

ASP. Net MVC

Module-1

-Anand P.K

Module

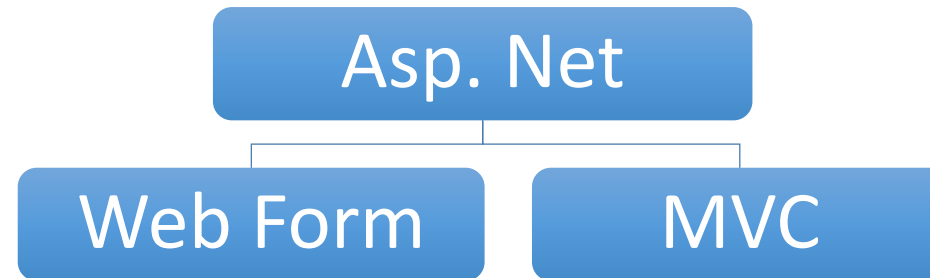
- Introduction to MVC
- Difference between MVC and Asp. Net Forms
- Page Processing in MVC
- Understanding Controller
- Working with Action Methods
- Understanding Register Route
- ViewData , ViewBag , TempData
- HTTP Verbs
- Forms Collections

- Razor Syntax
- Summary

Introduction to MVC

Introduction

- Asp. Net MVC is a light weight development framework for building Web application



Asp. Net

- Developing sites is faster (RAD)
- Built-in validation controls
- Built-in Navigations Controls & Login Control
- Built-in Controls to Interact with Database
- Easy to use UI like Ad Rotator, Tree View, etc...
- Good for non .NET programmer

Drawback of Asp. Net Web Forms

- Page size is heavy (loading page is slower & Most of the controls are server side)
- Server side code is difficult to integrate with HTML
- Page life cycle creates complexities
- Difficult to Test the application

Asp. Net MVC

- Easy to integrate server side code with HTML
 - Page size is lightweight (No Server side controls)
 - No Concept of Page Life cycle
-
- RAD May not be possible
 - Required highly skilled developers

Common Features

- **Master pages**
- **Data Binding**
- **Form Authentication**
- **Output and data caching**
- **Session**
- **Configuration System**

MVC

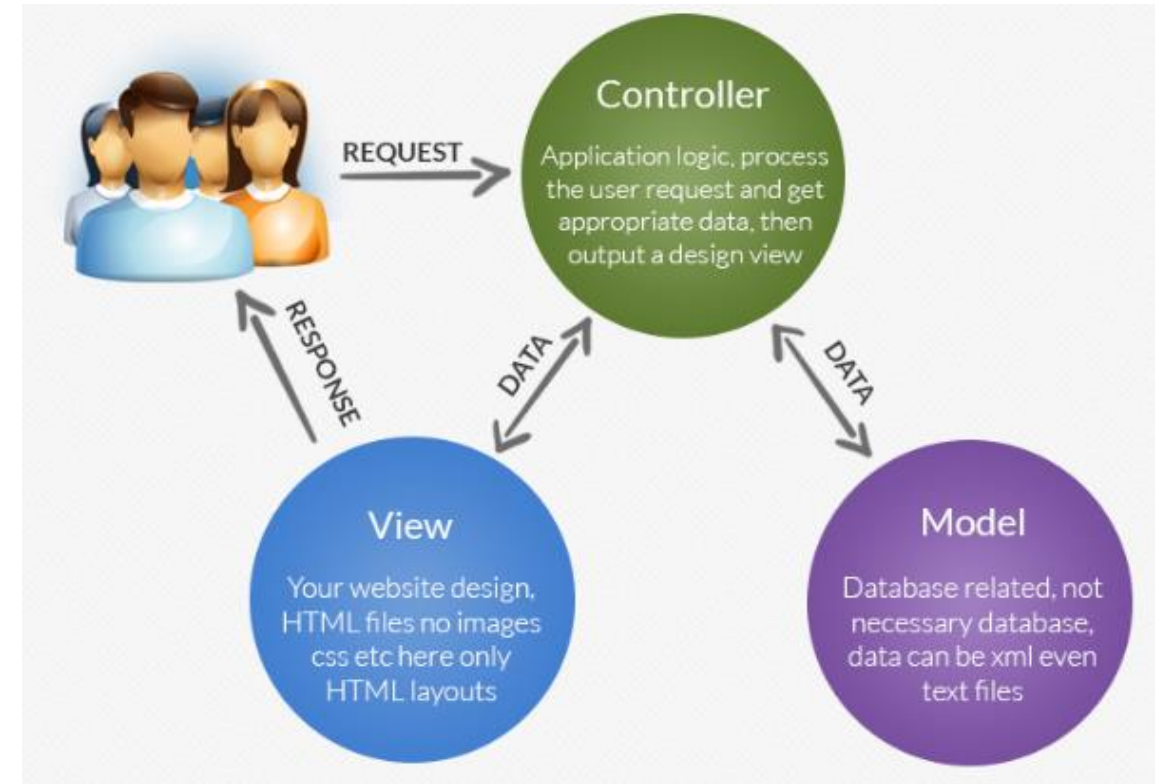
- The Model-View-Controller (MVC) architectural pattern separates an application into three main components

The model, The view, and The controller.

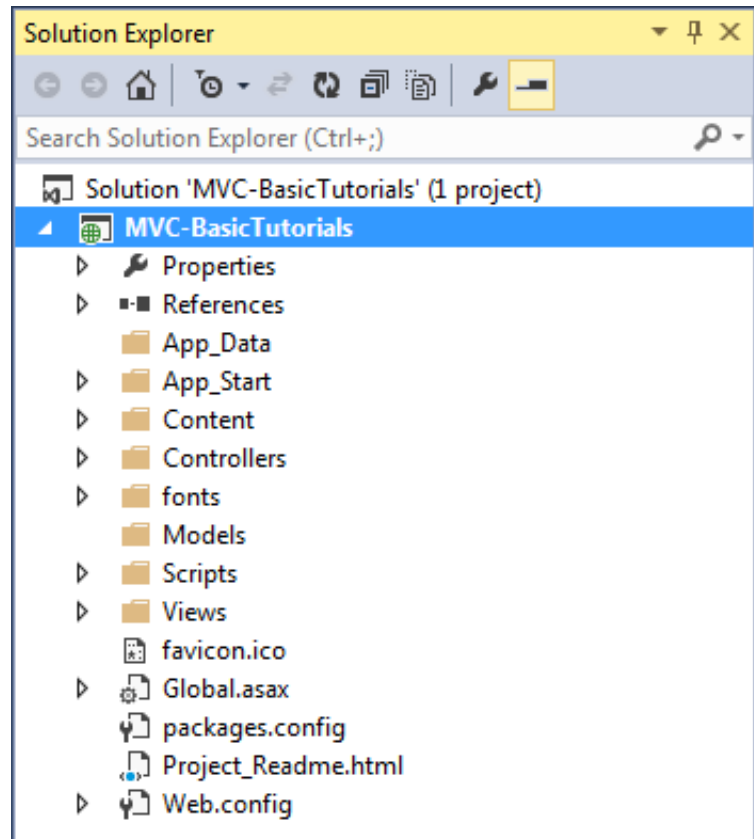
- The MVC framework is defined in the **System.Web.Mvc** namespace

M-V-C

- Model : Represents Data
- View : User Interface
- Controller : Handles User request



MVC 5 Folder Structure



App_Data

App_Data folder can contain application data files like LocalDB, .mdf files, xml files and other data related files

App_Start

These would be config files like AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs etc

Content

Content folder contains static files like css files, images and icons files. MVC 5 application includes bootstrap.css, bootstrap.min.css and Site.css by default

Scripts

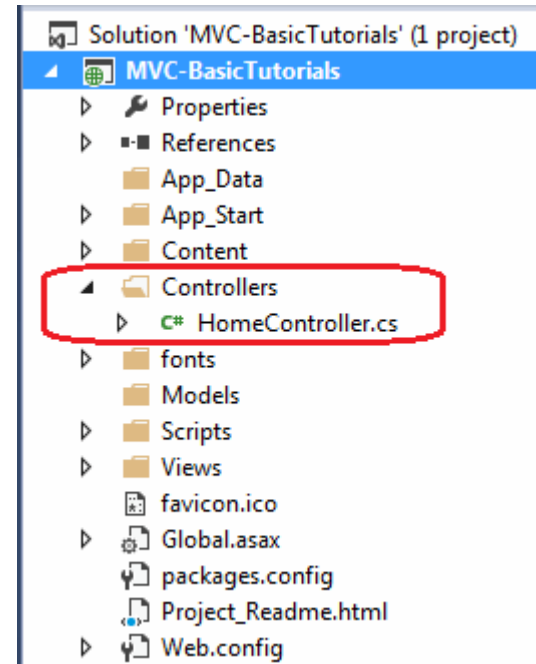
Scripts folder contains JavaScript or VBScript files for the application

Fonts

Fonts folder contains custom font files for your application.

Controller

- Controllers folder contains class files for the controllers. Controllers handles users' request and returns a response. MVC requires the name of all controller files to end with "Controller".

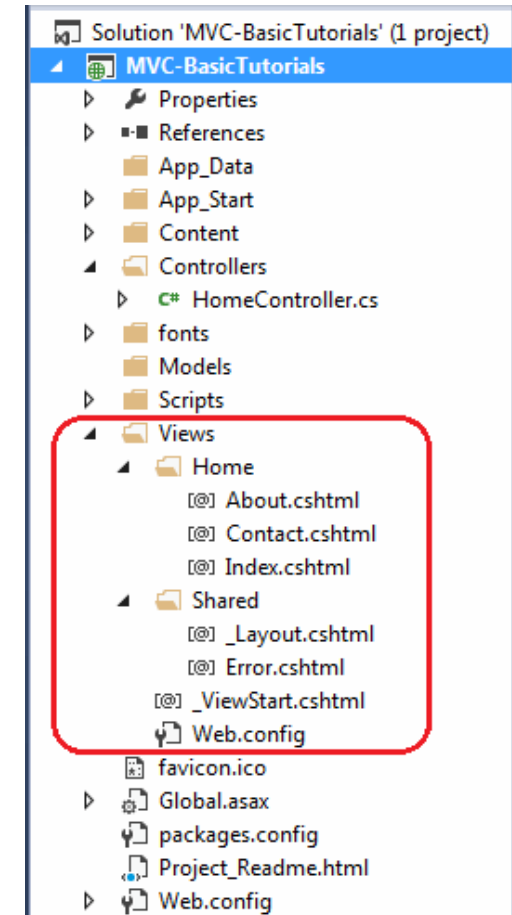


Model

- Models folder contains model class files. Typically model class includes public properties, which will be used by application to hold and manipulate application data.

View

- Views folder contains html files for the application. Typically view file is a .cshtml file where you write html and C# or VB.NET code.
- Views folder includes separate folder for each controllers. For example, all the .cshtml files, which will be rendered by HomeController will be in View > Home folder.
- Shared folder under View folder contains all the views which will be shared among different controllers e.g. layout files.



Folder Structure continued

Global.asax

Global.asax allows you to write code that runs in response to application level events, such as `Application_BeginRequest`, `application_start`, `application_error`, `session_start`, `session_end` etc.

Packages.config:

Packages.config file is managed by NuGet to keep track of what packages and versions you have installed in the application.

Web.config

Web.config file contains application level configurations.



Controller

- The Controller in MVC architecture handles any incoming URL request.
- Controller is a class, derived from the base class *System.Web.Mvc.Controller*.
- Controller class contains public methods called **Action** methods.
- Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.

Simple Controller

```
using System.Web.Mvc;
```

```
namespace MVC_BasicTutorials.Controllers
```

```
{
```

```
    public class StudentController : Controller
```

```
    {
```

```
        public string Index()
```

```
        {
```

```
            return "This is Index action method of StudentController";
```

```
        }
```

```
    }
```

Configure Route

- Every MVC application must configure (register) at least one route, which is configured by MVC framework by default. You can register a route in **RouteConfig** class, which is in RouteConfig.cs under **App_Start** folder.

```
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 }
        );
    }
}
```

Route to ignore

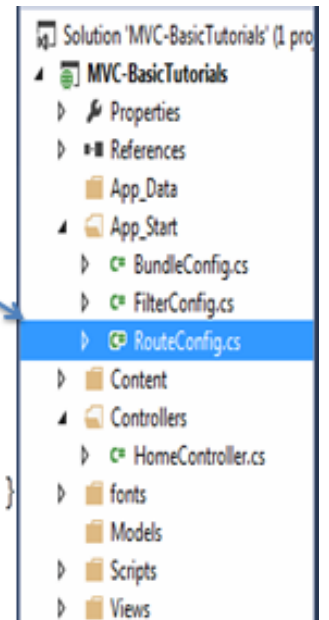
Route name

URL Pattern

Defaults for Route

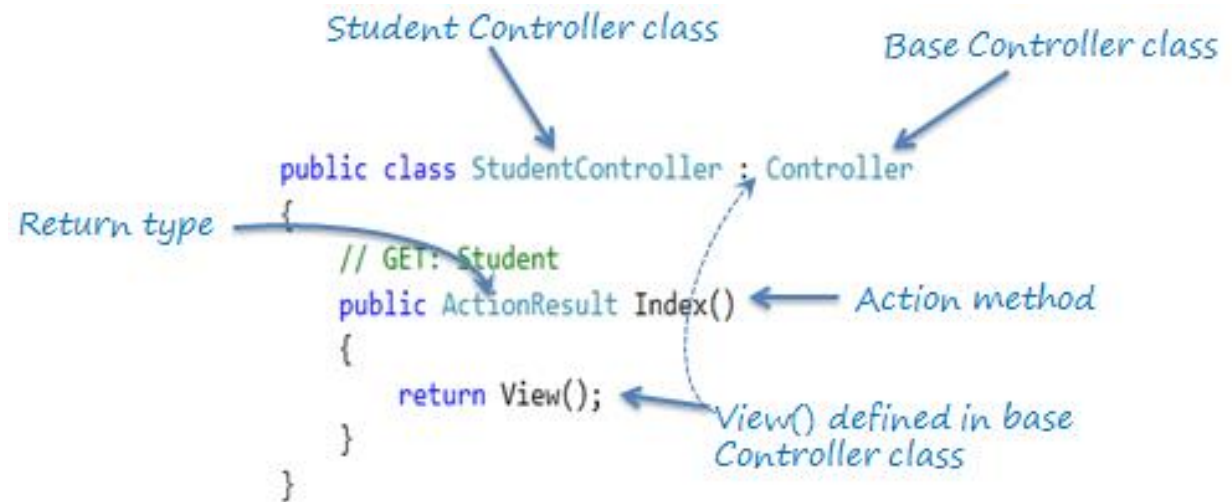
© TutorialsTeacher.com

RouteConfig.cs



Action Methods

- All the public methods of a Controller class are called Action methods. They are like any other normal methods with the following restrictions:
- Action method must be public. It cannot be private or protected
- Action method cannot be overloaded
- Action method cannot be a static method.



Action Result:

Result Class	Description
ViewResult	Represents HTML and markup.
EmptyResult	Represents No response.
ContentResult	Represents string literal.
FileContentResult/ FilePathResult/ FileStreamResult	Represents the content of a file
JavaScriptResult	Represent a JavaScript script.
JsonResult	Represent JSON that can be used in AJAX
RedirectResult	Represents a redirection to a new URL
RedirectToRouteResult	Represent another action of same or other controller
PartialViewResult	Returns HTML from Partial view
HttpUnauthorizedResult	Returns HTTP 403 status

Return Type

Result Class	Description	Base Controller method
ViewResult	Represents HTML and markup.	View()
EmptyResult	Represents No response.	
ContentResult	Represents string literal.	Content()
FileContentResult, FilePathResult, FileStreamResult	Represents the content of a file	File()
JavaScriptResult	Represent a JavaScript script.	JavaScript()
JsonResult	Represent JSON that can be used in AJAX	Json()
RedirectResult	Represents a redirection to a new URL	Redirect()
RedirectToRouteResult	Represent another action of same or other controller	RedirectToRoute()
PartialViewResult	Returns HTML	PartialView()
HttpUnauthorizedResult	Returns HTTP 403 status	

Types of action results

ViewResult

Contentresult

EmptyResult

RedirectResult

JavaScriptResult

```
return View("Index");
```

```
return Content("I its my content result");
```

```
return new EmptyResult();
```

```
return Redirect("/Home/index");
```

```
public JsonResult Examplejson()
```

```
{
```

```
List<string> mystring = newList<string>();
```

```
mystring.Add("HI");
```

```
mystring.Add("MVC");
```

```
return Json(mystring, JsonRequestBehavior.AllowGet);
```

```
}
```

PartialViewResult

```
public PartialViewResult ExamplePartialView()
```

```
{
```

```
return PartialView("Login");
```

```
}
```


Action Selectors

- Action selector is the attribute that can be applied to the action methods. It helps routing engine to select the correct action method to handle a particular request. MVC 5 includes the following action selector attributes:
 - ActionName
 - NonAction
 - ActionVerbs

Action Name

- ActionName attribute allows us to specify a different action name than the method name. Consider the following example.

```
[ActionName("find")]  
public ActionResult GetByld(int id)  
{  
    // get student from the database  
    return View();  
}  
  
//http://localhost/student/find/1
```

No Action

- NonAction selector attribute indicates that a public method of a Controller is not an action method. Use NonAction attribute when you want public method in a controller but do not want to treat it as an action method.

[NonAction]

```
public Student GetStudnet(int id)
{
    return studentList.Where(s => s.StudentId == id).FirstOrDefault();
}
```

Action Verbs

- The ActionVerbs selector is used when you want to control the selection of an action method based on a Http request method. For example, you can define two different action methods with the same name but one action method responds to an HTTP Get request and another action method responds to an HTTP Post request.

http://localhost/Student/Edit/1

HttpGET

```
public ActionResult Edit(int Id)
{
    var std = students.Where(s => s.StudentId == Id).FirstOrDefault();

    return View(std);
}
```

© TutorialsTeacher.com

http://localhost/Student/Edit

HttpPOST

```
[HttpPost]
public ActionResult Edit(Student std)
{
    //update database here..

    return RedirectToAction("Index");
}
```

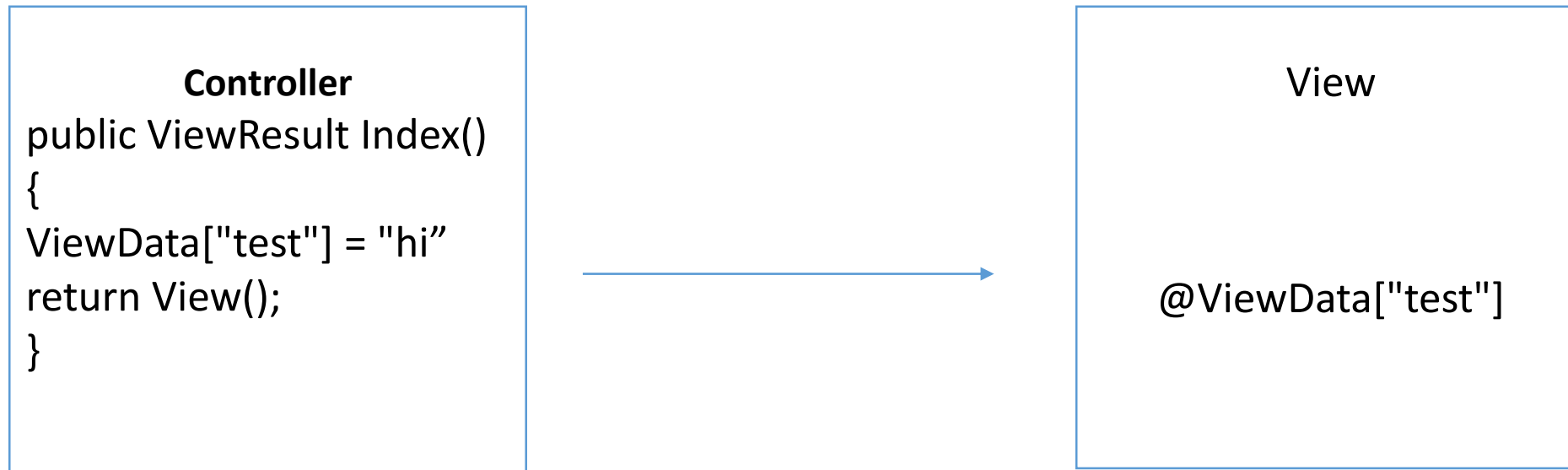


Passing Data from Controller to View

- We have three options to pass data from controller to view in asp.net mvc application those are
 - ViewData
 - ViewBag
 - TempData

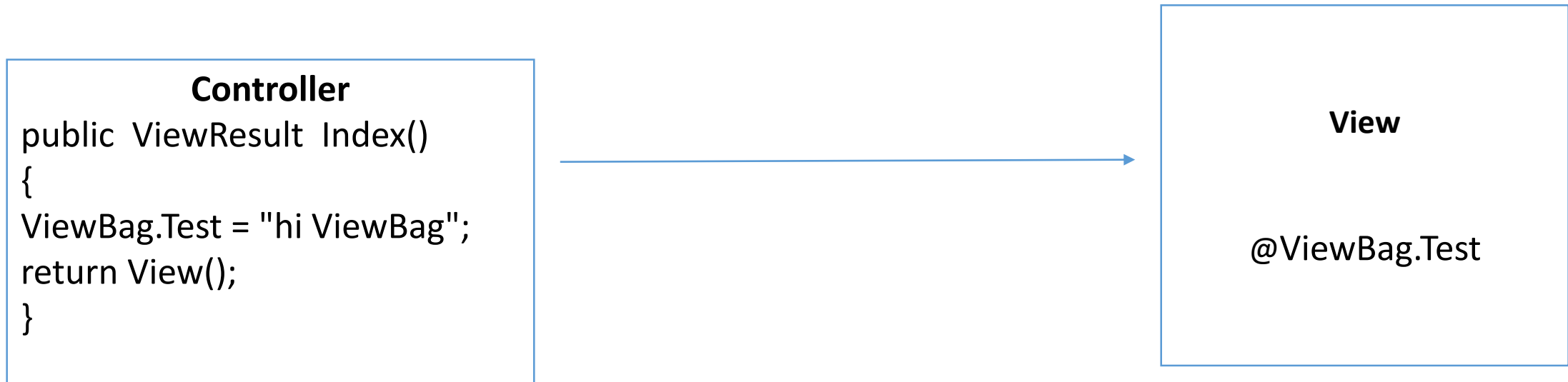
ViewData

- ViewData is used to pass data from controller to view and it contains null value when direction occurs.
- ViewData is a dictionary which can contain key-value pairs where each key must be string.



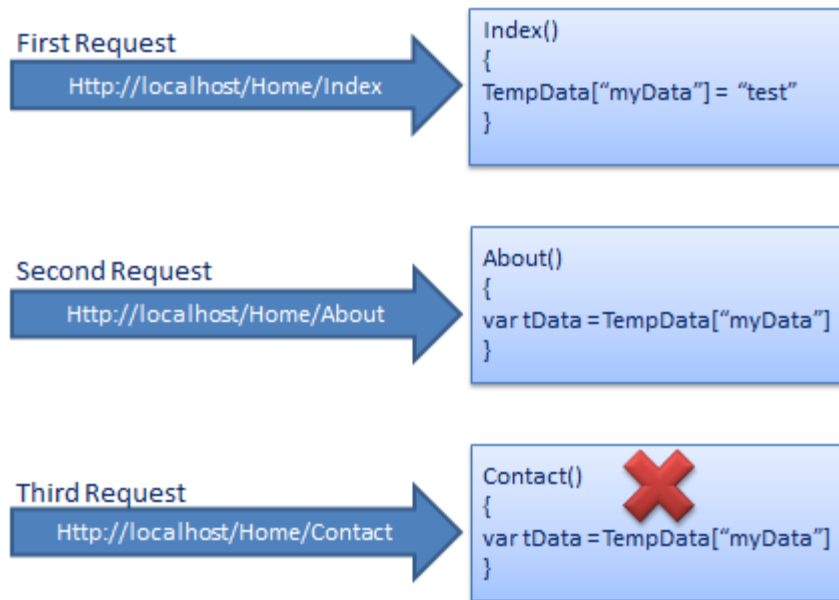
ViewBag

- ViewBag is used to pass data from controller to view and it does not required type casting. ViewBag contains a null value when redirection occurs
- The ViewBag is a dynamic type property



TempData

- TempData in mvc is used to pass data from Controller to Controller or Controller to view and it is used to pass data from the current request to the next request.



Call TempData.Keep()
to retain TempData
values in a third
consecutive request.

Razor Syntax

Introduction

- Razor view engine in asp.net mvc is syntax that allows you to write server side code on view
- Razor supports C# and Visual Basic programming languages.
- razor view engine code blocks are enclosed in "@{ ... }". Here "@" is character that tells beginning of Razor syntax. This code can be a single expression or entire code block

```
@{  
  
//Razor block  
  
}
```

```
@{  
var weekDay  
= DateTime.Now.DayOf  
Week;  
}
```

Multiple –statement line code

```
@{  
    var date = DateTime.Now.ToShortDateString();  
    var message = "Hello World";  
}
```

<h2>Today's date is: @date </h2>

<h3>@message</h3>

Declaring Variables

```
@{  
    string str = "";  
  
    if(1 > 0)  
    {  
        str = "Hello World!";  
    }  
}
```

```
<p>@str</p>
```

Display Text

- Use @: or <text>/<text> to display texts within code block.

```
@{  
    var date =  
DateTime.Now.ToShortDateString();  
    string message = "Hello World!";  
    @:Today's date is: @date <br />  
    @message  
}
```

```
@{  
    var date =  
DateTime.Now.ToShortDateString();  
    string message = "Hello World!";  
    <text>Today's date is:</text>  
    @date <br />  
    @message  
}
```

Comments

- This razor view syntax ("`@* *@`") tell that code inside this should be ignore from being executed. This syntax is common for both vb.net and c# language using Razor.

Eg:-

- `@* This is a Razor multiline comment *@`
- Shortcut key
 - Comment the block - (Ctrl E, C)
 - Uncomment the block - (Ctrl E, U)

If Condition

```
@if(DateTime.IsLeapYear(DateTime.Now.Year) )
{
    @DateTime.Now.Year @:is a leap year.
}
else {
    @DateTime.Now.Year @:is not a leap year.
}
```


Loops

```
@for (int i = 0; i < 5; i++) {  
    @i.ToString() <br />  
}
```

```
@foreach (var item in marks)  
{  
    @item<br/>  
}
```

HTML Helpers

Reusable Html Helper

```
@helper ListingItems(string[] items)
{
    <ol>
    @foreach (string item in items)
    {
        <li>@item</li>
    }
    </ol>
}
```

Reusable Html Helper

```
<h3>Programming Languages:</h3>
```

```
@ListingItems(new string[] { "C", "C++", "C#" })
```

```
<h3>Book List:</h3>
```

```
@ListingItems(new string[] { "How to C", "how to C++", "how to C#" })
```

Extension Method

```
public static class MyHelper {  
    public static MvcHtmlString DisplayList(this HtmlHelper h, string[] list)  
{  
        TagBuilder t = new TagBuilder("ul");  
        foreach (var item in list) {  
            TagBuilder itemtag = new TagBuilder("li");  
            itemtag.SetInnerText (item);  
            t.InnerHtml += itemtag.ToString();  
        }  
        return new MvcHtmlString(t.ToString());  
    }  
}
```

```
public ActionResult DisplayFruits()-----→ Controller
{
    ViewBag.Fruits = new string[] { "Apple", "Orange", "Pear",
    "Banana", "Grapes" };
    return View();

}
@Html.DisplayList((string[])@ViewBag.Fruits) -----→ View
```

UI Controls

Introduction

- HTML Helper is just a method that returns a HTML string. The string can represent any type of content that you want. For example, you can use HTML Helpers to render standard HTML tags like HTML `<input>`, `<button>` and `` tags etc.
- HTML helpers can be of two types
 - basic html helper
 - strongly type html helper

Basic HTML helper

- @Html.BeginForm
- @Html.EndForm
- @Html.TextBox
- @Html.TextArea
- @Html.Password
- @Html.Hidden
- @Html.CheckBox
- @Html.RadioButton
- @Html.DropDownList
- @Html.ListBox

Example

Textbox - `@Html.TextBox("UserName")`

TextArea - `@Html.TextArea("UserAddress")`

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

Hidden field - `@Html.Hidden("UserID")`

Checkbox - `@Html.CheckBox("Check")`

Radiobutton - `@Html.RadioButton("gender", "Male", true)`

Example

```
DropDownList - @Html.DropDownList("DrpCountry", new List<SelectListItem> {  
  
    new SelectListItem { Text = "select", Value = "0" },  
    new SelectListItem { Text = "India", Value = "1" },  
    new SelectListItem { Text = "USA", Value = "2" }  
})
```

```
Listbox - @Html.ListBox("ListName", new List<SelectListItem> {  
  
    new SelectListItem { Text = "One", Value = "1" },  
    new SelectListItem { Text = "Two", Value = "2" },  
    new SelectListItem { Text = "Three", Value = "3" }  
})
```

```
DisplayName - @Html.DisplayName("Displayname")
```



Strongly Typed html controls

Textbox - `@Html.TextBoxFor(model => model.UserName)`

TextArea - `@Html.TextAreaFor(model => model.UserAge)`

Password - `@Html.PasswordFor(model => model.UserAddress)`

Hidden field - `@Html.HiddenFor(model => model.UserID)`

Checkbox - `@Html.CheckBoxFor(m => m.Usergender)`

Radiobutton - `@Html.RadioButtonFor(m => m.CheckID, true, new { @checked = "checked" })`

Strongly Typed html controls

- **DropDownList** - `@Html.DropDownListFor(m => m.UserID, Model.listdropdown)`
-
- **Listbox** - `@Html.ListBoxFor(x => x.Listboxid, new SelectList(Model.listboxdata, "Id", "Name"))`
-
- **DisplayName** - `@Html.DisplayFor(m => m.Listboxid)`
-



Validations

Introduction

- ASP.NET MVC uses DataAnnotations attributes to implement validations
- DataAnnotations includes built-in validation attributes for different validation rules, which can be applied to the properties of model class.
- ASP.NET MVC framework will automatically enforce these validation rules and display validation messages in the view.
- The DataAnnotations attributes included in [System.ComponentModel.DataAnnotations](#) namespace.

Data annotations

Attribute	Description
Required	Indicates that the property is a required field
StringLength	Defines a maximum length for string field
Range	Defines a maximum and minimum value for a numeric field
RegularExpression	Specifies that the field value must match with specified Regular Expression
CreditCard	Specifies that the specified field is a credit card number
CustomValidation	Specified custom validation method to validate the field
EmailAddress	Validates with email address format
FileExtension	Validates with file extension
MaxLength	Specifies maximum length for a string field
MinLength	Specifies minimum length for a string field
Phone	Specifies that the field is a phone number using regular expression for phone numbers

Simple Example

```
public class Student
{
    public int StudentId { get; set; }

    [Required]
    public string StudentName { get; set; }

    [Range(5,50)]
    public int Age { get; set; }
}
```

Validation Message

- The `Html.ValidationMessage()` is an extension method, that is a loosely typed method. It displays a validation message if an error exists for the specified field in the `ModelState` object.

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

```
@Html.ValidationMessage("Name", "*")
```

```
@Html.ValidationMessageFor(m => m.Name, "*")
```

Validation summary

- The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object.

Edit

Student

```
@Html.ValidationSummary(true);
```

- The Name field is required.
- The Age field is required.

Name

Age

Save

[Back to List](#)

ModelState

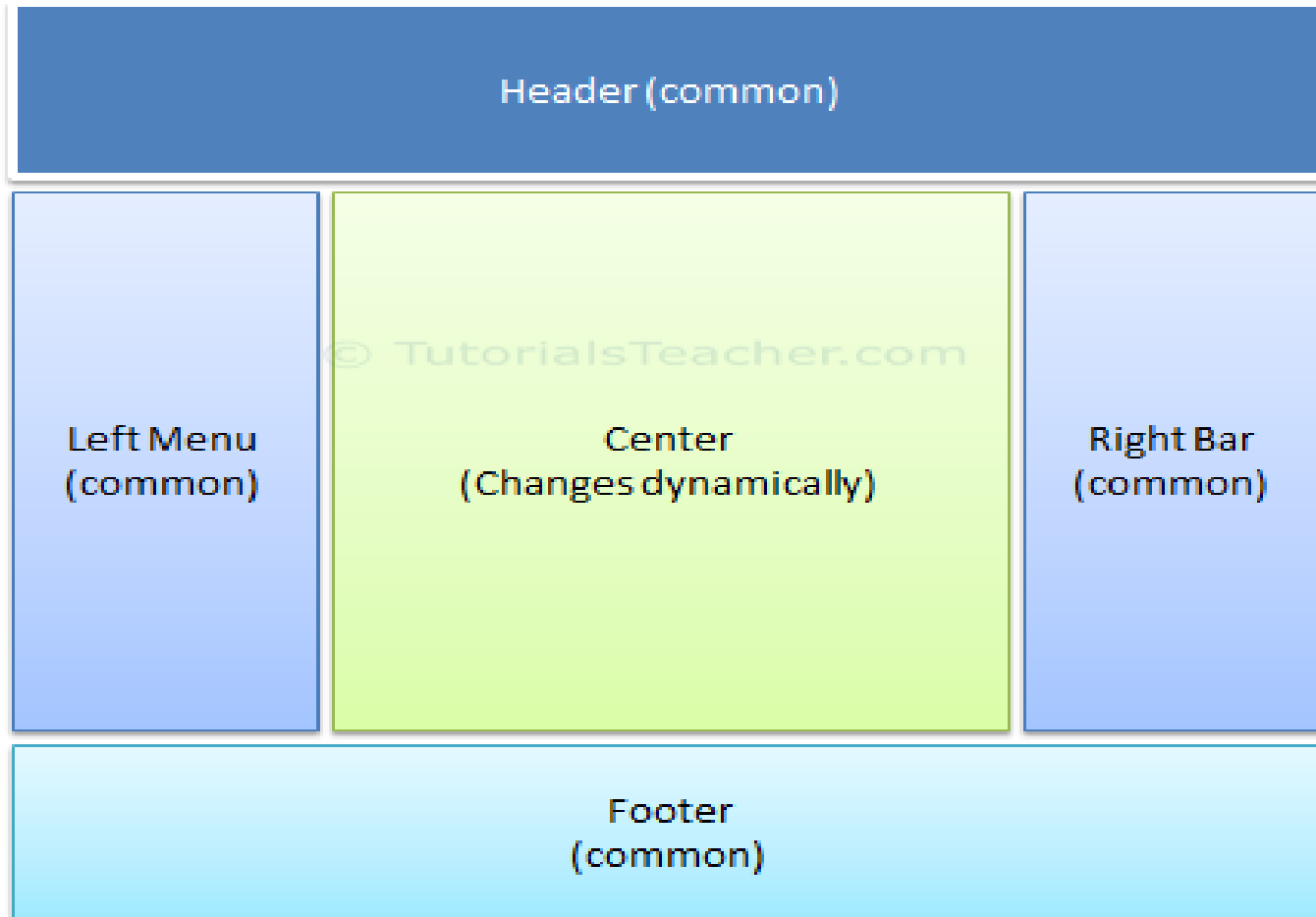
- the controller actions can query the ModelState to discover whether the request is valid and react accordingly.

```
if(!ModelState.IsValid)
{
    return View(model);
}
return RedirectToAction("Index");
```

Layout

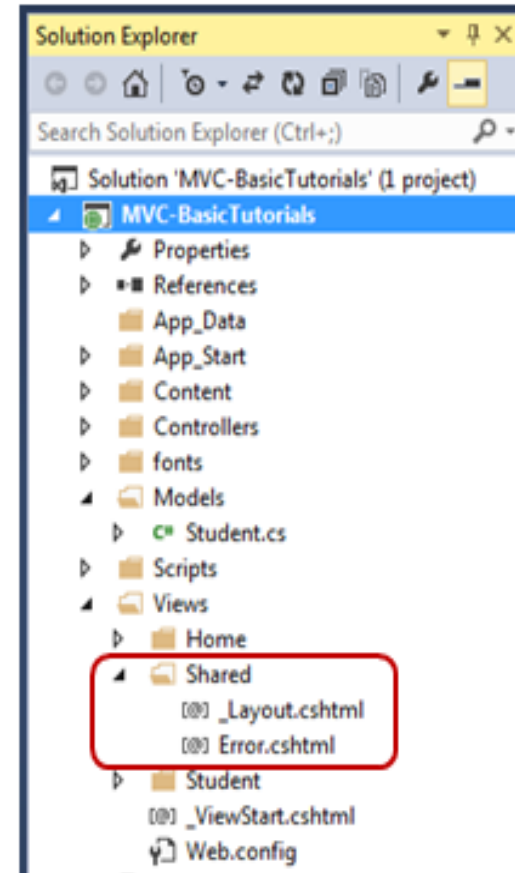
Introduction

- An application may contain common parts in the UI which remains the same throughout the application such as the logo, header, left navigation bar, right bar or footer section. ASP.NET MVC introduced a Layout view which contains these common UI parts, so that we don't have to write the same code in every page.
- The layout view is same as the master page of the ASP.NET webform application.



Layout

- Layout views are shared with multiple views, so it must be stored in the Shared folder.



_layoutdemo.cshtml

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width"/>
<title>_layoutdemo</title>
</head>
<body>
<table border="1px" style="width: 100%; text-align: center;">
<tr>
<td>
@RenderSection("headers")
</td>
</tr>
<tr>
<td style="height: 500px;">
@RenderBody()
// RenderBody
</td>
</tr>
<tr>
<td>
@RenderSection("footers")
</td>
</tr>
</table>
</body>
</html>
```

Index.cshtml

```
@{
    Layout = "~/Views/Shared/_layoutdemo.cshtml";
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width"/>
<title>Index</title>
</head>
<body>
<div>
<table style="width: 100%">
<tr>
<td>
@section headers{
<h1>Header Content</h1>
}
</td>
</tr>
<tr>
<td>
@section Footers{
<h1>Footer Content</h1>
}
</td>
</tr>
</table>
</div>
</body>
</html>
```

Hyperlinks

- asp.net mvc provide Html helper class (@Html.ActionLink) to generate hyperlinks or actionlinks.

```
@Html.ActionLink("About page","About")
```



Layout page in action method

- You can also specify which layout page to use in while rendering view from action method using View() method.

```
public ActionResult Index()  
{  
    return View("Index", "_myLayoutPage");  
}
```

Partial View

Introduction

- if we want to display some similar part of content in various part of web application then we need create a Partial View for that part. Generally partial can be reusable in multiple views and it helps us to reduce code duplication.

Example

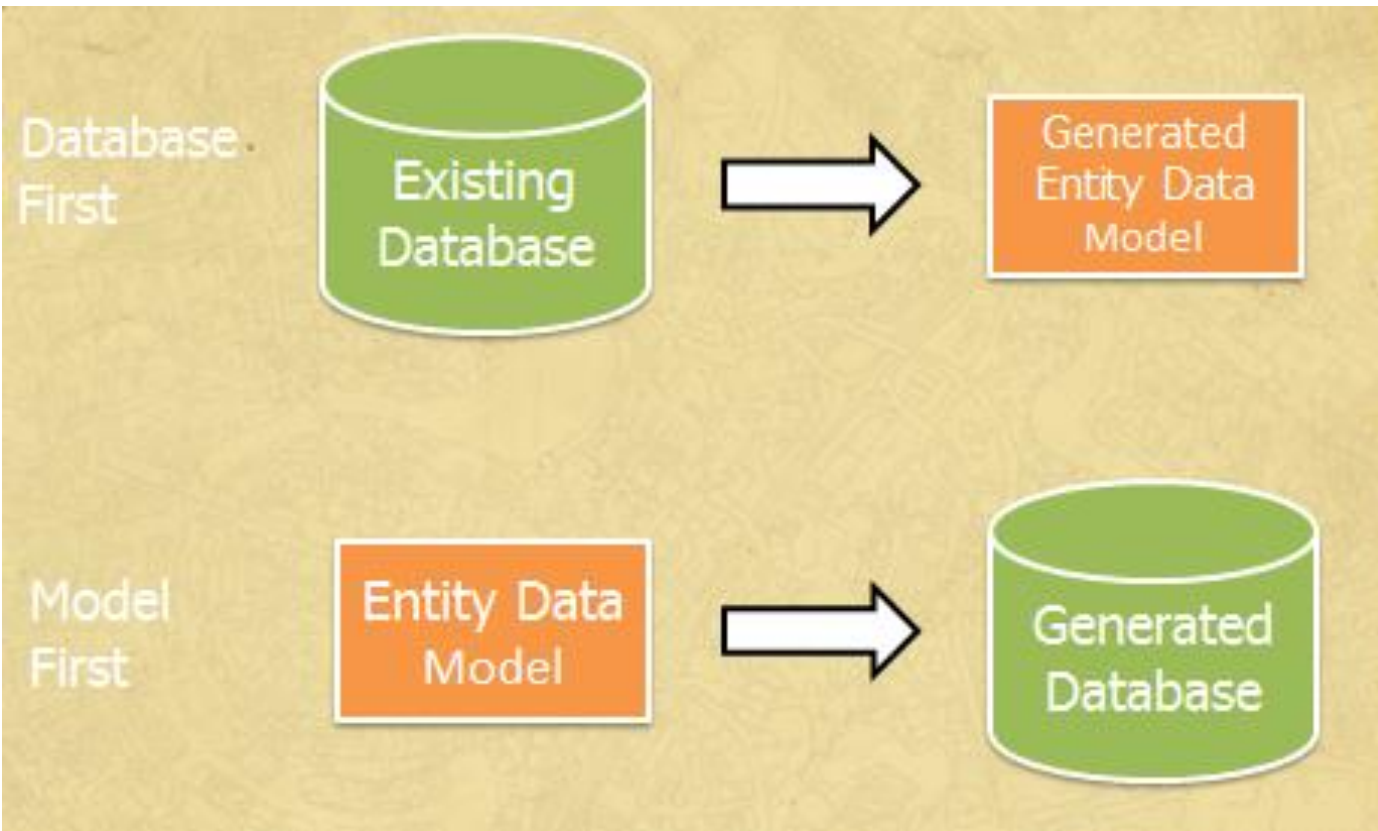
Action Method

```
public ActionResult  
PartialViewExample()  
{  
    return PartialView();  
}
```

View File

```
@Html.Partial("PartialViewExample")
```

Database in MVC



Two Approaches

- The Code First approach allows us to define the model from the code
- The Model First creates a model and classes from database

Code First Approach

```
public class customer
{
    [Key]
    public int cid { get; set; }
    public string cname { get; set; }
    public int age { get; set; }
}

public class CustomerContext : DbContext
{
    public CustomerContext()
    : base("name=mydb") { }
    public DbSet<customer> customers { get; set; }
}
```

```
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
```


Security

Introduction

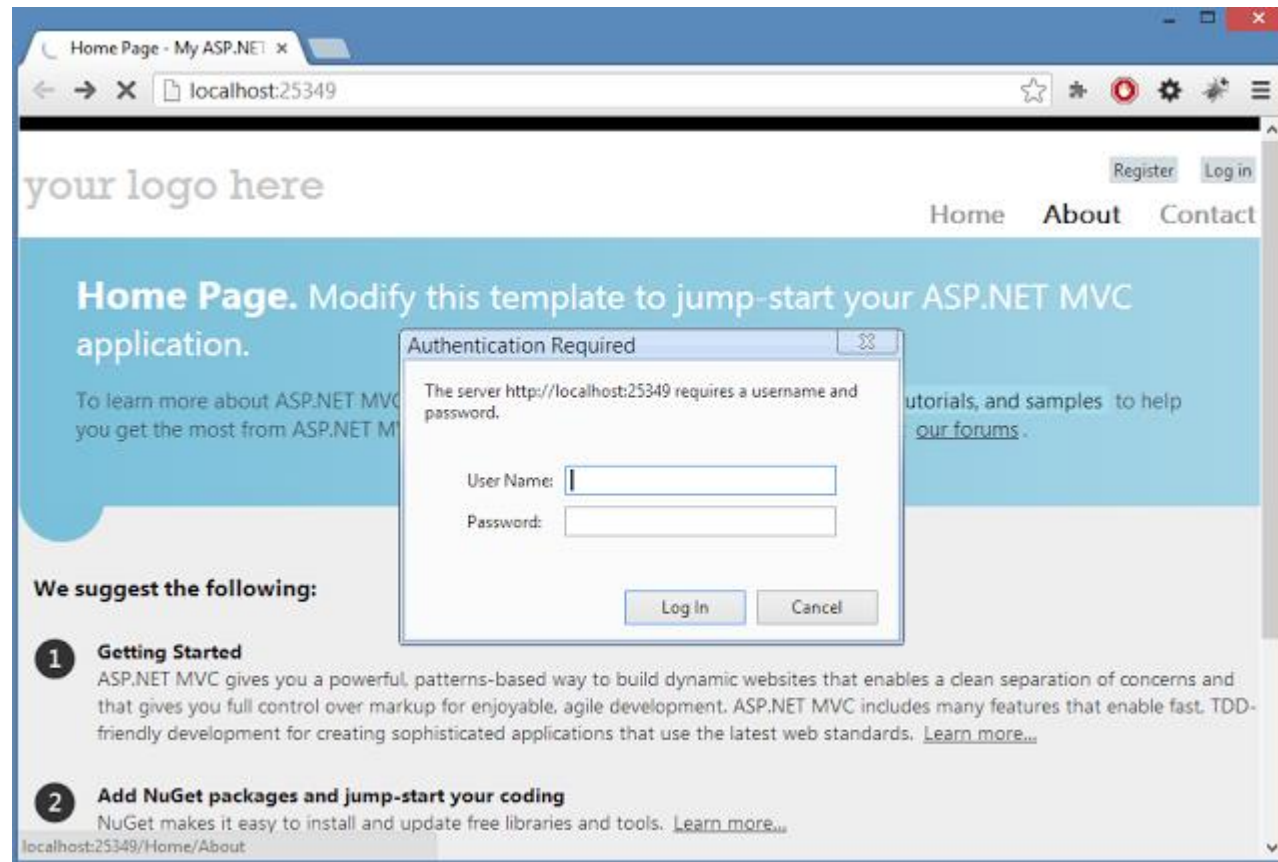
- To secure MVC application Asp.Net supports following kind of authenticati
 - Windows authentication
 - Forms authentication
- Windows authentications suitable for **intranet** environment
- Forms authentications suitable for **internet** environment

Windows authentication

- In Windows authentication the credentials are validated against **active directory**
- Active directory is a windows based folder which stores windows OS credentials
- To use windows authentication first we need to enable windows authentication in project property window
- Decorate [**Authorize**] attribute above all the method else it is treated as anonymous (every one can access)
- Anonymous type can be only used only if its enabled in property window.

Windows authentication

- The user provide his windows credentials which is prompts login screen



Enable authentication for specific user

- To enable authentication for specific users we need use to Authorization attribute in web.config file

```
<authorization>
```

```
  <allow users ="WIN-H7OCN9V67OI\anand"/>
```

```
  <deny users ="*" />
```

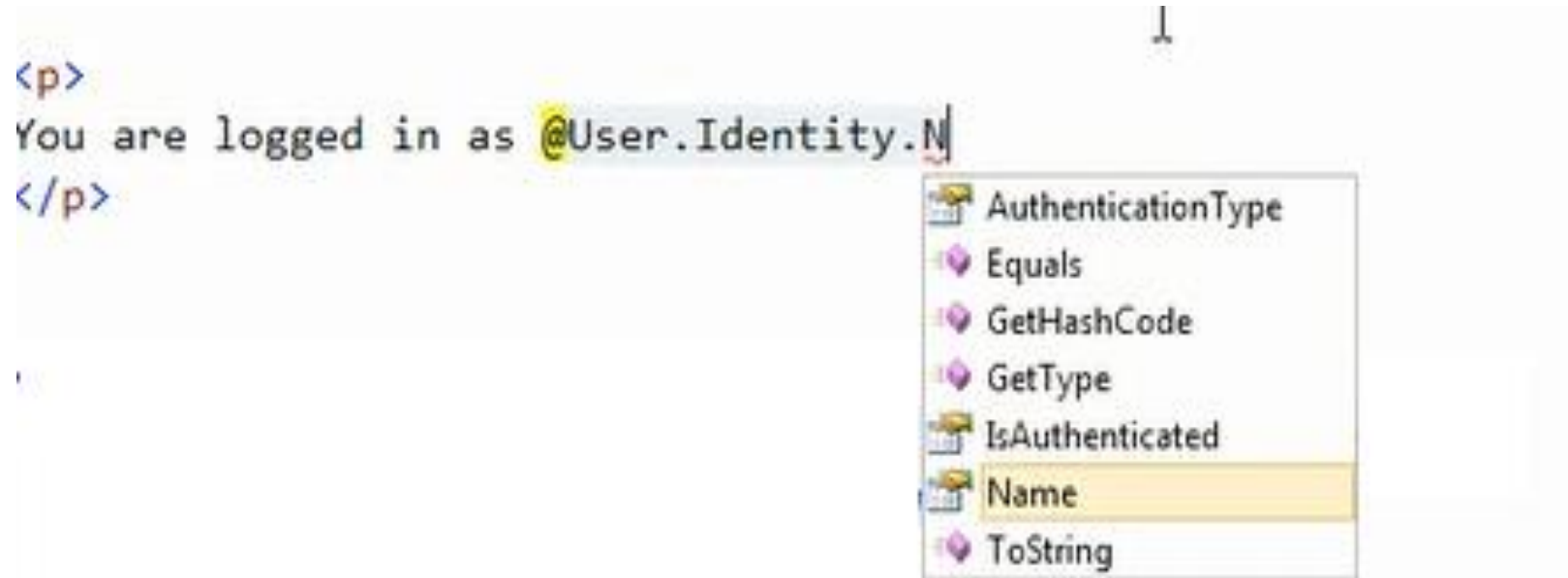
```
</authorization>
```

Remove <deny users ="*" /> tag if you need access method without authorize attribute.

Display username

- To display windows username use following code :

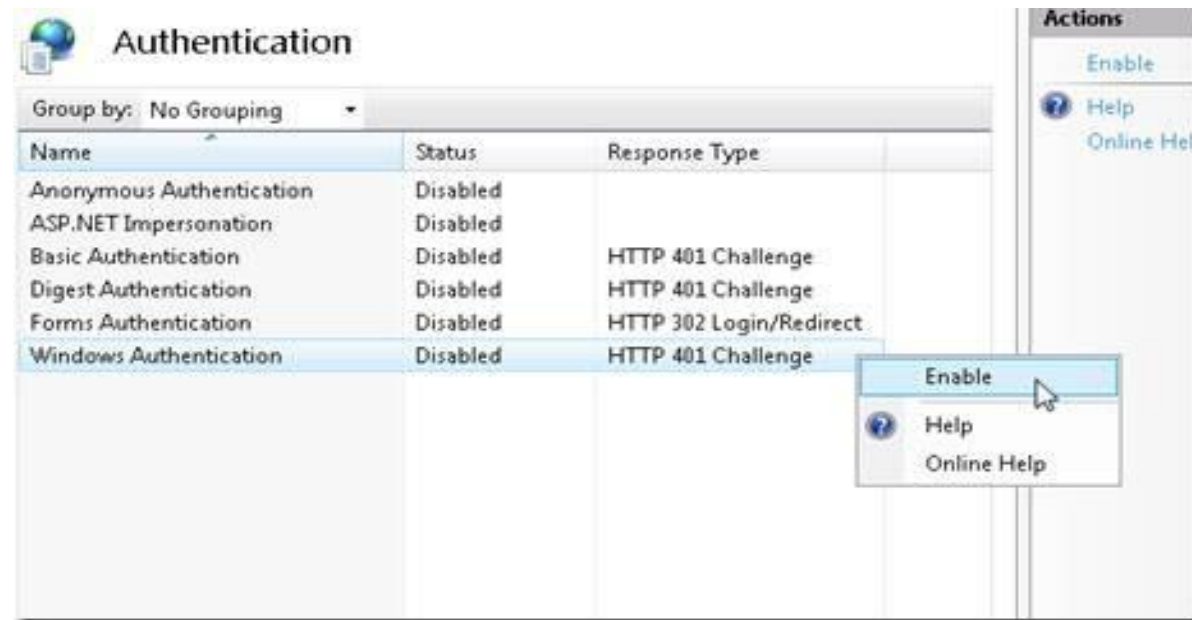
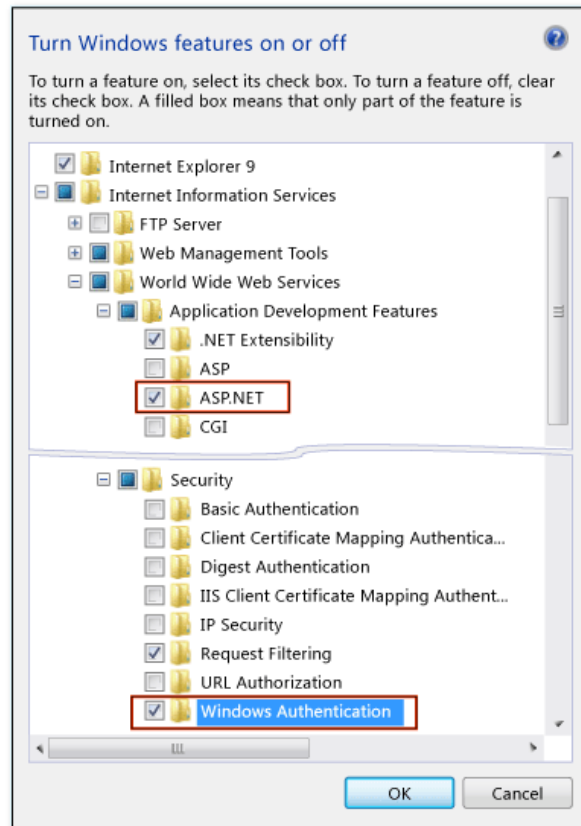
```
<p>  
You are logged in as @User.Identity.N  
</p>
```



The image shows a code editor with a snippet of HTML and a dropdown menu. The code is: `<p>You are logged in as @User.Identity.N</p>`. The dropdown menu is open, showing a list of properties and methods for the `@User.Identity.N` object. The `Name` property is highlighted. The list includes: `AuthenticationType`, `Equals`, `GetHashCode`, `GetType`, `IsAuthenticated`, `Name`, and `ToString`.

windows authentication in IIS Web server

- Install following windows component in order to use windows authentication from IIS web server



Forms Authentication

- Forms authentication is suitable for internet architecture as it supports a large number of users
- User information can be stored either in web.config file or in database
- To store user information in web.config file use credential tag under authentication section
- Decorate authorize attribute for every method else it is treated as an anonymous method (every one access it)

Web.config changes

```
<authentication mode="Forms">  
  <forms name="aspauth" loginUrl="/home/login">  
    <credentials passwordFormat="Clear">  
      <user name="ajay" password="india"/>  
    </credentials>  
  </forms>  
  
</authentication>  
<authorization>  
  <deny users ="?"/>  
</authorization>
```

Remove <deny user="?" /> if u want to access a method without authorize attribute

Controller

To validate username and password use FormsAuthentication Class from System.Web.Security namespace

[HttpPost]

```
public ActionResult login(FormCollection frm)
{
    string uname = frm["t1"];    string pwd = frm["t2"];
    if(FormsAuthentication.Authenticate(uname,pwd))    {
        return View("Index");    }
    else
    {
        return View();
    }
}
```

Bootstrap in MVC

Introduction

- Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites.
- responsive CSS adjusts to Desktops, Tablets and Mobiles



Bootstrap features

- BSGrid
- Tables
- Images
- Alerts
- Button
- Input controls

BS Grid

- Bootstrap grid system provides the quick and easy way to create responsive website layouts.
- Bootstrap's grid system allows up to 12 columns across the page.
- Bootstrap's grid system needs a container to hold rows and columns. A container is a simple `<div>` element with a class of `.container`

BS rows & columns

- Once container is created the user can create as many as rows required without any limitation
- A row is a simple <div> element with a class of .row

```
<div class="container">
```

```
<div class="row">
```

```
  <div class="col-sm-4" style="background-color:red;">.col-sm-4</div>
```

```
  <div class="col-sm-4" style="background-color:blue">.col-sm-8</div>
```

```
  <div class="col-sm-4" style="background-color:green">.col-sm-8</div>
```

```
</div>
```

```
</div>
```

Column

- Bootstrap uses different column class prefixes for different sized devices. These prefixes are shown in the table below:

Class Prefix	Device Size
.col-xs-	<768px
.col-sm-	768px to 991px
.col-md-	992px to 1199px
.col-lg-	≥ 1200px

Tables

Tables Classes

- .table
- table table-striped
- table table-bordered
- table table-hover
- table table-condensed

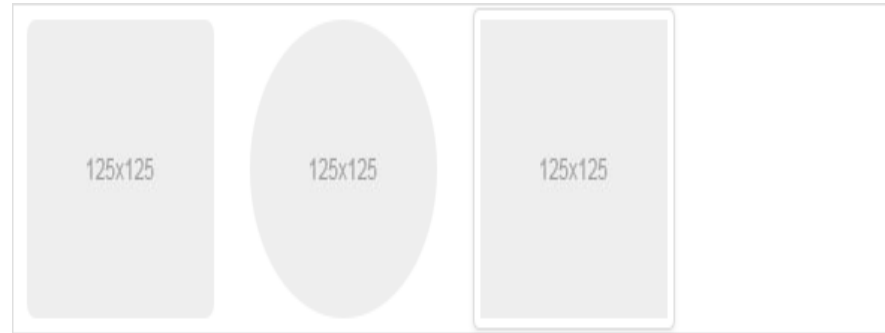
Responsive Table

- With Bootstrap 3 you can also create responsive tables to enable horizontal scrolling on small devices (screen width under 768px).
- viewing responsive tables on other devices having screen width larger than 768px, you will not see any difference.
- To make any table responsive just place the table inside a <div> element and apply the class .table-responsive

```
<div class="table-responsive">  
  <table class="table table-bordered">  
    <thead>  
      <tr>
```

Images

- Using the Bootstrap built-in classes you can easily style images such as making the round cornered or circular images, or give them effect like thumbnails.



```
  
  

```

Responsive images

- In Bootstrap you can make the images responsive too. Just add the class `.img-responsive` to the `` tag.
- The image will then scale nicely to the parent element.

```

```

Tabs

- `<ul class="nav nav-tabs">`
- `<li class="active">Home`
- `Menu 1`
- `Menu 2`
- `Menu 3`
- ``

Collapse

```
<button data-toggle="collapse" data-target="#demo">Click  
here</button>
```

```
<div id="demo" class="collapse">  
  welcome to india  
</div>
```


alert

- Alert boxes are used quite often to stand out the information that requires immediate attention of the end users such as warning, error or confirmation messages.
- With Bootstrap you can easily create elegant alert messages box for various purposes. You can also add an optional close button to dismiss any alert.

alerts

- You can create a simple Bootstrap warning alert message box by adding the contextual class `.alert-warning` to the `.alert` base class

```
<div class="alert alert-warning">  
  <a href="#" class="close" data-dismiss="alert">&times;</a>  
  <strong>Warning!</strong> There was a problem with your network  
connection.  
</div>
```

- Requires `bootstrap.min.js`

alerts

- alert alert-warning
- alert alert-danger fade in
- alert alert-warning fade in
- alert alert-success fade in
- alert alert-info fade in

If you want to enable the fading transition effect while closing the alert boxes, apply the classes `.fade` and `.in` to them along with the contextual class.

Form Control

- HTML forms are the integral part of the web pages and applications
- styling the form controls manually one by one with CSS are often tedious
- Bootstrap greatly simplifies the process of styling and alignment of form controls like labels, input fields, selectboxes, textareas, buttons, etc. through predefined set of classes.

Form Layout

- Bootstrap provides three different types of form layouts:
- Vertical Form (default form layout)
- Horizontal Form
- Inline Form

Vertical Layout

```
<form>
  <div class="form-group">
    <label for="inputEmail">Email</label>
    <input type="email" class="form-control" id="inputEmail" placeholder="Email">
  </div>
  <div class="form-group">
    <label for="inputPassword">Password</label>
    <input type="password" class="form-control" id="inputPassword" placeholder="Password">
  </div>
  <div class="checkbox">
    <label><input type="checkbox"> Remember me</label>
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```



The image shows a vertical login form. It consists of a container with a light gray border. Inside, there are four main sections: 1. An 'Email' section with a label 'Email' in bold and a text input field with a light gray border and placeholder text 'Email'. 2. A 'Password' section with a label 'Password' in bold and a password input field with a light gray border and placeholder text 'Password'. 3. A 'Remember me' section with a small square checkbox and the text 'Remember me'. 4. A 'Login' button, which is a blue rectangle with the word 'Login' in white text.

Horizontal Layout

- In horizontal form layout labels are right aligned and floated to left to make them appear on the same line as form controls. The horizontal form layout requires the various markup changes from a default form layout.



A horizontal form layout example. It features two rows of input fields. The first row has a label "Email" on the left and a text input field containing "Email" on the right. The second row has a label "Password" on the left and a text input field containing "Password" on the right. Below these fields is a checkbox labeled "Remember me". At the bottom is a blue button labeled "Login".

Email	<input type="text" value="Email"/>
Password	<input type="text" value="Password"/>

☐ Remember me

Login

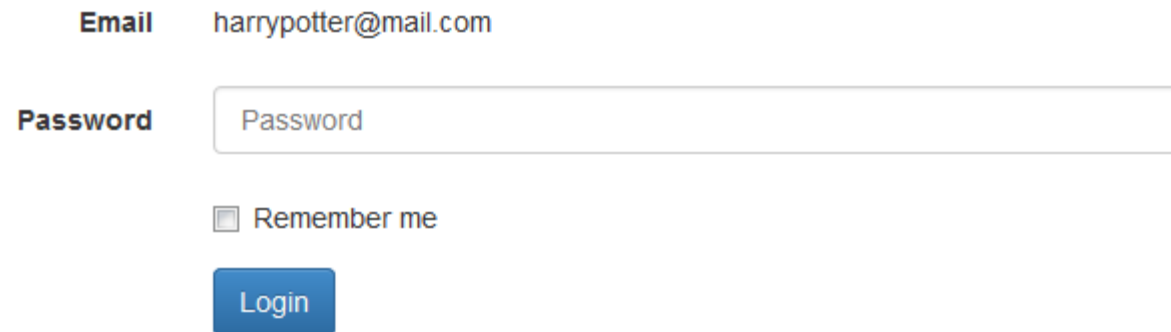
- Sometimes you might require to place the form controls side-by-side to compact the layout. You can do this easily by adding the Bootstrap class `.form-inline` to the `<form>` element.



A Bootstrap form example with the class `.form-inline`. It contains two input fields labeled "Email" and "Password" side-by-side, followed by a "Remember me" checkbox and a "Login" button.

Static Form Control

- There might be a situation when you need to place just plain text next to a form label instead of a form control. You can do this within a horizontal form layout by using the
- `.form-control-static` class on a `<p>` element,



The image shows a horizontal form layout. It features two labels, "Email" and "Password", positioned to the left of their respective input fields. The "Email" label is followed by the text "harrypotter@mail.com". The "Password" label is followed by a text input field containing the placeholder text "Password". Below these fields is a checkbox labeled "Remember me". At the bottom of the form is a blue "Login" button.

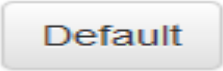






Email harrypotter@mail.com

Password Password

☐ Remember me

Login

Button

Button	Class	Description
	btn btn-default	Default gray button with gradient.
	btn btn-primary	Provides extra visual weight to indicate primary action button in a set of buttons.
	btn btn-info	Can be used as an alternative to the default button.
	btn btn-success	Indicates a successful or positive action.
	btn btn-warning	Indicates caution should be taken with this action.
	btn btn-danger	Indicates a dangerous or potentially negative action.
	btn btn-link	Deemphasize a button by making it look like a link while maintaining button behavior.

- The Bootstrap jumbotron component provides an excellent way to showcase the key content or information on a web page. Just wrap your featured content like heading, descriptions etc. in a `<div>` element and apply the class `.jumbotron` on it.

```
<div class="jumbotron">
```

```
  <h1>Learn to Create Websites</h1>
```

```
  <p>In today's world internet is the most popular way...</p>
```

```
  <p><a href="#" class="btn btn-primary btn-lg">Learn more</a></p>
```

```
</div>
```



Glyphicons

- Glyphicons are icon fonts which you can use in your web projects.
- Bootstrap provides 260 glyphicons from the Glyphicons Halflings set.
- Glyphicons can be used in text, buttons, toolbars, navigation, forms, etc.
- glyphicons can be found within the *fonts* folder
- Associated CSS rules are present within bootstrap.css and bootstrap-min.css
- ``

Examples

Envelope icon: 

Envelope icon as a link: 

Search icon: 

Search icon on a button: 

Search icon on a styled button: 

Print icon: 

Print icon on a styled link button: 

WebAPI

Introduction

- The ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices.



Web Api

- Use WebApi If you need to expose the data/information of your application to your clients and other people

For example,

- A mobile application requires a service.
- HTML 5 requires a service.
- Desktop PC and tablets require services.

Currently most device apps require Web API services.

Web Api

- The ASP. Net Web API Framework leverages both web standards such as JSON and XML
- Core concepts of Web API is similar to ASP. Net MVC such as routing and controllers.

HTTP Actions

GET (Read)

Retrieves the representation of the resource.

PUT(Update)

Update an existing resource.

POST (Create)

Create new resource.

DELETE (Delete)

Delete an existing resource.

[HttpGet]

- public List<Product> Get()
- public Product Get(int id)

[HttpPost]

- public List<Product> Post(Product p)

[HttpPut]

- public void Put(int id, Product p)

[HttpDelete]

- public List<Product> Delete(int id)

Product Model

```
public class Product
{
    public int pid { get; set; }
    public string pname { get; set; }
    public int price { get; set; }
    public int qty { get; set; }

}
```

Initialize Model in Web API

```
List<Product> li = new List<Product>()
{
    new Product(){ pid=100, pname="Books", price=530, qty=5},
    new Product(){ pid=200, pname="CD", price=100, qty=8},
    new Product(){ pid=300, pname="Toys", price=500, qty=12},
    new Product(){ pid=400, pname="Mobile", price=1000, qty=2},
    new Product(){ pid=500, pname="Bag", price=1200, qty=1},
};
```

Get Method to return product model

```
public List<Product> Get()
{
    return li;
}

// GET api/values/5
public Product Get(int id)
{
    return li.Find(t => t.pid == id);
}
```

Method to Insert

```
// POST api/values  
public List<Product> Post(Product p)  
{  
    li.Add(p);  
    return li;  
}
```

Metho to Delete

```
// DELETE api/values/5
public List<Product> Delete(int id)
{
    li.Remove( li.Find(t => t.pid == id));
    return li;

}
```


Using jQuery to access data

- `<div id="divResult"></div>`
- Enter the Product Id : `<input type="text" id="txtpid"
<hr />`
- `<button id="displayall">displayall</button>`
- `<button id="find">find</button>`
- `<button id="delete">delete</button>`
- `<button id="ShowDiv">ShowInsert/Hide</button>
`
-

- `<div id="addDiv">`
- Enter the pid : `
<input type="text" id="pid" />
`
- Enter the pname : `
<input type="text" id="pname" />
`
- Enter the price : `
 <input type="text" id="price" />
`
- Enter the qty : `
 <input type="text" id="qty" />
`
- `<button id="insert">insert</button>`
- `</div>`

Display All Item

```
$("#displayall").click(function () {  
  
    $.ajax({  
        url: 'api/Values/',  
        type: 'GET',  
        dataType: 'json',  
        success: function (data) {  
            WriteResponse(data);  
        },  
        error: function (x, y, z) {  
            alert(x + '\n' + y + '\n' + z);  
        }  
    });  
});
```

Logic to create table

```
function WriteResponse(product) {  
    var strResult =  
    "<table><th>pid</th><th>pname</th><th>price</th><th>qty</th>";  
    $.each(product, function (index, product) {  
        strResult += "<tr><td>" + product.pid + "</td><td> " +  
product.pname + "</td><td>" + product.price + "</td><td>" + product.qty +  
"</td></tr>";  
    });  
    strResult += "</table>";  
    $("#divResult").html(strResult);  
}
```

Find

```
$("#find").click(function () {  
    var id = $('#txtpid').val();  
    $.ajax({  
        url: 'api/Values/' + id,  
        type: 'GET',  
        dataType: 'json',  
        success: function (data) {  
            ShowEmployee(data);  
        },  
        error: function (x, y, z) {  
            alert(x + '\n' + y + '\n' + z);  
        }  
    });  
});
```

```
function ShowEmployee(product) {  
    if (product != null) {  
        var strResult = "<table><th>pid</th><th>pname</th><th>price</th><th>qty</th>";  
        strResult += "<tr><td>" + product.pid + "</td><td> " + product.pname + "</td><td>" +  
product.price + "</td><td>" + product.qty + "</td></tr>";  
        strResult += "</table>";  
        $("#divResult").html(strResult);  
    }  
    else {  
        $("#divResult").html("No Results To Display");  
    }  
}
```

Delete

```
$("#delete").click(function () {  
    var id = $('#txtpid').val()  
  
    $.ajax({  
        url: 'api/Values/' + id,  
        type: 'DELETE',  
        contentType: "application/json;charset=utf-8",  
        success: function (data) {  
            WriteResponse(data);  
        },  
        error: function (x, y, z) {  
            alert(x + '\n' + y + '\n' + z);  
        }  
    });  
});
```

```
$("#insert").click(function () {  
    var employee = {  
        pid: $('#pid').val(),  
        pname: $('#pname').val(),  
        price: $('#price').val(),  
        qty: $('#qty').val()  
    };  
});
```


Contd.. insert

```
$.ajax({  
    url: 'api/Values/',  
    type: 'POST',  
    data: JSON.stringify(employee),  
    contentType: "application/json;charset=utf-8",  
    success: function (data) {  
        WriteResponse(data);  
    },  
    error: function (x, y, z) {  
        alert(x + '\n' + y + '\n' + z);  
    }  
});  
});
```


Routing

- In MVC 5, to have more control over the URIs of a Web application, we can implement the route definition along with the action methods in the controller class, using *attributes*

```
[Route("Products/Electronics/{id}")]
```

```
public ActionResult GetElectronicItems(string id)
```

```
{  
    ViewBag.Id = id;  
    return View();  
}
```

To enable attribute based routing,
Add **routes.MapMvcAttributeRoutes** method in
the RouteConfig file.

Route Prefix

- If we have multiple action methods in a controller all using the same prefix we can use `RoutePrefix` attribute on the controller instead of putting that prefix on every action method.

```
[RoutePrefix("Products")]
public class ProdController : Controller
{
    [Route("Electronics")]
    public ActionResult Elec()
    {
        //Your code here
    }
}
```

URL:
<http://localhost:8080/Products/Electronics>