

Model and View in ASP.NET MVC

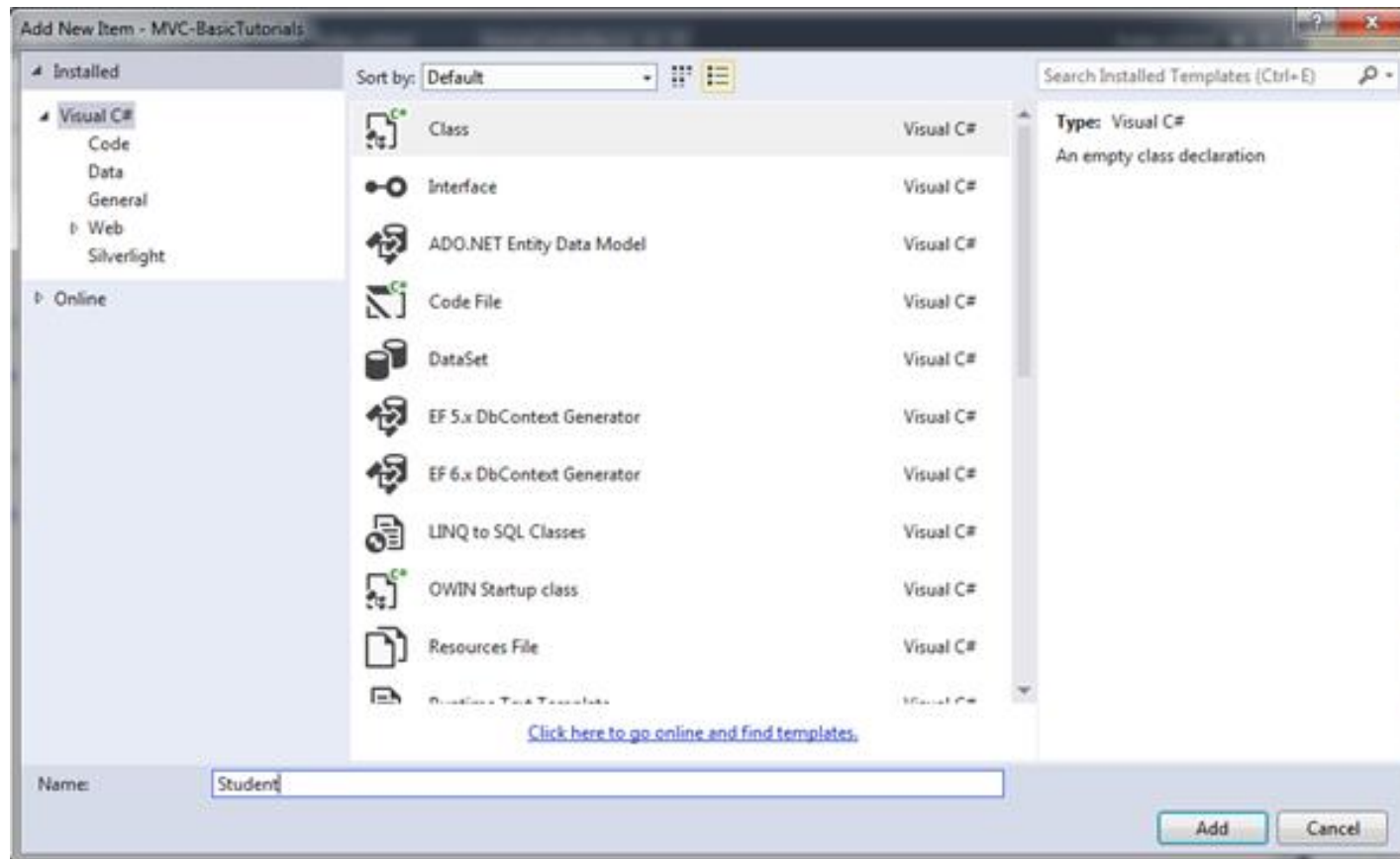
Tran Khai Thien

Introduction

- ▶ First, you will learn about the **Model** in ASP.NET MVC framework.
- ▶ Model represents domain specific data and business logic in MVC architecture. It maintains the data of the application. Model objects retrieve and store model state in the persistence store like a database.
- ▶ Model class holds data in public properties. All the Model classes reside in the Models folder in MVC folder structure.

Adding New Model

- ▶ Open our first MVC project created in previous step in the Visual Studio. Right click on Model folder -> Add -> click on Class..
- ▶ In the Add New Item dialog box, enter class name 'Student' and click **Add**.



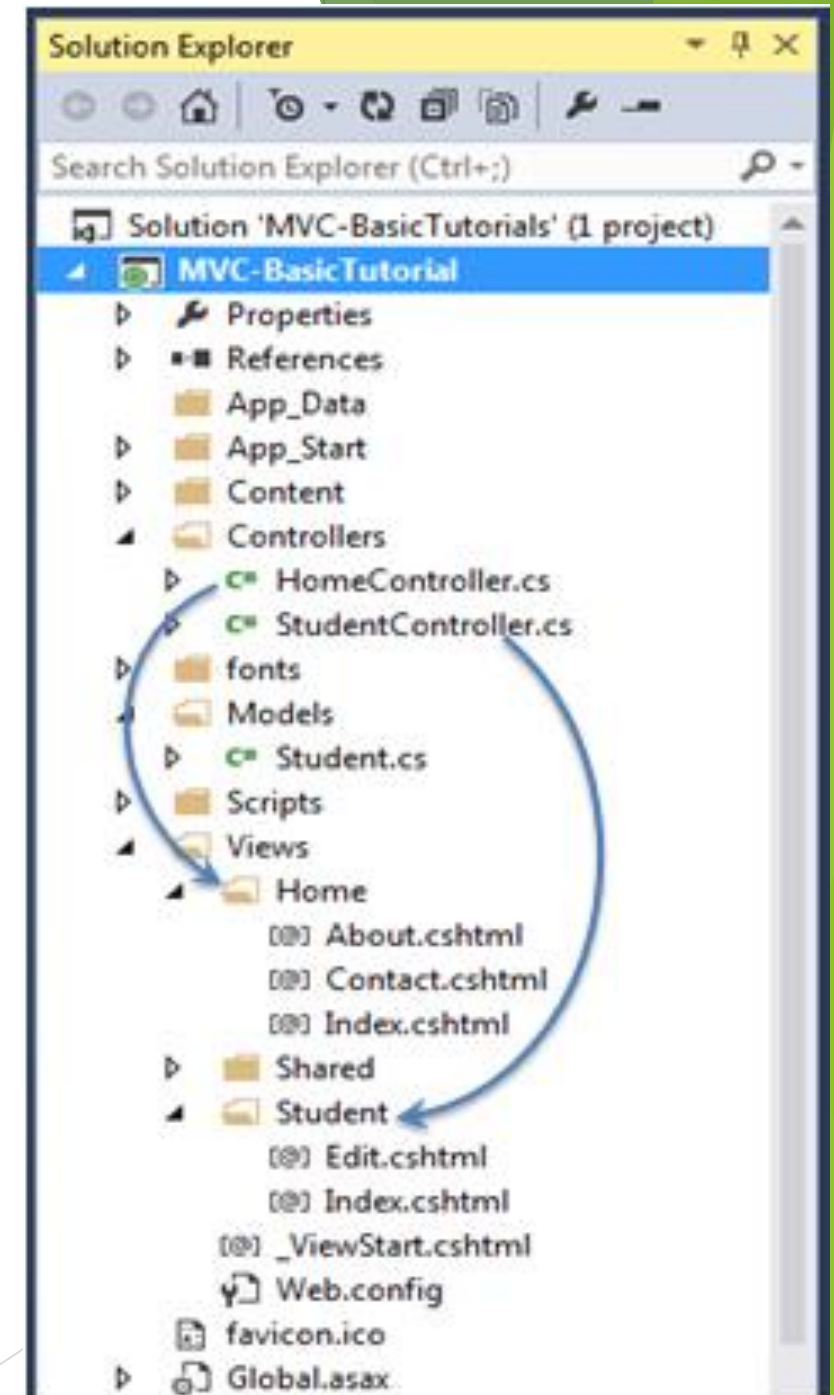
Adding a New Model

- This will add new Student class in model folder. Now, add Id, Name, Age properties as shown below.

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

View in ASP.NET MVC

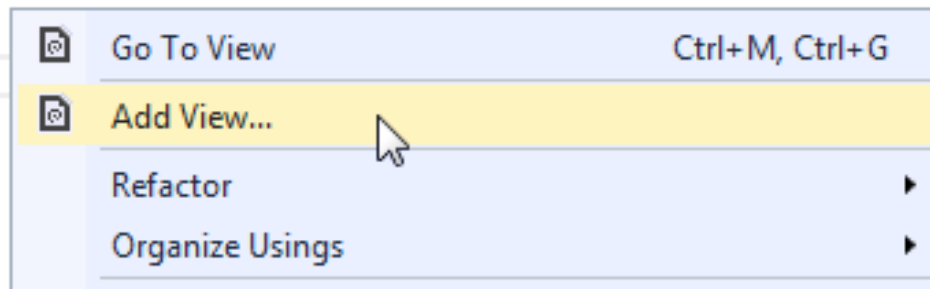
- ▶ **View** is a user interface. View displays data from the model to the user and also enables them to modify the data.
- ▶ ASP.NET MVC views are stored in **Views** folder. **Different action methods of a single controller class can render different views**, so the Views folder contains a separate 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 reside in Views > Student folder as shown below.



Adding a New View

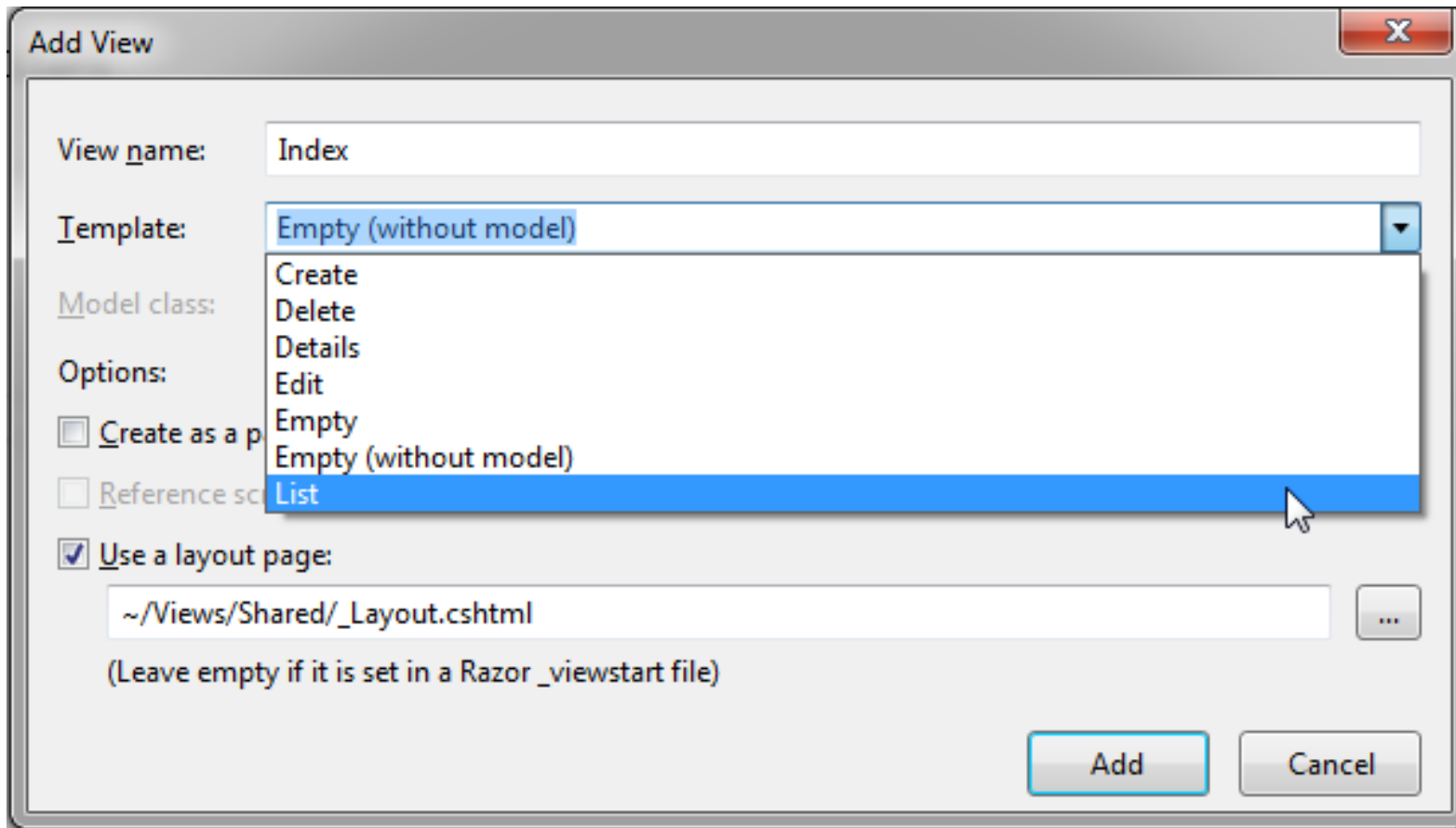
- We will create a view, which will be rendered from Index method of StudentController. So, open a StudentController class -> right click inside Index method -> click **Add View**.

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



Adding a New 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 list of students in the view.



Add View

View name: Index

Template: Empty (without model)

Model class:

Options:

☐ Create as a partial view

☐ Reference scaffolding model

☒ Use a layout page:

~/Views/Shared/_Layout.cshtml

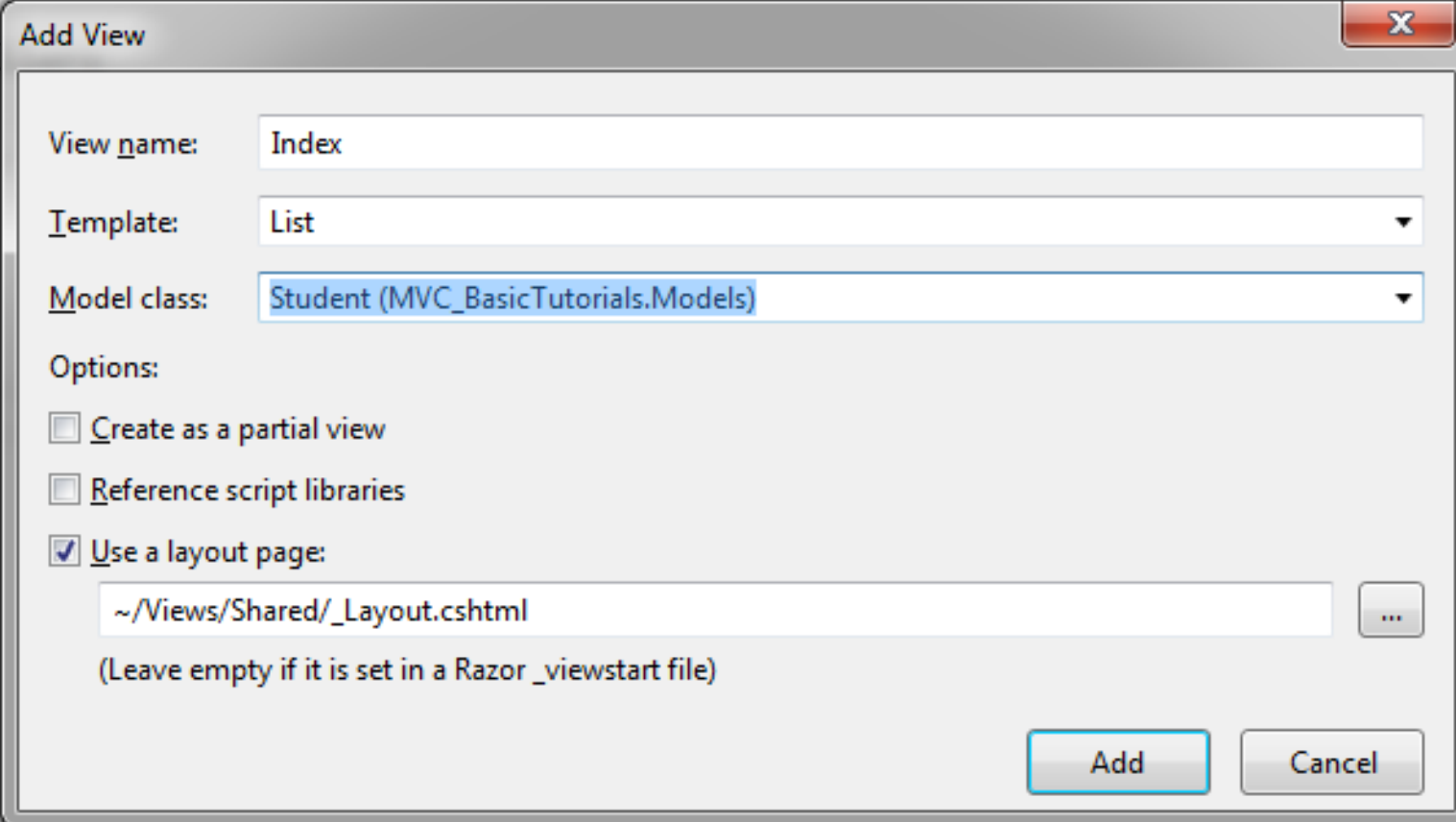
(Leave empty if it is set in a Razor _viewstart file)

Add Cancel

Adding a New View

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

▶



Add View

View name: Index

Template: List

Model class: Student (MVC_BasicTutorials.Models)

Options:

☐ Create as a partial view

☐ Reference script libraries

☒ Use a layout page:

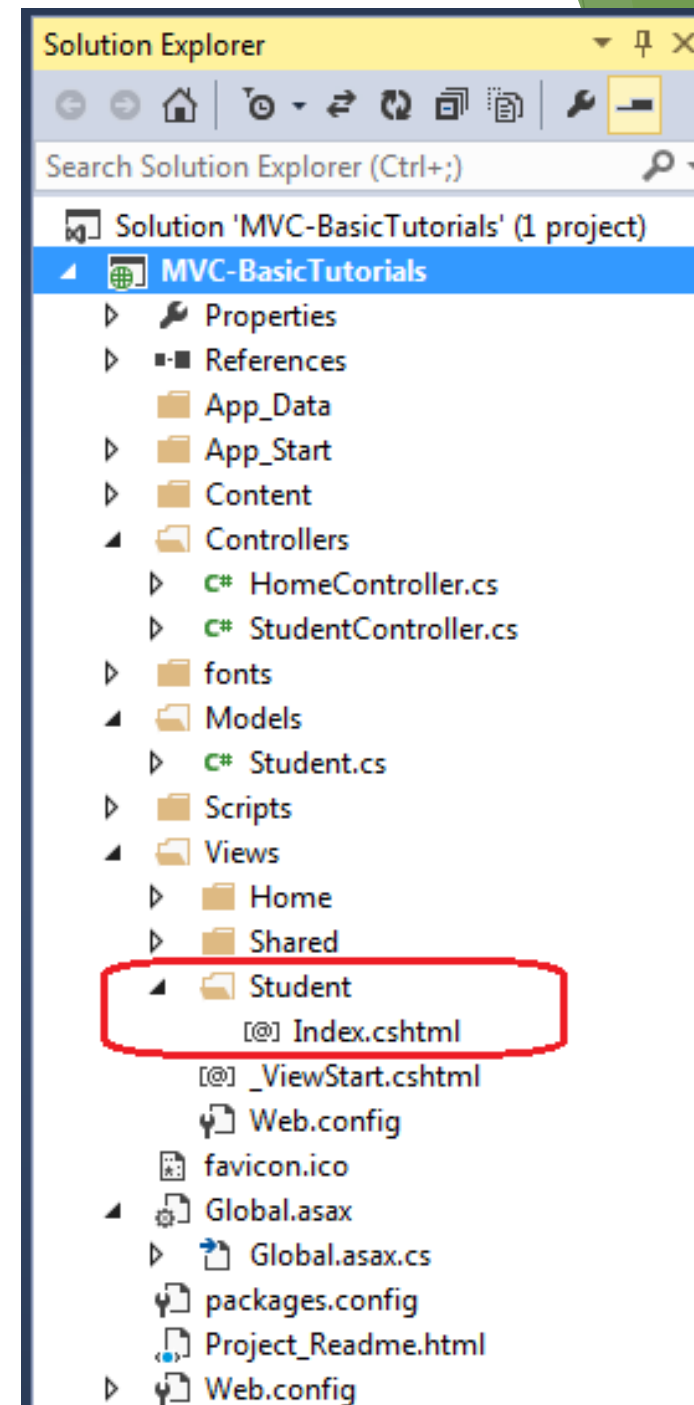
~/Views/Shared/_Layout.cshtml ...

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

Add Cancel

Adding a New View

- ▶ Check "Use a layout page" checkbox and select _Layout.cshtml page for this view and then click **Add** button. We will see later what is layout page but for now think it like a master page in MVC.
- ▶ This will create Index view under View -> Student folder as shown.



Adding a New View

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

Razor view engine: Microsoft introduced the Razor view engine and packaged with MVC 3. You can write a mix of html tags and server side code in razor view. Razor uses @ character for server side code instead of traditional <% %>. You can use C# or Visual Basic syntax to write server side code inside razor view. Razor view engine maximize the speed of writing code by minimizing the number of characters and keystrokes required when writing a view. Razor views files have .cshtml or vbhtml extension.

```
@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 }) |
                @Html.ActionLink("Details", "Details", new { id=item.StudentId }) |
                @Html.ActionLink("Delete", "Delete", new { id = item.StudentId })
            </td>
        </tr>
    }
</table>
```

Adding a New View

As you can see in the above Index view, it contains both **Html** and **razor** codes.

Inline razor expression starts with **@** symbol.
@Html is a **helper class** to generate html controls.

Razor syntax

```
@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 }) |
```

Html

Html helper

Adding a New View

The above Index view would look like below.

Application name Home About Contact		
Index		
Create New		
Name	Age	
John	18	Edit Details Delete
Steve	21	Edit Details Delete
Bill	25	Edit Details Delete
Ram	20	Edit Details Delete
Ron	31	Edit Details Delete
Chris	17	Edit Details Delete
Rob	19	Edit Details Delete
© 2014 - My ASP.NET Application		

Adding a New View

Points to Remember

- ▶ View is a User Interface which displays data and handles user interaction.
- ▶ Views folder contains separate folder for each controller.
- ▶ ASP.NET MVC supports Razor view engine in addition to traditional .aspx engine.
- ▶ Razor view files has .cshtml or .vbhtml extension.

Integrating Controller, View and Model:

- ▶ We have already created StudentController, model and view in the previous sections, but we have not integrated all these components in-order to run it.
- ▶ The following code snippet shows Student Controller and Student Model class and view created in the previous sections.

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

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

Integrating Controller, View and Model

Now, to run it successfully, we need to pass a model object from controller to Index view.

```
@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 }) |
                @Html.ActionLink("Details", "Details", new { id=item.StudentId }) |
                @Html.ActionLink("Delete", "Delete", new { id = item.StudentId })
            </td>
        </tr>
    }
</table>
```

Integrating Controller, View and Model

Now, to run it successfully, we need to pass a model object from controller to Index view. As you can see in the above Index.cshtml, it uses IEnumerable of Student as a model object. So we need to pass IEnumerable of Student model from the Index action method of StudentController class as shown below.

```
public class StudentController : Controller
{
    public ActionResult Index()
    {
        var studentList = new List<Student>{
            new Student() { StudentId = 1, StudentName = "John", Age = 18 } ,
            new Student() { StudentId = 2, StudentName = "Steve", Age = 21 } ,
            new Student() { StudentId = 3, StudentName = "Bill", Age = 25 } ,
            new Student() { StudentId = 4, StudentName = "Ram" , Age = 20 } ,
            new Student() { StudentId = 5, StudentName = "Ron" , Age = 31 } ,
            new Student() { StudentId = 4, StudentName = "Chris" , Age = 17 } ,
            new Student() { StudentId = 4, StudentName = "Rob" , Age = 19 }
        };
        return View(studentList);
    }
}
```


Integrating Controller, View and Model

Now, you can run the MVC project by pressing F5 and navigate to *http://localhost/Student*. You will see following view in the browser.

Application name Home About Contact		
Index		
Create New		
Name	Age	
John	18	Edit Details Delete
Steve	21	Edit Details Delete
Bill	25	Edit Details Delete
Ram	20	Edit Details Delete
Ron	31	Edit Details Delete
Chris	17	Edit Details Delete
Rob	19	Edit Details Delete

Razor Syntax

Inline expression:

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

```
<h1>Razor syntax demo</h1>  
<h2>@DateTime.Now.ToShortDateString()</h2>
```

Razor syntax demo
03-20-2018

Razor Syntax

Multi-statement Code block:

You can write multiple line of server side code enclosed in braces @{ ... }. Each line must ends with semicolon same as C#.

```
@{  
    var date = DateTime.Now.ToShortDateString();  
    var message = "Hello World";  
}  
  
<h2>Today's date is: @date </h2>  
<h3>@message</h3>
```

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

Razor Syntax

Display text from code block:

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

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

Razor Syntax

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

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

Razor Syntax

if-else condition:

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

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

Razor Syntax

Loop

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

```
0  
1  
2  
3  
4
```

Razor Syntax

Model:

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

```
@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>
```


Razor Syntax

Declare Variables:

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

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

Razor Syntax

Points to Remember:

- Use @ to write server side code.
- Server side code block starts with @{* code * }
- Use @: or <text></<text> to display text from code block.
- if condition starts with @if{ }
- for loop starts with @for
- @model allows you to use model object anywhere in the view.

HTML Helpers

HtmlHelper class generates html elements using the model class object in razor view. It binds the model object to html elements to display value of model properties into html elements and also assigns the value of the html elements to the model properties while submitting web form. So always use HtmlHelper class in razor view instead of writing html tags manually.

The following figure shows the use of HtmlHelper class in the razor view.

```
@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>
```

Extension method for
HtmlHelper

HtmlHelper

HTML Helpers

The following table lists HtmlHelper methods and html control each method generates.

HtmlHelper	Strogly Typed HtmlHelpers	Html Control
Html.ActionLink		Anchor link
Html.TextBox	Html.TextBoxFor	Textbox
Html.TextArea	Html.TextAreaFor	TextArea
Html.CheckBox	Html.CheckBoxFor	Checkbox
Html.RadioButton	Html.RadioButtonFor	Radio button
Html.DropDownList	Html.DropDownListFor	Dropdown, combobox
Html.ListBox	Html.ListBoxFor	multi-select list box
Html.Hidden	Html.HiddenFor	Hidden field
Password	Html.PasswordFor	Password textbox
Html.Display	Html.DisplayFor	Html text
Html.Label	Html.LabelFor	Label

HTML Helpers

The following table lists HtmlHelper methods and html control each method generates.

@Html.ActionLink("Login","Index","Login")

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-
width"/>
<title>Index</title>
<link href="/Content/site.css" rel="stylesheet"/>
<script src="/Scripts/modernizr-2.5.3.js"> </script>
</head>
<body>
<h2>Index</h2>
<form action="/PersonDetails/index" method="post">
</form>
<div>
<a href="/Login">Login</a>
</div>
<script src="/Scripts/jquery-1.7.1.js"> </script>
<script src="/Scripts/jquery.unobtrusive-ajax.js"> </script>
<script src="/Scripts/jquery.validate.js"> </script>
<script src="/Scripts/jquery.validate.unobtrusive.js"> </scri
pt>
</body>
</html>
```

HTML Helpers

Syntax of Basic HTML Helper Controls

Following are syntaxes of using basic html helper controls in ASP.NET MVC application.

Textbox - @Html.TextBox("UserName")

TextArea - @Html.TextArea("UserAddress")

Password - @Html.Password("UserPassword")

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

Checkbox - @Html.CheckBox("Check")

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

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

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

DisplayName - @Html.DisplayName("Displayname")

Editor - @Html.Editor("EditID")

HTML Helpers

Now if we run above code we will get basic html helper result like as shown below

UserDetails

Textbox

TextArea

Password

Hidden field

CheckBox

☐

RadioButton

☐ Male

☒ Female

DropDownList

ListBox

One

Two

Three

DisplayName Displayname

Editor

HTML Helpers

Points to Remember:

- HtmlHelper extension method generates html elements based on model properties.
- It is advisable to use "For" extension methods for compile time type checking e.g. TextBoxFor, EditorFor, CheckBoxFor etc.

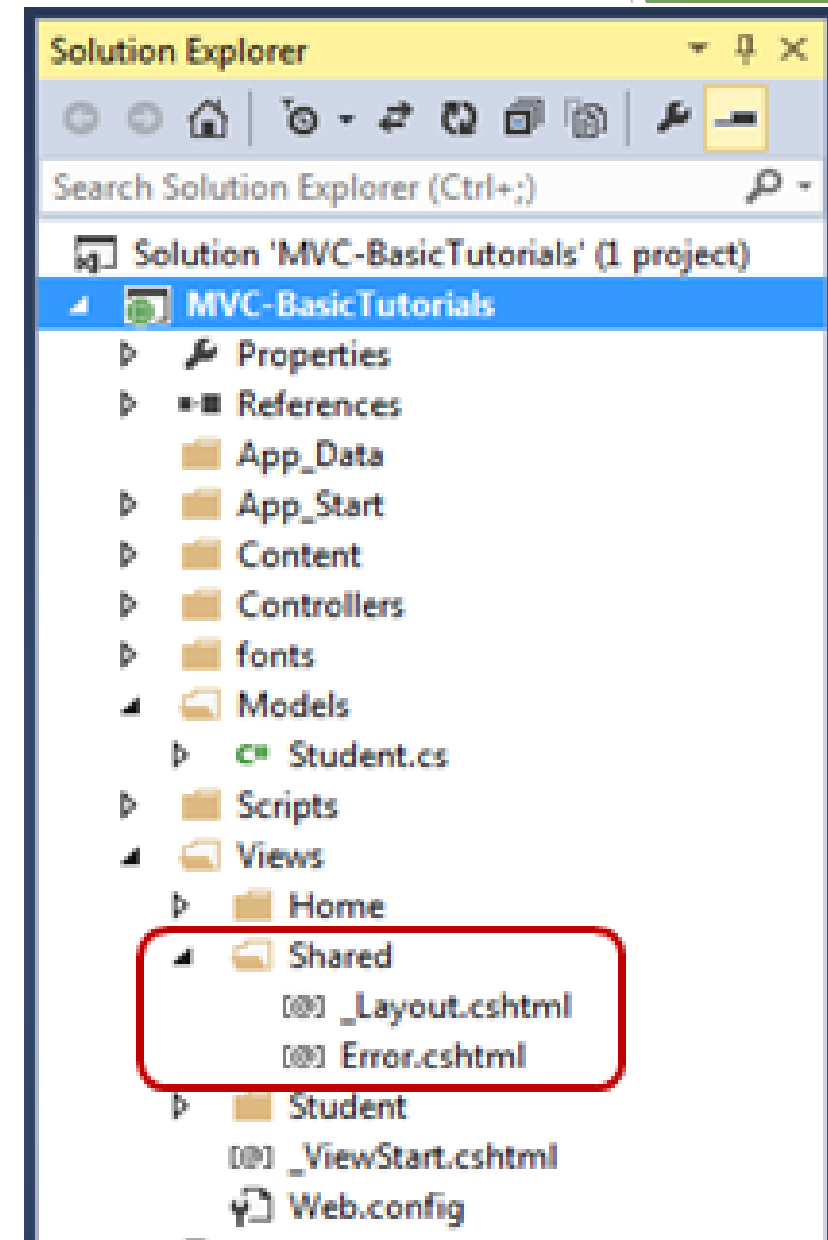
Ref: Slides Razor and Helper – ThS. Nguyễn Nghiêm

Layout View

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

Layout View

The razor layout view has same extension as other views, .cshtml or .vbhtml. Layout views are shared with multiple views, so it must be stored in the Shared folder. For example, when we created our first MVC application in the previous section, it also created `_Layout.cshtml` in the Shared folder as shown below.



Layout View

The following is an auto-generated _Layout.cshtml.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>@ViewBag.Title - My ASP.NET Application</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#navbar" >
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, null)
      </div>
```

The layout view contains html Doctype, head and body as normal html, the only difference is call to **RenderBody()** and **RenderSection()** methods.

RenderBody acts like a placeholder for other views. For example, Index.cshtml in the home folder will be injected and rendered in the layout view, where the RenderBody() method is being called. You will learn about these rendering methods later in this section.

Layout View

The following is an auto-generated _Layout.cshtml.

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
  </ul>
</div>
</div>
</div>
<div class="container body-content">
  @RenderBody()
  <hr />
  <footer>
    <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
  </footer>
</div>

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
</body>
</html>
```

The layout view contains html Doctype, head and body as normal html, the only difference is call to **RenderBody()** and **RenderSection()** methods.

RenderBody acts like a placeholder for other views. For example, Index.cshtml in the home folder will be injected and rendered in the layout view, where the RenderBody() method is being called. You will learn about these rendering methods later in this section.

Layout View


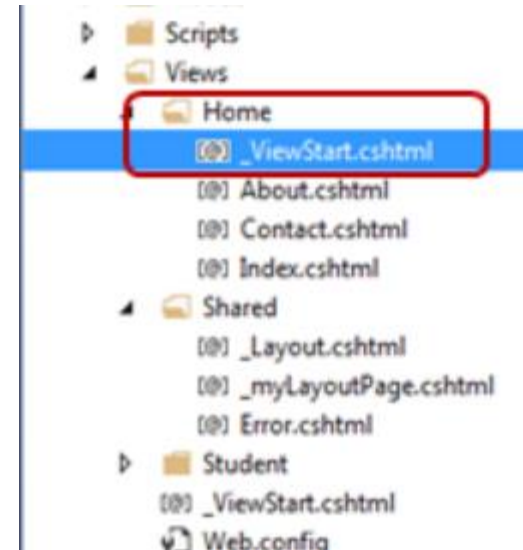
Use Layout View:

You can set the layout view in multiple ways, by using `_ViewStart.cshtml` or setting up path of the layout page using Layout property in the individual view or specifying layout view name in the action method.

_ViewStart.cshtml:

`_ViewStart.cshtml` is included in the Views folder by default. It sets up the default layout page for all the views in the folder and its subfolders using the Layout property.

The following `_ViewStart.cshtml` in the **Views** folder, sets the Layout property to `"~/Views/Shared/_Layout.cshtml"`. So now, `_layout.cshtml` would be layout view of all the views included in Views and its subfolders. So by default, all the views derived default layout page from `_ViewStart.cshtml` of Views folder.

A screenshot of a code editor showing the content of the `_ViewStart.cshtml` file. The code consists of a single line: `Layout = "~/Views/Shared/_myLayoutPage.cshtml";`. This line is enclosed in a red rectangular box. The editor's title bar shows the file name `_ViewStart.cshtml` and standard window controls.

Layout View

Setting Layout property in individual view:

You can also **override default layout page** set by `_ViewStart.cshtml` by setting `Layout` property in each individual `.cshtml` view. For example, the following `Index` view use `_myLayoutPage.cshtml` even if `_ViewStart.cshtml` set `_Layout.cshtml`.

```
@{
    ViewBag.Title = "Home Page";
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";
}
<div class="jumbotron">
    <h1>ASP.NET</h1>
    <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>
    <p><a href="http://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
</div>
<div class="row">
    <div class="col-md-4">
        <h2>Getting started</h2>
        <p>
            ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.
        </p>
        <p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301865">Learn more &raquo;</a></p>
    </div>
    <div class="col-md-4">
        <h2>Get more libraries</h2>
        <p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.</p>
        <p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301866">Learn more &raquo;</a></p>
    </div>
    <div class="col-md-4">
        <h2>Web Hosting</h2>
        <p>You can easily find a web hosting company that offers the right mix of features and price for your applications.</p>
        <p><a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301867">Learn more &raquo;</a></p>
    </div>
</div>
```

Layout View

Specify Layout page in ActionResult method:

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

The following example, View() method renders Index view using _myLayoutPage.cshtml.

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

    public ActionResult About()
    {
        return View();
    }

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

Layout View

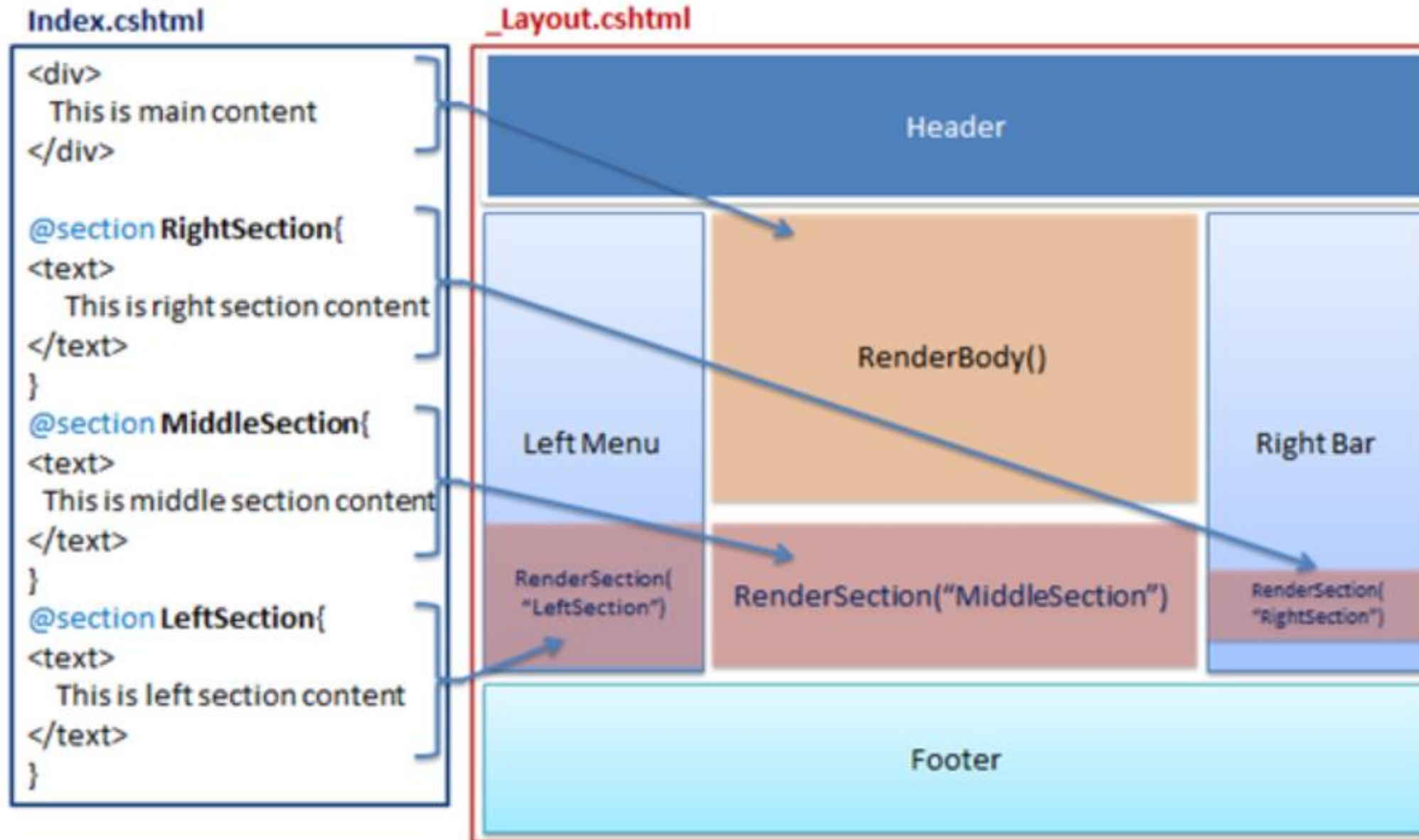
Rendering Methods:

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 RenderBody() method.
RenderSection(string name)	Renders a content of named section and specifies whether the section is required. RenderSection() is optional in Layout view.

Layout View

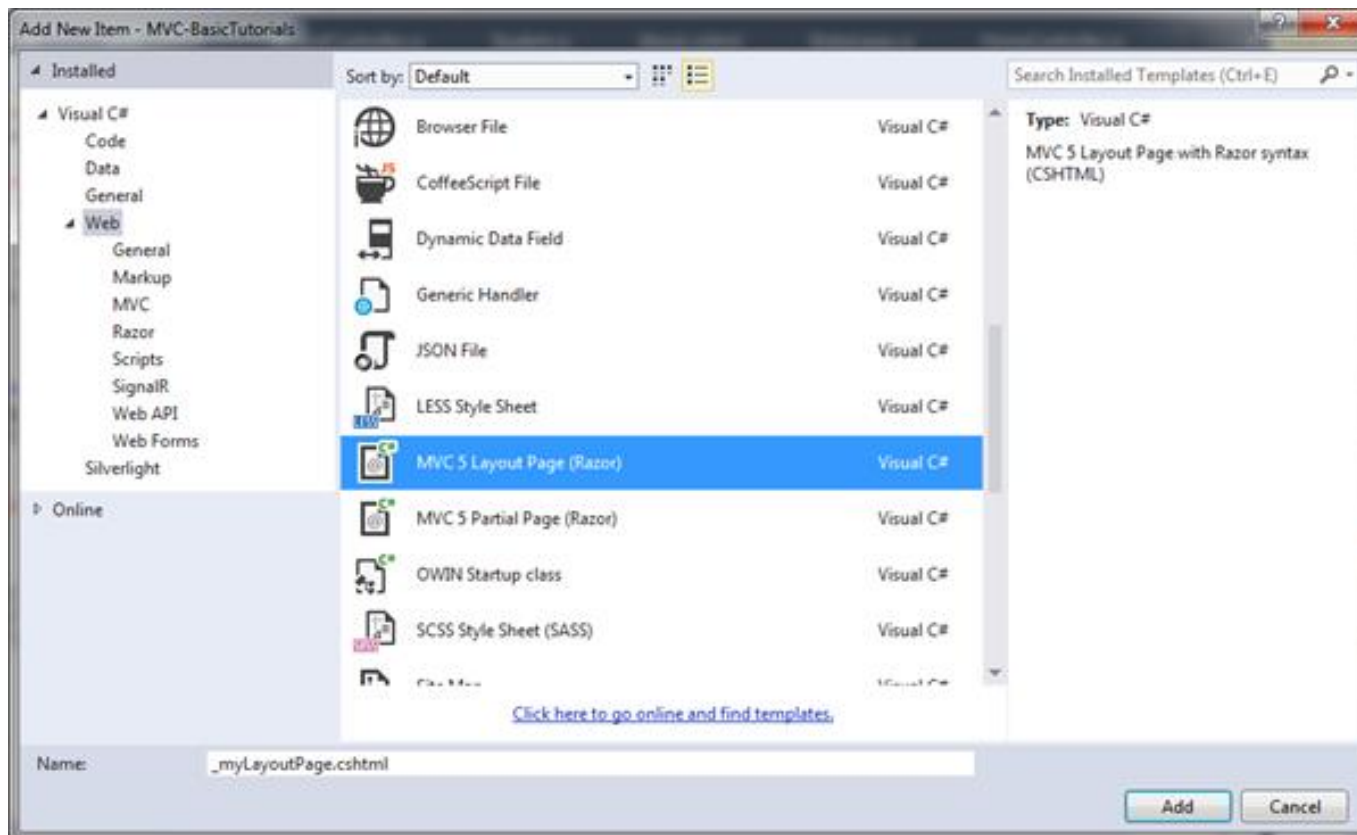
The following figure illustrates the use of **RenderBody** and **RenderSection** methods.



Layout View

Create Layout View:

- ▶ To create a new layout view in Visual Studio, **right click on shared folder** -> select **Add** -> click on **New Item..**
- ▶ In the Add New Item dialogue box, select **MVC 5 Layout Page (Razor)** and give the layout page name as **"_myLayoutPage.cshtml"** and click **Add**.



Layout View

Create Layout View:

- ▶ You will see _myLayoutPage.cshtml as shown below.

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

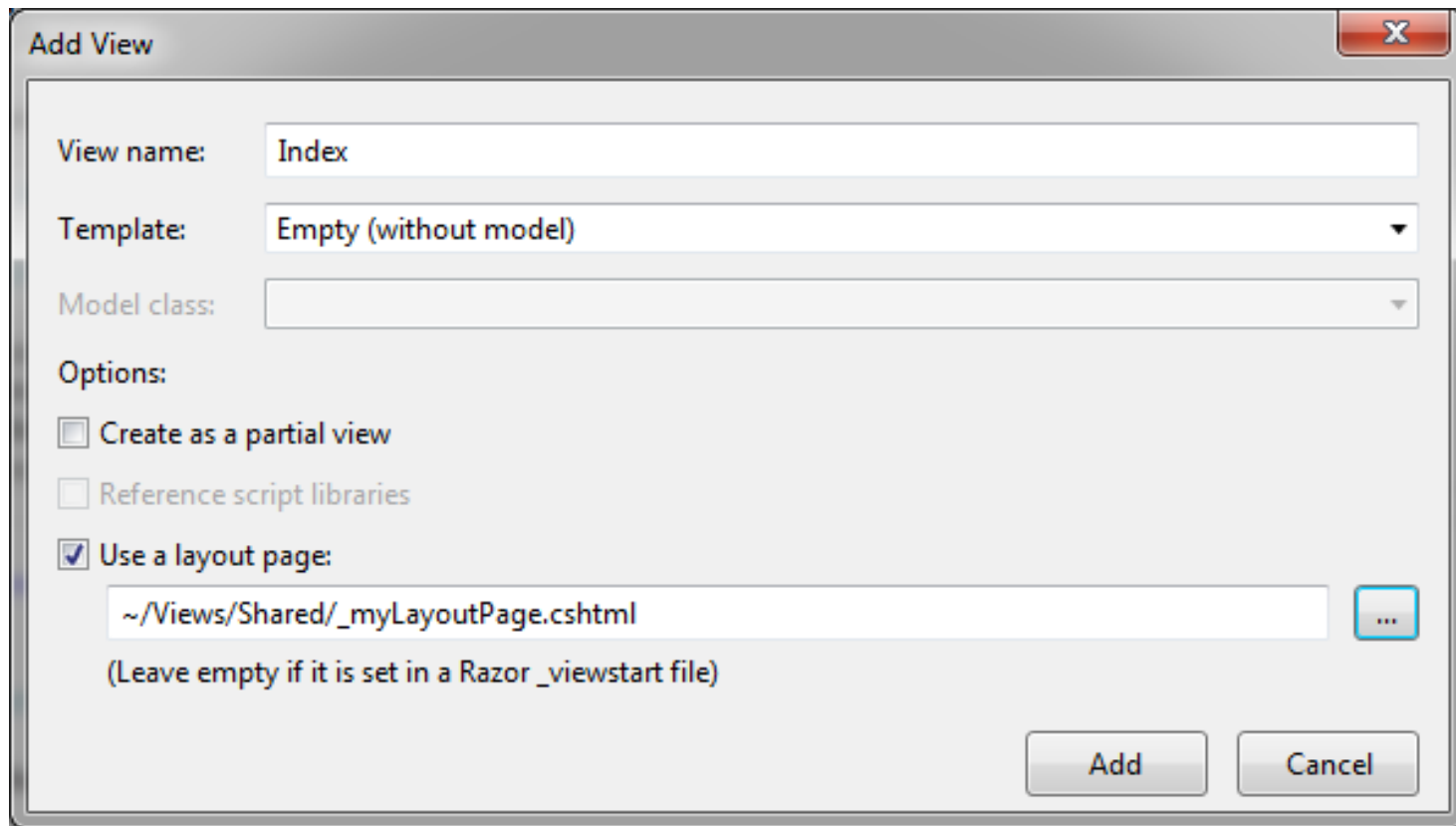
Layout View

Adding RenderSection in Layout: Now, add the <footer> tag with the `@RenderSection("footer",true)` method along with some styling as shown below. Please notice that we made this section as required. This means any view that uses `_myLayoutPage` as its layout view must include a footer section.

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>@ViewBag.Title</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  <div>
    @RenderBody()
  </div>
  <footer class="panel-footer">
    @RenderSection("footer", true)
  </footer>
</body>
</html>
```

Layout View

- ▶ Now, let's use this `_myLayoutPage.cshtml` with the Index view of **HomeController** (or adding a new Controller).
- ▶ You can add an empty Index view by right clicking on Index action method of HomeController and select Add View. Select Empty as a scaffolding template and `_myLayoutPage.cshtml` as layout view and click Add.



The screenshot shows the 'Add View' dialog box with the following configuration:

- View name:** Index
- Template:** Empty (without model)
- Model class:** (empty)
- Options:**
 - ☐ Create as a partial view
 - ☐ Reference script libraries
 - ☒ Use a layout page:
 - ~/Views/Shared/_myLayoutPage.cshtml
 - (Leave empty if it is set in a Razor _viewstart file)

The 'Add' button is highlighted at the bottom right of the dialog.

Layout View

- This will create Index.cshtml as shown below.

```
@{  
    ViewBag.Title = "Home Page";  
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";  
}  
  
<h2>Index</h2>
```

Layout View

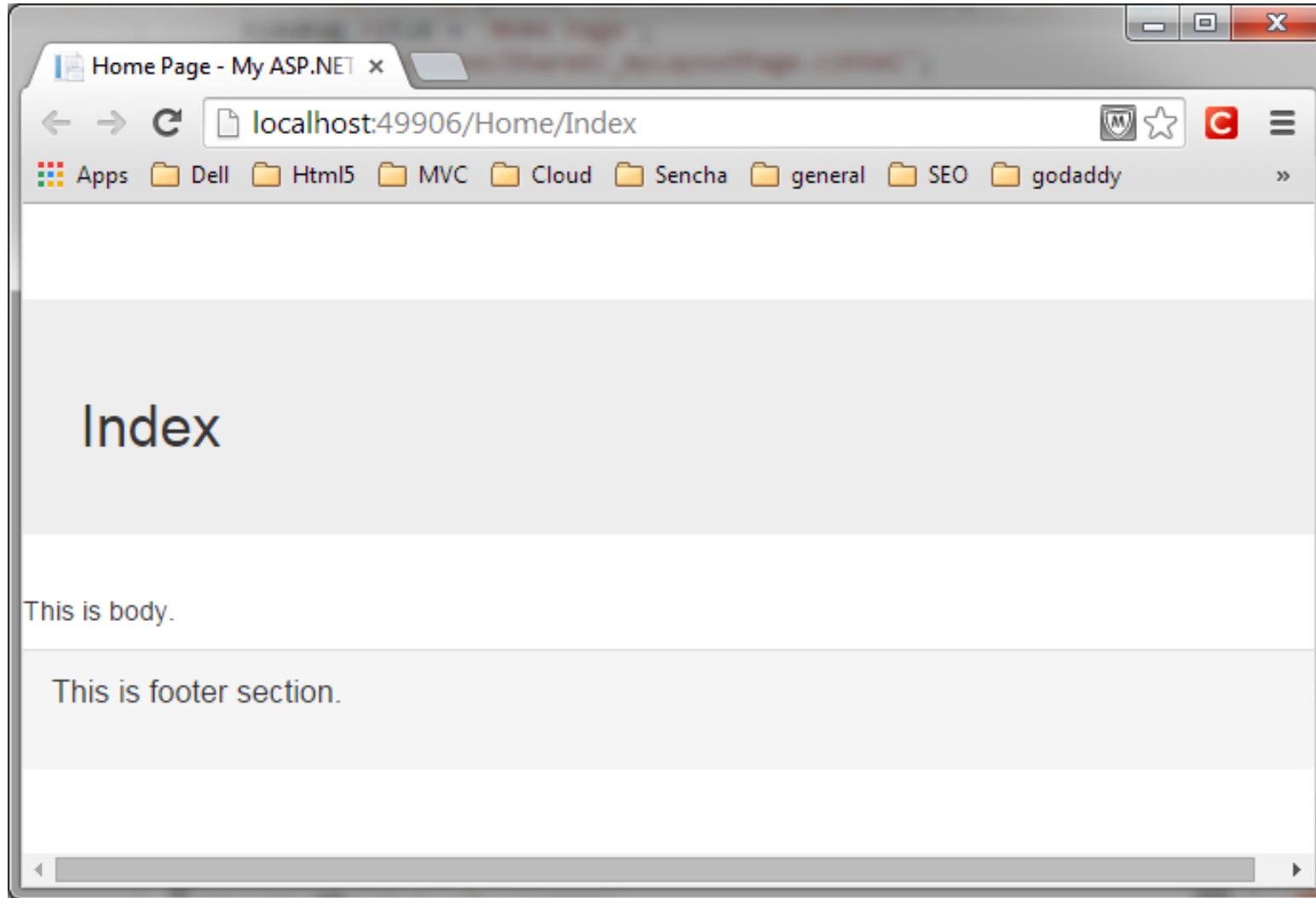
- **Adding Section in View and Rendering Sections:** So now, we have created Index view that uses our `_myLayoutPage.cshtml` as a layout view. We will now add footer section along with some styling because `_myLayoutPage` requires footer section.

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

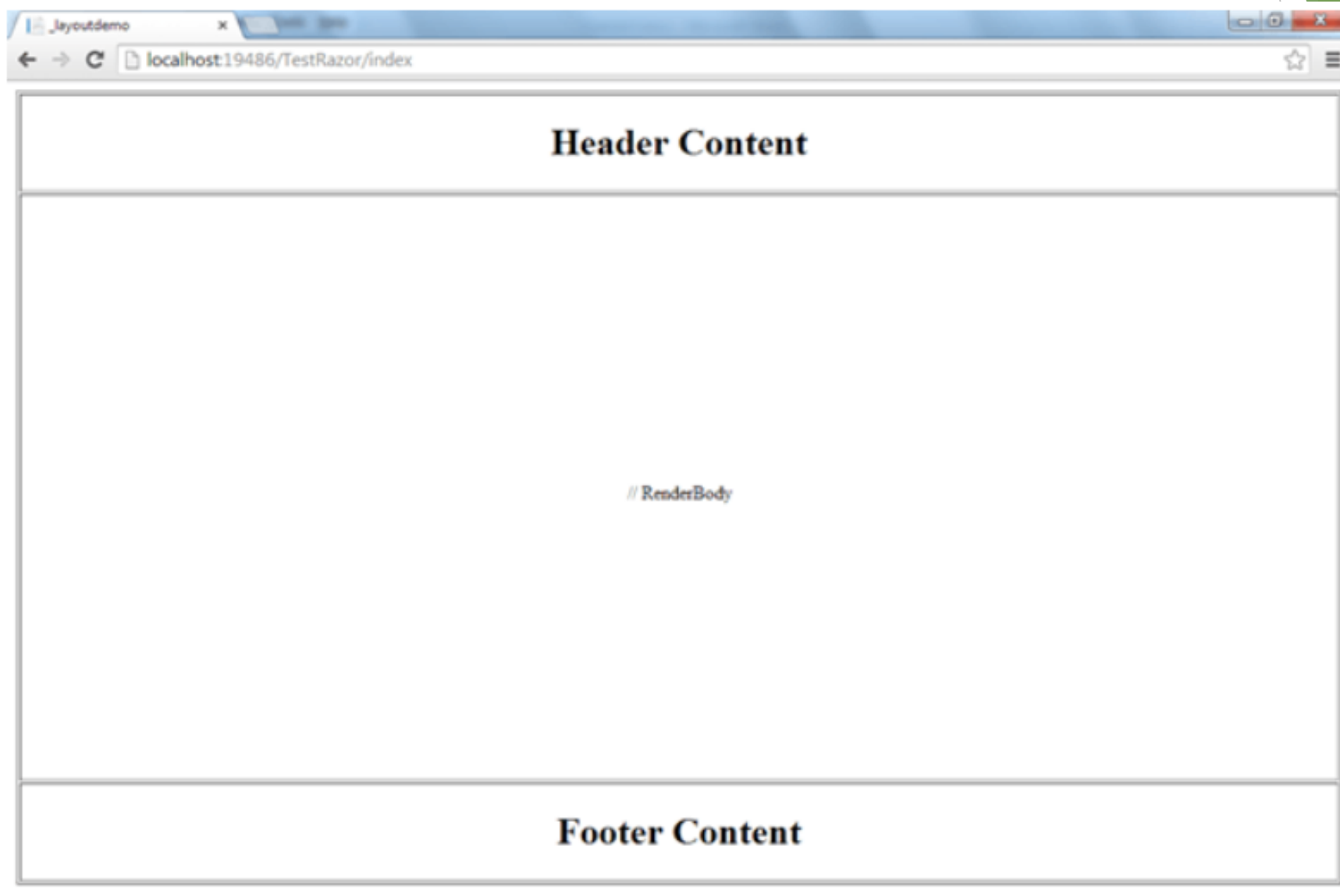
<div class="jumbotron">
    <h2>Index</h2>
</div>
<div class="row">
    <div class="col-md-4">
        <p>This is body.</p>
    </div>
    @section footer{
        <p class="lead">
            This is footer section.
        </p>
    }
</div>
```

Layout View

- Now, run the application and you will see Index view will contain body and footer part as shown below.



Layout View - Question?



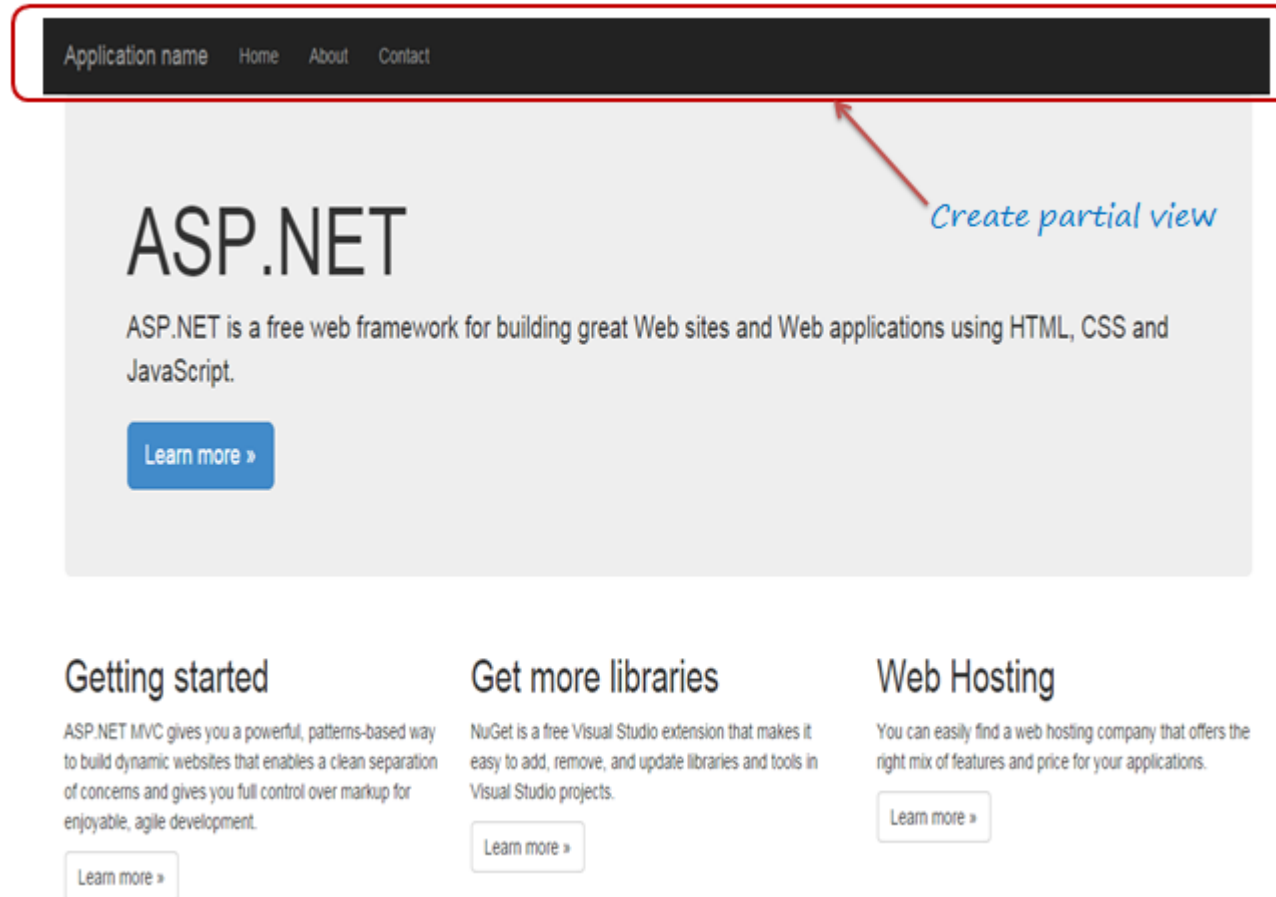
Layout View

Points to Remember:

- The Layout view contains common parts of a UI. It is same like masterpage of ASP.NET webforms.
- `_ViewStart.cshtml` file can be used to specify path of layout page, which in turn will be applicable to all the views of the folder and its subfolder.
- You can set the Layout property in the individual view also, to override default layout page setting of `_ViewStart.cshtml`
- Layout view uses two rendering methods: `RenderBody()` and `RenderSection()`.
- `RenderBody` can be used only once in the layout view, whereas the `RenderSection` method can be called multiple time with different name.
- `RenderBody` method renders all the content of view which is not wrapped in named section.
- `RenderSection` method renders the content of a view which is wrapped in named section.
- `RenderSection` can be configured as required or optional. If required, then all the child views must included that named section.

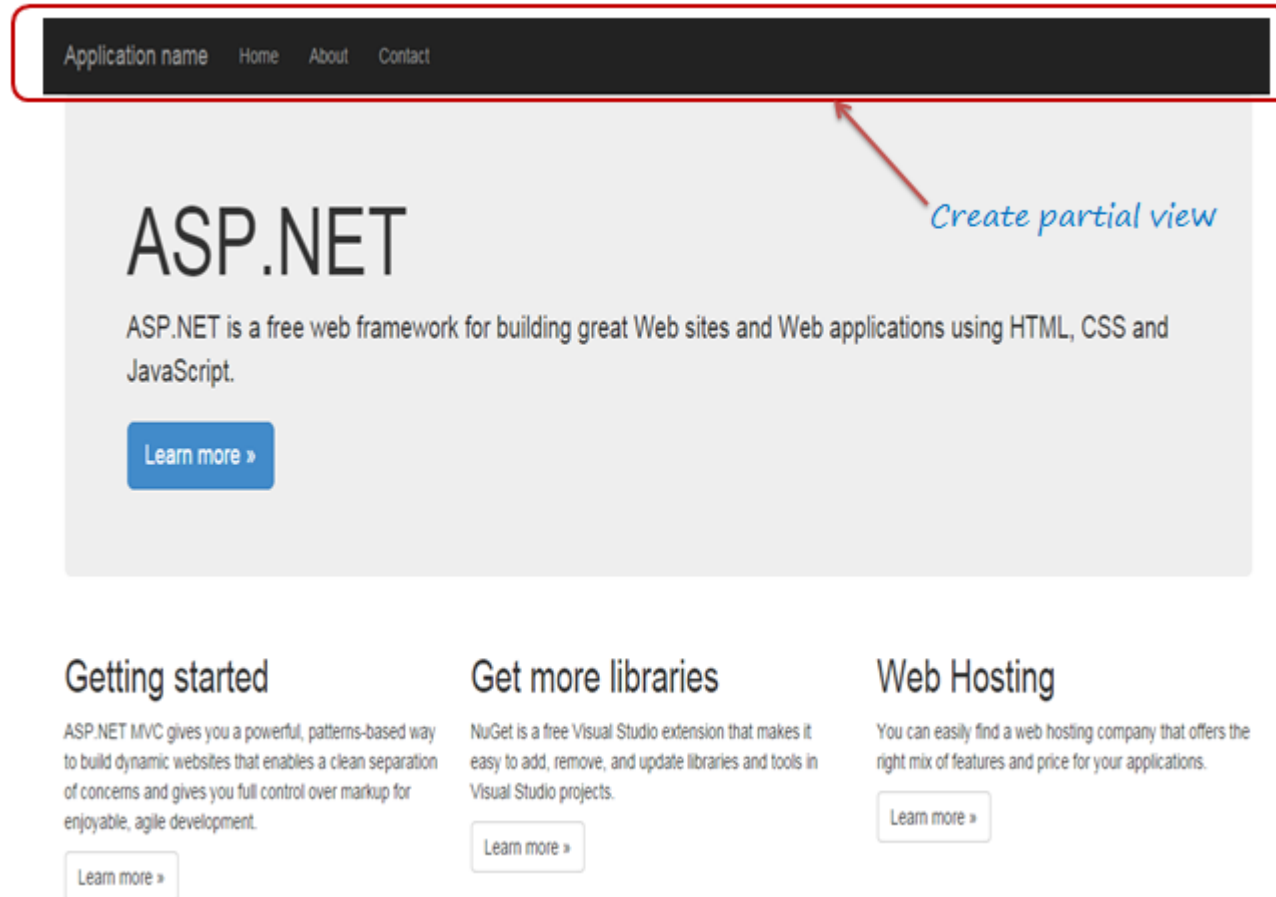
Partial View

Partial view is a reusable view, which can be used as a child view in multiple other views. It eliminates duplicate coding by reusing same partial view in multiple places. You can use the partial view in the layout view, as well as other content views.



Partial View

Partial view is a reusable view, which can be used as a child view in multiple other views. It eliminates duplicate coding by reusing same partial view in multiple places. You can use the partial view in the layout view, as well as other content views.



Partial View

For that create following items in our application:

- 1) Creating Model (CarModel)
- 2) Creating Controller (CarController)
- 3) Creating Partial view (Child)
- 4) Creating View (Parent)

Partial View

Creating Model

Now we will create model (CarModel) in our asp.net mvc application for that right click on Models Folder -> Select Add -> then select Class -> now new pop up will open in that select class and give name as **CarModel** and click Add button. Now open CarModel file and write the code like as shown below

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

namespace Tutorial5.Models
{
    public class CarModel
    {
        public int CarID { get; set; }
        public string CarType { get; set; }
        public int CarPrice { get; set; }
        public string CarColor { get; set; }

        public List<CarModel> Listcar { get; set; } // List of Car
    }
}
```

Partial View

Creating Controller

After completion of creating **CarModel** now we will create a Controller for that right click on Controllers folder -> select Add -> click on controller -> Give name as **CarController** and select template as Empty MVC Controller and click add.

```
using Tutorial5.Models;
namespace Tutorial5.Controllers
{
    public class CarController : Controller
    {
        public ActionResult Index()
        {
            CarModel objcar = new CarModel();
            objcar.CarID = 1;
            objcar.CarColor = "Brown";
            objcar.CarPrice = 20000;
            objcar.CarType = "sporty";

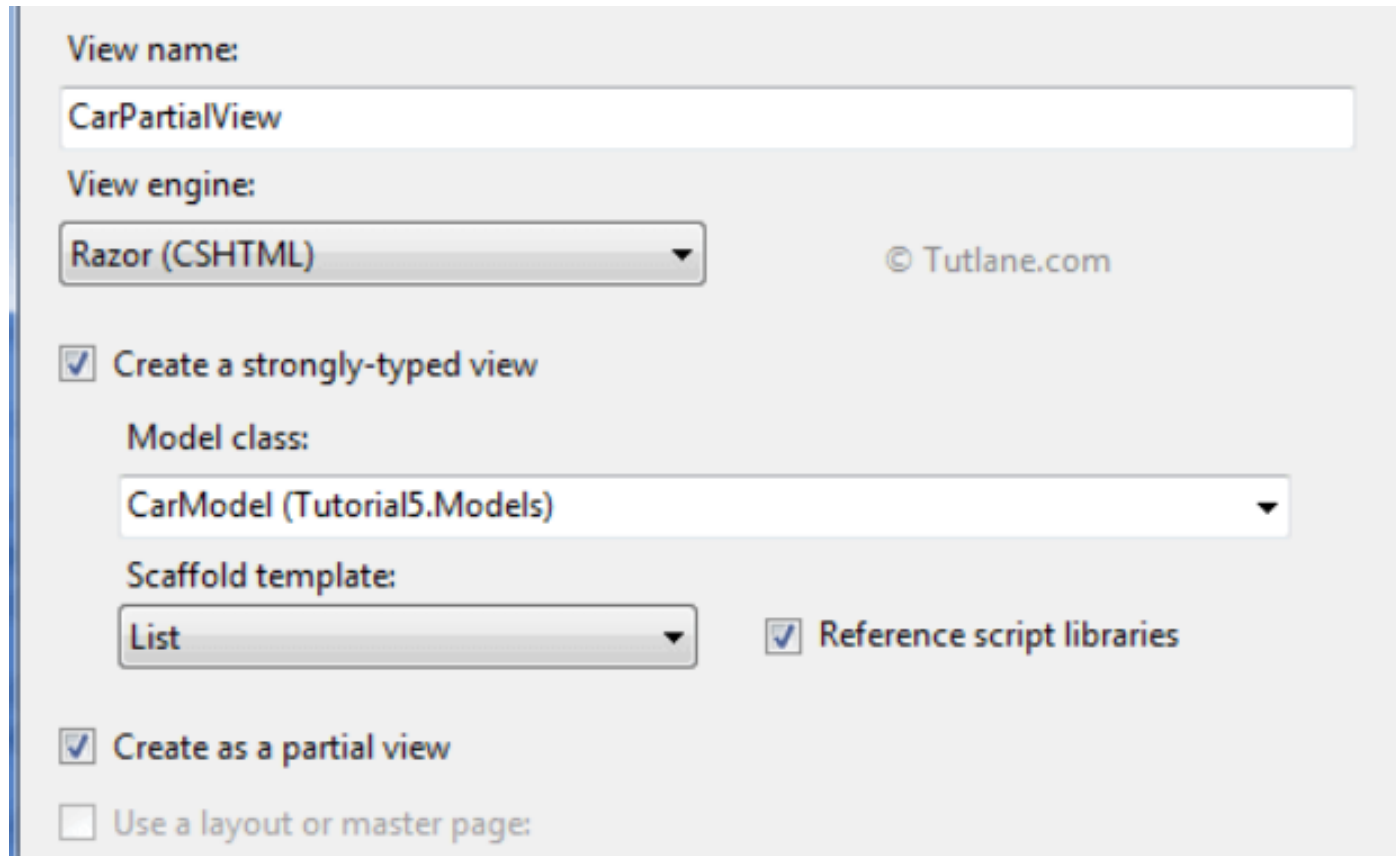
            List<CarModel> objlistcar = new List<CarModel>();
            objlistcar.Add(objcar); // add item in list
            objcar.Listcar = objlistcar; // assigning value to CarModel object
            return View(objcar); // Returning model
        }
    }
}
```

In this controller we are going to create an object of **CarModel** and assign value to it and also we will add data to **ListCar** which is list of type **Car** and then return a model. Now open our controller (**CarController**) and write the code like as shown.

Partial View

Creating Partial View

To create Partial view just right click inside the controller action method then a new dialog will pop up with Name Add View in that enter view name **CarPartialView** and select view engine as **Razor**. Here we are going to create a strongly-typed view that's why we need to select option **Create a strongly-typed view** and then in Model class give name as **CarModel**. Now select Scaffold template as **List** and select **Create as a partial view** option because we are going to create a partial view once everything done click on Add button. Our View options will be like as shown below image



View name:
CarPartialView

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
CarModel (Tutorial5.Models)

Scaffold template:
List

☒ Reference script libraries

☒ Create as a partial view

☐ Use a layout or master page:

© Tutlane.com

Partial View

After clicking on Add button the Scaffold template will generate code for us and It will generate a list with model which we assed to it that would be like as shown below

```
@model IEnumerable<Tutorial5.Models.CarModel>

<table style="background-color: rosybrown; border:dashed;">
<tr>
<td colspan="3"><h2>Child View</h2></td>
</tr>
<tr>
<th>
@Html.DisplayNameFor(model => model.CarID)
</th>
<th>
@Html.DisplayNameFor(model => model.CarType)
</th>
<th>
@Html.DisplayNameFor(model => model.CarPrice)
</th>
<th>
@Html.DisplayNameFor(model => model.CarColor)
</th>
</tr>
```

```
@foreach (var item in Model)
{
<tr>
<td>
@Html.DisplayFor(modelItem => item.CarID)
</td>
<td>
@Html.DisplayFor(modelItem => item.CarType)
</td>
<td>
@Html.DisplayFor(modelItem => item.CarPrice)
</td>
<td>
@Html.DisplayFor(modelItem => item.CarColor)
</td>
</tr>
}
</table>
```

Partial View

Creating View for Rendering Partial view

To create view just right click inside the controller action method then a new dialog will popup with Name Add View in that give view name **Index** and select View engine as **Razor**. Here we are going to create a strongly-typed view that why we need to select option **Create a strongly-typed view** and then in Model class give name as **CarModel**. Now select **Scaffold template** as Empty and Uncheck option **Use Layout or Master Page** lastely click Add button like as shown below

View name:
Index

View engine:
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:
CarModel (Tutorial5.Models)

Scaffold template:
Empty

☒ Reference script libraries

☐ Create as a partial view

☐ Use a layout or master page:

Partial View

After clicking on Add button the Scaffold template will generate code for us. It will generate an Empty view with model you had passed to it that will be like as shown below

```
@model Tutorial5.Models.CarModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width"/>
<title>Index</title>
</head>
<body>
<div>

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

Partial View

Now we have to render partial view inside Index view. For rendering view we are going to use Html Helper class that will be like as shown below

```
@Html.Partial("Partial View Name" , "Model which is required")
```

Once we add partial view inside of view that will be like as shown:

```
@model Tutorial5.Models.CarModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width"/>
<title>Index</title>
</head>
<body style="background-color:whitesmoke;">
<div>
@Html.Partial("CarPartialView", Model.Listcar)
</div>
</body>
</html>
```

Partial View

Finally just run application and check output that will be like as shown below. The outer most view is Parent view and the view with Brown color is Partial view (Child view)



Question #1

Which of the following view file types are supported in MVC?

A. ☐ .cshtml

B. ☐ .vbhtml

C. ☐ .aspx

D. ☐ All of the above

Question #2

HtmlHelper class _____.

A. ☐ Generates html elements

B. ☐ Generates html view

C. ☐ Generates html help file

D. ☐ Generates model data

Question #3

Which of the following view contains common parts of UI?

A. ☐ Partial view

B. ☐ Html View

C. ☐ Layout view

D. ☐ Razor view