout="MouseOut('btnSubmit')" />
```

- **Client side validation in asp.net mvc is unobtrusive.** To turn on client side validation and unobtrusive JavaScript, make sure the following 2 keys under appSettings element within web.config file are turned on. This will turn on client side validation and unobtrusive JavaScript for the entire application.

```
<appSettings>
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
</appSettings>
```

- **Is it possible to turn these features on/off using code?**
  Yes, the features can be enabled or disabled in Application_Start() event handler in Global.asax as shown below. This will turn on client side validation and unobtrusive JavaScript for the entire application.

```
protected void Application_Start()
{
    HtmlHelper.UnobtrusiveJavaScriptEnabled = true;
    HtmlHelper.ClientValidationEnabled = true;
}
```

- **Is it possible to turn these features on/off for a specific view?**
  Yes, include the following code, on a view where you want to enable/disable these features.

```
@{
    Html.EnableClientValidation(true);
    Html.EnableUnobtrusiveJavaScript(true);
}
```

- **How is Unobtrusive validation implemented in asp.net mvc?**
  Using **"data"** attributes. For example, if we use **"Required"** attribute, on Name property and if we enable client side validation and unobtrusive JavaScript, then the generated HTML for "Name" field is as shown below.

```
<input class="text-box single-line"
      data-val="true"
      data-val-required="The Name field is required."
      id="Name"
      name="Name"
      type="text"
      value="Sara Nan" />
```

data-val=”true”, indicates that the unobtrusive validation is turned on for this element.
data-val-required="The Name field is required.", indicates that the "Name" field is required and the associated error message will be displayed if the validation fails. These data attributes are used by jQuery validation plugin for client side validation.

Ajax with asp.net mvc

- **ASP.NET AJAX enables a Web application to retrieve data from the server asynchronously and to update portions of the existing page.** So these, partial page updates make web application more responsive and hence improves user experience.

- For more info:

https://csharp-video-tutorials.blogspot.com/2013/09/part-92-ajax-with-aspnet-mvc.html

What is Ajax and why should we use it

- **What is AJAX**
  AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML. AJAX enable web applications to retrieve data from the server asynchronously. Web application using AJAX enables partial page updates, ie. only the related section of the page is updated, without reloading the entire page.
- **Advantages of AJAX**
  1. AJAX applications are non blocking. As AJAX requests are asynchronous, the user doesn't have to wait for the request processing to complete. Even while the request is still being processed by the server, the application remains responsive and the user can interact with the application. When the request processing is complete, the user interface is automatically updated. This is not the case with, synchronous requests. The user interface is blocked and the user cannot do anything else until the request has completed processing.

  2. Better performance and reduced network traffic. AJAX enables an application to send and receive only the data that is required. As a result, there is reduced traffic between the client and the server and better peofrmance.

  3. No screen flicker. An AJAX response consists of only the data that is relevant to the request. As a result, only a portion of the page is updated avoiding full page refresh and screen flickers.
- **Disadvantages of AJAX:**
  1. AJAX requests cannot be bookmarked easily
  2. AJAX relies on JavaScript. If JavaScript is disabled, AJAX application won't work.
  3. Harder to debug
  4. Search Engine like Google, Bing, Yahoo etc cannot index AJAX pages.
- **Applications using AJAX**
  1. Many web sites like Google, bing, youtube, yahoo use AJAX to implement AutoComplete feature.
  2. Gmail use AJAX to implement AutoSave feature
  3. Gmail use AJAX to implement RemoteValidation i.e to validate if a user name is already in use, when creating a gmail account.
- **4.** Facebook use AJAX, to load data as you keep scrolling down.
- **AJAX is generally used to implement features like**
  1. AutoComplete
  2. AutoSave
  3. Remote validation etc.

OnBegin, OnComplete, OnSuccess and OnFailure properties of AjaxOptions class

- **Using the following 4 properties of the AjaxOptions class**, we can associate JavaScript functions that get called on the client at different stages of an AJAX request/response cycle.
  1. OnBegin - The associated JavaScript function is called before the action method is invoked
  2. OnComplete - The associated JavaScript function is invoked after the response data has been instantiated but before the page is updated.
  3. OnSuccess - The associated JavaScript function is invoked after the page is

updated.
4. OnFailure - The associated JavaScript function is invoked if the page update fails.

- **Please Note:**
  OnBegin property can also be used to cancel the invocation of the action method. The JavaScript function that is associated with **"OnBegin"** property is invoked before the action method is invoked. So if that associated JavasSript function returns false, then the action method will not be invoked. Your JavaScript function may look something like this.

```
function Validate()
{
   if (condition)
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

LoadingElementDuration property of AjaxOptions class

- **LoadingElementDuration** property is used to control the animaion duration of LoadingElement. The value for LoadingElementDuration property must be specified in milliseconds. By default, the LoadingElement fades in and fades out.
- But I noticed that, irrespective of whatever duration you set, the **LoadingElement** was fading in and out at the same speed. I have done some research on google and came across the following stackoverflow article which explained the fix for the issue.
  http://stackoverflow.com/questions/8928671/asp-net-mvc-3-loadingelementduration-not-working

  To fix the issue, we need to parse the duration to integer.
  Open **"jquery.unobtrusive-ajax.js"** and change the following line
  FROM

```
duration = element.getAttribute("data-ajax-loading-duration") || 0;
```

  TO

```
duration = parseInt(element.getAttribute("data-ajax-loading-duration")) || 0;
```

  After you have made the change, reference "jquery.unobtrusive-ajax.js" file in the **"Index"** view

```
<script src="~/Scripts/jquery.unobtrusive-ajax.js" type="text/javascript"></script>
```

  Finally set, **LoadingElementDuration** and test your application

```
@Ajax.ActionLink("All", "All",
   new AjaxOptions
   {
     HttpMethod = "GET",
     UpdateTargetId = "divStudents",
     InsertionMode = InsertionMode.Replace,
     LoadingElementId = "divloading",
     OnBegin = "ClearResults",
     OnSuccess = "CountRows",
```

```
  LoadingElementDuration = 2000
})
```

What is JavaScript minification

- **What is JavaScript minification?**
  JavaScript minification is the process of reducing the size of JavaScript file, by removing comments, extra white spaces, new line characters and using shorter variable names.
- **What are the advantages of JavaScript minification?**
  As the minified JavaScript files are very small, they will be downloaded much faster and consumes less bandwidth. Search engines like google considers page load time as one of the parameters to rank the pages.
- **Is any of the functionality lost because of minification?**
  No, the functionality will be exactly the same as before.
- **Are there any tools available to minify JavaScript?**
  There are lot of free tools available on the web. Just GOOGLE by using "Minify JavaScript" search term.
- **What is the difference between jquery.js and jquery.min.js?**
  jquery.min.js is the minified version, where as jquery.js is the non-minified version. In your production website always use minified files as they download faster and consumes less bandwidth.
- **What is the downside of JavaScript minification?**
  They are harder to read and debug. However, for development and debugging we can use non-minified versions. Just before deployment, minify and use the minified versions on the production environment.

What is CDN - Content Delivery Network

- CDN stands for Content Delivery Network. CDN is a network of computers that exist all over the world. For a web application there are 2 ways we can make the jQuery library available
- **Benefits or advantages of using a CDN:**
  1. Caching benefits - If other web sites use jquery and if the user has already visited those websites first, jquery file is cached. If the user then visits your website, the cached jquery file is used without having the need to download it again.

  2. Speed benefits - The jquery file from the nearest geographical server will be downloaded.

  3. Reduces network traffic - As the jQuery file is loaded from a CDN, this reduces the network traffic to your web server.

  4. Parallelism benefit - There is a limitation on how many components can be downloaded in parallel. This limitation is per hostname. For example,
  1. If the browser allows only 2 components to be downloaded in parallel per hostname and
  2. If a web page requires 4 components and
  3. If all of them are hosted on a single host and

4. If 2 components take 1 second to download
Then to download all the 4 components, it is going to take 2 seconds.

However, if 2 of the components are hosted on a different host(server), then all the 4 components can be downloaded in parallel and in 1 second.

- **A CDN is an external resource and beyond our control. So, what if, CDN is down? How can we fall back from CDN to use local copy of Jquery.**

  The following code checks if jQuery is loaded and if it isn't, the jquery script will be loaded from our web server.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"
        type="text/javascript">
</script>

<script type="text/javascript">
    window.jQuery || document.write('<script src="/MVCDemo/Scripts/jquery-
1.7.1.min.js">\x3C/script>')
</script>
```

  \x3C is hexadecimal for **<**