Views in MVC

View is the component involved with the application's User Interface (UI). A view is responsible for the UI. These Views are generally bind from the model data and have extensions such as html, aspx, cshtml, vbhtml, etc. The view displays the data coming from the model. A view is an HTML template which will be binding and displaying HTML controls with data. The ".cshtml" file use the Razor view engine. And .cshtml views are use C# programming. A view can contain "HTML" and "C#" code. It is a combination of c# and Html (.cshtml)

For rendering these static and dynamic content to the browser, MVC Framework utilizes View Engines. View Engines are basically markup syntax implementation, which are responsible for rendering the final HTML to the browser.

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.

ASPX Engine − ASPX or the Web Forms engine is the default view engine that is included in the MVC Framework since the beginning. Writing a code with this engine is similar to writing a code in ASP.NET Web Forms.

Types of Views

There are three different types of views

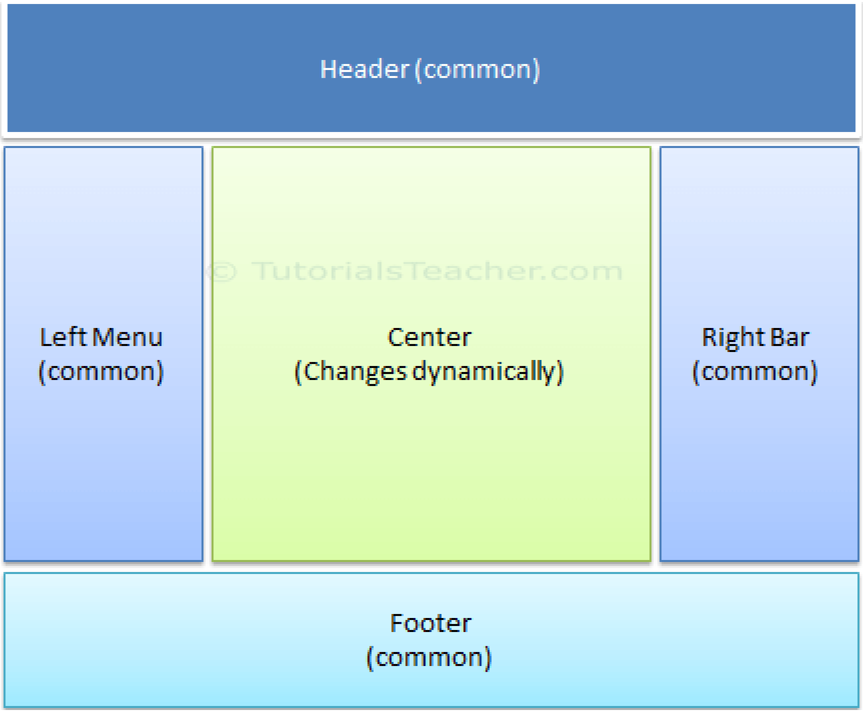
1. Layout (HTML ,CSHTML, VBHTML)
2. View (HTML, CSHTML, VBHTML)
3. Partial View (HTML, CSHTML, VBHTML)

Layout View

An application may contain a specific UI portion that remains the same throughout the application, such as header, left navigation bar, right bar, or footer section. ASP.NET MVC introduced a Layout view which contains these common UI portions so that we don't have to write the same code in every page. An application UI may contain a header footer, left menu bar and right bar, that remains the same on every page. Only the center section changes dynamically, as shown below.

The layout view is the same as the master page of the ASP.NET webform application.

The layout view allows you to define a common site template, which can be inherited in multiple views to provide a consistent look and feel in multiple pages of an application. The layout view eliminates duplicate coding and enhances development speed and easy maintenance. The layout view for the above sample UI would contain a Header, Left Menu, Right bar, and Footer sections. It has a placeholder for the center section that changes dynamically, as shown below.



The layout view has the same extension as other views, .cshtml or .vbhtml. Layout views are shared with multiple views, so it must be stored in the **Shared** folder. By default, a layout view **_Layout.cshtml** is created when you Create MVC application using Visual Studio

The **Shared** folder contains views, layout views, and partial views, which will be shared among multiple controllers.

Using Layout View

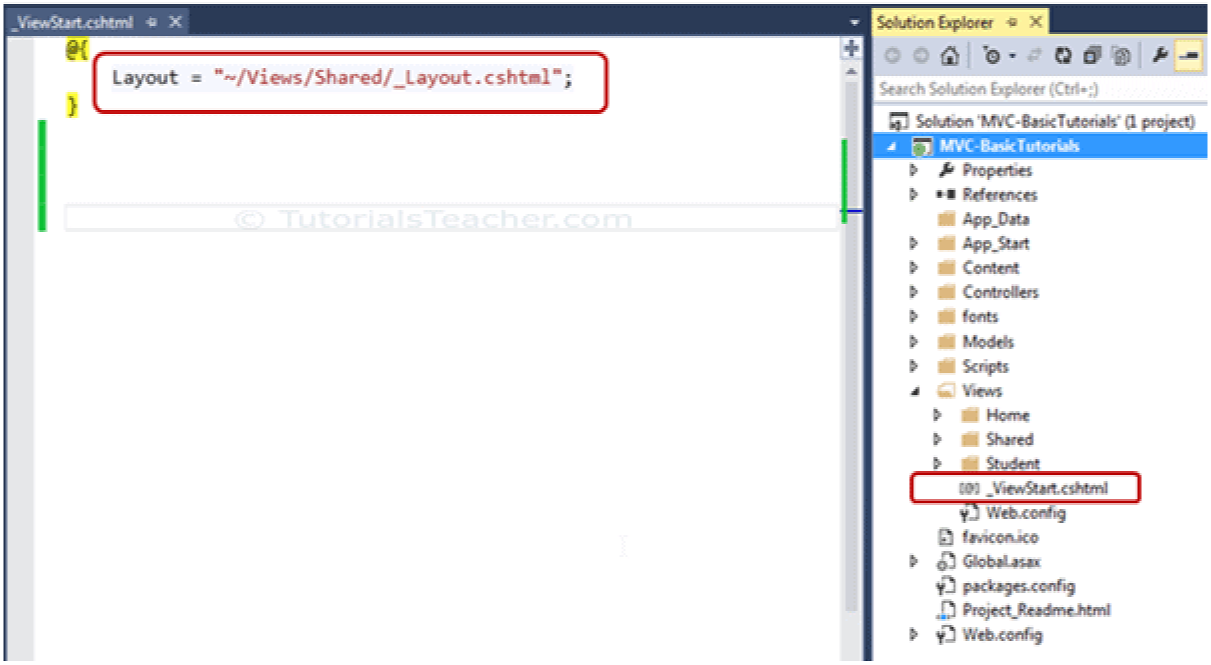
The views which will be displayed in a placeholder `RenderBody()` are called child views. There are multiple ways to specify which layout view will be used with which child views. You can specify it in a common `_ViewStart.cshtml`, in a child view, or in an action method.

ViewStart

The default `_ViewStart.cshtml` is included in the `Views` folder. It can also be created in all other `Views` sub-folders. It is used to specify common settings for all the views under a folder and sub-folders where it is created.

Set the `Layout` property to a particular layout view will be applicable to all the child views under that folder and its sub-folders.

For example, the following `_ViewStart.cshtml` in the **Views** folder sets the `Layout` property to `"~/Views/Shared/_Layout.cshtml"`. So, the `_layout.cshtml` would be a layout view of all the views included in `Views` and its subfolders.

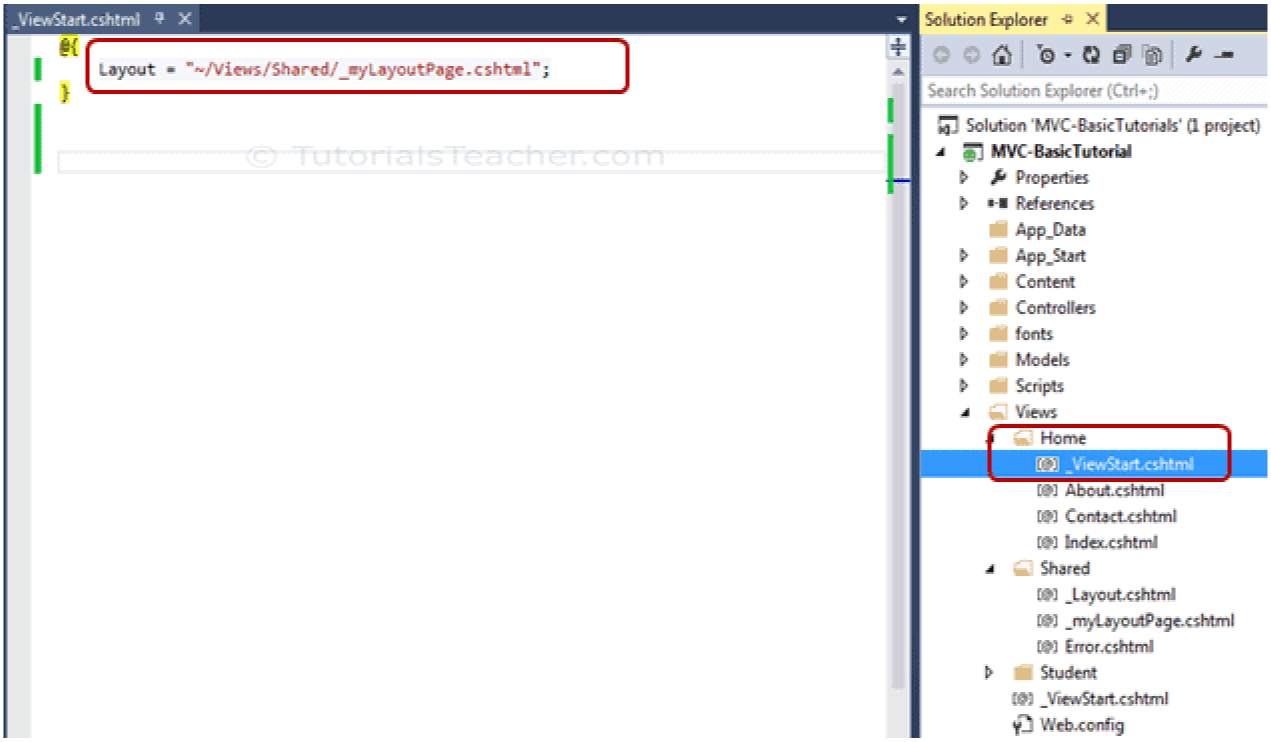


```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Setting Layout View in _ViewStart.cshtml

The `_ViewStart.cshtml` can also be created in the sub-folders of the `View` folder to set the default layout page for all the views included in that particular subfolder.

For example, the following `_ViewStart.cshtml` in the `Home` folder sets the `Layout` property to `_myLayoutPage.cshtml`. So now, `Index.cshtml`, `About.cshtml` and `Contact.cshtml` will display in the `_myLayoutPage.cshtml` instead of default `_Layout.cshml`.



```
@{
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";
}
```

Specify Layout View in a Child View

You can also override the default layout view setting of `_ViewStart.cshtml` by setting the `Layout` property in each child view. For example, the following `Index.cshtml` view uses the `_myLayoutPage.cshtml` even if `_ViewStart.cshtml` sets the `_Layout.cshtml`.

```
@{
    ViewBag.Title = "Home Page";
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";
}
```

Specify Layout Page in Action Method

Specify the layout view name as a second parameter in the `View()` method, as shown below. By default, layout view will be searched in the `Shared` folder.

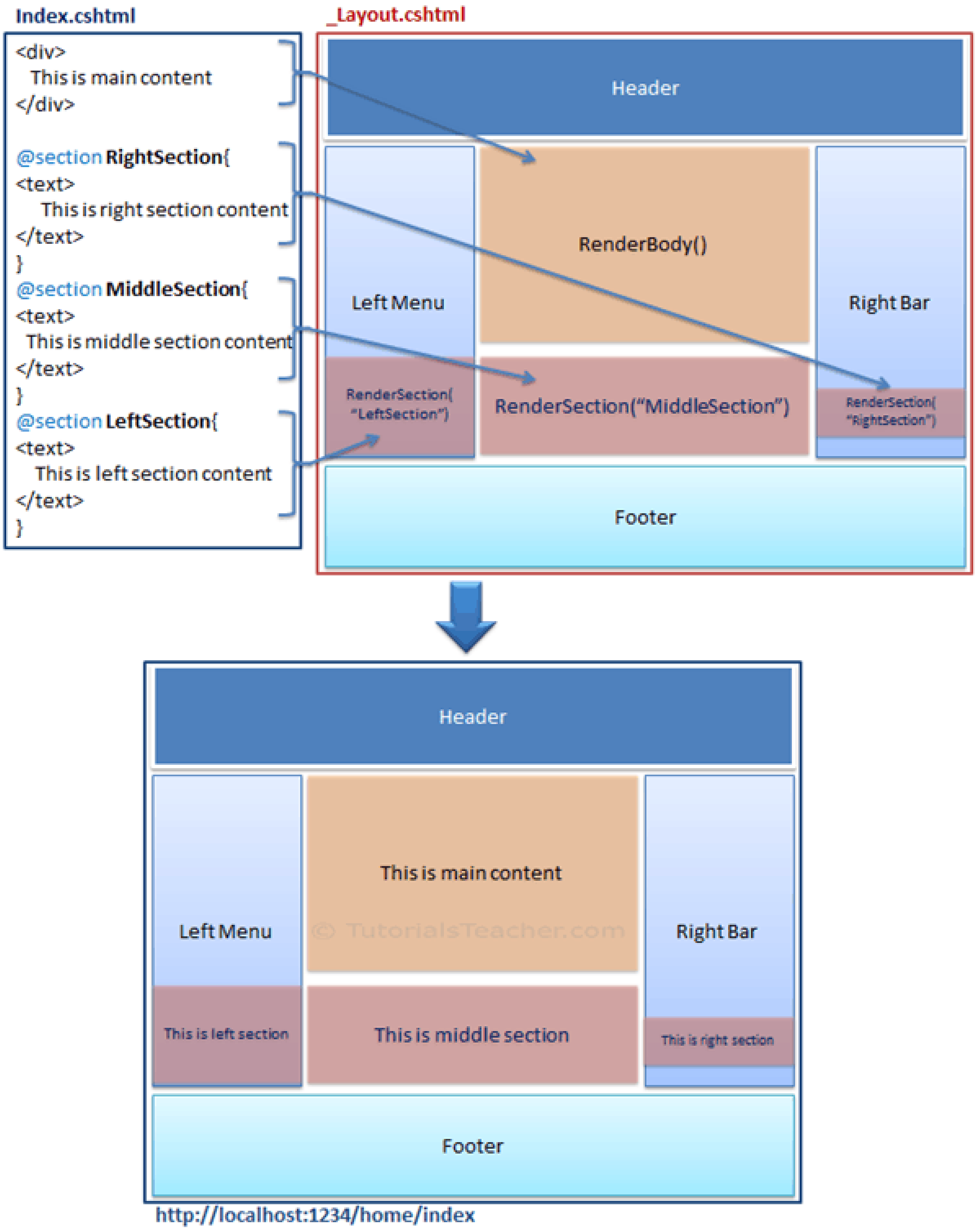
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("Index", "_myLayoutPage");
    }
}
```

Rendering Methods

ASP.NET MVC layout view renders child views using the following methods.

Method	Description
RenderBody()	Renders the portion of the child view that is not within a named section. Layout view must include the `RenderBody()` method.
RenderSection(string name)	Renders a content of named section and specifies whether the section is required.

The following figure illustrates the use of the `RenderBody()` and `RenderSection()` methods.



```
<div>
  This is main content
</div>

@section RightSection{
<text>
   This is right section content
</text>
}
@section MiddleSection{
<text>
  This is middle section content
</text>
}
@section LeftSection{
<text>
   This is left section content
</text>
}
```

As you can see in the above figure, the `_Layout.cshtml` includes the `RenderBody()` method and `RenderSection()` method. Above, `Index.cshtml` contains the named sections using `@section` where the name of each section matches the name specified in the `RenderSection()` method of a layout view `_Layout.cshtml`, e.g. `@Section`

RightSection. At run time, the named sections of Index.cshtml, such as LeftSection, RightSection, and MiddleSection will be rendered at appropriate place where the RenderSection() method is called. The rest of the Index.cshtml view, which is not in any of the named section, will be rendered in the RenderBody() is called.

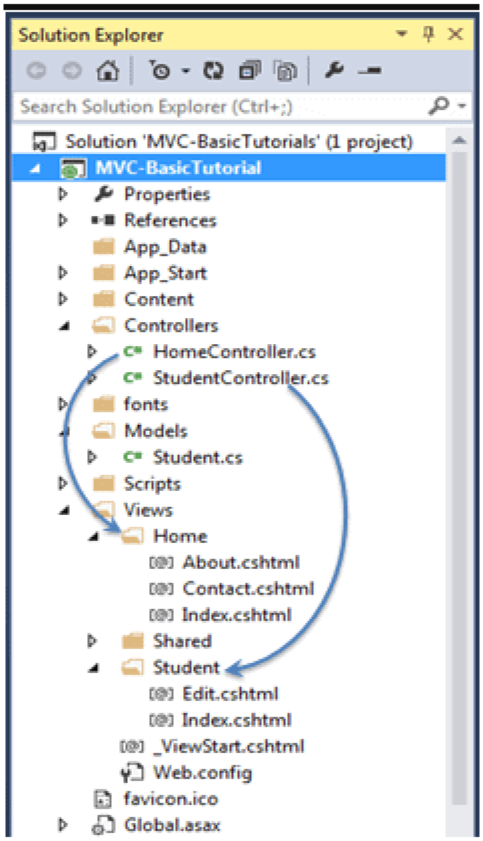
Let's create a new layout view to understand the above render methods in the next section.

Views

A view is used to display data using the model class object. The **Views** folder contains all the view files in the ASP.NET MVC application.

A controller can have one or more action methods, and each action method can return a different view. In short, a controller can render one or more views. So, for easy maintenance, the MVC framework requires a separate sub-folder for each controller with the same name as a controller, under the **Views** folder.

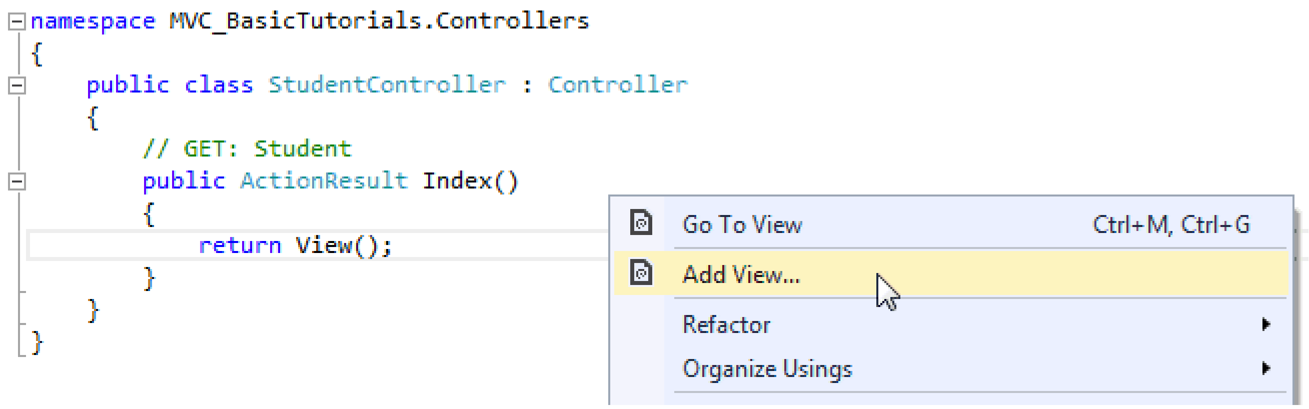
For example, all the views rendered from the HomeController will resides in the **Views** > **Home** folder. In the same way, views for StudentController will resides in **Views** > **Student** folder, as shown below.



Creating a View

You can create a view for an action method directly from it by right clicking inside an action method and select **Add View...**.

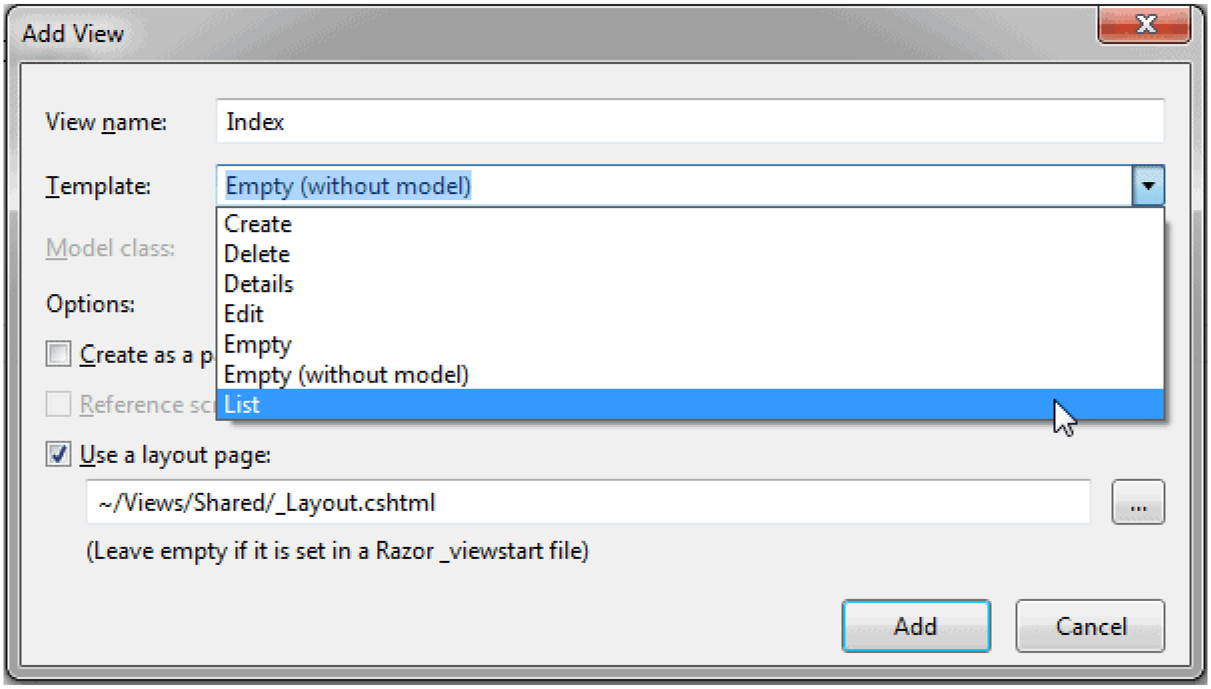
The following creates a view from the `Index()` action method of the `StudentContoller`, as shown below.



```
namespace MVC_BasicTutorials.Controllers
{
    public class StudentController : Controller
    {
        // GET: Student
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Create a View from Action Method

This will open the **Add View** dialogue box, shown below. It's good practice to keep the view name the same as the action method name so that you don't have to explicitly specify the view name in the action method while returning the view.



Add View

View name: Index

Template: Empty (without model)

Model class:

Options:

☐ Create as a p

☐ Reference sc

☑ Use a layout page:

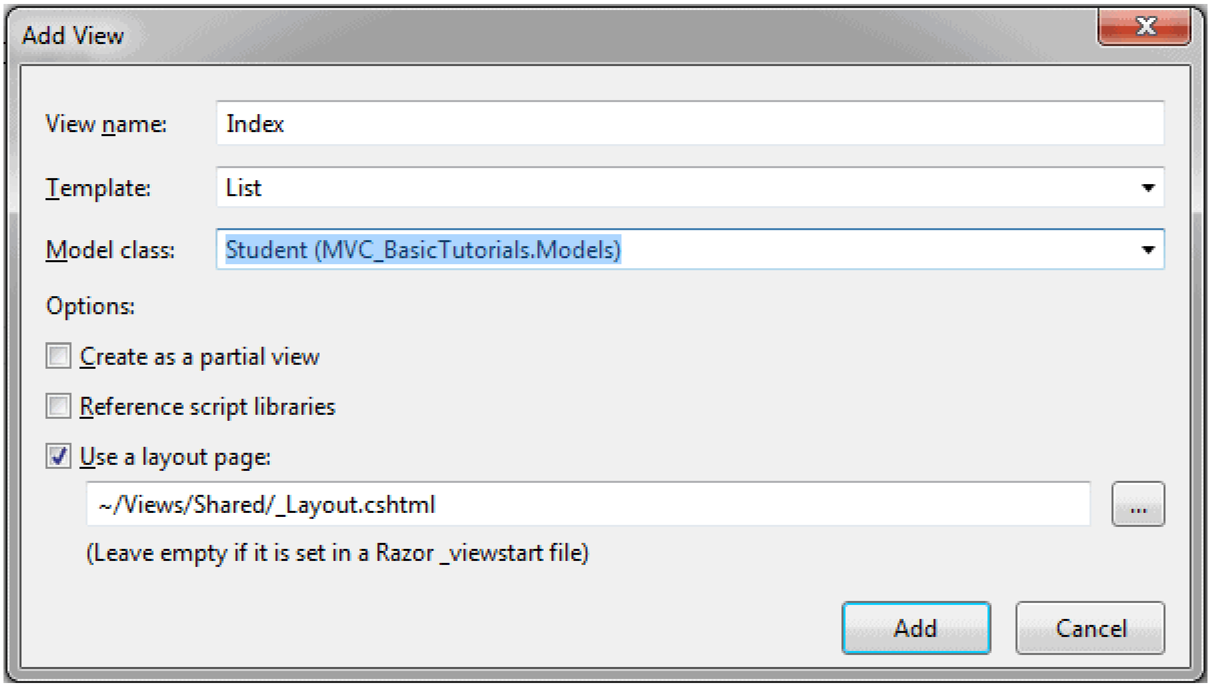
~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Add a View

Select the scaffolding template. Template dropdown will show default templates available for Create, Delete, Details, Edit, List, or Empty view. Select "List" template because we want to show the list of students in the view.

Now, select `Student` from the model class dropdown. The model class dropdown automatically displays the name of all the classes in the `Model` folder. We have already created the `Student` model class in the previous section, so it would be included in the dropdown.



View name: Index

Template: List

Model class: Student (MVC_BasicTutorials.Models)

Options:

□ Create as a partial view

□ Reference script libraries

☑ Use a layout page:

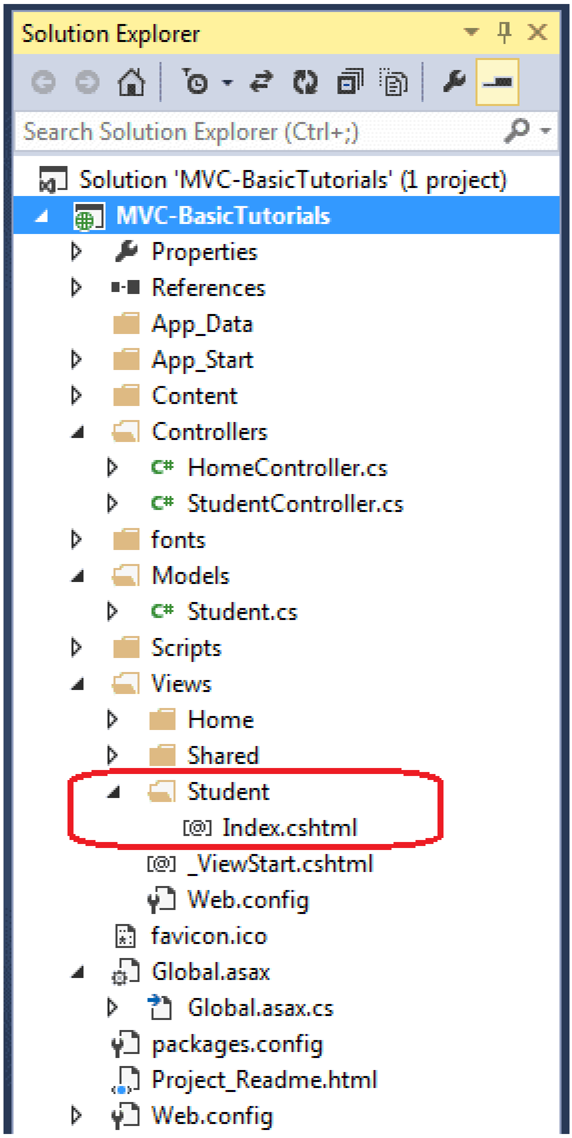
~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Add a View

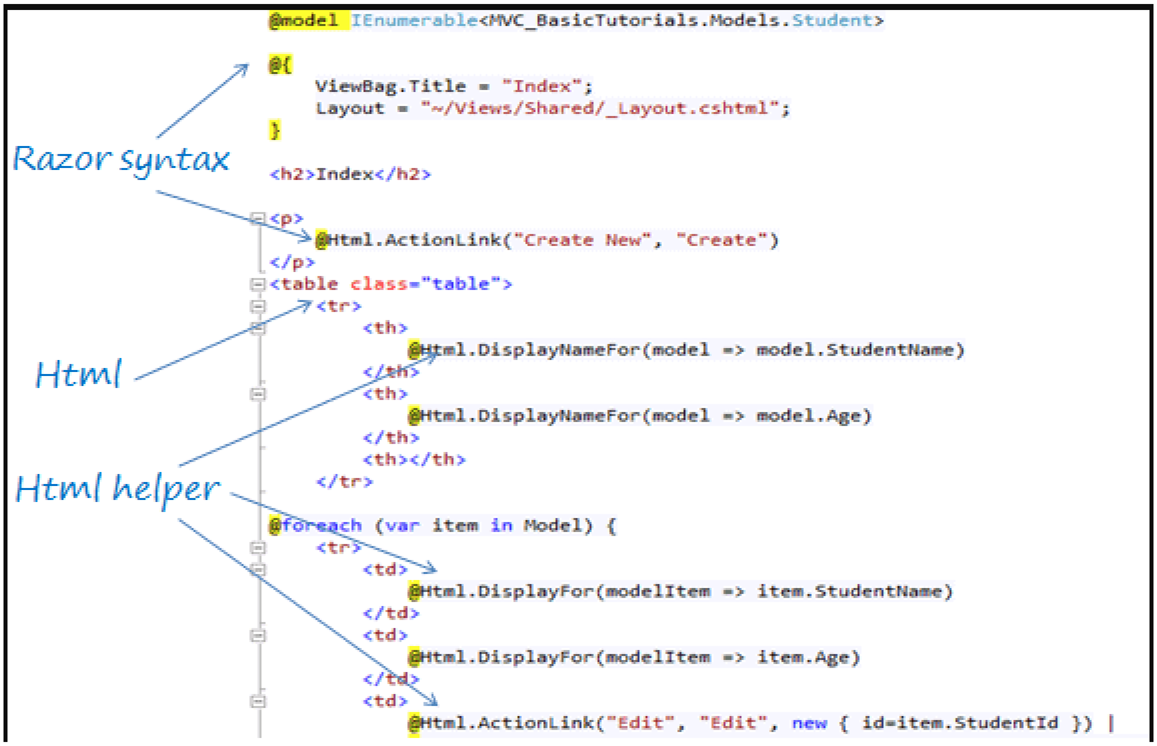
Check "Use a layout page" checkbox and select the default `_Layout.cshtml` page for this view and then click **Add** button.

This will create the `Index` view under **View** -> **Student** folder, as shown below:



View

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



```
@model IEnumerable<MVC_BasicTutorials.Models.Student>

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.StudentName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Age)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.StudentName)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Age)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |
```

Every view in the ASP.NET MVC is derived from `WebViewPage` class included in `System.Web.Mvc` namespace.

Partial Views

A partial view is a reusable portion of a web page. It is `.cshtml` or `.vbhtml` file that contains HTML code. It can be used in one or more Views or Layout Views. You can use the same partial view at multiple places and eliminates the redundant code.

Let's create a partial view for the following menu, so that we can use the same menu in multiple layout views without rewriting the same code everywhere.

A partial view is a Razor markup file (.cshtml) without an **@page** directive that renders HTML output *within* another markup file's rendered output.

The term *partial view* is used when developing either an MVC app, where markup files are called *views*, or a Razor Pages app, where markup files are called *pages*. This topic generically refers to MVC views and Razor Pages **pages as *markup files***.

When to use partial views

Partial views are an effective way to:

Break up large markup files into smaller components.

- In a large, complex markup file composed of several logical pieces, there's an advantage to working with each piece isolated into a partial view. The code in the markup file is manageable because the markup only contains the overall page structure and references to partial views.
- Reduce the duplication of common markup content across markup files.

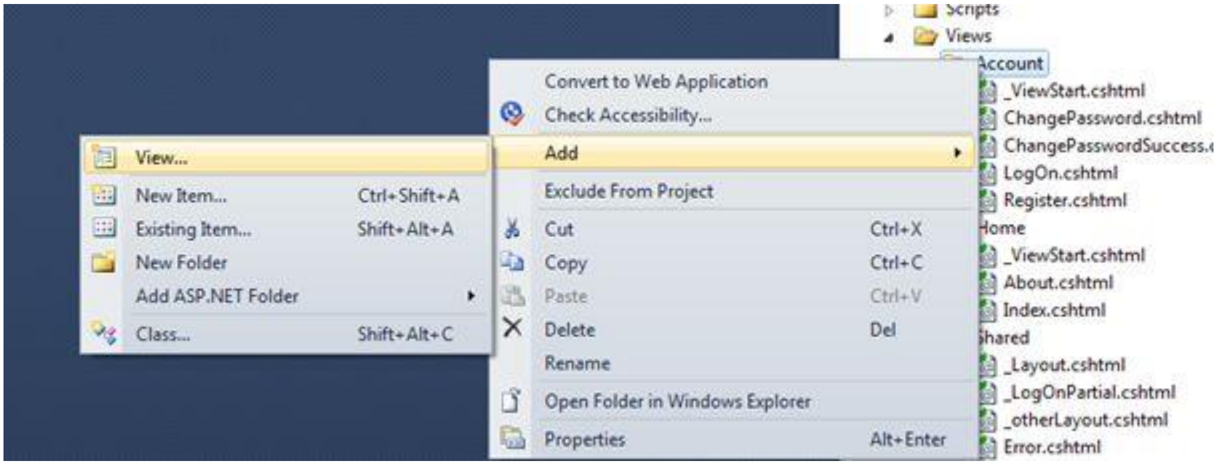
When the same markup elements are used across markup files, a partial view removes the duplication of markup content into one partial view file. When the markup is changed in the partial view, it updates the rendered output of the markup files that use the partial view.

Partial views shouldn't be used to maintain common layout elements. Common layout elements should be specified in **_Layout.cshtml** files.

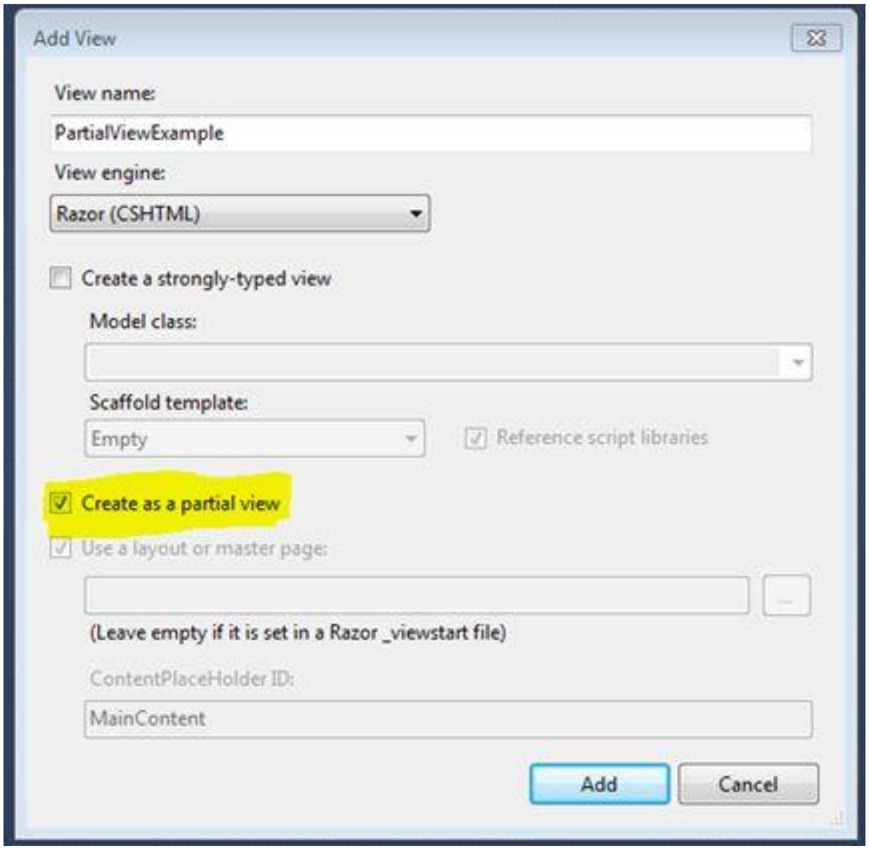
Don't use a partial view where complex rendering logic or code execution is required to render the markup. Instead of a partial view, use a **view component**.

Creating Partial View

To create a partial view, right-click on view -> shared folder and select Add -> View option. In this way we can add a partial view.



It is not mandatory to create a partial view in a shared folder but a partial view is mostly used as a reusable component, it is a good practice to put it in the "shared" folder.



Add View

View name: PartialViewExample

View engine: Razor (CSHTML)

☐ Create a strongly-typed view

Model class:

Scaffold template: Empty

☑ Reference script libraries

☑ Create as a partial view

☑ Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID: MainContent

HTML helper has two methods for rendering the partial view: Partial and RenderPartial.

```
<div>
    @Html.Partial("PartialViewExample")
</div>
<div>
    @{
        Html.RenderPartial("PartialViewExample");
    }
</div>
```

@Html.RenderPartial

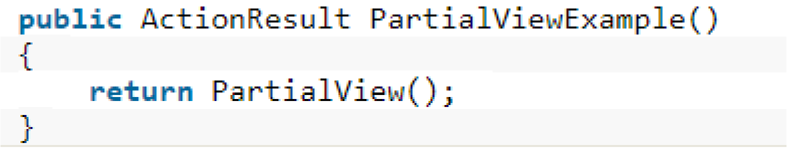
The result of the RenderPartial method is written directly into the HTTP response, it means that this method used the same TextWriter object as used by the current view. This method returns nothing.

@Html.Partial

This method renders the view as an HTML-encoded string. We can store the method result in a string variable. The Html.RenderPartial method writes output directly to the HTTP response stream so it is slightly faster than the Html.Partial method.

Partial View from Controller Action.

Returning a Partial view from the Controller's Action method:



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