Views:

View is the one of the three components in MVC application. View is responsible for providing the user interface (UI) to the user. View displays data from the model to the user and also enables them to modify the data.

In MVC application, all browser requests are mapped to controller and its action method. Controller action method will return the view to the user.

ASP.NET MVC views are stored in **Views** folder. Different action methods of a single controller class can render different views, so the Views folder contains a separate folder for each controller with the same name as controller, in order to accommodate multiple views. For example, views, which will be rendered from any of the action methods of HomeController, resides in Views > Home folder. In the same way, views which will be rendered from StudentController, will resides in Views > Student folder.

* The Views folder contains one folder for each controller.

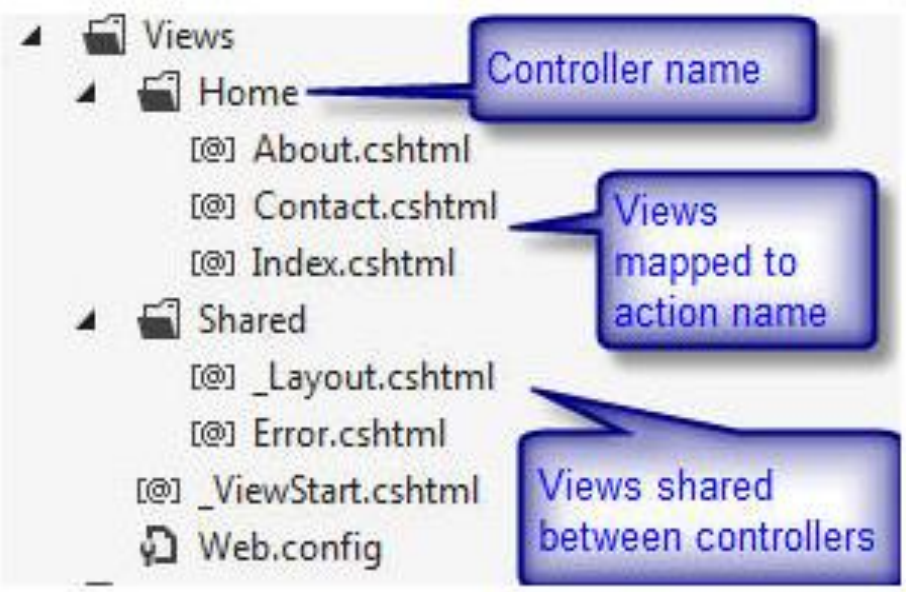
* By default inside the Views folder has created an Account folder, a Home folder, and a Shared folder.

* The Account folder contains pages for registering and logging in to user accounts.

* The Home folder is used for storing application pages like the index page, about page and the contact page.

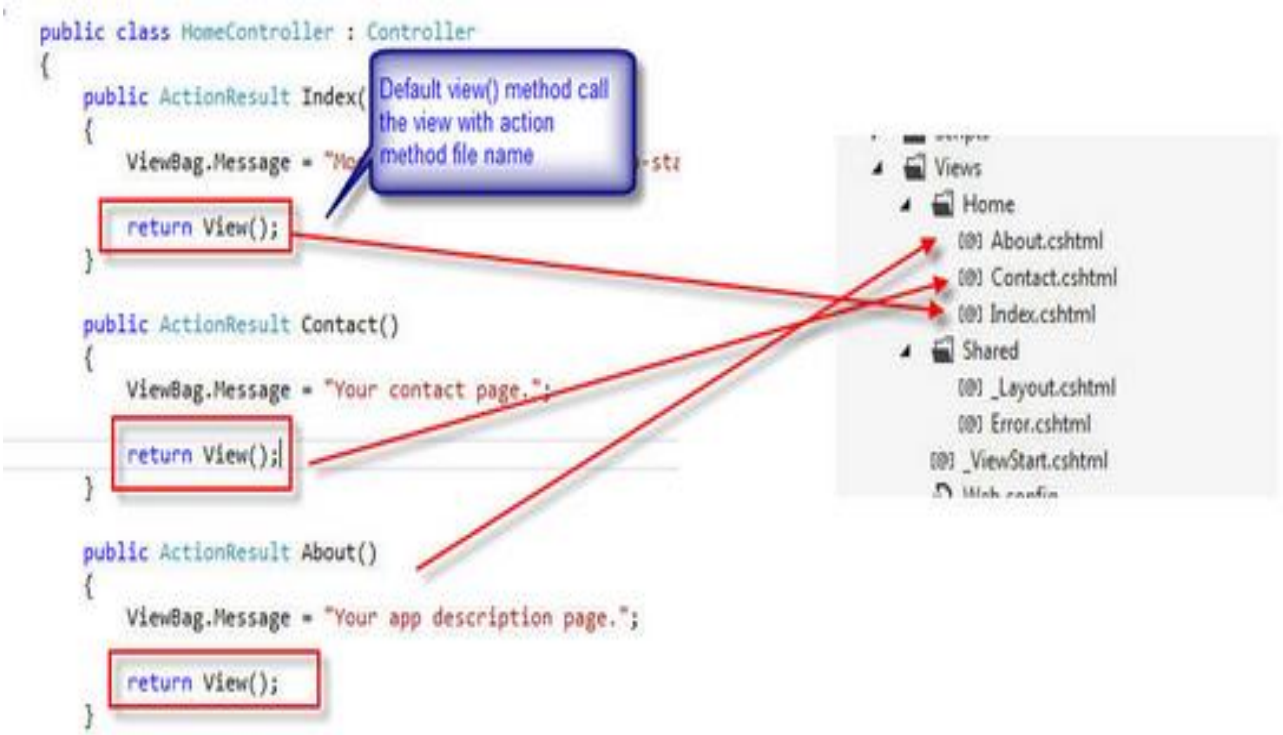
* The Shared folder is used to store views (layout pages) which can be shared between controllers.

* All the views are placed inside the 'Views' folder with /Views/ [Controller Name] / [Action Name] .cshtml format



* Depending upon the language selection, view files will have different file type e.g. .cshtml, .vbhtml

* In the below code you can see that View() method called inside the action method is not mentioned with view name. In this case action method applies a convention to locate the view which has same name as the action within the /Views/ControllerName directory



* Action method can call the different view without using the convention

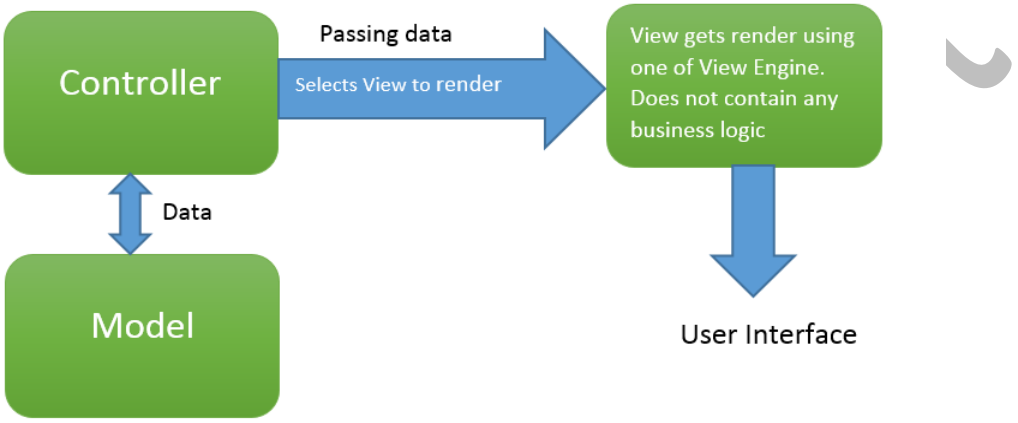
```
public class TestController : Controller
{
    public ActionResult Index()
    {
      return View("Home");
    }
}
```

* You can use the tilde syntax to provide the full path to the view

```
public class TestController : Controller
{
    // GET: Demo
    public ViewResult Index()
    {
      return View("~/Views/Test/Home.cshtml");
    }
}
```

* Views are processed by a special execution engine called View Engine. In other words we can say: View contains markup that gets render with View Engine. View Engine generates View in ASP.NET MVC framework.

* We can say View is User Interface in MVC application and it never talks to database or model directly. It does not contain any business or application logic. Controller returns a View to be rendered.



View Engine :->

View Engine is responsible for rendering the view into html form to the browser. By default, Asp.net MVC support Web Form (ASPX) and Razor View Engine. Views are usually some kind of mix-up of HTML and a programming language. There are many third party view engines (like Spark, Nhaml etc.) that are also available for Asp.net MVC. Now, Asp.net MVC is open source and can work with other third party view engines like Spark, Nhaml.

The view engine is what's responsible for rendering our view, and converting our code into glorious HTML. As such, they are directly responsible for HOW we need to write code in our views.

A view engine is what MVC uses to find and render the views we are requesting from the controller

"View Engine in ASP.NET MVC is used to translate our views to HTML and then render to response."

In MVC, View engine is the one that works between our View and browser to provide valid HTML output to our browser by compiling the code inside our View. There are many view engines available and some of them are following:

- ASPX
- Razor

- Spark
- NHaml
- NDJango
- Hasic
- Brail
- Bellevue
- Sharp Tiles
- String Template
- Wing Beats
- SharpDOM

Currently most developers prefer to use Razor view engine as it provides very convenient way of programming. All of these view engines may not support ASP.NET MVC.

The ASP.NET MVC Framework comes with two built-in view engines:

1. Razor Engine: Razor is a markup syntax that enables the server side C# or VB code into web pages. This server side code can be used to create dynamic content when the web page is being loaded. Razor is an advanced engine as compared to ASPX engine and was launched in the later versions of MVC i.e. ASP.NET MVC 3.0.

2. Web Forms/ASPX View Engine: ASPX or the Web Forms engine is the default view engine that is included in the MVC Framework since the beginning. Writing code with this engine is very similar to writing code in ASP.NET Web Forms.

For example, ASP.NET comes with its own view engine out of the box. That is the one where views have lots of tags like <% %>, <%= %>, and <%: %>. It uses the .aspx file extension.

With ASP.NET MVC3, another out-of-the-box view engine was added, Razor, which has a more appealing syntax, e.g. @Model.UserName . Razor views files have .cshtml or .vbhtml extension.

Note: Even we can use multiple view engines in parallel in ASP.NET MVC, though it wouldn't recommend it unless necessary. Most of the ASP.NET MVC Application using “Razor” view engine from ASP.NET MVC 3.0 version.

Example:

ASPX Engine (*.aspx)	Razor Engine (*.cshtml)

<%
int x = 123;
if(x%2 == 0)
{%>
 x is Even Number
<%}
else
{%>
 x is Odd Number
<%}
%>
	@{
	int x = 123;
	if(x%2 == 0)
	{
	 x is Even Number
	}
	else
	{
	 x is Odd Number
	}
	}

Difference between Razor View Engine and ASPX View Engine:

Razor View Engine Vs Web Form (ASPX) View Engine

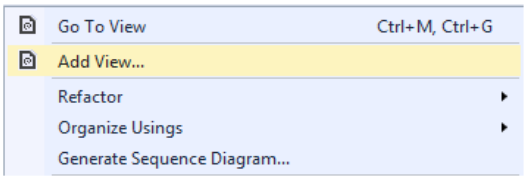
Razor View Engine	Web Form View Engine
Razor Engine is an advanced view engine that was introduced with MVC3. This is not a new language but it is a new markup syntax.	Web Form Engine is the default view engine for the Asp.net MVC that is included with Asp.net MVC from the beginning.
The namespace for Razor Engine is System.Web.Razor.	The namespace for WebForm Engine is System.Web.Mvc.WebFormViewEngine.
The file extensions used with Razor Engine are different from Web Form Engine. It has .cshtml (Razor with C#) or .vbhtml (Razor with VB) extension for views, partial views, and editor templates and for layout pages.	The file extensions used with Web Form Engine are also like Asp.net Web Forms. It has .aspx extension for views, .ascx extension for partial views & editor templates and .master extension for layout/master pages.
Razor has new and advance syntax that are compact, expressive and reduces typing.	Web Form Engine has the same syntax like Asp.net Web Forms uses for .aspx pages.

Razor syntax are easy to learn and much clean than Web Form syntax. Razor uses @ symbol to make the code like as: @Html.ActionLink("SignUp", "SignUp")	Web Form syntax are borrowed from Asp.net Web Forms syntax that are mixed with html and sometimes make a view messy. WebForm uses <% and %> delimiters to make the code like as: <%: Html.ActionLink("SignUp", "SignUp") %>
By default, Razor Engine prevents XSS attacks (Cross-Site Scripting Attacks) means it encodes the script or html tags like <,> before rendering to view.	Web Form Engine does not prevent XSS attacks means any script saved in the database will be fired while rendering the page
Razor Engine is little bit slowly as compared to WebForm Engine.	Web Form Engine is faster than Razor Engine.
Razor Engine, doesn't support design mode in visual studio means you cannot see your page look and feel.	Web Form engine support design mode in visual studio means you can see your page look and feel without running the application.
Razor Engine support TDD (Test Driven Development) since it is not depend on System.Web.UI.Page class.	Web Form Engine doesn't support TDD (Test Driven Development) since it depend on System.Web.UI.Page class which makes the testing complex.

Create New View:

We will create a view, which will be rendered from Index method of TestController. So, open a TestController class -> Right Click inside Index method -> click Add View...

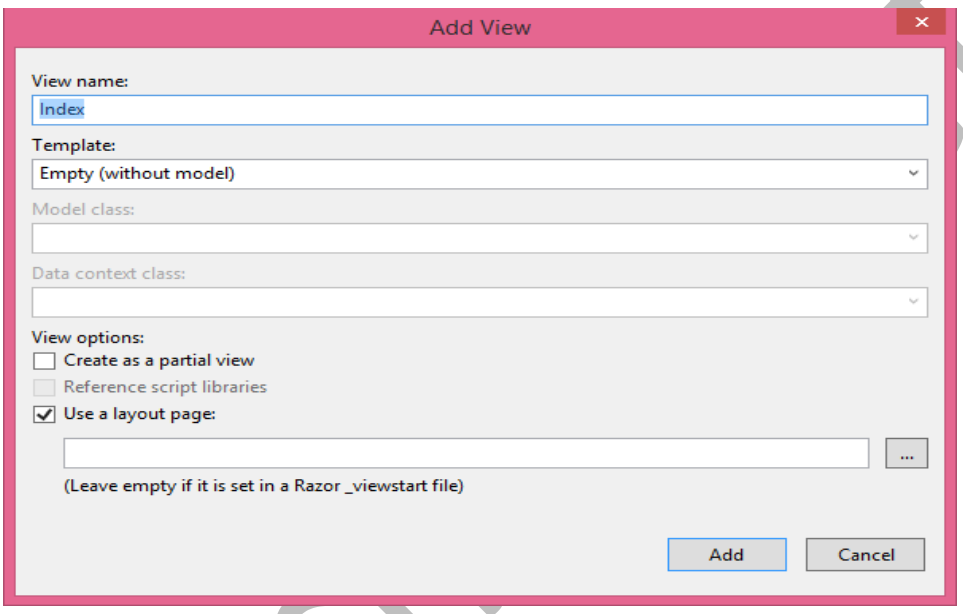
```
using System.Web.Mvc;

namespace TestWebApplication.Controllers
{
    0 references
    public class TestController : Controller
    {
        0 references
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



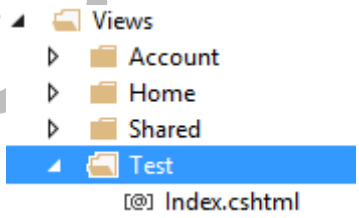
In the Add View dialogue box, keep the view name as Index. It's good practice to keep the view name the same as the action method name so that you don't have to specify view name explicitly in the action method while returning the view.

Select the scaffolding template. Template dropdown will show default templates available for Create, Delete, Details, Edit, Empty, Empty (without model), and List view. Select "Empty (without model)" template.

Leave the rest of the defaults as is, and click "Add" button



This will create Index view under Views -> Test folder as shown below:

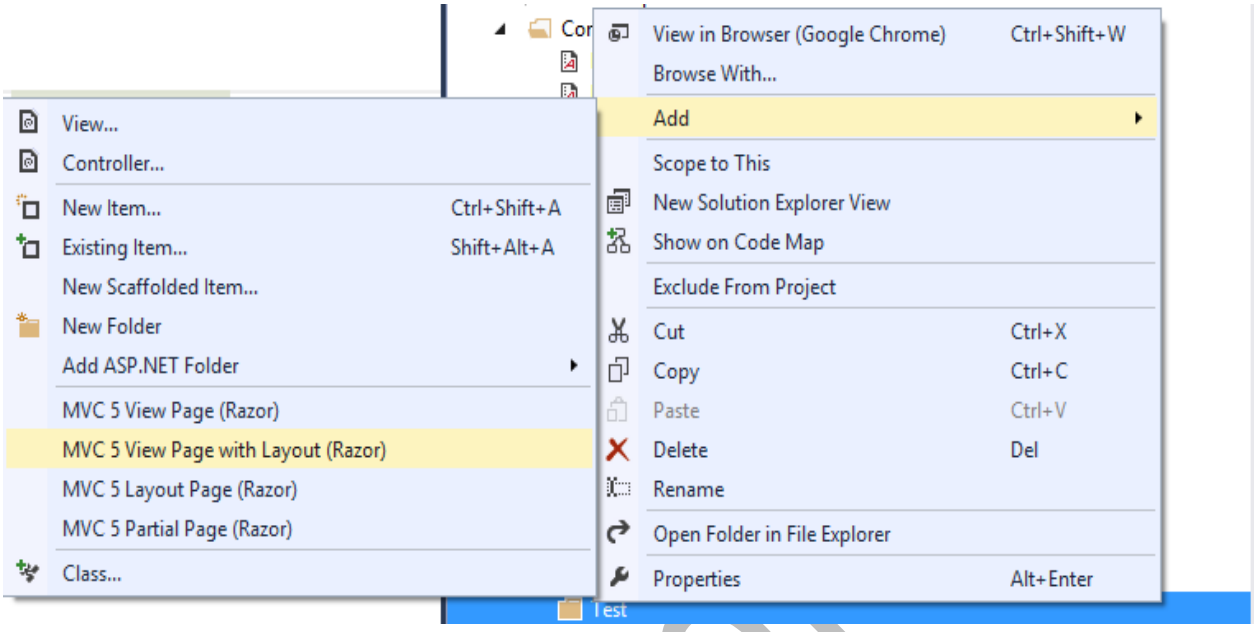


The following code snippet shows an Index.cshtml created above.

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

As you can see in the above Index view, it contains both Html and razor codes. Inline razor expression starts with @ symbol.

We can also add a View, by right clicking on subfolder in Views and select Add from context menu and then select template MVC 5 View Page with Layout.



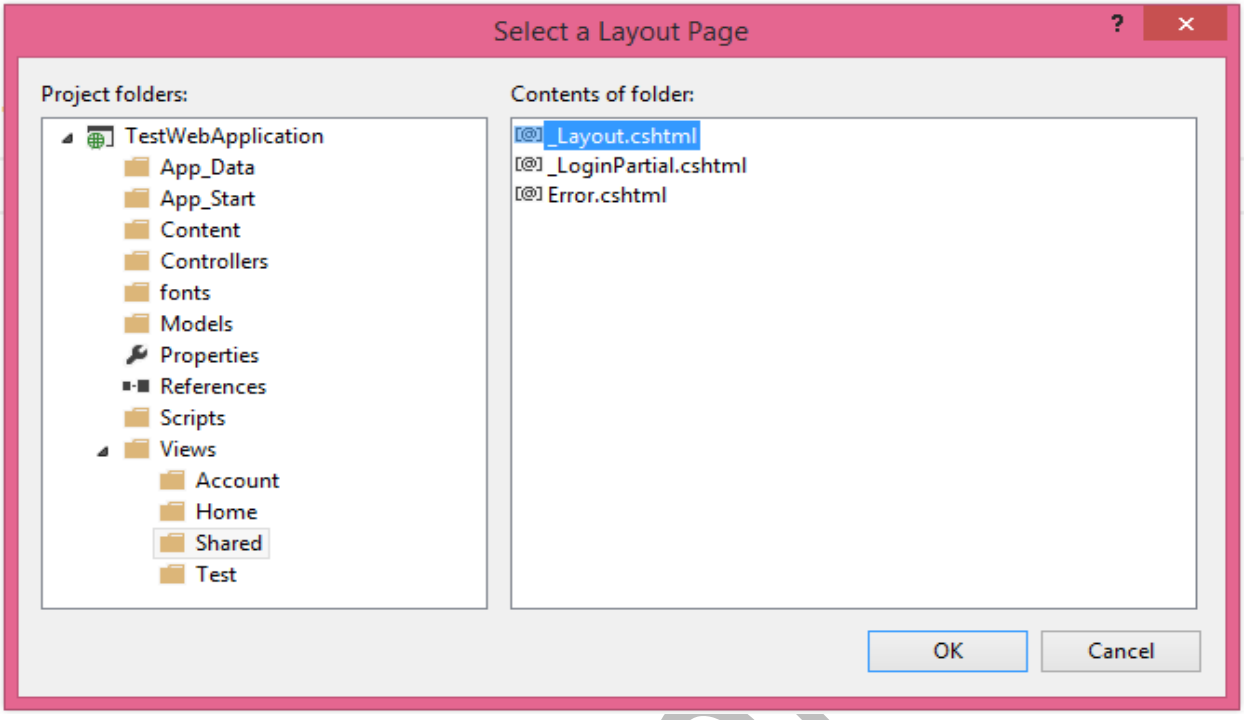
There are four options for Views.

1. MVC 5 View Page
2. MVC 5 View Page with Layout
3. MVC 5 Layout Page
4. MVC5 Partial Page

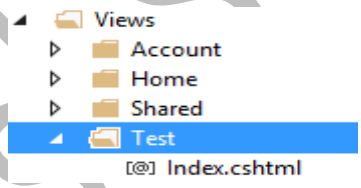
Let us start with adding a MVC5 View Page with Layout. Give name of the View as of the action name. For example, if we have action in controller as below:

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

Then we will create a View with name Index since action name is Index and it is returning view. Next we need to select layout page. Let us go ahead and select a Layout.cshtml from Shared folder.



After adding View, we will find that inside **Views->Test** folder **index.cshtml** being added as following:



By default following code gets created in Index view:

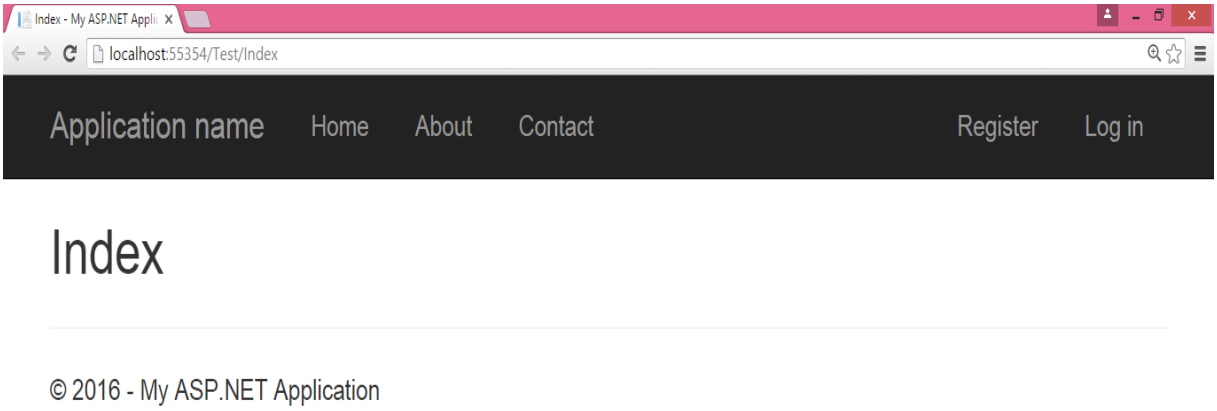
```
1
2  @{
3      ViewBag.Title = "Index";
4  }
5
6  <h2>Index</h2>
7
8
```

Note: By default in the Index.cshtml, a layout i.e. _Layout.cshtml is set from Shared folder and remaining thing as per need we can add it here.

Let us explore above code line by line,

- We are setting layout of view as _Layout.cshtml from Shared folder.
- ViewBag is used to pass parameters. We are setting Title of the View as Index.
- On DOM creating some HTML elements like <h2>.

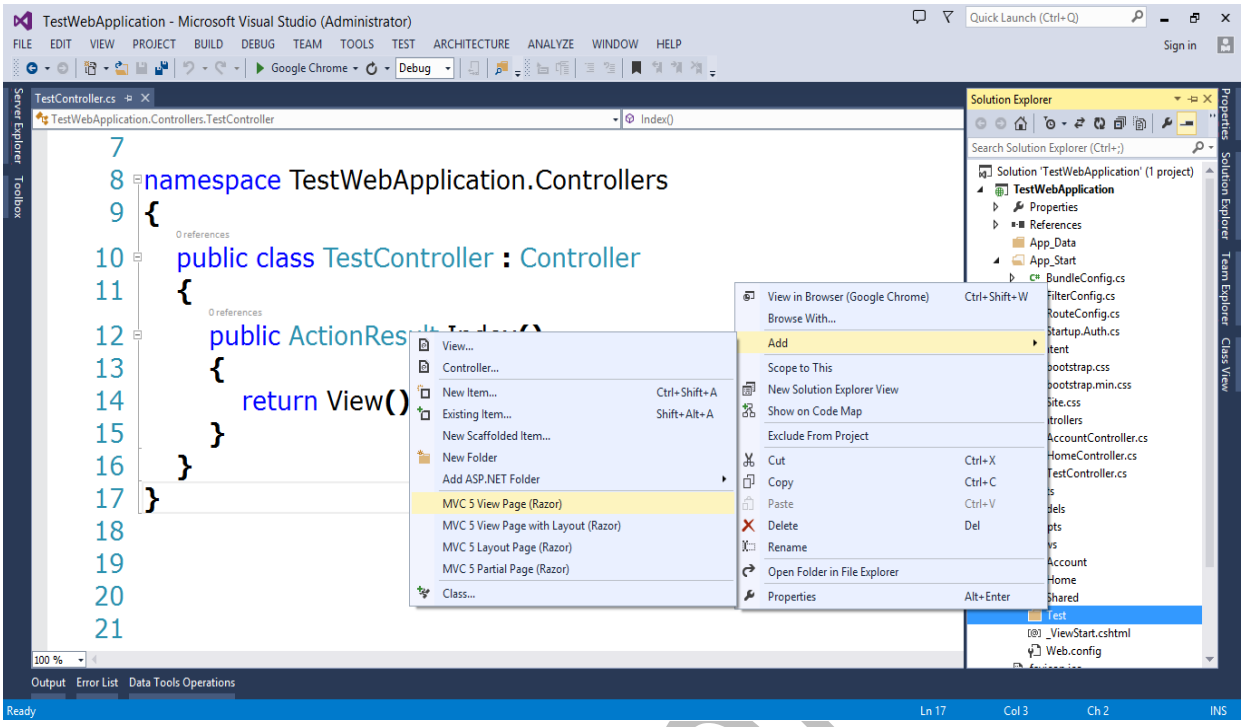
Now let us go ahead and run application. On navigating Index action in Test controller, we will get index view rendered in browser.



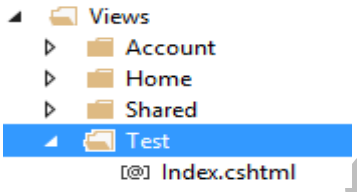
Let us start with adding a MVC5 View Page without Layout. Give name of the View as of the action name. For example, if we have action in controller as below:

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

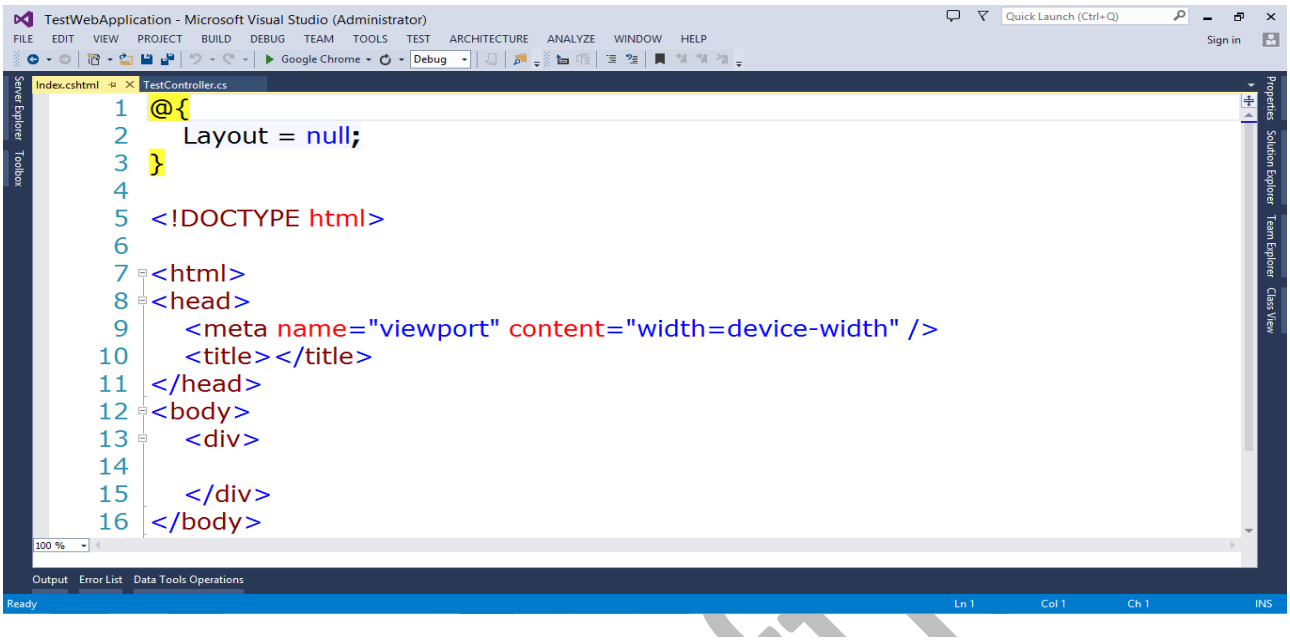
Then we will create a View with name Index since action name is Index and it is returning view. So to create a View Page without Layout, right click Test folder in Views, select Add -> MVC 5 View Page (Razor) as following:



After adding View, we will find that inside **Views->Test** folder **index.cshtml** being added as following:



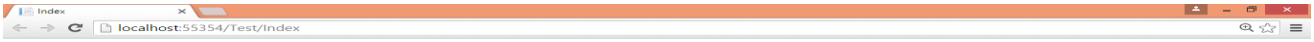
By default following code gets created in Index view



Let us explore above code line by line,

- We are setting layout of view as null means no layout page applied.
- DOCTYPE is used as HTML.
- HTML DOM is created with default structure.

Now let us go ahead and run application. On navigating Index action in Test controller, you will get index view rendered in browser.



Welcome to ASP.NET MVC Application

Some of the important points are as follows,

- View never perform business logic
- View never interacts with Model
- Controller talk to Model, performs business logic and pass data to View