Authentication and Authorization in MVC

ASP.NET SECURITY

Security is the crucial part of any web base application. All the web based application must be secure from every point of view because confidential data process in web site so unauthorized access should not be happen.

Authentication and Authorization

1. **Windows authentication**
2. **Form-based authentication**
3. **Passport authentication**

Authentication It is the process in which developer ensure the identity of the user on the basis of login information first user authenticate whether the authorize user is accessing the web site or not

Authorization Every user has a different level of access permission like admin has the different and user has different rights. So in authorization process it is ensure whether a particular user granted to access any resource or not. It is kind of permission to access the particular resource

A web.config file is xml based configuration file with the help of this file we can change the behavior of asp.net and also use for security compliances

1. **Windows authentication** It is default authentication mode in asp.net in which user provides the login information and submit the form then information check if information is correct then user gets the form in which user id, password and key is mentioned. It is more useful in intranet environment because in the intranet we have a network domain and all user have windows account based on the level permission grant to user for

accessing the resources in which request directly goes to IIS [Internet information services] to provide the authentication process.

2. **Form-based authentication** In which session management techniques are involved in which user provide the credential either credential are store in database or in text file after proper authentication of user id and password stored in the cookies for in that session so user activity will be trace on the basis of information which is stored in cookies

3. **Passport authentication** it is the centralized service provided by Microsoft it allow user to create a single registration and the same name and password can be use in any area of website where passport authentication is implemented like MSN and Hotmail if you have user id and password of MSN you can access the Hotmail also due to passport authentication

What does federated user mean?

What Is Federated Login. Federated login enables users to use a single authentication ticket/token to obtain access across all the networks of the different IT systems. As a result, once the identity provider's authentication is complete, they now also have access to the other federated domains.

BUNDLING AND MINIFICATION

Bundling and minification techniques were introduced in MVC 4 to improve request load time. Bundling allow us to load the bunch of static files from the server into one http request.

Advantage of BUNDLING and MINIFICAITON

1. It improves the web page load time.
2. This is the performance optimization technique

Suppose in a particular web page 25 css and 30 Java Script files so total 55 files are there so for load complete web page browser required **55 http load request**. Bundle compress or combine the multiple file in single file so single file can be download in single request

Minification reduces the file size so overall load of the website will be reducing. So how the file in minimize in minificaiton process it removes the comment, Remove white spaces, Shorten Variable name so after doing these sort of minification we received the minified version of java script and css files

Jscompressor.com

Bundle Types:

MVC 5 includes following bundle classes in System.web.Optimization namespace:

ScriptBundle: ScriptBundle is responsible for JavaScript minification of single or multiple script files.

StyleBundle: StyleBundle is responsible for CSS minification of single or multiple style sheet files.

DynamicFolderBundle: Represents a Bundle object that ASP.NET creates from a folder that contains files of the same type.

All the above bundle classes are included in *System.Web.Optimization.Bundle* namespace and derived from Bundle class.

Points to Remember :

1. Bundling and Minification minimize static script or css files loading time therby minimize page loading time.
2. MVC framework provides ScriptBundle, StyleBundle and DynamicFolderBundle classes.
3. ScriptBundle does minification of JavaScript files.
4. StyleBundle does minification of CSS files.

Minification techniques were introduced in MVC 4 to improve request load time. Bundling allow us to load the bunch of static files from the server into one http request.

MINIFICAITON

1. It improves the web page load time.
2. This is the performance optimization technique
3. MVC framework provides ScriptBundle, StyleBundle and DynamicFolderBundle classes.
4. ScriptBundle does minification of JavaScript files.
5. StyleBundle does minification of CSS files.

Minification reduces the file size so overall load of the website will be reducing. So how the file in minimize in minificaiton process it removes the comment, Remove white spaces, Shorten Variable name so after doing these sort of minification we received the minified version of java script and css files. After making the javascript or css file we can minified these file so the overall size will be reduce and performance of the website will be increase minification can be done online most of the site are available for minfication process we only have to put the complete java script file and online site will produce the minified form of the java script file and then we can use the minified files for example JSCOMPRESSOR

CURD OPERATION IN MVC

```
@{
    Layout = null;
}
```

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
    <style>
        #first {
            height:20px;
            width:70px;
            float:left;
            line-height:20px;
            border:1px solid red;
            font-size:18px;
            color:red;
        }
        #second {
            height:20px;
            width:70px;
            float:left;
            line-height:20px;
```

```
            border:1px solid red;
            font-size:18px;
            color:red;
        }
    </style>

</head>
<body>
    <form action="/Home/Index" method="post">
    <table>
        <tr>
            <td>Enter Name</td>
            <td><input type="text" name="EMP_NAME" /></td>
        </tr>
        <tr>
            <td>Enter City</td>
            <td><input type="text" name="EMP_CITY" /></td>
        </tr>
        <tr>
            <td>Enter AGE</td>
            <td><input type="text" name="EMP_AGE" /></td>
        </tr>
```

```
        <tr>
            <td>Enter Salary</td>
            <td><input type="text" name="EMP_SALARY" /></td>
        </tr>
        <tr>
            <td><input type="submit" value="Save Record"
name="insert" /></td>

        </tr>

    </table>
        </form>
    <div>
        <div id="first">
            Create
        </div>
        <div id="second">
          <a href="/Home/Index">Display</a>
        </div>
        <div id="third">

        </div>
        <div id="fourth">
```

```
@{
    if(ViewBag.tab!="")
    {
    @Html.Raw(ViewBag.tab )
    }
    }
        </div>
    </div>
</body>
</html>
```

Home controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ENTITYFRAMEWORKDEMO.Models;
namespace ENTITYFRAMEWORKDEMO.Controllers
{
    public class HomeController : Controller
    {
```

```
        //
        // GET: /Home/
        //create DataBase Object as Global
        ENTITYDATABASEEntities db = new
ENTITYDATABASEEntities();
        public ActionResult Index()
        {
            List<TBL_EMP> Lst = db.TBL_EMP.ToList();
            string table = "<table><tr
style='background:orange;color:white;font-size:20px;text-
align:center;font-weight:bold;'><th>Emplyee
Id</th><th>Emplyee Name</th><th>Emplyee City</th><th>Emplyee
Age</th><th>Salary</th><th>Delete</th><th>Update</th><th>Det
ails</th></tr>";
            for (int i = 0; i < Lst.Count; i++)
            {
                table += "<tr
style='background:red;color:white;font-size:20px;text-
align:center;font-weight:bold;'><td>" + Lst[i].EMP_ID +
"</td><td>" + Lst[i].EMP_NAME + "</td><td>" +
Lst[i].EMP_CITY + "</td><td>" + Lst[i].EMP_AGE + "</td><td>"
+ Lst[i].EMP_SALARY + "</td><td><a
href='/Home/Delete?empid=" + Lst[i].EMP_ID +
"'>Delete</a></td><td><a
href='/home/updatepage?update="+Lst[i].EMP_ID+"'>Update</a><
/td><td><a
href='/Home/Details?dt="+Lst[i].EMP_ID+"'>Details</a></td></
tr>";
```

```
            }
            table += "</table>";
            ViewBag.tab = table;
            return View();
        }
        [HttpPost]
        public ActionResult Index(TBL_EMP emp)
        {
              db.TBL_EMP.Add(emp);
                db.SaveChanges();
                Response.Write("<script>alert('Records saved
into database ');</script>");
                Response.Redirect("/Home/Index");
                return View();
        }
        public ActionResult Delete()
        {
         int id =
int.Parse(Request.QueryString["empid"].ToString());
          //var del = new TBL_EMP { EMP_ID = id };
         // db.TBL_EMP.Attach(del);
         TBL_EMP emp = db.TBL_EMP.SingleOrDefault(a =>
a.EMP_ID == id);
```

```
            db.TBL_EMP.Remove(emp);
            db.SaveChanges();
             Response.Write("<script>alert('Record deleted
')</script>");
             Response.Redirect("/Home/Index");
              return View();
        }
        public ActionResult Details()
        {
             int id =
int.Parse(Request.QueryString["dt"].ToString());
             TBL_EMP emp = db.TBL_EMP.SingleOrDefault(a =>
a.EMP_ID == id);
             return View(emp);
        }
        [HttpGet]
        public ActionResult updatepage()
        {
             int id =
int.Parse(Request.QueryString["update"].ToString());
             TBL_EMP emp = db.TBL_EMP.SingleOrDefault(a =>
a.EMP_ID == id);
             return View(emp);
        }
```

```
        [HttpPost]
        public ActionResult updatepage(TBL_EMP emp)
        {
            //int id =
int.Parse(Request.QueryString["update"].ToString());
            TBL_EMP  em = db.TBL_EMP.First(x => x.EMP_ID ==
emp.EMP_ID);
            em.EMP_NAME = emp.EMP_NAME;
            em.EMP_SALARY = emp.EMP_SALARY;
            em.EMP_AGE = emp.EMP_AGE;
            em.EMP_CITY = emp.EMP_CITY;
            db.SaveChanges();
            Response.Redirect("/home/index");
            return View();
        }
    }
}
```

Details

```
@{
    Layout = null;
}

<!DOCTYPE html>
```

```
@using ENTITYFRAMEWORKDEMO.Models
@model TBL_EMP
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Details</title>
</head>
<body>
    <div>
        <table>

            <tr>
                <td>Employee Name</td>
                <td>
                    <input type="text"
value="@Model.EMP_NAME" readonly="true"/>
                </td>
                <td>Emplyee City</td>
                <td>
                    <input type="text"
value="@Model.EMP_CITY" />
                </td>
                <td>Employee Age</td>
```

```
                <td>
                    <input type="text"
value="@Model.EMP_AGE" />
                </td>
                <td>Salary</td>
                <td>
                    <input type="text"
value="@Model.EMP_SALARY" />
                </td>
            </tr>

        </table>
    </div>
</body>
</html>
<!DOCTYPE html>
```

UPDRADE

```
@using ENTITYFRAMEWORKDEMO.Models
@model TBL_EMP
<html>
```

```
<head>
    <meta name="viewport" content="width=device-width" />
    <title>updatepage</title>
</head>
<body>
    <div>
        <form action="/home/updatepage" method="post">
        <table>
             <tr><td>Emp Id</td><td><input type="text"
name="emp_id" value="@Model.EMP_ID" readonly="true"
/></td></tr>
            <tr><td>Name</td><td><input type="text"
name="emp_name" value="@Model.EMP_NAME" /></td></tr>
            <tr><td>Age</td><td><input type="text"
name="emp_city"  value="@Model.EMP_CITY"/></td></tr>
            <tr><td>City</td><td><input type="text"
name="emp_age" value="@Model.EMP_AGE" /></td></tr>
            <tr><td>Salary</td><td><input type="text"
name="emp_salary" value="@Model.EMP_SALARY"/></td></tr>
            <tr><td><input type="submit" /></td></tr>

        </table>
            </form>
    </div>
```

```
</body>
</html>
```

Delete separately using LAMBDA EXPRESSION

```
public ActionResult Delete()
        {
         int id =
int.Parse(Request.QueryString["empid"].ToString());
          //var del = new TBL_EMP { EMP_ID = id };
         // db.TBL_EMP.Attach(del);
         TBL_EMP emp = db.TBL_EMP.SingleOrDefault(a =>
a.EMP_ID == id);
          db.TBL_EMP.Remove(emp);
          db.SaveChanges();
           Response.Write("<script>alert('Record deleted
')</script>");
          Response.Redirect("/Home/Index");
           return View();
                                    }
```

Entity Framework

it is an open source ORM(Object Relational mapper) Framework developed by MicroSoft .EntityFramework is also used internal ADO.Net .

Object Relational Mapping framework automatically creates classes based on database tables. It can also automatically generate necessary SQL to create database tables based on classes

=>How we can performs CRUD Operation By Using EF .

---

1)Features of EF :-

1) **Configuration**

Any kind of configuration is not required, automatic it establish the connection string with database

2) **Command**

Command is not required

3) **Object oriented approach**

Every table or data base represent in the form of class.

Important method for CRUD operation

---

1. for insertion

Add(above 4.0) or AddObject (below 4.0)

2. For save the record

SaveChanges(Object of Table)

Important Steps of storing record in database using EFWORK

1. Entity Frame model creation save and rebuild
2. Add Controller
3. Using name space <ProjectName>.models
4. Create the object
   ```
   ENTITYDATABASEEntities db = new ENTITYDATABASEEntities();
   ```
5. Create post index method with parameter(TBL_emp emp)
6. db.TBL_EMP.Add(emp);
7. db.SaveChanges(); //commit records permanent storage
8. response.redirect("/home/index"); without refresh record will be display on the screen.

Display Record using Dynamic table

1. if you want to display records on view of html
2. Bind table records with List
   ```
   List<TBL_EMP> lst = db.TBL_EMP.ToList();
   ```
3. Create a table header
   ```
   string table = "<table border='1' cellpadding='10' cellspacing='10' style='color:red'><tr><th>Employee Id</th><th>Employee Name</th><th>Employee City</th><th>Employee Age</th><th>Employee Salary</tr>";
   ```
4. Execute a Loop
5. for (int i = 0; i < lst.Count; i++)

6. table+="<tr><td>"+lst[i].EMP_ID+"</td><td>"+lst[i].EMP_NAME+"</td><td>"+lst[i].EMP_CITY+"</td><td>"+lst[i].EMP_AGE+"</td><td>"+lst[i].EMP_SALARY+"</td></tr>";
7. Store dynamic table in ViewBag
   `ViewBag.tbl = table;`

8. Now use Razor Block in View

```
@{
        if(ViewBag.tbl!="")
        {
            @Html.Raw(ViewBag.tbl);
        }
        }
```

- **Deletion**

```
public ActionResult Delete()
        {
          int                          id                          =
int.Parse(Request.QueryString["empid"].ToString());
           //var del = new TBL_EMP { EMP_ID = id };
          // db.TBL_EMP.Attach(del);
          TBL_EMP   emp   =   db.TBL_EMP.SingleOrDefault(a   =>
a.EMP_ID == id);
           db.TBL_EMP.Remove(emp);
           db.SaveChanges();
            Response.Write("<script>alert('Record      deleted
')</script>");
```

```
            Response.Redirect("/Home/Index");
             return View();
              }
```

Index Coading

```
public ActionResult Index()
        {
            List<TBL_EMP> Lst = db.TBL_EMP.ToList();
            string table = "<table><tr
style='background:orange;color:white;font-size:20px;text-
align:center;font-weight:bold;'><th>Emplyee
Id</th><th>Emplyee Name</th><th>Emplyee City</th><th>Emplyee
Age</th><th>Salary</th><th>Delete</th><th>Update</th><th>Det
ails</th></tr>";
            for (int i = 0; i < Lst.Count; i++)
            {
                table += "<tr
style='background:red;color:white;font-size:20px;text-
align:center;font-weight:bold;'><td>" + Lst[i].EMP_ID +
"</td><td>" + Lst[i].EMP_NAME + "</td><td>" +
Lst[i].EMP_CITY + "</td><td>" + Lst[i].EMP_AGE + "</td><td>"
+ Lst[i].EMP_SALARY + "</td><td><a
href='/Home/Delete?empid=" + Lst[i].EMP_ID +
"'>Delete</a></td><td><a href='#'>Update</a></td><td><a
href='/Home/Details?dt="+Lst[i].EMP_ID+"'>Details</a></td></
tr>";
            }
            table += "</table>";
```

```
            ViewBag.tab = table;
            return View();
             }
```

- **Details**

```
public ActionResult Details()
        {
             int                    id                    =
int.Parse(Request.QueryString["dt"].ToString());
             TBL_EMP  emp  =  db.TBL_EMP.SingleOrDefault(a  =>
a.EMP_ID == id);
            return View(emp);
        }
```

Veiw of Details

```
@{
    Layout = null;
}

<!DOCTYPE html>
@using ENTITYFRAMEWORKDEMO.Models
@model TBL_EMP
<html>
<head>
```

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

            <tr>
                <td>Employee Name</td>
                <td>
                    <input type="text" value="@Model.EMP_NAME" readonly="true"/>
                </td>
                <td>Emplyee City</td>
                <td>
                    <input type="text" value="@Model.EMP_CITY" />
                </td>
                <td>Employee Age</td>
                <td>
                    <input type="text" value="@Model.EMP_AGE" />
                </td>
```

```
                <td>Salary</td>
                <td>
                    <input type="text"
value="@Model.EMP_SALARY" />
                </td>
            </tr>

        </table>
    </div>
</body>
</html>
```

Entity Frame work

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ENTITYFRAMEWORKDEMO.Models;
namespace ENTITYFRAMEWORKDEMO.Controllers
```

```
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        //create DataBase Object as Global
        ENTITYDATABASEEntities db = new ENTITYDATABASEEntities();
        public ActionResult Index()
        {
            List<TBL_EMP> Lst = db.TBL_EMP.ToList();
            string table = "<table><tr style='background:orange;color:white;font-size:20px;text-align:center;font-weight:bold;'><th>Emplyee Id</th><th>Emplyee Name</th><th>Emplyee City</th><th>Emplyee Age</th><th>Salary</th><th>Delete</th><th>Update</th><th>Details</th></tr>";
            for (int i = 0; i < Lst.Count; i++)
            {
                table += "<tr style='background:red;color:white;font-size:20px;text-align:center;font-weight:bold;'><td>" + Lst[i].EMP_ID + "</td><td>" + Lst[i].EMP_NAME + "</td><td>" + Lst[i].EMP_CITY + "</td><td>" + Lst[i].EMP_AGE + "</td><td>" + Lst[i].EMP_SALARY + "</td><td><a href='/Home/Delete?empid="+Lst[i].EMP_ID+"'>Delete</a></td><
```

```
td><a                          href='#'>Update</a></td><td><a
href='#'>Details</a></td></tr>";
            }
            table += "</table>";
            ViewBag.tab = table;
            return View();
        }
        [HttpPost]
        public ActionResult Index(TBL_EMP emp)
        {
              db.TBL_EMP.Add(emp);
                db.SaveChanges();
                Response.Write("<script>alert('Records  saved
into database ');</script>");
                Response.Redirect("/Home/Index");
                return View();
        }
        public ActionResult Delete()
        {
         int                          id                          =
int.Parse(Request.QueryString["empid"].ToString());
          var del = new TBL_EMP { EMP_ID = id };
          db.TBL_EMP.Attach(del);
```

```
            db.TBL_EMP.Remove(del);
            db.SaveChanges();

             Response.Write("<script>alert('Record     deleted
')</script>");
            Response.Redirect("/Home/Index");
             return View();
        }
    }
}
```

View code

```
@{
    if(ViewBag.tab!="")
    {
    @Html.Raw(ViewBag.tab )
    }
    }
<col width='130'><col width='130'>
```

FILE UPLOAD PROCEDURE FROM CLIENT TO SERVER

STEP 1.

View Part

```
<input type="file"  name="fupic" class="form-control" /><br />
```

Controller part

```
HttpPostedFileBase FUPic = Request.Files["fupic"];

string File = FUPic.FileName; //Only name will be store in FileName
```

HttpPostedFileBase is a class provide the access to file which is uploaded by the client

Note: Here “fupic” is the name in html file control

Step 2

Create a folder in Server

Step 3

```
String n = File.FileName;
```

```
            if (n != null)
            {

file.SaveAs(Server.MapPath("~/Content/Profiles/"+file.FileNa
me));
            }
```

Filters

Filters are basically use to perform any logic before or after an action method is being called so according to the need of project a single or multiple filters can be applied in a controller. There is a provision of built-in filter and custom filters in MVC.

Filter general meaning though which something has to pass before actual processing take place it change the flow of execution.

List of common Built-in Action Filters

S.no	Name of Filter	Purpose
1.	OutputCache	It will cache the last state and proceed according to the condition specified in outputcache
2.	Authorize	Only authorized user will access the action method no anonymous user are permitted

3.	ValidateInput	Turn on or off request validation
4.	ValidateAntiForgeryToken	Help to prevent the cross site request forgeries.
5.	HandleError	Using in Excepiton handling cases

Example of OutputCache Action Filter

```
[OutputCache(Duration=15)]
        public ActionResult Index()
        {

            string dt = DateTime.Now.ToLongTimeString();
            ViewBag.dtime = dt;
            return View();
        }
```

Example of OutputCache Authorize Filter

```
[Authorize]
        public ActionResult Process()
        {
            return View();
        }
```

For understanding the concept of validate input one thing very important that tags cannot be submit in the form due to high security reasons because due to tags some unauthorized data can be access using script tags by default validate input is true

But manually it can be set as false.

Example of OutputCache ValidateInputFilter

```
[HttpPost]
  [ValidateInput(false)]
  public ActionResult Process(string txtname)
  {
      ViewBag.result = txtname;
      return View();
  }
```

Very Important ActionFilter

Some illegal user can copy the code from one site and paste or integrate the code in their site and can submit the data in my site so we can prevent this kind of copied by ValidateAntiForgeryToken

STEAL CODE

```
<form action="http://localhost:1336/Home/Process"
method="post">

    Enter Project Name <input type="text" name="txtname" />

    <input type="submit" id="btn" value="submit" />
```

```
</form>
```

So it a two step process to prevent

Step1

```
[HttpPost]
        [ValidateInput(false)]
        [ValidateAntiForgeryToken()]
        public ActionResult Process(string txtname)
        {
            ViewBag.result = txtname;
            return View();
        }
```

Step2

```
<form action="/Home/Process" method="post">
            @Html.AntiForgeryToken();

            Enter Project Name <input type="text"
name="txtname" />
            <input type="submit" id="btn" value="submit" />

        </form>
```

HTML HELPER CLASSES IN ASP.NET MVC

We can build an ASP.NET MVC application without using them, but HTML Helpers helps in the rapid development of a view. HTML Helpers are more lightweight as compared to ASP.NET Web Form controls as they do not use ViewState and do not have event models.

In ASP.Net web forms, developers are using the toolbox for adding controls on any particular page. However, in ASP.NET MVC application there is no toolbox available to drag and drop HTML controls on the view. In ASP.NET MVC application, if you want to create a view it should contain HTML code. So those developers who are new to MVC especially with web forms background finds this a little hard.

To overcome this problem, ASP.NET MVC provides HtmlHelper class which contains different methods that help you create HTML controls programmatically

HTML Helpers are categorized into three types:

1. Inline HTML Helpers
2. Built-in HTML Helpers
3. Custom HTML Helpers

There are different types of helper methods.

- **Createinputs** – Creates inputs for text boxes and buttons.
- **Createlinks** – Creates links that are based on information from the routing tables.
- **Createforms** – Create form tags that can post back to our action, or to post back to an action on a different controller.

The following is the list of Html Helper controls.

- Html.Beginform
- Html.EndForm
- Html.Label
- Html.TextBox
- Html.TextArea

- Html.Password
- Html.DropDownList
- Html.CheckBox
- Html.RedioButton
- Html.ListBox
- Html.Hidden

Below are Strongly Type Html Helper methods, this will allow us to check compile time errors. We get Model's Property intelligence at Runtime.

- Html.LabelFor
- Html.TextBoxFor
- Html.TextAreaFor
- Html.DropDownListFor
- Html.CheckBoxFor
- Html.RadioButtonFor
- Html.ListBoxFor
- Html.HiddenFor

Example of Using HTML HELPER

```
@using (Html.BeginForm("Index", "Home", FormMethod.Post))

{

   <br />


      @Html.Label("Enter Emp Name")

      @Html.TextBox("EmpName")

   <br />


      @Html.Label("Enter Emp Designation")

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

```
    <br />
      @Html.Label("Enter Employee Email Address")
      @Html.TextBox("EmpEmail")
    <br />
}
```

```
                @Html.Label("Enter Name")
                @Html.TextBox("txt1")
                <br />
                @Html.TextBox("txt2")
                <br />
                ColdDrink
                @Html.CheckBox("Colddrink", false)
```

HTML HELPER WITH CSS and other attribute and property

Example No 1

```
@Html.TextBox("txtbox","",new{
                    style="font-size:24px; background-
color:lightgreen"
    })
```

- Here new is behaving like the anonymous function

Example No 2

```
@Html.TextBox("txtbox","",new{
                    style="font-size:24px; background-
color:lightgreen",
                    title="Enter Company information",
                    placeholder="Fill these field"
    })
```

- The short hint is displayed in the input field before the user enters a value.
- Title is the hint which display when the user hold the curser

Example 3 Third How to call the class in HTML HELPER

```
@Html.TextBox("txt3","",new{
            @class="format"
        })
```

TextArea Using HTML HELPER

```
@Html.TextArea("info","",10,20,null)
```

HTML Element

Example

TextBox

@Html.TextBox("Textbox1", "val")
Output: <input id="Textbox1" name="Textbox1" type="text" value="val" />

TextArea

@Html.TextArea("Textarea1", "val", 5, 15, null)
Output: <textarea cols="15" id="Textarea1" name="Textarea1" rows="5">val</textarea>

Password

@Html.Password("Password1", "val")
Output: <input id="Password1" name="Password1" type="password" value="val" />

Hidden Field

@Html.Hidden("Hidden1", "val")
Output: <input id="Hidden1" name="Hidden1" type="hidden" value="val" />

CheckBox

@Html.CheckBox("Checkbox1", false)
Output: <input id="Checkbox1" name="Checkbox1" type="checkbox" value="true" /> <input name="myCheckbox" type="hidden" value="false" />

RadioButton

@Html.RadioButton("Radiobutton1", "val", true)
Output: <input checked="checked" id="Radiobutton1" name="Radiobutton1" type="radio" value="val" />

Drop-down list

@Html.DropDownList (“DropDownList1”, new SelectList(new [] {"Male", "Female"}))
Output: <select id="DropDownList1" name="DropDownList1"> <option>M</option> <option>F</option> </select>

Multiple-select

Html.ListBox(“ListBox1”, new MultiSelectList(new [] {"Cricket", "Chess"}))
Output: <select id="ListBox1" multiple="multiple" name="ListBox1"> <option>Cricket</option> <option>Chess</option> </select>

```
<body>
    <div>
        <h2>Html Helper Classes</h2>
        <hr />
        <div style="margin-left:300px;">
       @using (Html.BeginForm("index","home",FormMethod.Post))
       {
           @Html.TextBox("txt1", "Enter", new { placeholder = "Enter your name" })
           <br />
           <br />
           @Html.TextBox("txt2", null, new { placeholder = "Enter email " })
           <br />
           <br />
```

```
@Html.RadioButton("Gender", "Male")
@Html.Label("Male");

      <br />
@Html.RadioButton("Gender", "Female")
@Html.Label("Female");
<br />

<br />
@Html.CheckBox("Check")
@Html.Label("I agree the terms and condtions")

<br />
@Html.TextArea("txtmsg", "Type Here")
<br />
@Html.Label("Enter Password ")

<br />
@Html.Password("txtpass")
<br />

<input type="submit" value="Submit" />

}
```

```
            </div>
        <hr />
            @{
                if(IsPost)
                {

                    @Html.Raw(ViewBag.table);
                }
            }
    </div>
</body>
```

```
[HttpPost]
        public ActionResult Index(string txt1, string txt2, string
Gender,
string Check, string msg, string pass)
        {
            string tbl = "<table border='1' cellpadding='10'
cellspacing='10'><tr><td>Information</td></tr>";

            tbl += "<tr><td>Name</td><td>" + txt1 + "</td></tr>";
            tbl += "<tr><td>Email</td><td>" + txt2 + "</td></tr>";
            tbl += "<tr><td>Gender</td><td>" + Gender + "</td></tr>";
            tbl += "<tr><td>Disclaimer</td><td>" + Check +
"</td></tr>";

            tbl += "</table>";
            ViewBag.table = tbl;



            return View();
        }
```

Array in Jquery or Java script

---

var n = [];

var n = ["C++", "C", "Java", "JavaScript", "Asp.net", "Python", "PHP"];

Ajax Implementation Using JSON and Jquery

Traditional web applications rely on synchronous method calls to process data. The entire web page is posted to the server each time data is submitted through the page, causing performance delays.

This problem can be overcome by using AJAX. AJAX allows web applications to call methods asynchronously. Instead of posting the entire page to the web server, only the required data is posted. This improves the overall performance of the web applications.

As we all know, AJAX means Asynchronous JavaScript and XML. It is a client-side script that communicates to and from a server/database without the need for a postback or a complete page refresh. The Ajax speeds up response time.

Implementation of Ajax can be done in two way in ASP.Net Application

- using Update Panel and,
- using jQuery

JSON[Java Script Object Notation]

What is the use of Json ?

JSON is very commonly used concept in MVC and other technology. It is a data interchange medium and a very light weighted code between client and server. It is language independent so any technology and language can use the concept of JSON. It is mainly use when AJAX call and functionality implement in project so it provide the partial rendering without refreshing the entire heavy page of web page it can transfer the data to server as well it can receive the data from server.

JsonResult is actually a special ActionResult , which suggest the View Engine that the object of JSON type will be returned rather than normal HTML

1. JSON objects are surrounded by curly braces {}.
2. JSON objects are written in key/value pairs.
3. Keys and values are separated by a colon.

Uses of JSON

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

Characteristics of JSON

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.

```
$(document).ready(function () {

            jsonobj = { "name": "Raman", "Age": "24", "Marks": "90" };
```

```
var res = "";

res += "Name " + jsonobj.name + "<br>";

res += "Age " + jsonobj.Age + "<br>";

res += "Marks " + jsonobj.Marks;

$("#div1").html(res);     });
```

JSON OBJECT AS ARRAY

```
jsonobj = [{ "Name": "Mahira", "Age": "24", "City": "Mumbai" }, { "Name": "Raj",
"Age": "34", "City": "Lucknow" }];

for(i=0;i<=1;i++)

{

    res += jsonobj[i].Name + "<br>";

    res += jsonobj[i].Age + "</br>";

    res += jsonobj[i].City + "</br>";

    res += "_____________________________</br>";

}


   $("#div1").html(res);
```

```
jsonobj = [{ "name": "Raman", "Age": "24", "Marks": "90" }, { "name": "Ram",
"Age": "30", "Marks": "95" }];

            var res = "";

            for (i = 0; i < jsonobj.length; i++)

            {

                res+=(jsonobj[i].name + " " + jsonobj[i].Age + " " +
jsonobj[i].Marks+"<br>");

            }

            $("#div1").html(res);
```

View Coading

```
$.getJSON("/Admin/GetAttendence", {Rn:rollno,Atten:attendance},
function (msg) {

            if (msg == "success")

                alert("saved records");


        })
```

Controller Method

```
public ActionResult GetAttendence(string Rn, string Atten)

        {

            string msg = "success";

            return Json(msg, JsonRequestBehavior.AllowGet);

            //return View();
```

```
        }
```

Insertion and searching Using Json Ajax implementation with page refresh

While creating the function related to Json make sure post function should not be created

```
$(document).ready(function () {
            $("#btn").click(function () {
                var name, marks;
                nam = $("#name").val();
                marks = $("#marks").val();
                $.getJSON("/Home/insertion", { n: nam, m: marks }, function (msg)
{
                    if (msg == "success") {
                        alert('Record Inserted');
                    }
                    else
                        alert('Record not inserted');

                });
```

Controller part coading

```
public ActionResult insertion(string n,string m)
        {
```

```
            int marks = int.Parse(m);
            string msg;

            string q = "insert into tbl_student values('" + n + "','" + marks +
"')";
            bool j = dm.insertupdatedelete(q);
            if (j == true)
                msg = "success";
            else
                msg="failure";



            return (Json(msg, JsonRequestBehavior.AllowGet));


        }
```

Search Record

```
$("#searchbtn").click(function () {
                var ssid = $("#sid").val();
                $.getJSON("/Home/search", { searchid: ssid }, function (msg) {
                    if (msg == "find") {
                        $("#span1").html("Record Found Successfully");
                    }
                    else {
```

```
                $("#span1").html("Record Not Found Successfully");
            }
        });
    });
```

Controller part coading for search

```
public ActionResult search(string searchid)
        {
            int id = int.Parse(searchid);
            string msg;
            string q="select * from tbl_student where
studentid='"+id+"'";
            DataTable dt = dm.readbulkdata(q);
            if (dt.Rows.Count > 0)
                msg = "find";
            else
                msg = "not found";
            return (Json(msg,
JsonRequestBehavior.AllowGet));



                                    }
```

Lamda Expression

```
static void Main(string[] args)
        {
            Console.WriteLine("lamda expression");
            List<int> lst = new List<int>();
            lst.Add(2);
            lst.Add(3);
            lst.Add(4);
            lst.Add(5);
```

```
            lst.Add(6);

            foreach (int num in lst)
            {
                Console.WriteLine("Value "+num);
            }
            List<int> oddlist = lst.Where(x => x % 2 !=
0).ToList();
            Console.WriteLine("Odd number of the list");
            foreach (int num in oddlist)
            {
                Console.WriteLine("Odd Value " + num);
            }

            Console.ReadKey();
        }
```

```
            List<int> lst = new List<int>();
            lst.Add(2);
            lst.Add(3);
            lst.Add(4);
            lst.Add(5);
            lst.Add(6);


            List<int> listodd = lst.Where(num => num % 2 !=
0).ToList();


            foreach (int n in listodd)
            {
                Console.WriteLine("Value " + n);

            List<string> lst = new List<string> { "Ali",
"Aman", "Rahil", "Deepak", "Ajay" };
            string x = "";
            for (int i = 0; i < lst.Count; i++)
            {
```

```
                x = x+" "+lst.FindAll(str =>
str.StartsWith("A"));
                Console.WriteLine("Name starts with  A "+x);
            }
```

LINQ [Language Integrated Query Language]

LINQ has its own Query Processing Engine. The information returned by a LINQ query is a collection of in-memory object which may be enumerated.

The LINQ concept treats the data source as an Object, rather than a Database. So we may say, its an object that is queried. LINQ may query any type of data source,

1. Whenever we write the sql query it does not show any compile time error but linq show the compile time error if you use wrong syntax
2. Just like sql query linq is also use to fetch the data from data base
3. It is included in the C# that is why it is called integrated query language
4. It was introduce in .NET 3.0
5. It is use to query the data from the various data sources like XML document, data set , web services, Collection Generics, SQL Database.
6.

MVC Architecture

The MVC Architecture is existing in software engineering from the long time. Mostly all the language presently using MVC. MVC is a popular way of organizing your code

MVC stands for Model, View and Controller. MVC separates application into three components - Model, View and Controller.

Any software or application is divided into three part **User Interface** by which user interact to the application and **flow of control and logic** and the **storage**

Model

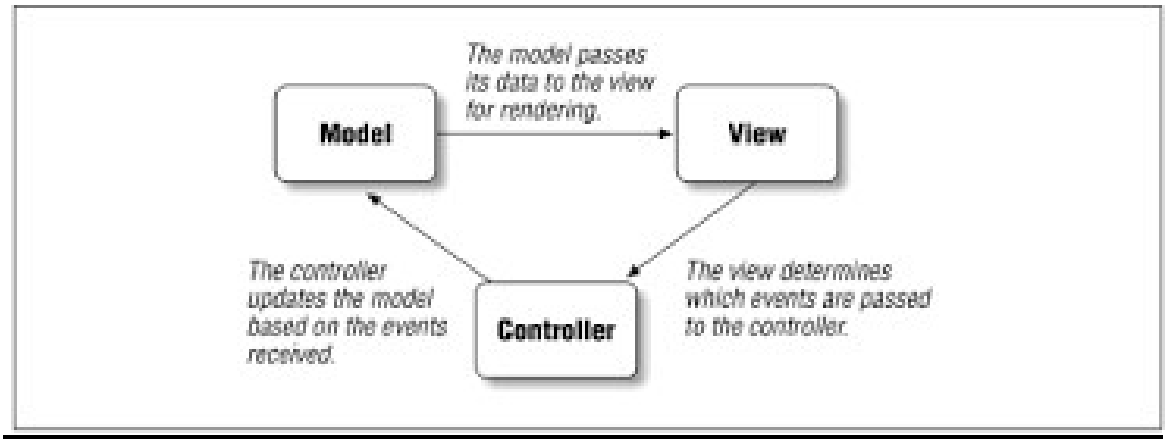
1. It is a data and business logic
2. It maintain the data of the application
3. It provide the mechanism to store the data and retrieve the data whenever it is required
4. It is the essential component of the application
5. Model provides the source of data that manage and control by controller and display on view
6. Data can be store in the file or huge data store in Data base management systems
7. Model respond to controller when any request comes from controller end

View

1. View is the user interface
2. User interact with the help of View
3. It provides all the thing that user can see and respond, like button , check boxes ,
   menus etc
4. View component is used for all the UI logic of the application and these are the components that display the application's user interface (UI)
5. View only interact with the controller not with model whenever any event occur it send to controller

Controller

1. Controller handles user's requests and renders appropriate View with Model data.
2. Controllers act as an interface between Model and View components.
3. Controller co-ordinate with model and view both



View Engine

Before stating the description of View Engine. We discuss one thing that is suppose we create any web form related to HTML TAGS. That web page will be treated as static but suppose we want to add some c# or any other programming language codes in between HTML so it is not possible. So View Engines provides the facility to mix HTML and Programming Codes.

So in Asp.net we have the two options regarding the View Engine

1. **ASPX View Engine or Web Form View Engine (Old)**
2. **Razor View Engine. (Latest)**

Note: Some third party View Engine (like Spark , Nhaml, Brail, SharpTiles, Hasic and many more) support is also available in ASP.NET we give the preference to default Veiw Engine of ASP.NET

Difference between ASPX View Engine and Razor View Engine

s.no	Razor View Engine	ASPX or Web Form View Engine
1	It is advance view engine that was introduce with MVC3	It is the default View Engine for Asp.net MVC and it is included from the beginning.
2	The name space for the Razor View Engine is **System.Web.Razor**	The name space for the Aspx View Engine is **System.Web.Mvc.WebFormViewEngine**
3	The extension of Razor View Engine is **.cshtml (Razor with c#)**	The extension of Aspx View Engine is **.aspx**
4	Razor has new and advance syntax which is easy to understand and usable and provide very clear view and reduce typing.	Comparatively ASPX View Engine Code is complicated
5	Razor use @ symbol to make the code **For Eg @{** **int a=2,b=4,c=0;** **c=a+b;** **}**	Aspx View Engine use **<% and %>** **For Eg <%:** **int a=2,b=4,c=0;** **c=a+b;** **%>**
6	Razor does not support design mode in visual studio mean we cannot see the design without running the application.	While Aspx support the design mode in visual studio we can see the design without running application.
7	Razor View is little bit slow in comparison of Aspx View	Comparatively faster
8	Razor View Engine prevent **Cross Site Scripting Attack**s means it encodes the script or html tags before rendering to view	Aspx View Engine does not prevent
9	Razor Engine Support Test Driven Deployment	Aspx doesn't support Test Driven Deployment
10	The Razor View Engine is compatible with a unit testing framework	Not very much compatible with the unit testing
11	Razor also supports the concept of layout pages	Aspx support the concept of Master Page

Advantages of Razor View Engine

1. It is easy to use and implement
2. Razor helps us to minimize the coding and provide us a fast and fluid coding work flow.
3. Powerful built-in validation of markup that helps us to avoid unwanted runtime exceptions due to errors in the view.
4. The code looks clean.
5. Razor does not require any special tool to write markup. We can also write our markup code with any old plain text editor like Notepad.
6. The @model directive provides a cleaner and more concise way to define a strongly typed model
7. Improves readability of the markup and code

Razor Code Syntax

1. **Razor Expression**

   These are single Line C# Statement
   @Statements

2. **Razor Block**

   These are multiline Statement
   @
   {
   Statements
   }

Note: So due to these Razor engine is capable to identify which is HTML code and What are c# codes