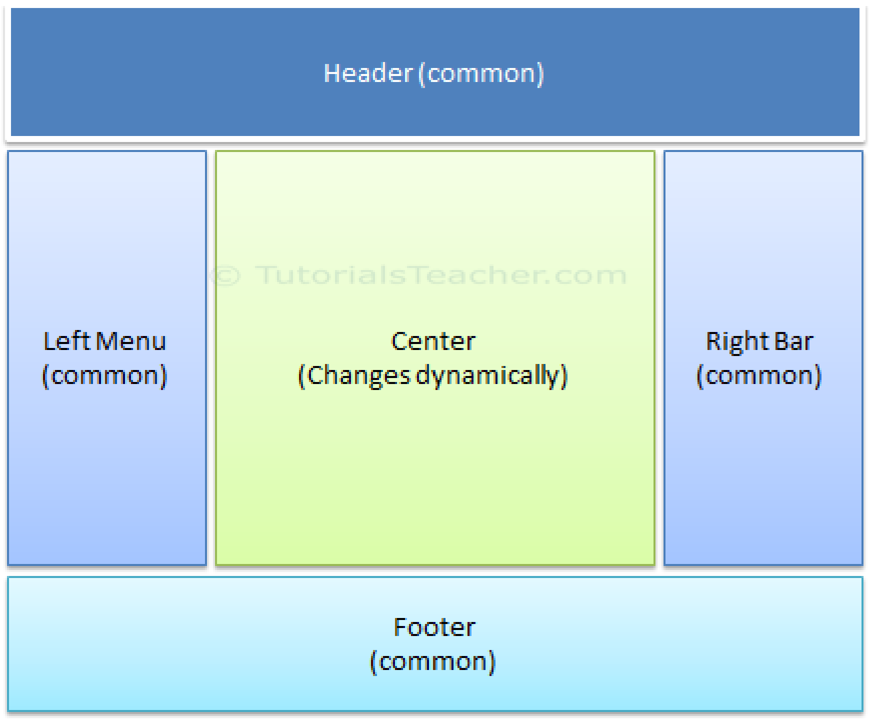
Layout View

In this section, you will learn about the layout view in ASP.NET MVC.

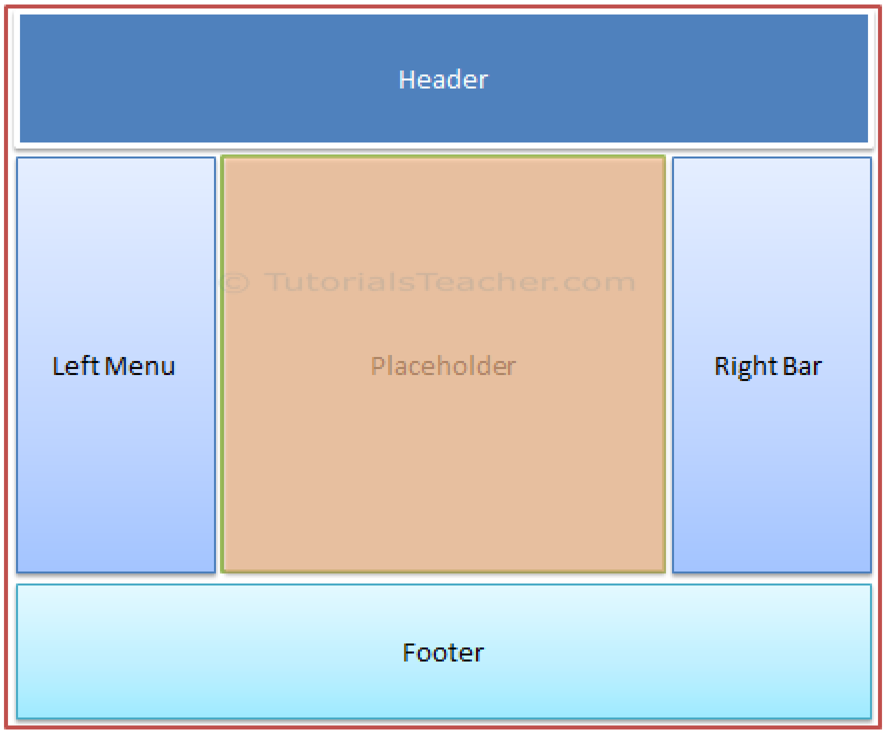
An application may contain common parts in the UI which remains the same throughout the application such as the logo, header, left navigation bar, right bar or footer section. ASP.NET MVC introduced a Layout view which contains these common UI parts, so that we don't have to write the same code in every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

For example, an application UI may contain Header, Left menu bar, right bar and footer section that remains same in every page and only the centre section changes dynamically as shown below.



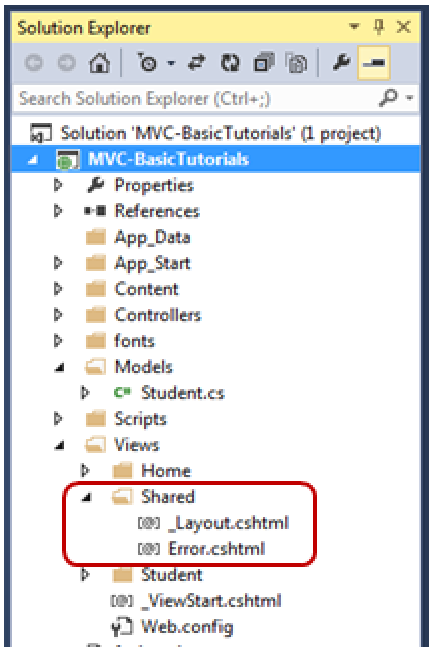
Sample Application UI Parts

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 Views in Shared Folder

The following is an auto-generated _Layout.cshtml.

_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-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new { area = ""
}, new { @class = "navbar-brand" })
            </div>
            <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>
```

As you can see, 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.

Use Layout View

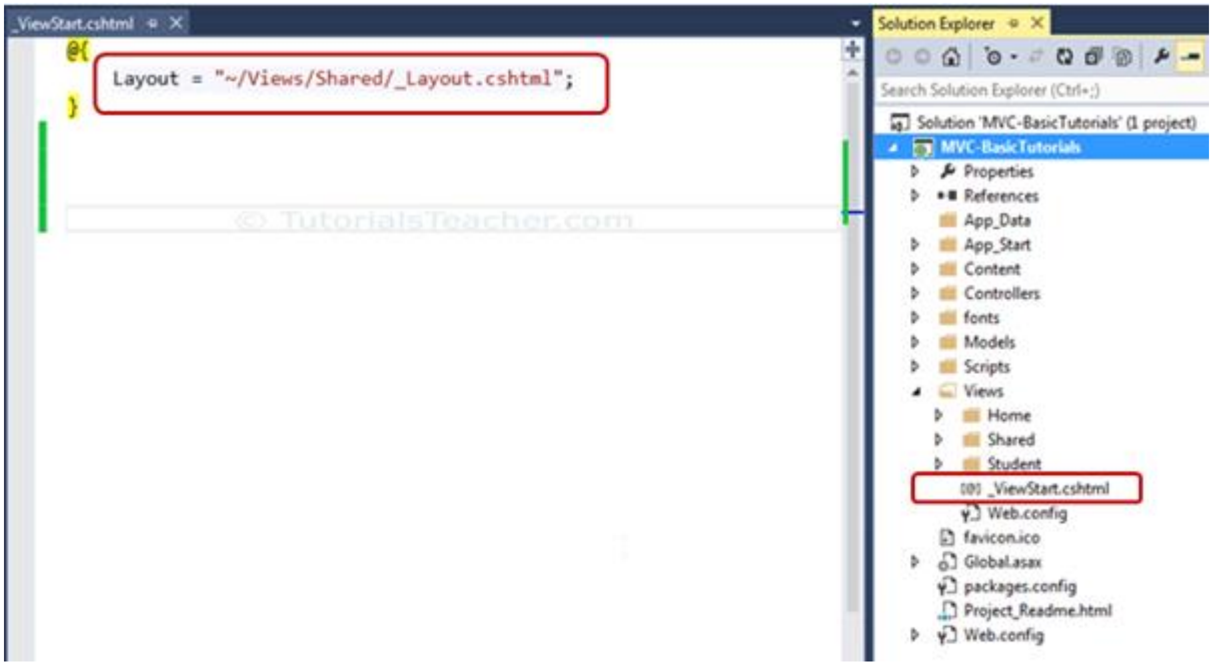
You must be wondering that how would the View know which layout view to use?

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. You can assign a valid path of any Layout page to the Layout property.

For example, 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.

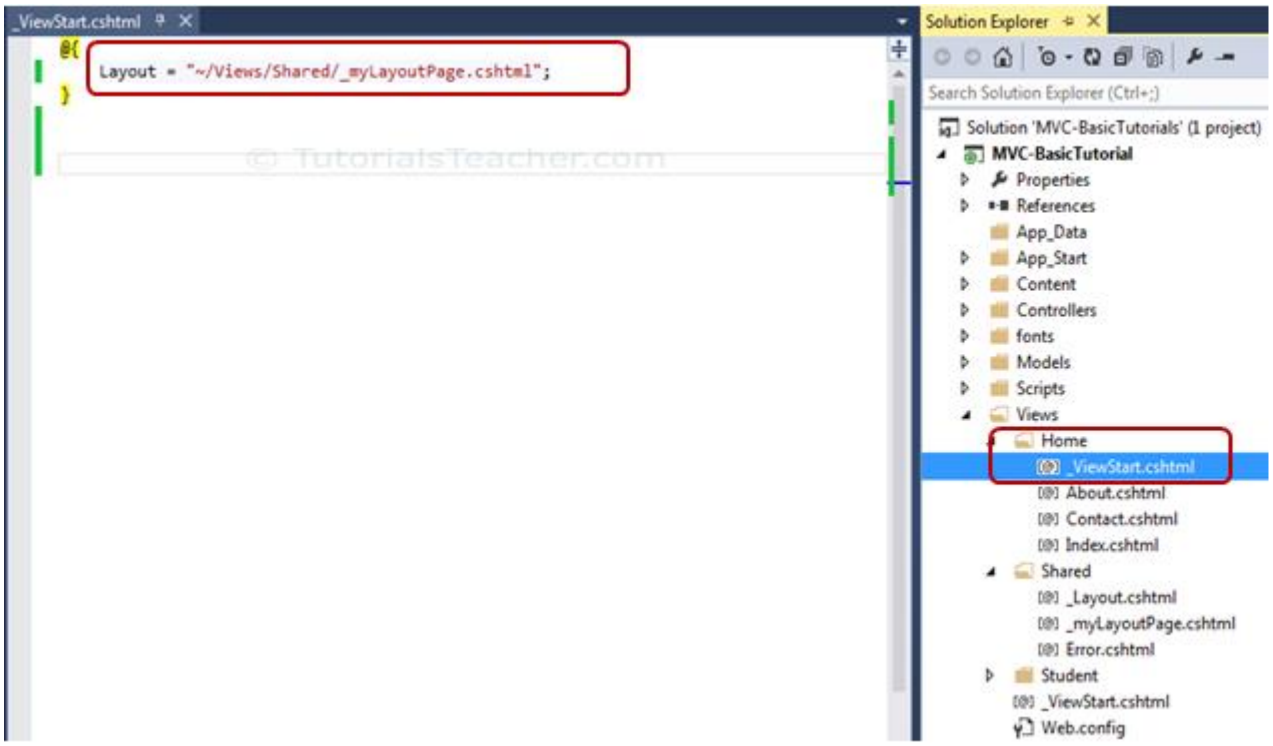


```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

_ViewStart.cshtml

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

For example, the following _ViewStart.cshtml in Home folder, sets Layout property to _myLayoutPage.cshtml. So this _ViewStart.cshtml will influence all the views included in the Home folder only. So now, Index, About and Contact views will be rendered in _myLayoutPage.cshtml instead of default _Layout.cshml.



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

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.

Index View:

```
@{
    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>
```

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.

Example: Specify Layout View in Action Method

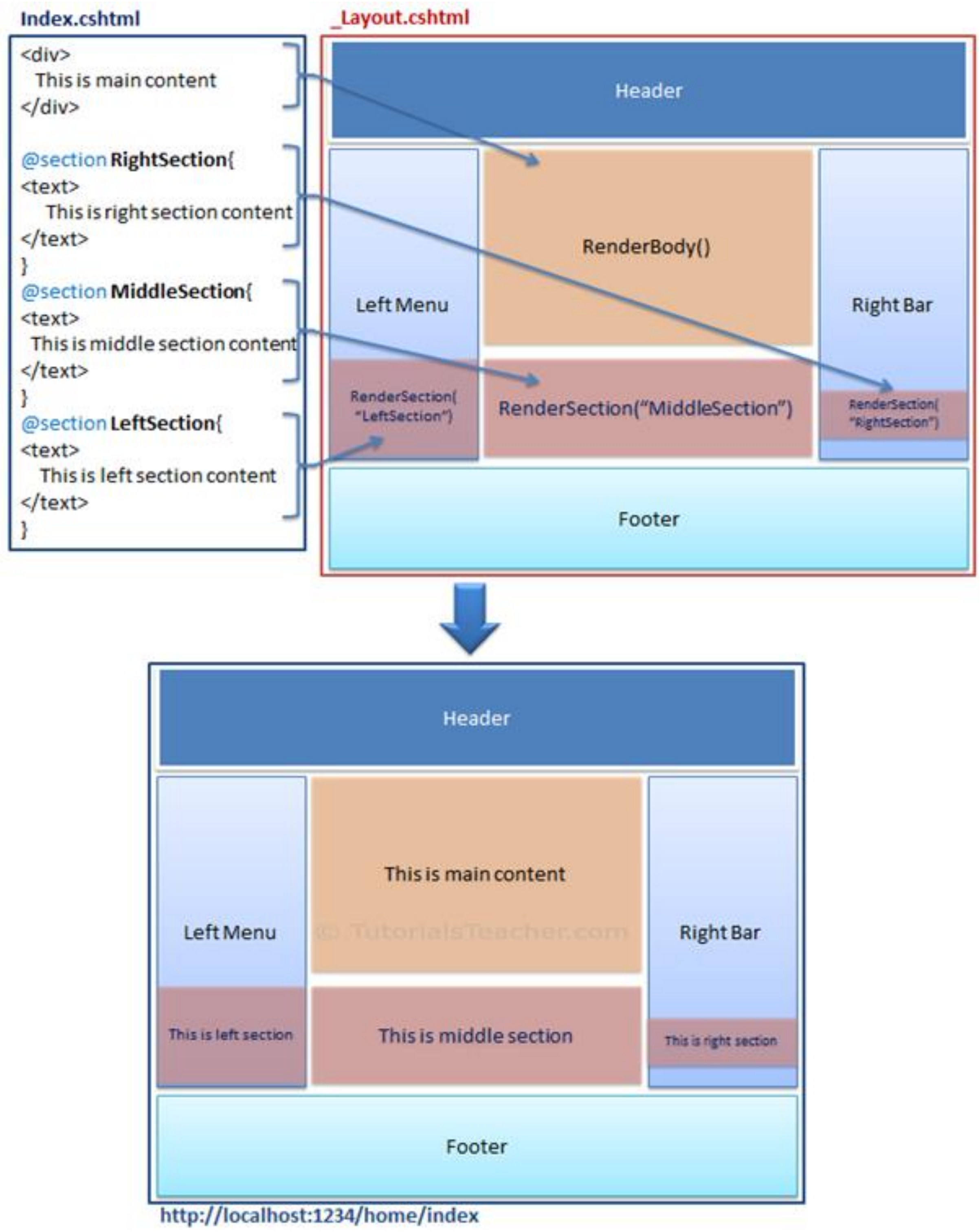
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("Index", "_myLayoutPage");
    }

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

    public ActionResult Contact()
    {
        return 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.

The following figure illustrates the use of 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>
}
```

Rendering Methods

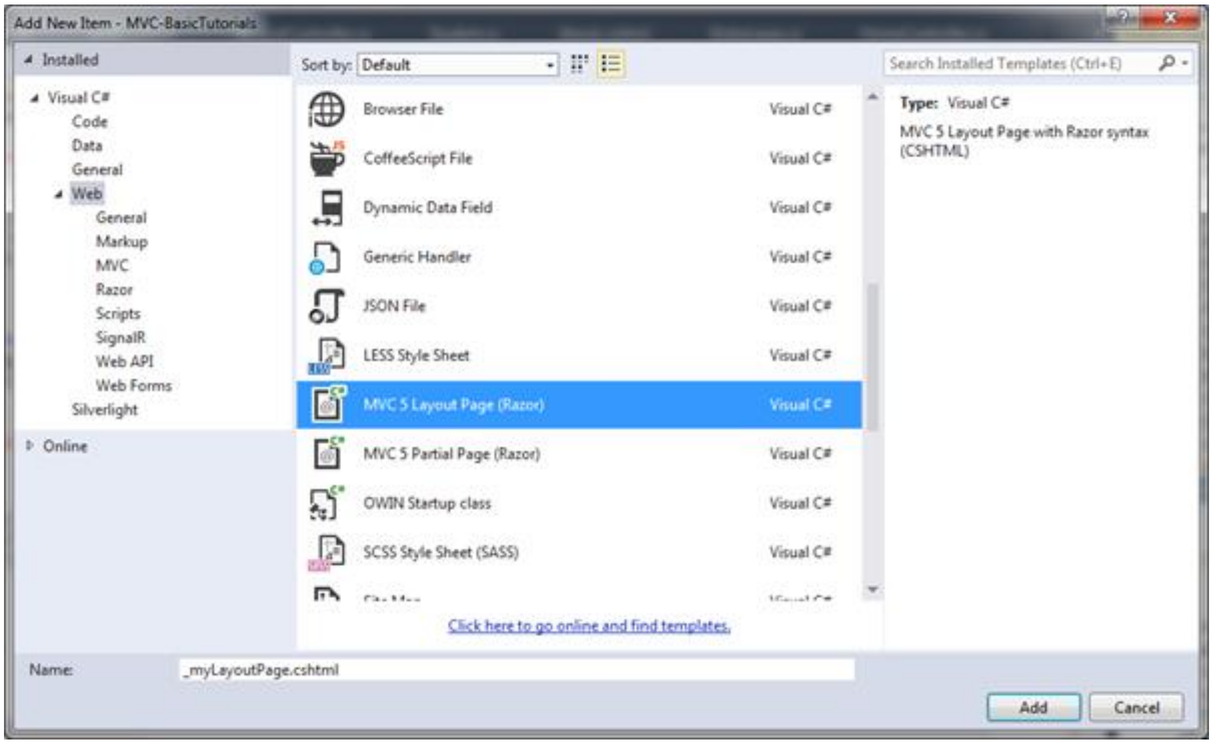
As you can see in the above figure, _Layout.cshtml includes RenderBody() method and RenderSection() method. RenderSection methods specify name of a section such as LeftSection, MiddleSection and RightSection in the above figure. Index.cshtml defines named section using @section where name of each section matches with the name specified in RenderSection method of _Layout.cshtml, such as @Section RightSection etc. At run time, the named sections of Index.cshtml such as LeftSection, RightSection and MiddleSection will be rendered at appropriate place where RenderSection method is called. Rest of the part of Index view, which is not in any of the named section will render where RenderBody() method is being called.

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

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**.


Add New Item - MVC-BasicTutorials
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Silverlight
Online
Sort by: Default
Browser File Visual C#
CoffeeScript File Visual C#
Dynamic Data Field Visual C#
Generic Handler Visual C#
JSON File Visual C#
LESS Style Sheet Visual C#
MVC 5 Layout Page (Razor) Visual C#
MVC 5 Partial Page (Razor) Visual C#
OWIN Startup class Visual C#
SCSS Style Sheet (SASS) Visual C#
Click here to go online and find templates.
Search Installed Templates (Ctrl+E)
Type: Visual C#
MVC 5 Layout Page with Razor syntax (CSHTML)
Name: _myLayoutPage.cshtml
Add
Cancel
Rendering Methods

You will see _myLayoutPage.cshtml as shown below.

_myLayoutPage.cshtml:

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

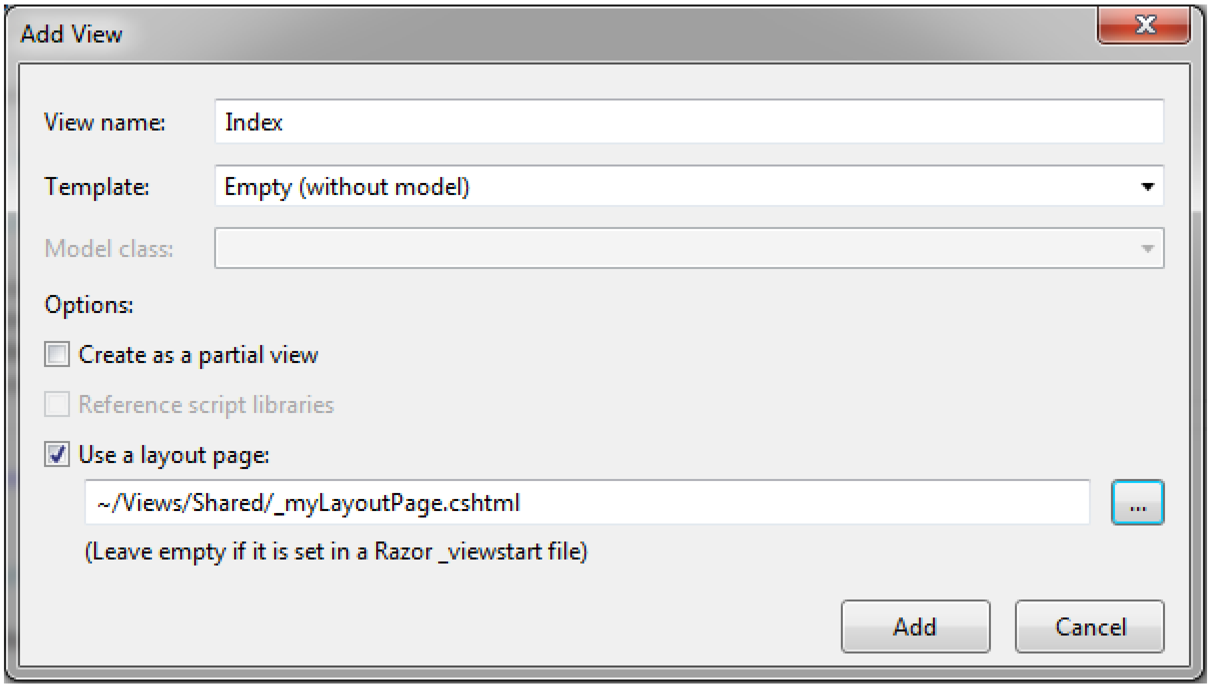
Now, add the <footer> tag with the `RenderSection("footer",true)` method alongwith some styling as shown below. Please notice that we made this section as required. This means any view that uses _myLayoutPage as its layout view must include a footer section.

Example: Adding RenderSection

```
<!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>
```

Now, let's use this _myLayoutPage.cshtml with the Index view of HomeController.

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.



View name: Index

Template: Empty (without model)

Model class:

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

~/Views/Shared/_myLayoutPage.cshtml

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

Add Index View

This will create Index.cshtml as shown below.

Index view:

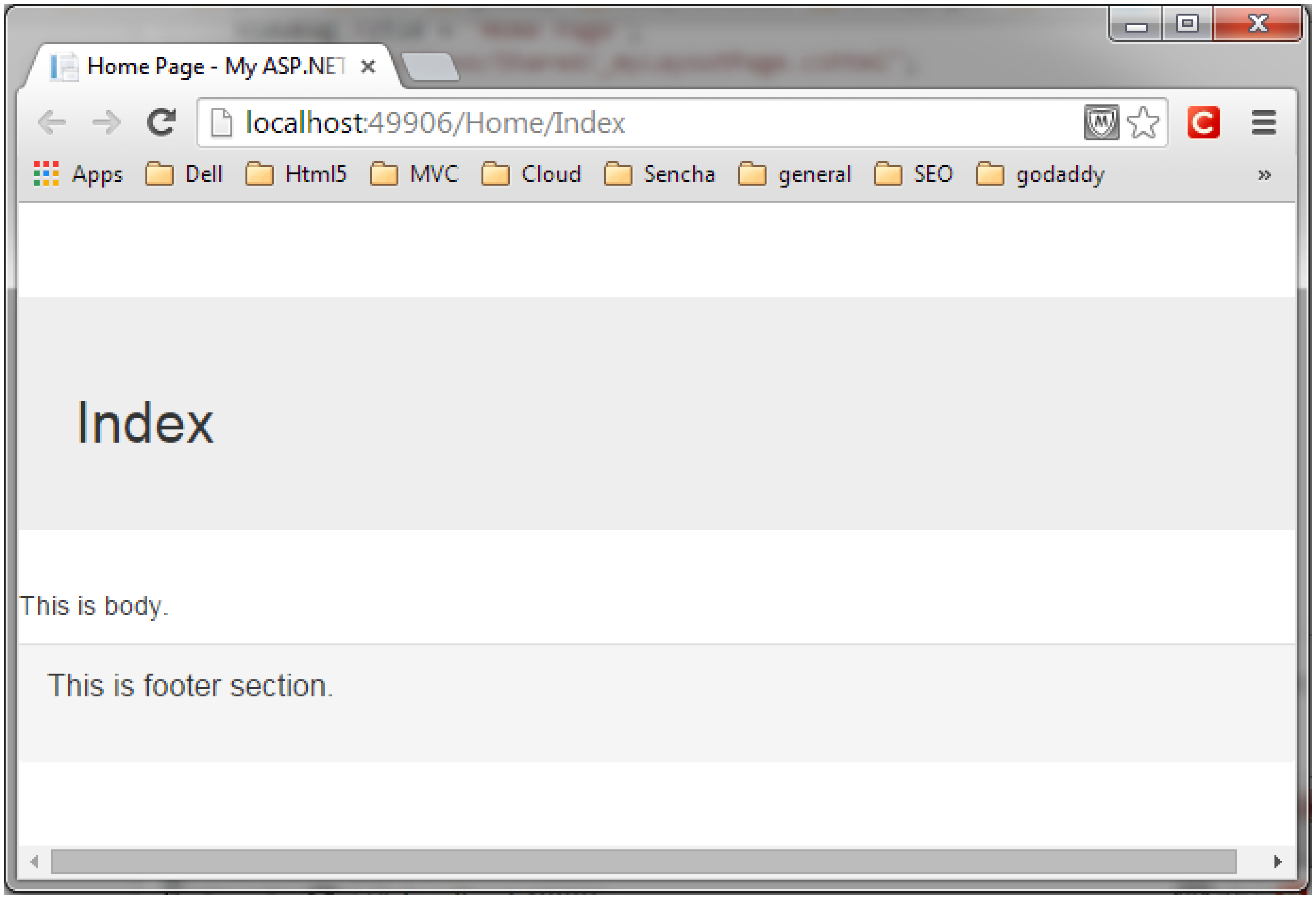
```
@{
    ViewBag.Title = "Home Page";
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";
}

<h2>Index</h2>
```

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.

Index view:

```
@{
    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>
```

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



Index

This is body.

This is footer section.

Index View

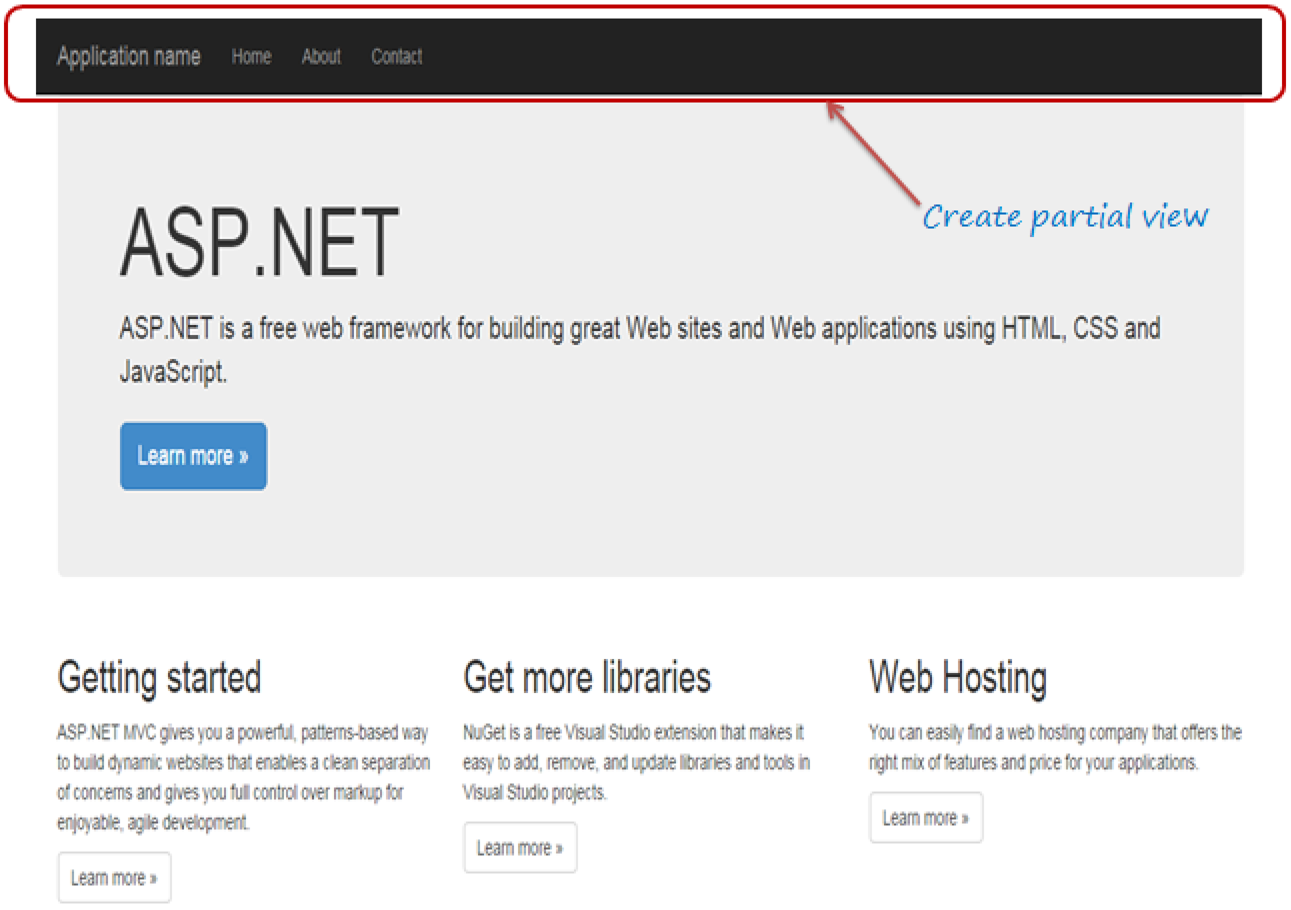
Thus, you can create new layout view with different rendering methods.

Partial View

In this section you will learn about partial views in ASP.NET MVC.

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.

To start with, let's create a simple partial view for the following navigation bar for demo purposes. We will create a partial view for it, so that we can use the same navigation bar in multiple layout views without rewriting the same code everywhere.



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Learn more »

Getting started

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.

Learn more »

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

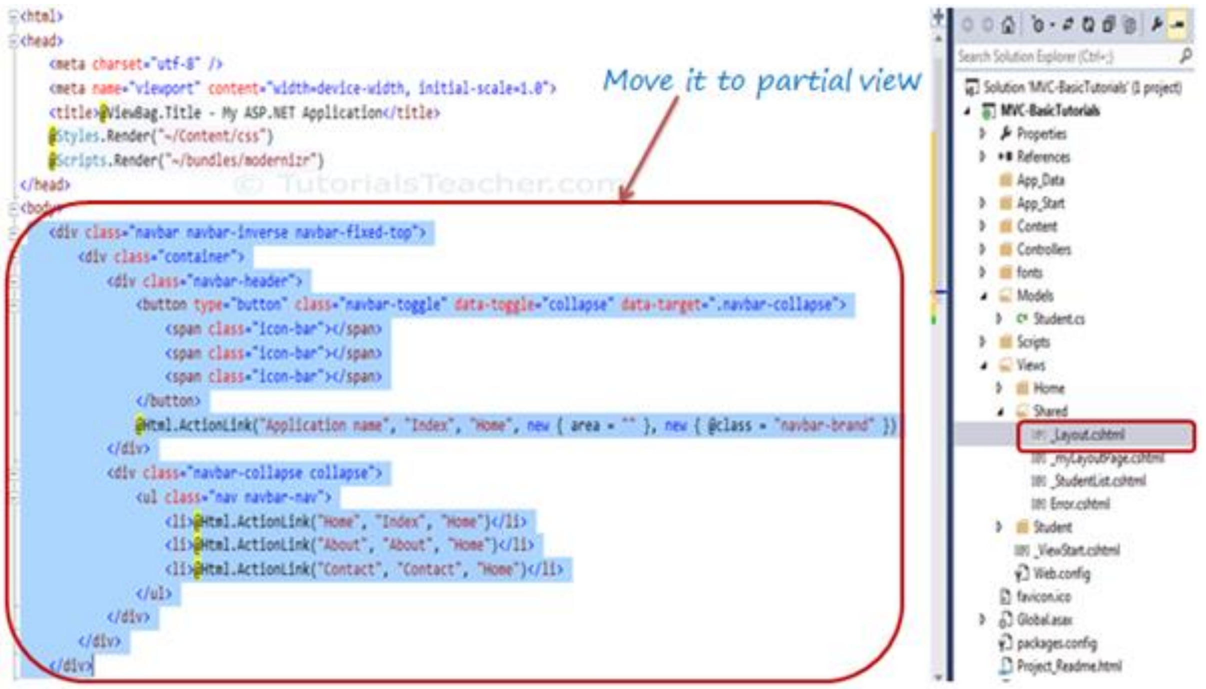
Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

Learn more »

Partial View

The following figure shows the html code for the above navigation bar. We will cut and paste this code in a seperate partial view for demo purposes.



```
<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-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <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>
```

Partial Views

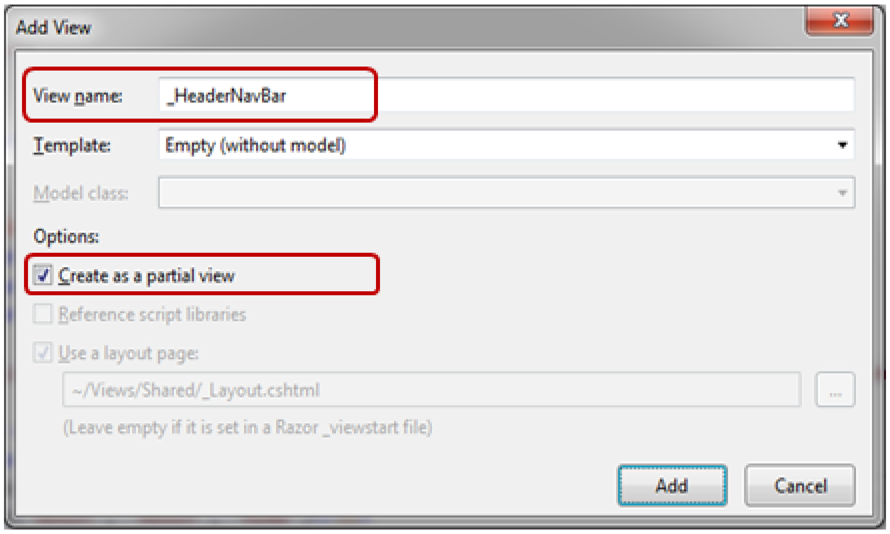
Create a New Partial View

To create a partial view, right click on Shared folder -> select **Add** -> click on **View..**

Note:

If a partial view will be shared with multiple views of different controller folder then create it in the Shared folder, otherwise you can create the partial view in the same folder where it is going to be used.

In the Add View dialogue, enter View name and select "Create as a partial view" checkbox and click Add.


Add View

View name: _HeaderNavBar

Template: Empty (without model)

Model class:

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)

We are not going to use any model for this partial view, so keep the Template dropdown as Empty (without model) and click **Add**. This will create an empty partial view in Shared folder.

Now, you can cut the above code for navigation bar and paste it in _HeaderNavBar.cshtml as shown below:

Example: Partial View _HeaderNavBar.cshtml

```
<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-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", new { area = "" },
new { @class = "navbar-brand" })
        </div>
        <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>
```

Thus, you can create a new partial view. Let's see how to render partial view.

Render Partial View

You can render the partial view in the parent view using html helper methods: Partial() or RenderPartial() or RenderAction(). Each method serves different purposes. Let's have an overview of each method and then see how to render partial view using these methods.

Html.Partial()

@Html.Partial() helper method renders the specified partial view. It accept partial view name as a string parameter and returns MvcHtmlString. It returns html string so you have a chance of modifing the html before rendering.

The following table lists overloads of the Partial helper method:

Helper Method	Description
MvcHtmlString Html.Partial(string partialViewName)	Renders the given partial view content in the referred view.
MvcHtmlString Html.Partial(string partialViewName,object model)	Renders the partial view content in the referred view. Model parameter passes the model object to the partial view.
MvcHtmlString Html.Partial(string partialViewName, ViewDataDictionary viewData)	Renders the partial view content in the referred view. View data parameter passes view data dictionary to the partial view.
MvcHtmlString Html.Partial(string partialViewName,object model, ViewDataDictionary viewData)	Renders the partial view content in the referred view. Model parameter passes the model object and View data passes view data dictionary to the partial view.

Html.RenderPartial()

The RenderPartial helper method is same as the Partial method except that it returns void and writes resulted html of a specified partial view into a http response stream directly.

Helper method	Description
RenderPartial(String partialViewName)	Renders the specified partial view
RenderPartial(String partialViewName, Object model)	Renders the specified partial view and set the specified model object
RenderPartial(String partialViewName, ViewDataDictionary viewData)	Renders the specified partial view, replacing its ViewData property with the specified ViewDataDictionary object.
RenderPartial(String partialViewName, Object model, ViewDataDictionary viewData)	Renders the specified partial view, replacing the partial view's ViewData property with the specified ViewDataDictionary object and set the specified model object

Html.RenderAction()

The RenderAction helper method invokes a specified controller and action and renders the result as a partial view. The specified Action method should return PartialViewResult using the Partial() method.

Name	Description
RenderAction(String actionName)	Invokes the specified child action method and renders the result in the parent view.
RenderAction(String actionName, Object routeValue)	Invokes the specified child action method using the specified parameters and renders the result inline in the parent view.
RenderAction(String actionName, String controllerName)	Invokes the specified child action method using the specified controller name and renders the result inline in the parent view.
RenderAction(String actionName, RouteValueDictionary routeValues)	Invokes the specified child action method using the specified parameters and renders the result inline in the parent view.
RenderAction(String actionName, String controllerName, Object routeValue)	Invokes the specified child action method using the specified parameters and controller name and renders the result inline in the parent view.
RenderAction(String actionName, String controllerName, RouteValueDictionary routeValues)	Invokes the specified child action method using the specified parameters and controller name and renders the result inline in the parent view.

So now, we can use any of the above rending methods to render the _HeaderNavBar partial view into _Layout.cshtml. The following layout view renders partial view using the RenderPartial() method.

Example: Html.RenderPartial()

```
<!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>
    @{
      Html.RenderPartial("_HeaderNavBar");
    }
    <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>
```

Note:

RenderPartial returns void, so a semicolon is required at the end and so it must be enclosed in the braces.

The following layout view uses the Partial method to render partial view_HeaderNavBar.cshtml.

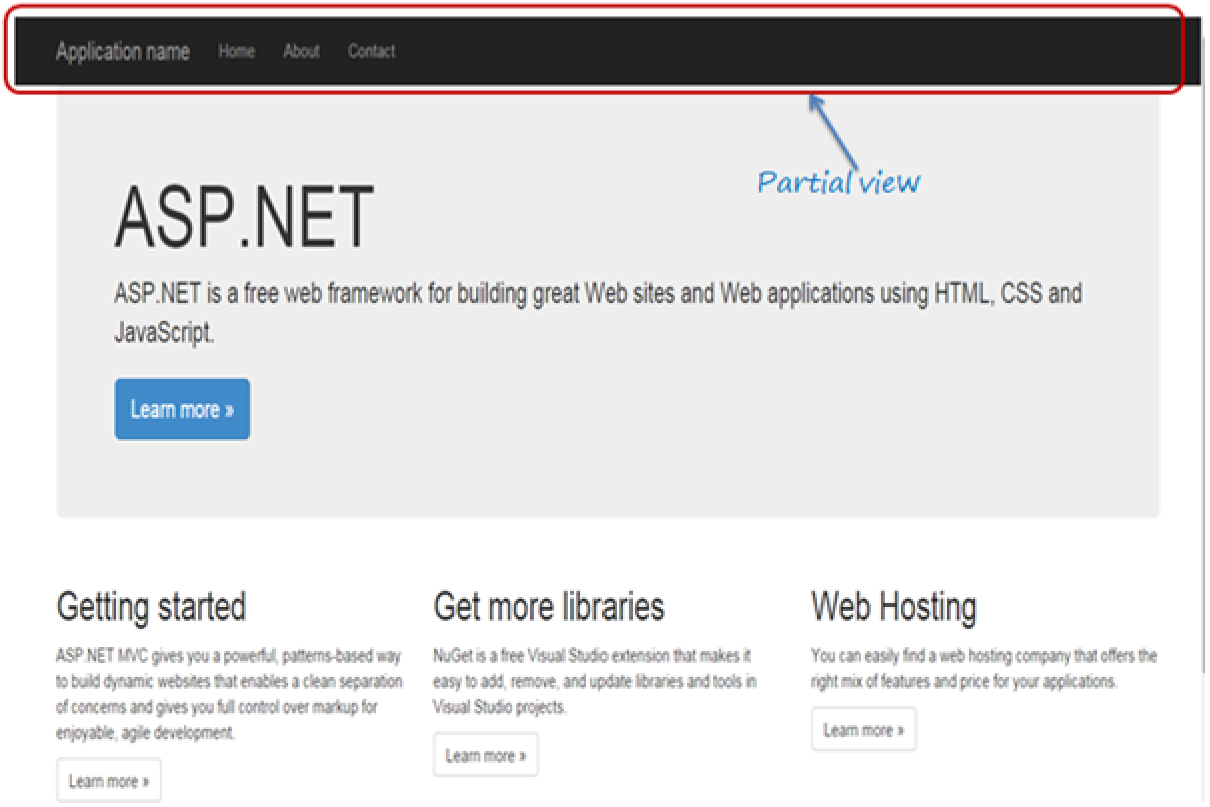
Example: Html.Partial()

```
<!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>
    @Html.Partial("_HeaderNavBar")
    <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>
```

Note:

@Html.Partial() method doesn't need to be in code block because it returns a html string.

You will see following UI in browser when you run the application.



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

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.

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

Index.cshtml

So in this way, you can use partial view without any differences in the UI.