Introduction to ASP.net Ajax

AJAX in ASP.net stands for Asynchronous JavaScript and XML. This technique is used to develop rich web applications that are interactive, responsive, and dynamic in nature.

By the method of receiving and sending data to the server in the background, the web pages are updated using AJAX in ASP.net. Without even reloading the web page, we will be able to update the website part by part as well. This update of the website is done asynchronously using AJAX in ASP.net.

In AJAX, usually, multiple technologies are used together to create these dynamic web pages. Some of these technologies include -

- XSLT
- XHTML
- CSS
- JavaScript
- Document Object Model
- XML
- XMLHttpRequest object

The interactive animation that we see on modern web pages is mainly because of AJAX. In common web pages that do not use AJAX, one needs to reload the entire web page in order to reflect content changes in any parts or all. Now, we are going to discuss the advantages, disadvantages of AJAX in ASP.net and how to transfer data through AJAX in ASP.net.

Advantages of AJAX in ASP.net

Some of these advantages of AJAX in ASP.net are described below -

- Ajax in ASP.net improves the responsiveness and interactivity of a website.
- Ajax in ASP.net helps in reducing the traffic between server and client.
- Ajax in ASP.net reduces the cross-browser issues as it is cross-browser friendly.

- In Ajax in ASP.net, we can use a single web page or SPA to be able to handle several features and apply multi-purpose applications to it.
- In Ajax in ASP.net, we can use APIs or Application Programming Interfaces and because these work seamlessly with JavaScript and HTTP methods, it makes it an enormous advantage to building dynamic web applications.

Disadvantages of AJAX in ASP.net

- The size of a data request is largely increased as all these data requests are URL encoded.
- It highly depends on JavaScript as it is a JavaScript built-in. Therefore, if a user disables JavaScript in the browser, then AJAX stops working.
- Indexing of an AJAX application cannot be done using Google-like search engines.
- Since all the files are downloaded on the client-side in an AJAX application, security is scarce in these applications.
- Within the AJAX, the server information is completely inaccessible.

The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If we create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

The UpdatePanel Control

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control

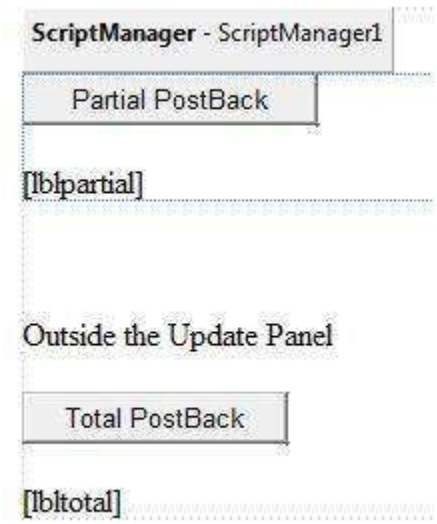
inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

Example

Add an AJAX web form in our application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:



The source file is as follows:

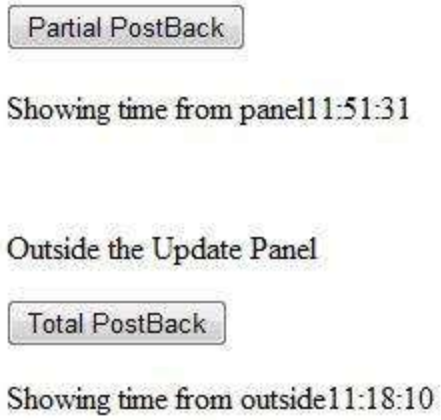
```
<form id="form1" runat="server">
  <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
  </div>

  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
```

```
        <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Partial PostBack"/>
        <br />
        <br />
        <asp:Label ID="lblpartial" runat="server"></asp:Label>
      </ContentTemplate>
    </asp:UpdatePanel>

    <p> </p>
    <p>Outside the Update Panel</p>
    <p>
      <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack" />
    </p>

    <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

Both the button controls have same code for the event handler:

```
string time = DateTime.Now.ToLongTimeString();
lblpartial.Text = "Showing time from panel" + time;
lbltotal.Text = "Showing time from outside" + time;
```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
ContentTemplateContainer	Retrieves the dynamically created template container object and used for adding child controls programmatically.
IsInPartialRendering	Indicates whether the panel is being updated as part of the partial post back.
RenderMode	Shows the render modes. The available modes are Block and

	Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control

The following table shows the methods of the update panel control:

Methods	Description
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content of an UpdatePanel control.

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

UpdateMode	ChildrenAsTriggers	Effect
Always	False	Illegal parameters.
Always	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back.

Conditional	False	UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh.
Conditional	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh.

The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DynamicLawet="true"
AssociatedUpdatePanelID="UpdatePanel1" >

  <ProgressTemplate>
    Loading...
  </ProgressTemplate>

</asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

Properties	Description
AssociatedUpdatePanelID	Gets and sets the ID of the update panel with which this control is associated.
Attributes	Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control.
DisplayAfter	Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.
DynamicLawet	Indicates whether the progress template is dynamically rendered.
ProgressTemplate	Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.

Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

Methods	Description
GetScriptDescriptors	Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality.
GetScriptReferences	Returns a list of client script library dependencies for the UpdateProgress control.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
</Triggers>
```

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">

  <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000">
      </asp:Timer>

    <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" >
      </asp:Label>
  </ContentTemplate>

</asp:UpdatePanel>
```

Example: Simple Calculator.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Simple_calculator_Ajax.aspx.cs" Inherits="AspnetAjax.ajaxdemo" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Simple Calculator</title>

</head>
<body style="width: 300">
```

```
<form id="form1" runat="server" style="background-color:aqua">

    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <table align="center" >
        <tr>
            <td>First Number</td>
            <td>
                <asp:TextBox ID="TextBox1" runat="server" Height="30px" Width="128px">

                </asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>Second number</td>
            <td>
                <asp:TextBox ID="TextBox2" runat="server" Height="29px" Width="127px">

                </asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Button ID="Button1" runat="server" Text="Submit"
OnClick="Button1_Click" />
            </td>

        </tr>

        <tr>
            <td>Sum</td>
            <td>
                <asp:UpdateProgress ID="UpdateProgress1" runat="server" >
                    <ProgressTemplate>
                      <h1 style="font-size:50px"> <div style="background-color:antiquewhite" >
                         Loading.....
                          </div></h1>
                    </ProgressTemplate>

                </asp:UpdateProgress>
                <asp:Label ID="l1" runat="server"></asp:Label>
            </td>
        </tr>
```

```
            <tr>
                <td>Difference</td>
                <td>
                    <asp:Label ID="l2" runat="server"></asp:Label>
                </td>
            </tr>

            <tr>
                <td>Prod</td>
                <td>
                    <asp:Label ID="l3" runat="server"></asp:Label>
                </td>
            </tr>
            <tr>
                <td>Div</td>
                <asp:Label ID="Label4" runat="server" Text="Label"></asp:Label>
                <td>
                    <asp:Label ID="l4" runat="server"></asp:Label>
                </td>
            </tr>
        </table>
            </ContentTemplate>
        </asp:UpdatePanel>

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

Simple Calculator.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AspnetAjax
{
    public partial class ajaxdemo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        if(!IsPostBack)
            {
```

```
            TextBox1.Focus();
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(10000);

        int x = Convert.ToInt32(TextBox1.Text.ToString().Trim());
        int y = Convert.ToInt32(TextBox2.Text.ToString().Trim());

        int sum = x + y;
        int dif = x - y;
        int pro = x * y;
        int div = x / y;

        l1.Text = sum.ToString();
        l2.Text = dif.ToString();
        l3.Text = pro.ToString();
        l4.Text = div.ToString();
    }
  }
}
```

Example 2: TimerDemoAjax.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="TimerDemoAjax.aspx.cs"
Inherits="AspnetAjax.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

      <asp:ScriptManager ID="sm1" runat="server">
      </asp:ScriptManager>
      <asp:UpdatePanel ID="up1" runat="server">
        <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000" OnTick="getdate"></asp:Timer>
```

```
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
          </ContentTemplate>
        </asp:UpdatePanel>

      </div>
    </form>
</body>
</html>
```

TimerDemoAjax.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AspnetAjax
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Label1.Text = DateTime.Now.ToLongTimeString();
      }
      protected void getdate(object sender, EventArgs e)
      {
         Label1.Text = DateTime.Now.ToLongTimeString();

      }

   }
}
```

Introduction to ASP.Net MVC

ASP.NET MVC is open-source software from Microsoft. Its web development framework combines the features of MVC (Model-View-Controller) architecture, the most up-to-date ideas and techniques from agile development and the best parts of the existing ASP.NET platform. This tutorial provides a complete picture of the MVC framework and teaches us how to build an application using this tool.

Benefits of ASP.NET MVC

Following are the benefits of using ASP.NET MVC −

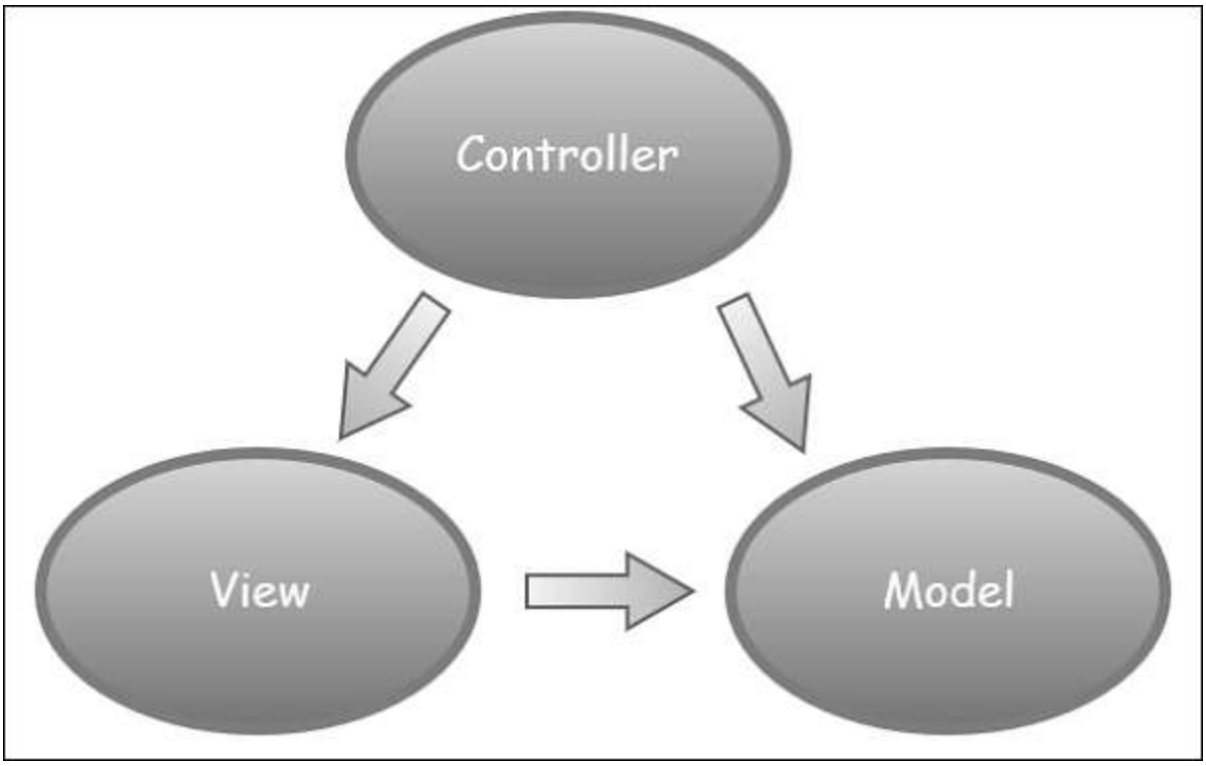
- Makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- Enables full control over the rendered HTML and provides a clean separation of concerns.
- Direct control over HTML also means better accessibility for implementing compliance with evolving Web standards.
- Facilitates adding more interactivity and responsiveness to existing apps.
- Provides better support for test-driven development (TDD).
- Works well for Web applications that are supported by large teams of developers and for Web designers who need a high degree of control over the application behavior.

The MVC (Model-View-Controller) design pattern has actually been around for a few decades, and it's been used across many different technologies. Everything from Smalltalk to C++ to Java, and now C Sharp and .NET use this design pattern to build a user interface.

Following are some salient features of the MVC pattern −

- Originally it was named Thing-Model-View-Editor in 1979, and then it was later simplified to Model- View-Controller.
- It is a powerful and elegant means of separating concerns within an application (for example, separating data access logic from display logic) and applies itself extremely well to web applications.
- Its explicit separation of concerns does add a small amount of extra complexity to an application’s design, but the extraordinary benefits outweigh the extra effort.

The MVC architectural pattern separates the user interface (UI) of an application into three main parts.



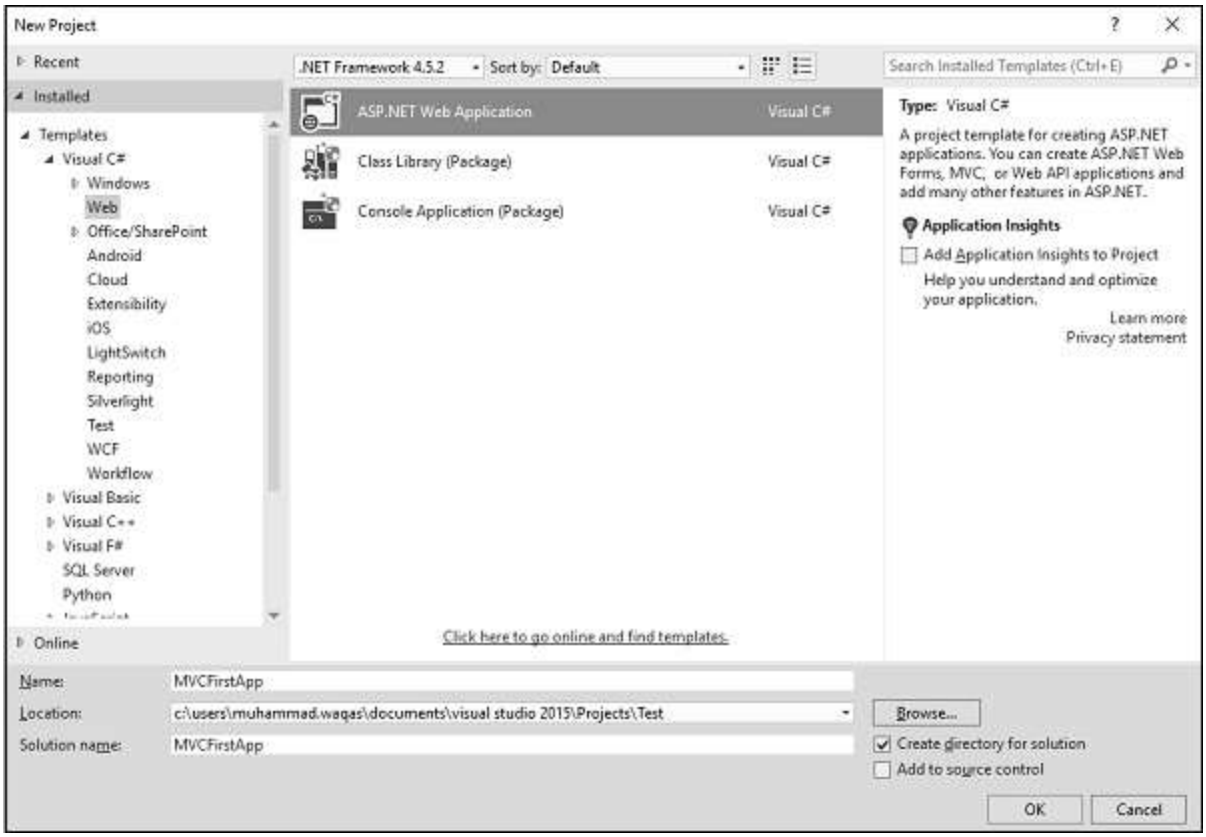
- **The Model** − A set of classes that describes the data we are working with as well as the business logic.
- **The View** − Defines how the application's UI will be displayed. It is a pure HTML, which decides how the UI is going to look like.
- **The Controller** − A set of classes that handles communication from the user, overall application flow, and application-specific logic.

Create ASP.Net MVC Application

Following are the steps to create a project using project templates available in Visual Studio.

Step 1 − Open the Visual Studio. Click File → New → Project menu option.

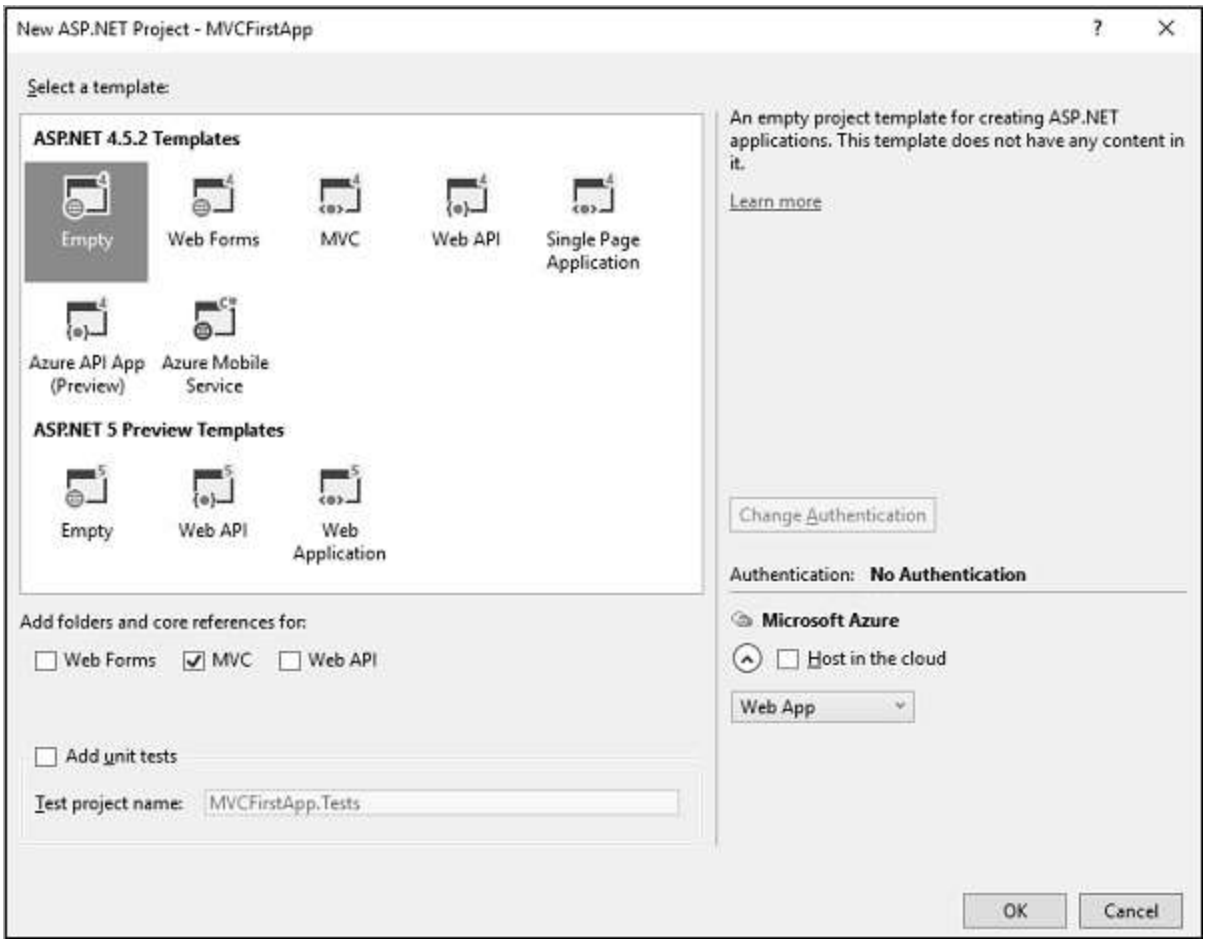
A new Project dialog opens.



Step 2 − From the left pane, select Templates → Visual C# → Web.

Step 3 − In the middle pane, select ASP.NET Web Application.

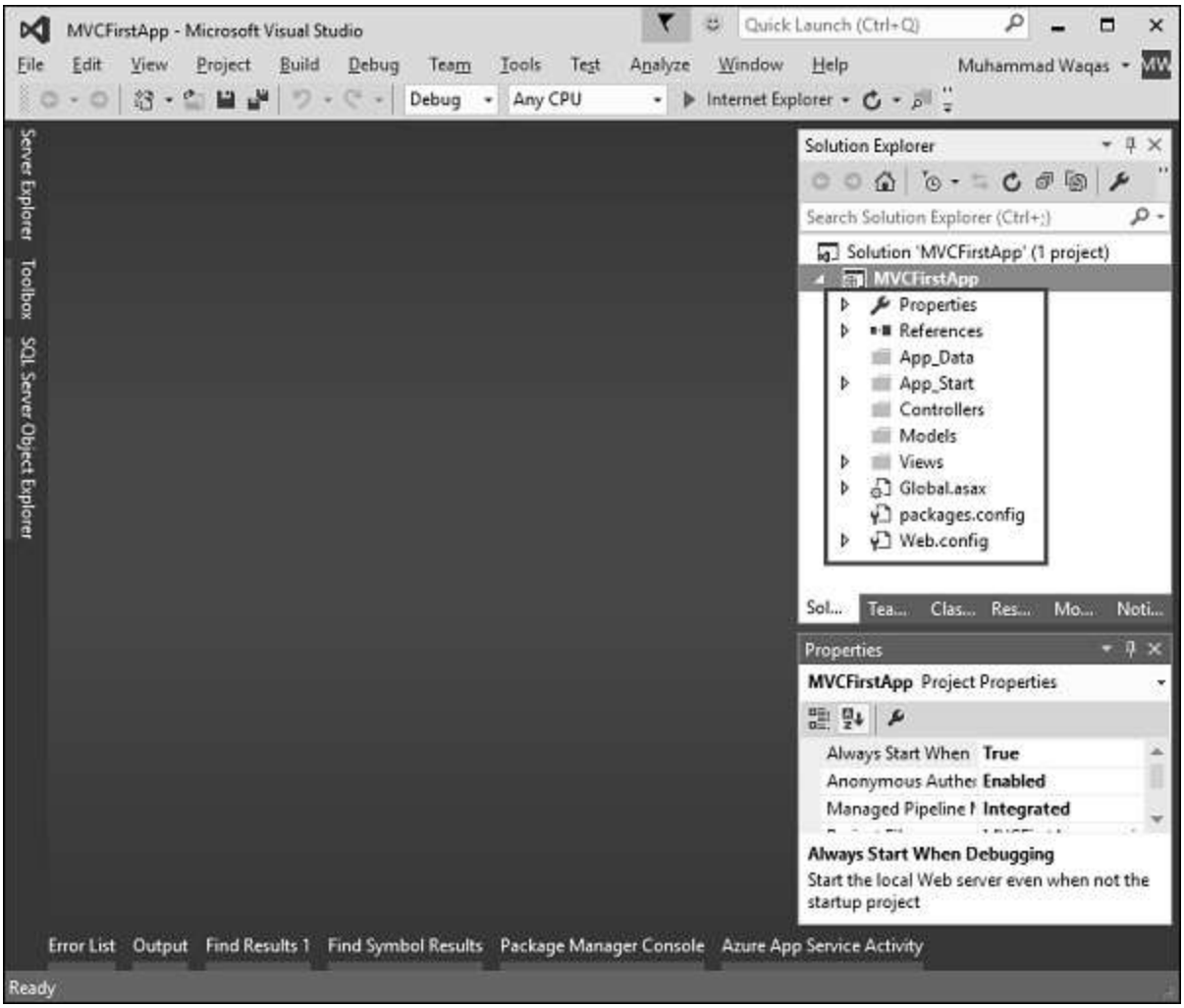
Step 4 − Enter the project name, MVCFirstApp, in the Name field and click ok to continue. We will see the following dialog which asks we to set the initial content for the ASP.NET project.



Step 5 − To keep things simple, select the 'Empty' option and check the MVC checkbox in the Add folders and core references section. Click Ok.

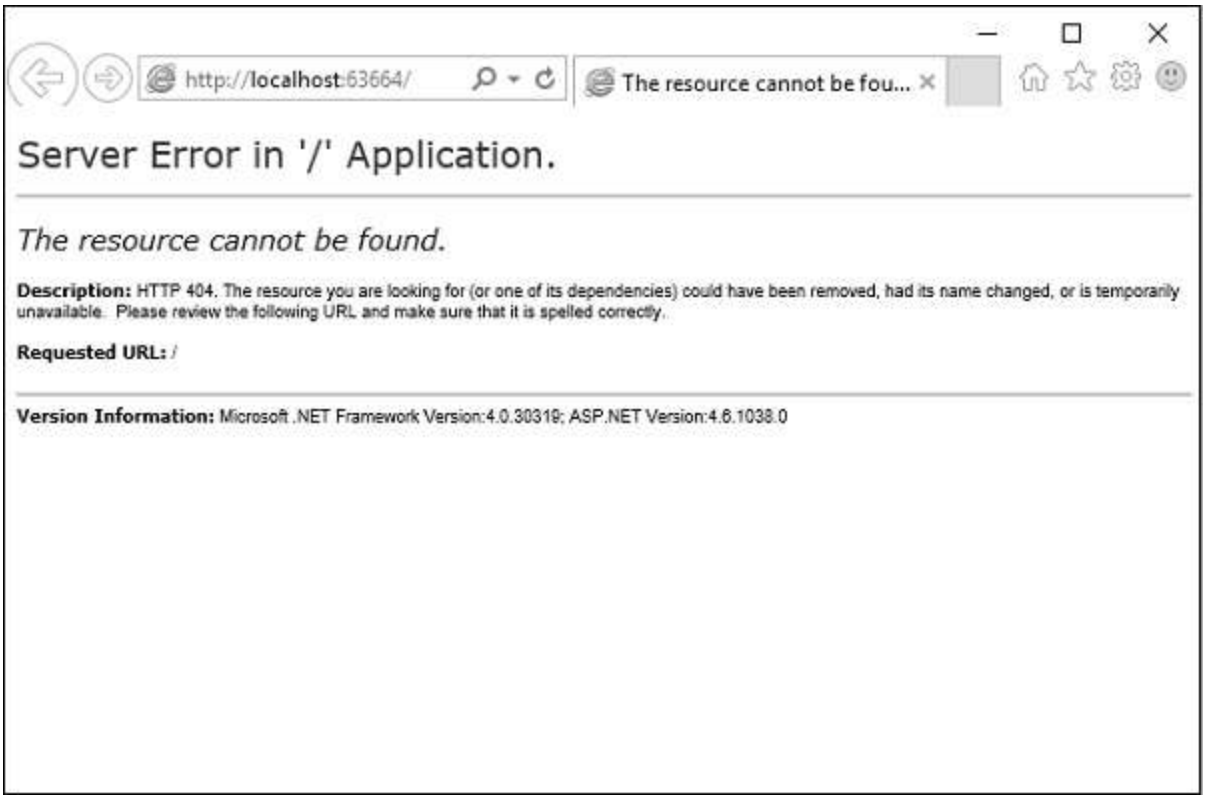
It will create a basic MVC project with minimal predefined content.

Once the project is created by Visual Studio, we will see a number of files and folders displayed in the Solution Explorer window.



As we know that we have created ASP.Net MVC project from an empty project template, so for the moment the application does not contain anything to run.

Step 6 − Run this application from Debug → Start Debugging menu option and we will see a **404 Not Found** Error.

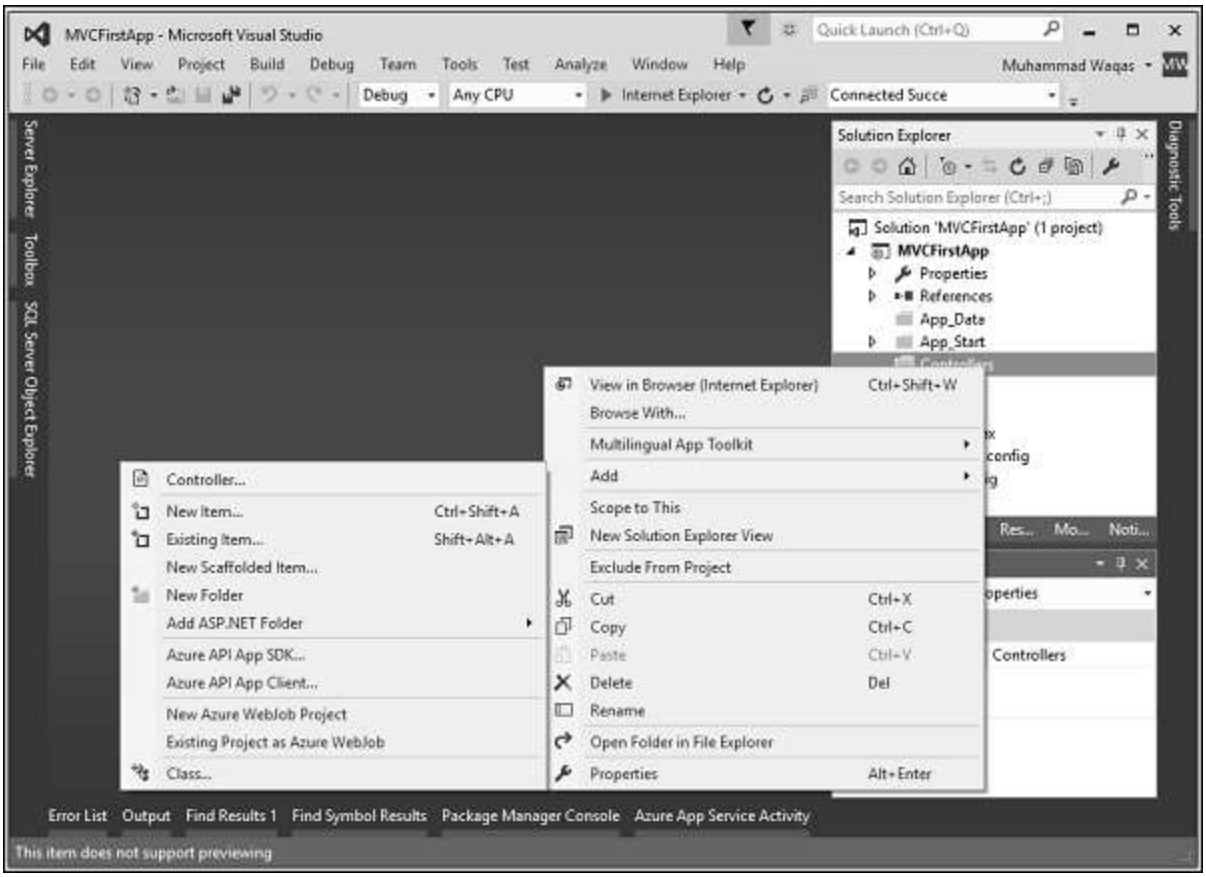


The default browser is, Internet Explorer, but we can select any browser that we have installed from the toolbar.

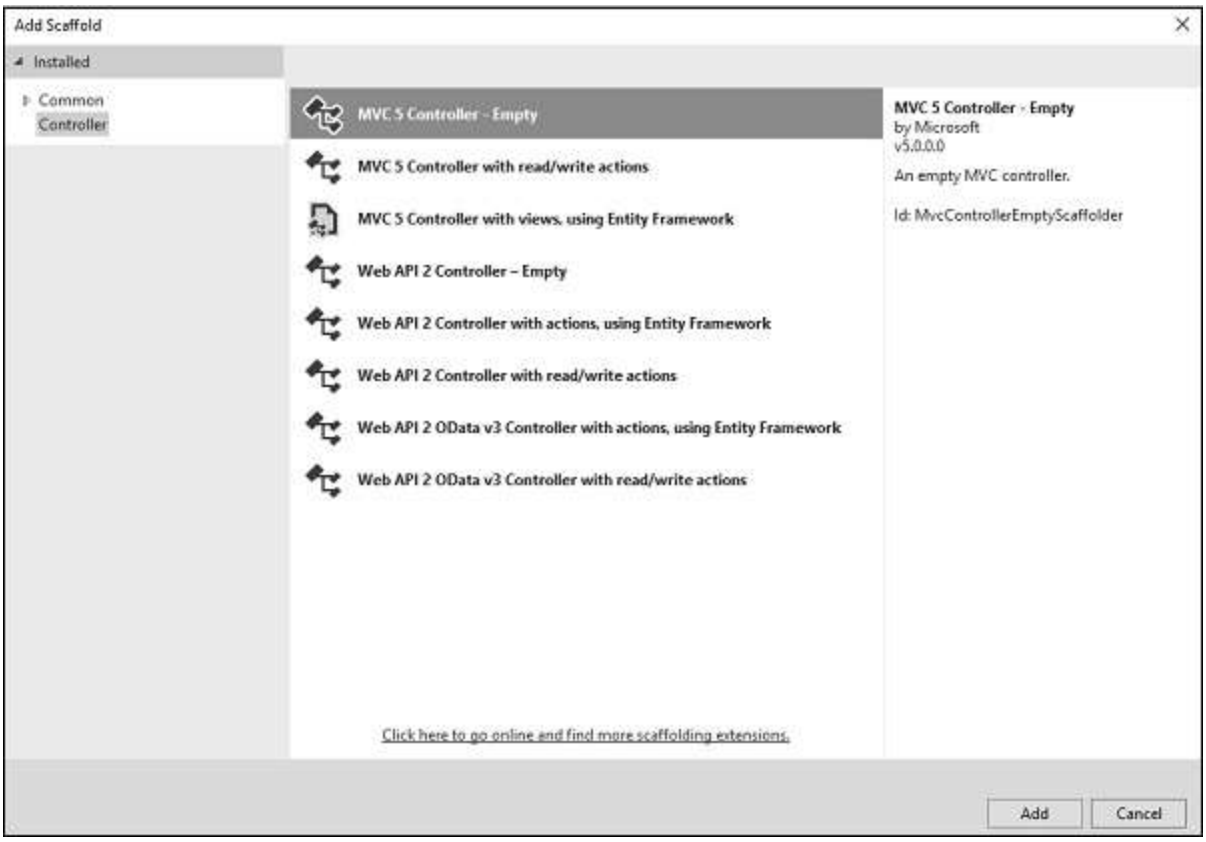
Add Controller

To remove the 404 Not Found error, we need to add a controller, which handles all the incoming requests.

Step 1 − To add a controller, right-click on the controller folder in the solution explorer and select Add → Controller.

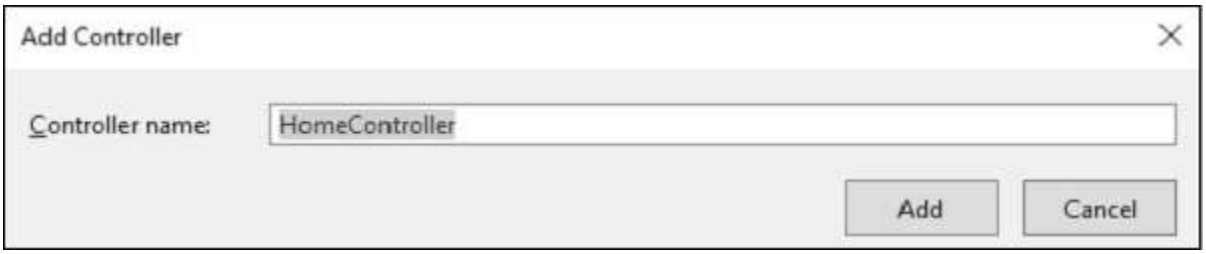


It will display the Add Scaffold dialog.



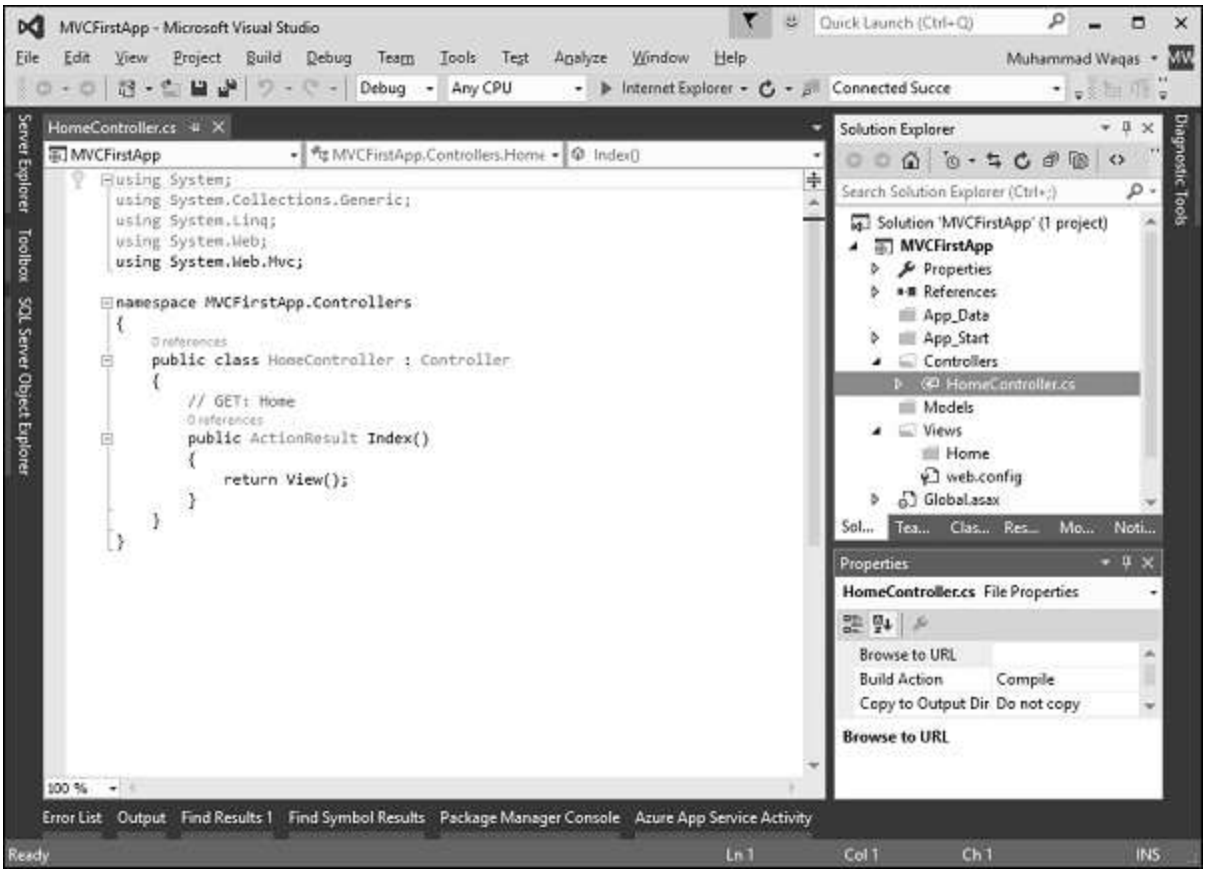
Step 2 − Select the MVC 5 Controller – Empty option and click 'Add' button.

The Add Controller dialog will appear.



Step 3 − Set the name to HomeController and click the Add button.

We will see a new C# file HomeController.cs in the Controllers folder, which is open for editing in Visual Studio as well.

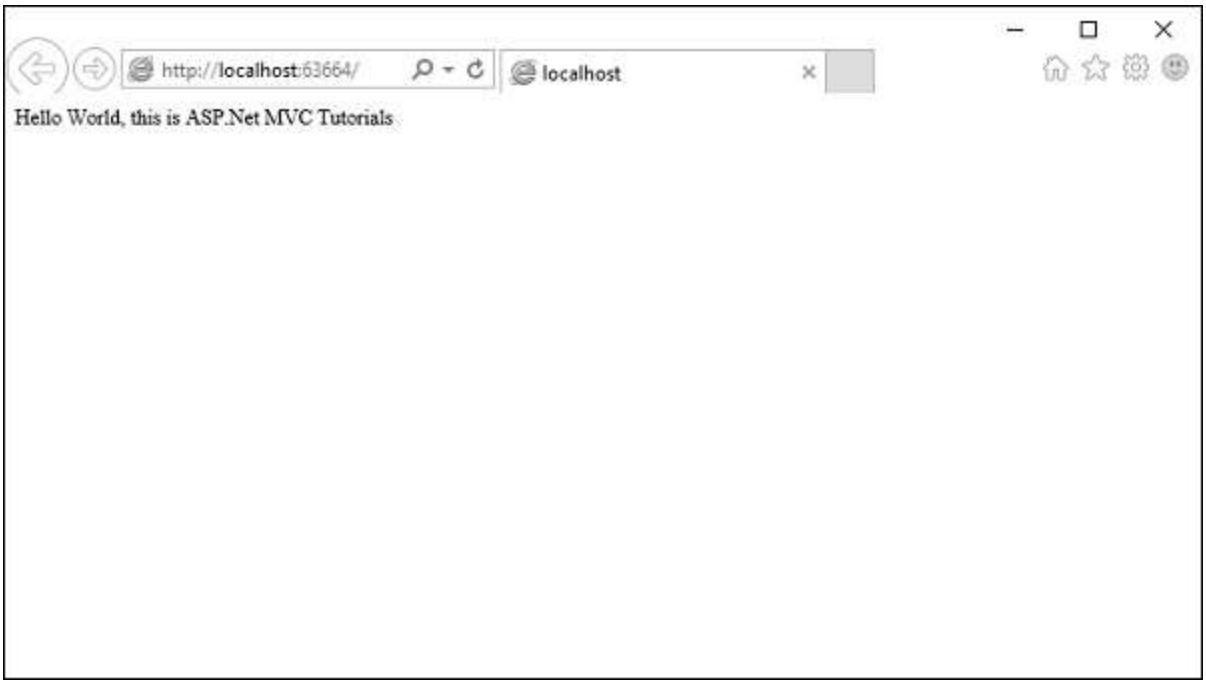


Step 4 − To make this a working example, let's modify the controller class by changing the action method called **Index** using the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCFirstApp.Controllers {
  public class HomeController : Controller {
    // GET: Home
    public string Index(){
      return "Hello World, this is ASP.Net MVC Tutorials";
    }
```

```
  }
}
```

Step 5 − Run this application and we will see that the browser is displaying the result of the Index action method.



Example:

Model: Student.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebApplication12.Models
{
    public class Student
    {
        public int StudentId { get; set; }
        public string StudentName { get; set; }
        public int Age { get; set; }
    }}
```

View: Index.cshtml

```
@model IEnumerable<WebApplication12.Models.Student>
<html>

<body>

<table>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.StudentName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Age)
        </th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.StudentName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Age)
        </td>
    </tr>
}

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

Controller: HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using WebApplication12.Models;


namespace WebApplication12.Controllers
{
   public class HomeController : Controller
   {
      // GET: Home
      public ActionResult Index()
```

```
        {
            var studentList = new List<Student>{
                    new Student() { StudentId = 1, StudentName = "Ram", Age = 18 } ,
                    new Student() { StudentId = 2, StudentName = "Shyam",  Age = 21 } ,
                    new Student() { StudentId = 3, StudentName = "Hari",  Age = 25 } ,
                    new Student() { StudentId = 4, StudentName = "Ram" , Age = 20 }
            };
            // Get the students from the database in the real application


            return View(studentList);


        }
    }
}
```

Razor Syntax

Razor is one of the view engines supported in ASP.NET MVC. Razor allows we to write a mix of HTML and server-side code using C# or Visual Basic. Razor view with visual basic syntax has .vbhtml file extension and C# syntax has .cshtml file extension.

Razor syntax has the following Characteristics:

- **Compact**: Razor syntax is compact, enabling we to minimize the number of characters and keystrokes required to write code.
- **Easy to Learn**: Razor syntax is easy to learn where we can use our familiar language C# or Visual Basic.
- **Intellisense**: Razor syntax supports statement completion within Visual Studio.

Inline expression

Start with @ symbol to write server-side C# or VB code with HTML code. For example, write @Variable_Name to display the value of a server-side variable, e.g., DateTime.Now returns the current date and time. So, write @DateTime.Now to display the current date and time, as shown below. A single line expression does not require a semicolon at the end of the expression.

C# Razor Syntax

```
<h1>Razor syntax demo</h1>
```

```
<h2>@DateTime.Now.ToShortDateString()</h2>
```
Output:

Razor syntax demo
08-09-2014

Multi-statement Code block

We can write multiple lines of server-side code enclosed in braces `@{ ... }`. Each line must end with a semicolon the same as C#.

Example: Server side Code in Razor Syntax

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

<h2>Today's date is: @date </h2>
<h3>@message</h3>
```
Output:

Today's date is: 08-09-2014
Hello World!

Display Text from Code Block

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

Example: Display Text in Razor Syntax

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

Today's date is: 08-09-2014
Hello World!

Display text using <text> within a code block, as shown below.

Example: Text in Razor Syntax

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

Output:

Today's date is: 08-09-2014
Hello World!

if-else condition

Write if-else condition starting with @ symbol. The if-else code block must be enclosed in braces { }, even for a single statement.

Example: if else in Razor

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

Output:

2014 is not a leap year.

For loop

Example: for loop in Razor

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

Output:

0
1
2
3
4

Model

Use @model to use model object anywhere in the view.

Example: Use Model in Razor

```
@model Student

<h2>Student Detail:</h2>
<ul>
   <li>Student Id: @Model.StudentId</li>
   <li>Student Name: @Model.StudentName</li>
   <li>Age: @Model.Age</li>
</ul>
```

Output:

Student Detail:

- Student Id: 1
- Student Name: John
- Age: 18

Declare Variables

Declare a variable in a code block enclosed in brackets and then use those variables inside HTML with @ symbol.

Example: Variable in Razor

```
@{
   string str = "";

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

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

Output:Hello World!