



ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
VNUHCM - UIT

LẬP TRÌNH WEB

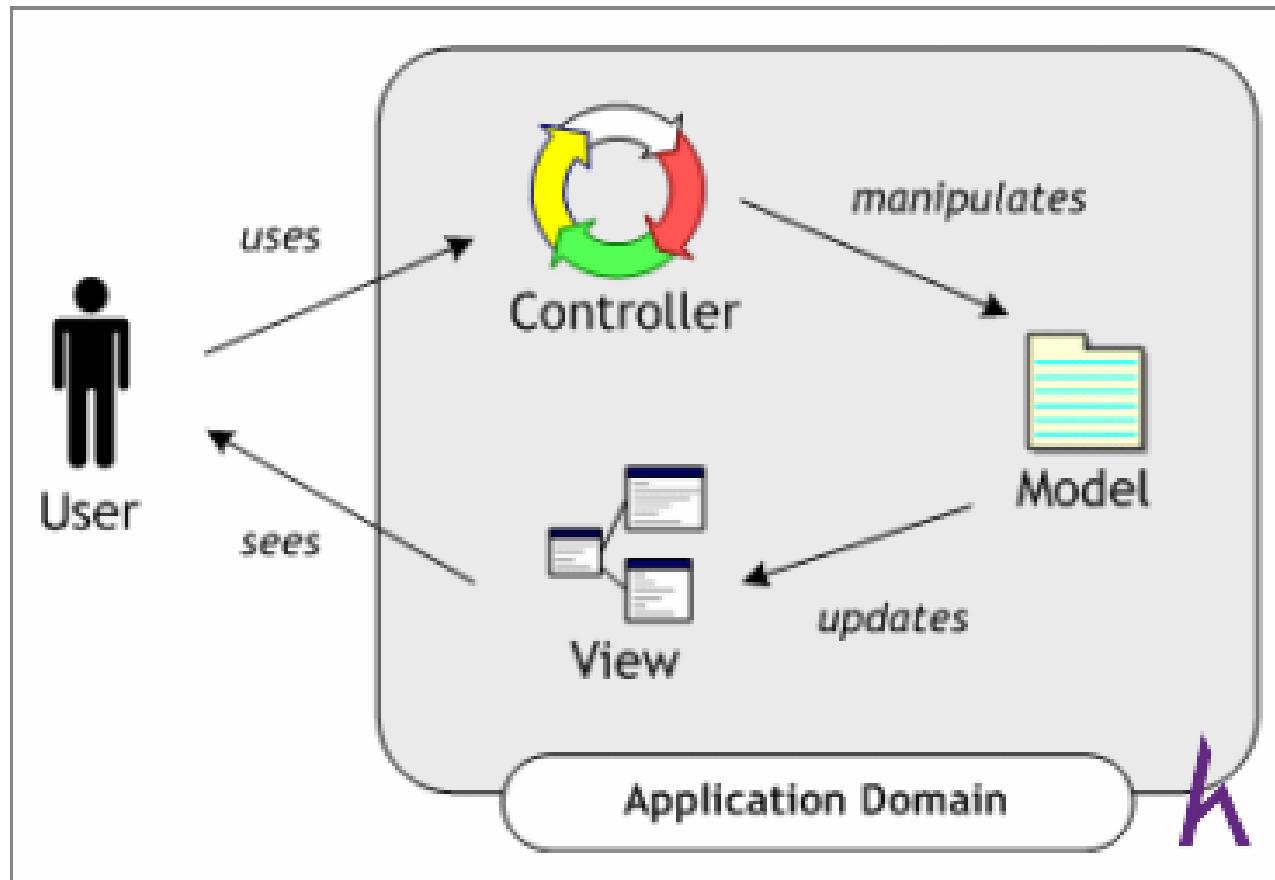
CHƯƠNG 3 LÀM VIỆC VỚI VIEW

NỘI DUNG

- 1 • Kiến trúc MVC
- 2 • Khái niệm về View
- 3 • Tạo View cho một Action
- 4 • Truyền dữ liệu từ Controller về View
- 5 • Sử dụng partial view
- 6 • Thiết lập layout
- 7 • Cách tạo Layout View trong MVC

1. KIẾN TRÚC MVC

❖ MVC (Model – View - Controller)



2. KHÁI NIỆM VỀ VIEW

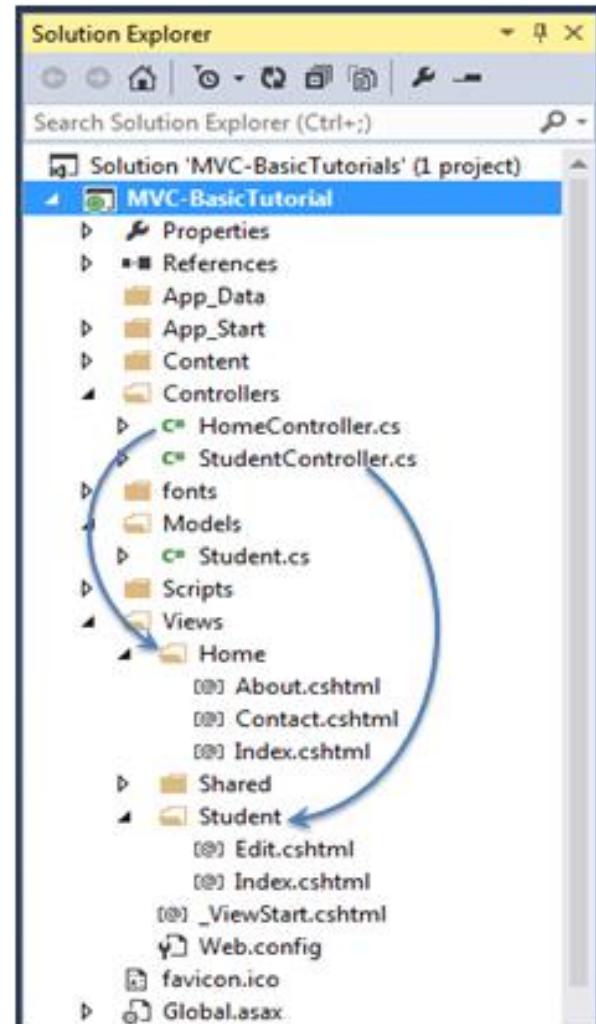
- ❖ **View** là một giao diện người dùng. **View** chính là các thành phần chịu trách nhiệm hiển thị các thông tin lên cho người dùng thông qua giao diện. Thông thường, các thông tin cần hiển thị được lấy từ thành phần **Models**.
- ❖ Ví dụ, đối tượng Product có một “Edit” view bao gồm các textboxes, các dropdowns và checkboxes để chỉnh sửa các thuộc tính của sản phẩm; có một “Display” view gồm 2 dòng, cột dòng là ProductID, dòng sau là OrderDate... để xem thông tin về sản phẩm.

2. KHÁI NIỆM VỀ VIEW

- ❖ View ASP.NET MVC được lưu trong thư mục Views. Các phương thức hành động khác nhau của một lớp controller duy nhất có thể hiển thị các Views khác nhau, do đó, thư mục Views chứa một thư mục riêng cho mỗi controller có cùng tên controller, để phù hợp với nhiều Views
- ❖ Ví dụ: các Views sẽ được hiển thị từ bất kỳ phương thức hành động nào của HomeController, nằm trong thư mục Views -> Home. Tương tự các Views sẽ được hiển thị từ StudentController, sẽ nằm trong thư mục Views -> Student như được hiển thị bên dưới.

2. KHÁI NIỆM VỀ VIEW

❖ Chú ý: Trong hình 1, các bạn thấy có 2 thư mục Home và Student trong thư mục Views, tương ứng với 2 controller **HomeController** và **StudentController**. Trong mỗi thư mục, số lượng các View khác nhau, tùy thuộc nghiệp vụ, logic của controller đó.



2. KHÁI NIỆM VỀ VIEW

- ❖ Microsoft đã giới thiệu công cụ **Razor** được đóng gói với MVC 3. Bạn có thể viết hỗn hợp các thẻ html và mã phía máy chủ trong **razor**. **Razor** sử dụng ký tự @ cho mã phía máy chủ thay vì <% %>. Bạn có thể sử ngôn ngữ C # hoặc Visual Basic để viết mã phía máy chủ bên trong **Razor**.
- ❖ **Razor** tối đa hóa tốc độ viết mã bằng cách giảm thiểu số lượng ký tự và tổ hợp phím cần thiết khi viết trong View. Các tập tin Razor có phần mở rộng **.cshtml** hoặc **.vbhtml**.

2. KHÁI NIỆM VỀ VIEW

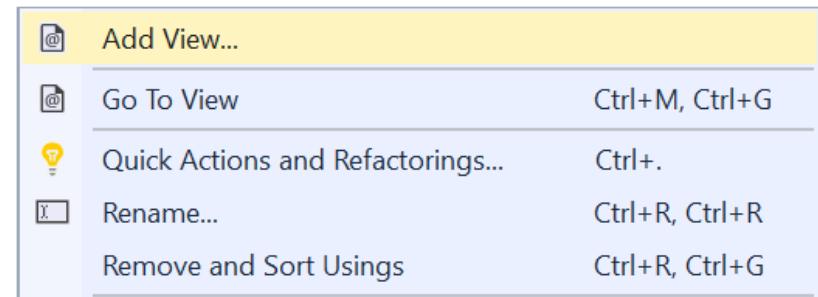
- ❖ ASP.NET MVC hỗ trợ các loại tập tin:

Phần mở rộng	Giải thích
.cshtml	C# Razor, Hỗ trợ C# có thẻ html
.vbhtml	Visual Basic Razor, Hỗ trợ Visual Basic có thẻ html
.aspx	ASP.Net web form
.ascx	ASP.NET web control

3. TẠO VIEW CHO MỘT ACTION

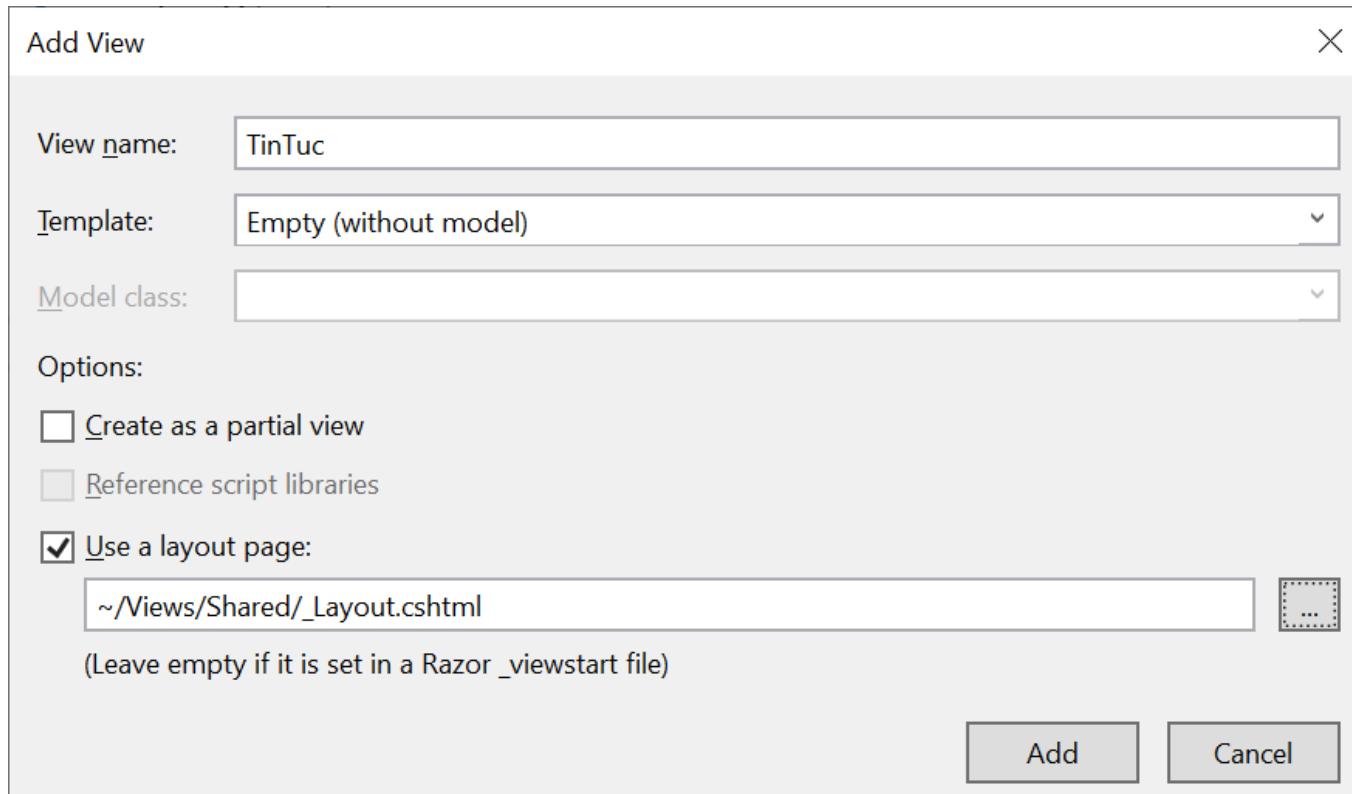
- ❖ Tạo mới View, Click chuột phải vào hành động cần tạo View. Ví dụ **ActionResult TinTuc()**, chọn **Add View...**

```
public class HomeController : Controller
{
    public ActionResult Index()...
    public ActionResult About()...
    public ActionResult Contact()...
    public ActionResult TinTuc()
    {
        return View();
    }
}
```



3. TẠO VIEW CHO MỘT ACTION

- ❖ Trong cửa sổ Add View, các bạn nhập các thông số, sau đó nhấn nút Add để tiến hành tạo View.



3. TẠO VIEW CHO MỘT ACTION

❖ Các tùy chọn trong Add View:

- **ViewName**: tên view, mặc định sẽ là tên action trong Controller, có thể thay đổi tên khác.
- **Template**: lựa chọn các mẫu View có sẵn.
- **Create as a partial view**: tạo view có kiểu là partial view (sẽ giới thiệu trong phần PartialView)
- **Use a layout page**: chọn Layout Page mặc định cho View (sẽ giới thiệu trong phần Layout Page).

3. TẠO VIEW CHO MỘT ACTION

- ❖ View được tạo mới như sau (Code của trang TinTuc.cshtml):

The screenshot shows a code editor window with the tab bar at the top. The active tab is "TinTuc.cshtml". To its right is another tab labeled "HomeController.cs". The main area of the editor contains the following C# code:

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

The code consists of eight numbered lines. Lines 1, 2, 3, 4, 6, and 8 are plain text. Line 3 contains the assignment statement `ViewBag.Title = "TinTuc";` with "TinTuc" highlighted in red. Line 6 contains the opening tag of a heading element, `

##

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewData trong MVC ASP.NET:

- ViewData là một thuộc tính của Controller base class, nó trả về một đối tượng ViewDataDictionary. ViewDataDictionary là một đối tượng dictionary cho phép lưu dữ liệu dạng key-value. Key phải là một chuỗi không phân biệt chữ hoa thường.
- Để truyền dữ liệu vào view chúng ta cần gán giá trị vào dictionary sử dụng key. Chúng ta có thể lưu bất kỳ số lượng key-value nào cần thiết trong ViewData.

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

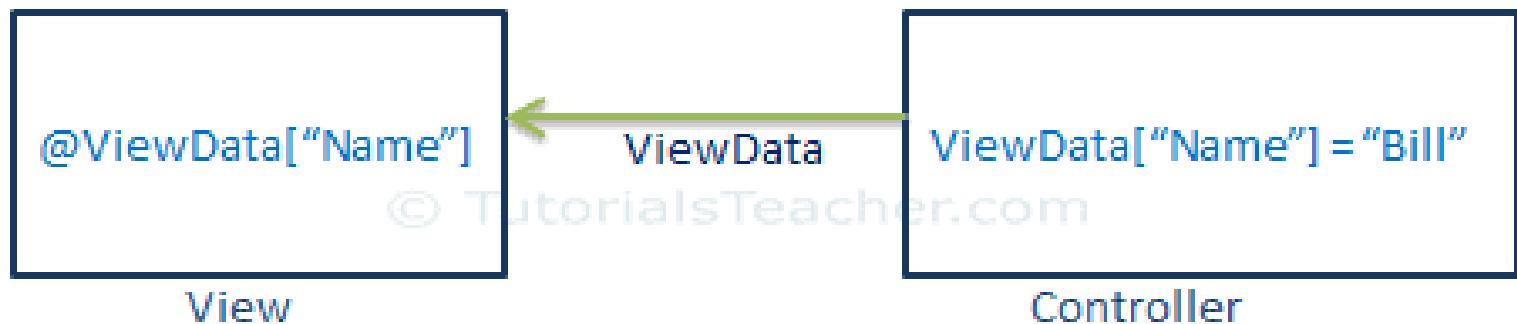
❖ ViewData trong MVC ASP.NET:

- ViewData truyền dữ liệu sang View từ Controller.
- Khi chúng ta gọi phương thức View trong Controller action, ViewData sẽ tự động gán vào View. Trong View chúng ta có thể truy cập giá trị được lưu trong ViewData cũng sử dụng key. Dữ liệu được lưu trong ViewData tồn tại chỉ trong request đó. Khi View được tạo xong cho client thì đối tượng ViewData đó cũng bị hủy.

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewData trong MVC ASP.NET:

- Hình dưới đây minh họa cho ViewData.



4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewData trong MVC ASP.NET:

- Ví dụ sau minh họa cách truyền dữ liệu từ controller sang view bằng ViewData.

```
public ActionResult Index()
{
    IList<Student> studentList = new List<Student>();
    studentList.Add(new Student(){ StudentName = "Bill" });
    studentList.Add(new Student(){ StudentName = "Steve" });
    studentList.Add(new Student(){ StudentName = "Ram" });

    ViewData["students"] = studentList;

    return View();
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewData trong MVC ASP.NET:

- Bây giờ danh sách sinh viên có thể được truy cập trong view như sau:

```
<ul>
@foreach (var std in ViewData["students"] as IList<Student>)
{
    <li>
        @std.StudentName
    </li>
}
</ul>
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewData trong MVC ASP.NET:

- Chúng ta có thể sử dụng KeyValuePair thêm dữ liệu cho ViewData như bên dưới.

Ví dụ sử dụng KeyValuePair trong ViewData

```
public ActionResult Index()
{
    ViewData.Add("Id", 1);
    ViewData.Add(new KeyValuePair<string, object>("Name", "Bill"));
    ViewData.Add(new KeyValuePair<string, object>("Age", 20));

    return View();
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewBag trong MVC ASP.NET:

- Là một `Dynamic ViewData` object, nó là một lớp bao bọc (wrap) `ViewData` để cho phép truy cập vào object một cách linh hoạt.
- `ViewBag` cũng cho phép chúng ta sử dụng dynamic properties (dùng dấu chấm thay vì ngoặc vuông như `ViewData`). Sử dụng `ViewBag` cũng tương tự như `ViewData` nhưng sẽ tiện lợi hơn vì nó **không cần phải ép kiểu**.

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewBag trong MVC ASP.NET:

```
namespace MVC_BasicTutorials.Controllers
{
    public class StudentController : Controller
    {
        IList<Student> studentList = new List<Student>() {
            new Student(){ StudentID=1, StudentName="Steve", Age = 21 },
            new Student(){ StudentID=2, StudentName="Bill", Age = 25 },
            new Student(){ StudentID=3, StudentName="Ram", Age = 20 },
            new Student(){ StudentID=4, StudentName="Ron", Age = 31 },
            new Student(){ StudentID=5, StudentName="Rob", Age = 19 }
        };
        // GET: Student
        public ActionResult Index()
        {
            ViewBag.TotalStudents = studentList.Count();

            return View();
        }
    }
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewBag trong MVC ASP.NET:

- Nay, trong view Index.cshtml, chúng ta có thể truy cập thuộc tính ViewBag.TotalStudents và hiển thị tổng số sinh viên như sau:

```
<label>Total Students:</label> @ViewBag.TotalStudents
```

- ViewBag không yêu cầu ép kiểu trong khi lấy các giá trị.

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ ViewBag và ViewData trong MVC ASP.NET:

```
public ActionResult Index()
{
    ViewBag.Id = 1;

    ViewData.Add("Id", 1); // throw runtime exception as it already has "Id" key
    ViewData.Add(new KeyValuePair<string, object>("Name", "Bill"));
    ViewData.Add(new KeyValuePair<string, object>("Age", 20));

    return View();
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

- TempData trong ASP.NET MVC có thể được sử dụng để lưu trữ dữ liệu tạm thời có thể được sử dụng trong yêu cầu tiếp theo. TempData sẽ bị xóa sau khi hoàn thành một yêu cầu tiếp theo.
- TempData rất hữu ích khi chúng ta muốn chuyển dữ liệu từ một phương thức hành động này sang một phương thức hành động khác của cùng một controller. Đây là loại từ điển có nguồn gốc từ TempDataDipedia.

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

Ví dụ: TempData

```
public class HomeController : Controller
{
    // GET: Student
    public HomeController()
    {
    }

    public ActionResult Index()
    {
        TempData["name"] = "Test data";
        TempData["age"] = 30;

        return View();
    }
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

```
public ActionResult About()
{
    string userName;
    int userAge;

    if(TempData.ContainsKey("name"))
        userName = TempData["name"].ToString();

    if(TempData.ContainsKey("age"))
        userAge = int.Parse(TempData["age"].ToString());

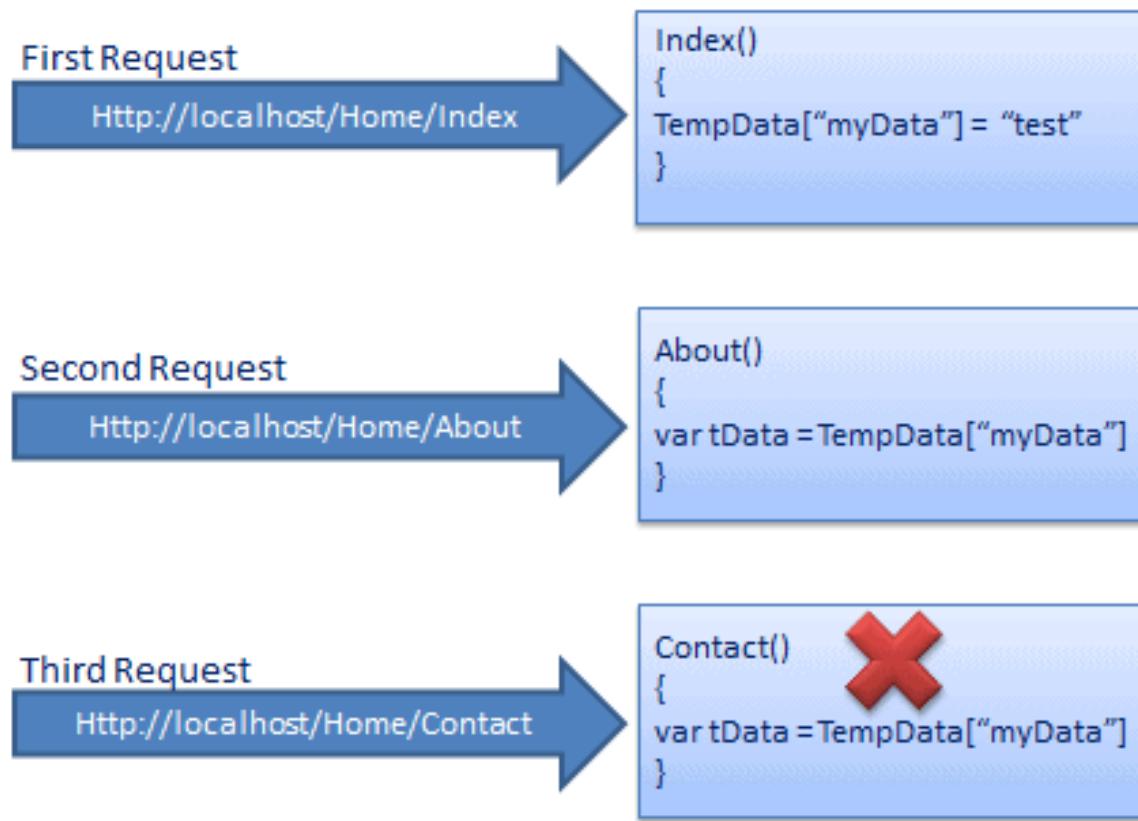
    // do something with userName or userAge here

    return View();
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

- Hình dưới đây minh họa TempData.



4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

- Gọi `TempData.Keep()` để giữ lại các giá trị TempData trong yêu cầu thứ ba tiếp theo.

Ví dụ `TempData.Keep()`

```
public class HomeController : Controller
{
    public HomeController()
    {

    }

    public ActionResult Index()
    {
        TempData["myData"] = "Test data";
        return View();
    }
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

```
public ActionResult About()
{
    string data;

    if(TempData["myData"] != null)
        data = TempData["myData"] as string;

    TempData.Keep();

    return View();
}

public ActionResult Contact()
{
    string data;

    if(TempData["myData"] != null)
        data = TempData["myData"] as string;

    return View();
}
```

4. TRUYỀN DỮ LIỆU TỪ CONTROLLER VỀ VIEW

❖ TempData trong MVC ASP.NET:

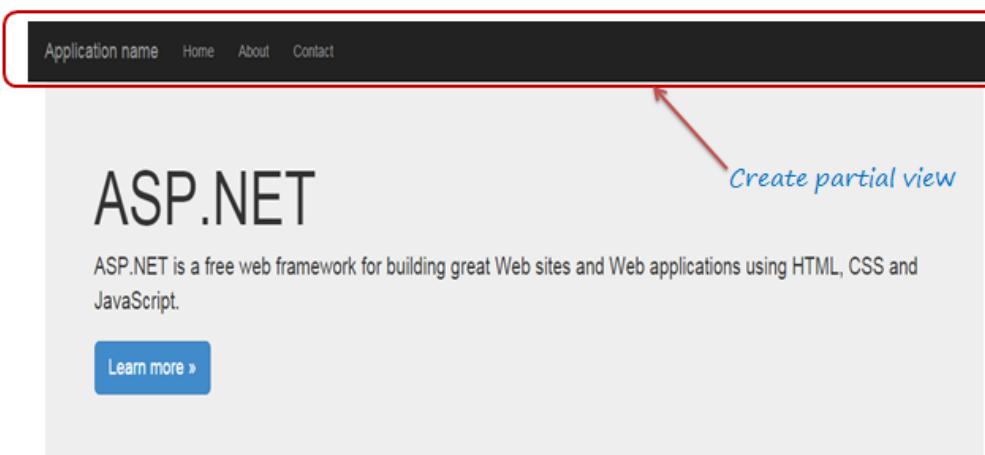
- TempData có thể được sử dụng để lưu trữ dữ liệu giữa hai yêu cầu liên tiếp.
- TempData là một kiểu TempDataDictionary.
- TempData sử dụng Session để lưu trữ dữ liệu.
- Giá trị của TempData phải được ép kiểu trước khi sử dụng.
- TempData chỉ có thể được sử dụng để lưu trữ các thông báo một lần như thông báo lỗi, thông báo xác thực.
- Gọi `TempData.Keep()` để giữ lại các giá trị TempData trong yêu cầu thứ ba tiếp theo.

5. SỬ DỤNG PARTIAL VIEW

❖ Partial view trong ASP.NET MVC là một view sử dụng lại, có thể được sử dụng như là một view con trong nhiều view khác. Partial view giúp loại bỏ trùng lặp mã bằng cách sử dụng lại cùng một partial view ở nhiều view khác. Chúng ta có thể sử dụng partial view là một phần trong layout view, cũng như nội dung view khác.

5. SỬ DỤNG PARTIAL VIEW

- ❖ Để bắt đầu, tạo một partial view đơn giản chứa thanh điều hướng như ảnh demo. Sau đó sẽ sử dụng partial view này cho các view khác.



5. SỬ DỤNG PARTIAL VIEW

- ❖ Hình dưới đây cho thấy mã html cho thanh điều hướng ở trên. Chúng ta sẽ cắt và dán mã này trong view một phần riêng biệt cho mục đích demo.

The screenshot shows the Visual Studio interface. On the right is the Solution Explorer with the project 'MVC-BasicTutorials' selected, showing files like _Layout.cshtml, _myLayoutPage.cshtml, _StudentList.cshtml, Error.cshtml, Student.cshtml, ViewStart.cshtml, Web.config, favicon.ico, Global.asax, and packages.config. On the left is the code editor with the _Layout.cshtml file open. A red box highlights the entire `<div class="navbar navbar-inverse navbar-fixed-top">` block, which contains the navigation bar code. A blue arrow points from the text "Move it to partial view" to the top of this highlighted block.

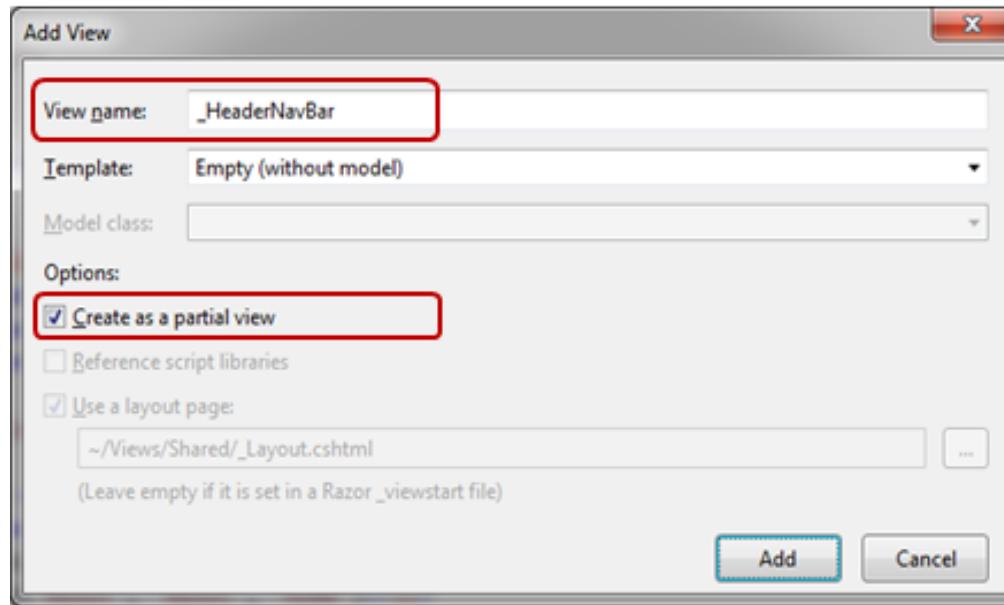
```
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>ViewBag.Title - My ASP.NET Application</title>
    <Styles.Render("~/Content/css")/>
    <Scripts.Render("~/bundles/modernizr")/>
</head>
<body>
    <div class="navbar 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>
</body>

```

5. SỬ DỤNG PARTIAL VIEW

❖ Cách tạo một Partial View mới:

- Click phải chuột trên thư mục shared --> Add --> View...
- Trong hộp thoại Add View, nhập tên View và chọn checkbox "Create as a partial view" và nhấp vào Add.



5. SỬ DỤNG PARTIAL VIEW

- ❖ Bây giờ, bạn có thể cắt mã ở trên cho thanh điều hướng và dán nó vào **_HeaderNavBar.cshtml** như hình dưới đây:

Ví dụ: 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>
```

5. SỬ DỤNG PARTIAL VIEW

❖ Hiển thị một partial view:

➤ Chúng ta có thể hiển thị một partial view trong view cha sử dụng các phương thức:

✓ Partial()

✓ RenderPartial()

✓ RenderAction().

✓ Mỗi phương thức có mục đích khác nhau.

5. SỬ DỤNG PARTIAL VIEW

❖ `Html.Partial()`

- Phương thức `@Html.Partial()` sử dụng chèn một partial view vào trong một view xác định. Tham số của phương thức `@Html.Partial()` là tên của partial view và trả về chuỗi html.
- Bảng sau liệt kê phương thức nạp chồng của Partial helper:

5. SỬ DỤNG PARTIAL VIEW

Phương thức Helper	Ý nghĩa
<i>MvcHtmlString</i> Html.Partial(string partialViewName)	Hiển thị nội dung trong partial view.
<i>MvcHtmlString</i> Html.Partial(string partialViewName, object model)	Hiển thị nội dung trong partial view được tham chiếu đến view. Các tham số trong Model truyền cho đối tượng model đến với partial view.
<i>MvcHtmlString</i> Html.Partial(string partialViewName, ViewDataDictionary viewData)	Hiển thị nội dung trong partial view được tham chiếu đến view. Tham số dữ liệu trong View được truyền qua bộ tự diễn mới đến partial view.
<i>MvcHtmlString</i> Html.Partial(string partialViewName, object model, ViewDataDictionary viewData)	Hiển thị nội dung trong partial view được tham chiếu đến view. Tham số model chuyển đổi từ model và View dữ liệu chuyển từ diễn dữ liệu sang partial view.

5. SỬ DỤNG PARTIAL VIEW

❖ `Html.RenderPartial()`

➤ Phương thức `@Html.renderPartial` giống như phương thức `@Html.Partial`, ngoại trừ việc nó trả về `void` và viết kết quả html của partial view xác định vào một luồng http trả về trực tiếp.

5. SỬ DỤNG PARTIAL VIEW

Phương thức Helper	Ý nghĩa
RenderPartial(String partialViewName)	Hiển thị partial view được chỉ định.
RenderPartial(String partialViewName, Object model)	Hiển thị partial view được chỉ định và thiết lập đối tượng model
RenderPartial(String partialViewName, ViewDataDictionary viewData)	Hiển thị partial view được chỉ định, thay thế thuộc tính ViewData bằng đối tượng ViewDataDictionary.
RenderPartial(String partialViewName, Object model, ViewDataDictionary viewData)	Hiển thị partial view được chỉ định, thay thế thuộc tính ViewData bằng đối tượng ViewDataDictionary và thiết lập đối tượng model được chỉ định.

5. SỬ DỤNG PARTIAL VIEW

❖ `Html.RenderAction()`

➤ Phương thức RenderAction gọi một controller xác định và action cụ thể và tạo ra kết quả như một partial view. Phương thức action cụ thể sẽ trả về PartialViewResult sử dụng phương thức Partial().

5. SỬ DỤNG PARTIAL VIEW

Tên phương thức	Giải thích
RenderAction(String actionPerformed)	Gọi phương thức hành động con đã chỉ định và hiển thị kết quả trong parent view.
RenderAction(String actionPerformed, Object routeValue)	Gọi phương thức hành động con được chỉ định bằng cách sử dụng các tham số đã chỉ định và hiển thị trong parent view.
RenderAction(String actionPerformed, String controllerName)	Gọi phương thức hành động con được chỉ định bằng cách sử dụng tên của trình điều khiển đã chỉ định và hiển thị nội tuyến kết quả trong parent view.
RenderAction(String actionPerformed, RouteValueDictionary routeValues)	Gọi phương thức hành động con được chỉ định bằng cách sử dụng các tham số đã chỉ định và hiển thị nội tuyến kết quả trong parent view.

5. SỬ DỤNG PARTIAL VIEW

Tên phương thức	Giải thích
RenderAction(String actionPerformed, String controllerName, Object routeValue)	Gọi phương thức hành động con được chỉ định bằng cách sử dụng các tham số và tên của trình điều khiển đã chỉ định và hiển thị nội tuyến kết quả trong parent view.
RenderAction(String actionPerformed, String controllerName, RouteValueDictionary routeValues)	Gọi phương thức hành động con được chỉ định bằng cách sử dụng các tham số và tên của trình điều khiển đã chỉ định và hiển thị nội tuyến kết quả trong parent view.

5. SỬ DỤNG PARTIAL VIEW

- ❖ Bây giờ, chúng ta có thể sử dụng bất kỳ phương thức nào ở trên để hiển thị partial view `_HeaderNavBar` thành `_Layout.cshtml`.
- ❖ Layout sau đây hiển thị partial view bằng phương thức `RenderPartial()`.

5. SỬ DỤNG PARTIAL VIEW

```
<!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>
```

5. SỬ DỤNG PARTIAL VIEW

```
<!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>
```

5. SỬ DỤNG PARTIAL VIEW

- ❖ Kết quả chạy ứng dụng:

The screenshot shows a web page with a navigation bar at the top containing 'Application name', 'Home', 'About', and 'Contact' links. A red box highlights the 'Application name' link. Below the navigation bar, the word 'ASP.NET' is displayed in large letters. A blue arrow points from the text 'Partial view' to the 'Application name' link in the navigation bar.

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

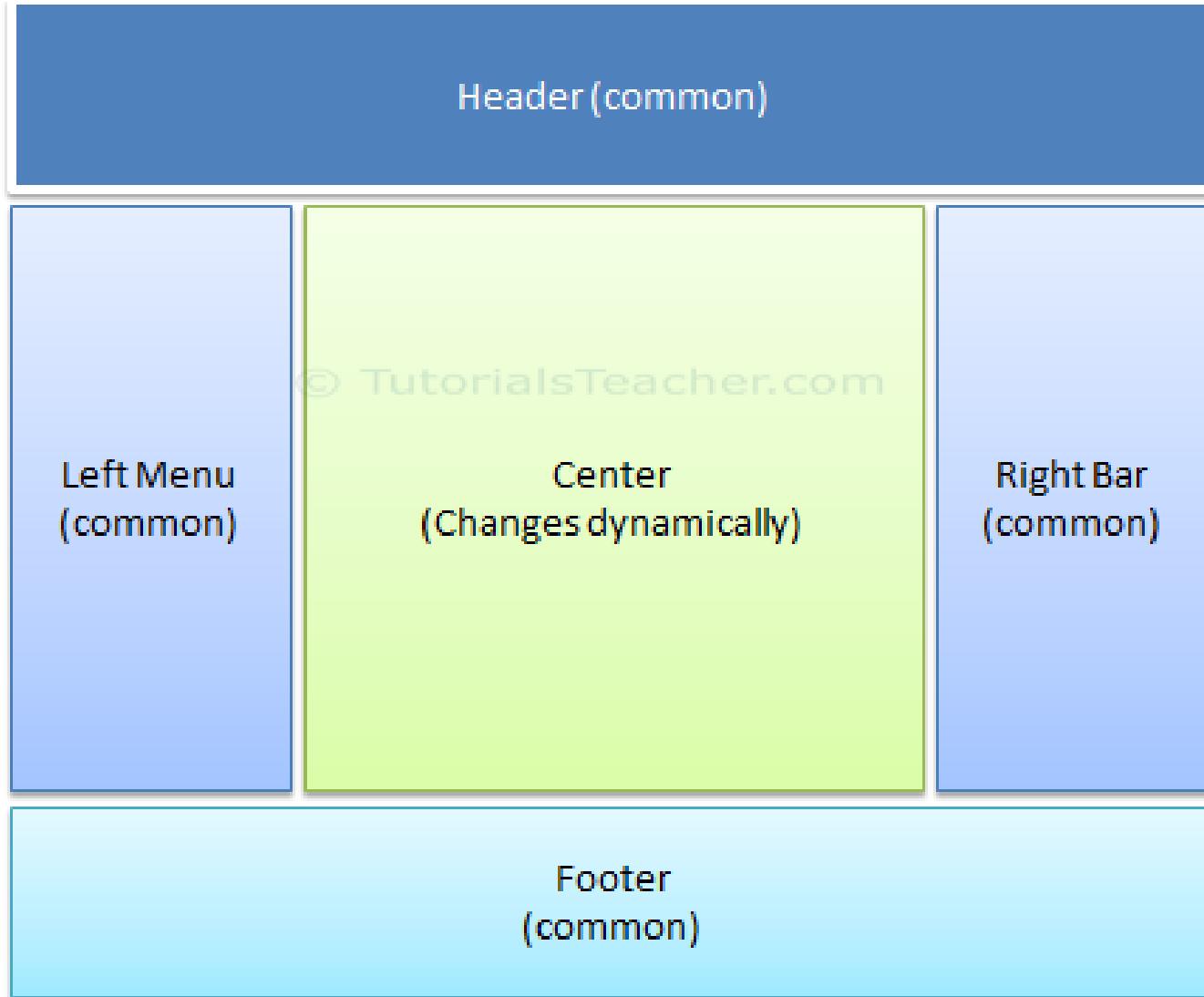
You can easily find a web hosting company that offers the right mix of features and price for your applications.

Learn more »

6. LAYOUT VIEW

- ❖ Một ứng dụng có thể chứa các phần chung trong giao diện người như logo, tiêu đề, menu trái, menu phải hoặc footer. ASP.NET MVC đã giới thiệu một giao diện Bố cục chứa các phần UI chung này, do đó chúng ta không phải viết cùng một mã trong mỗi trang.
- ❖ Giao diện bố cục giống như trang chính (master page) của ứng dụng WebForm ASP.NET.

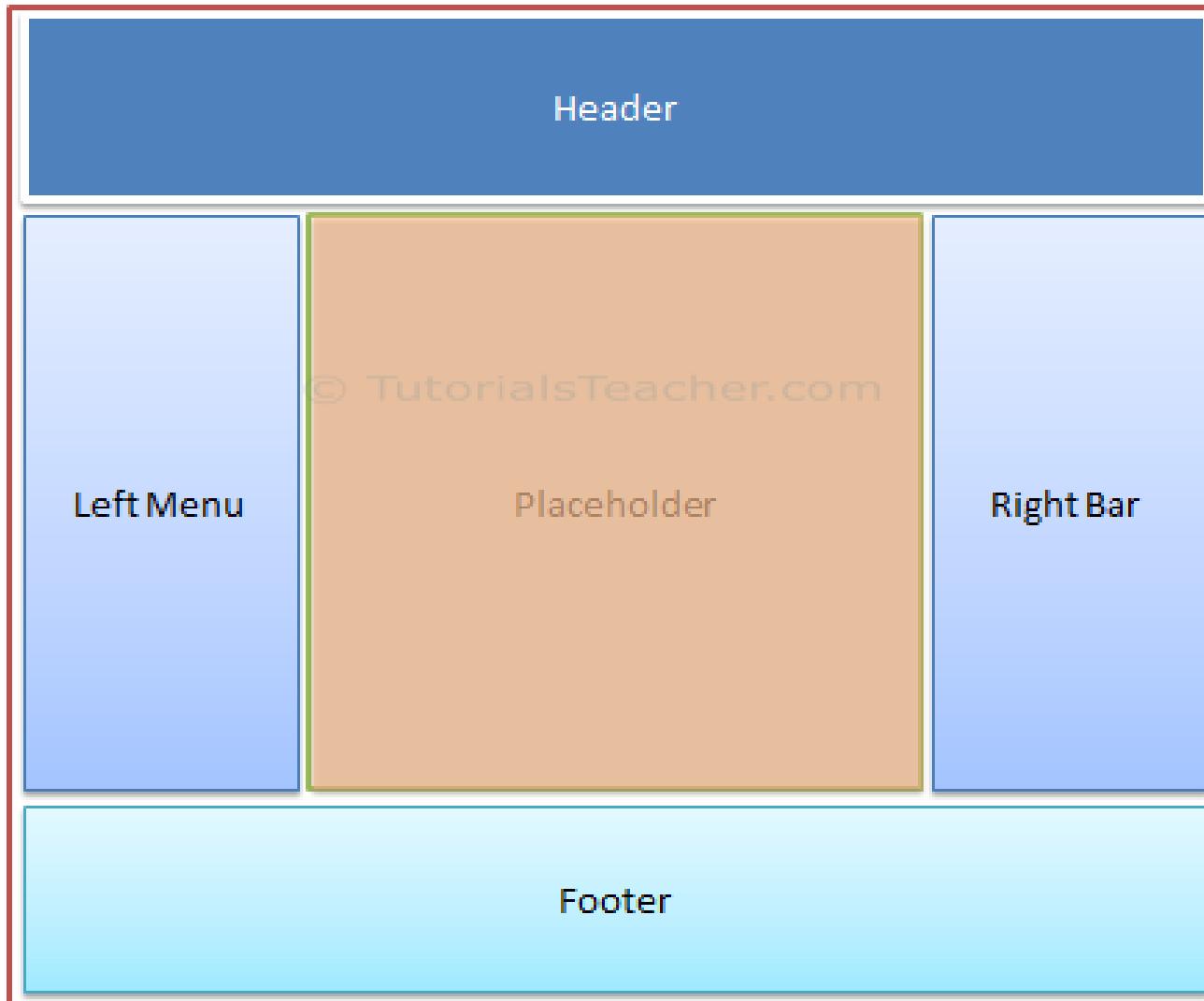
6. LAYOUT VIEW



6. LAYOUT VIEW

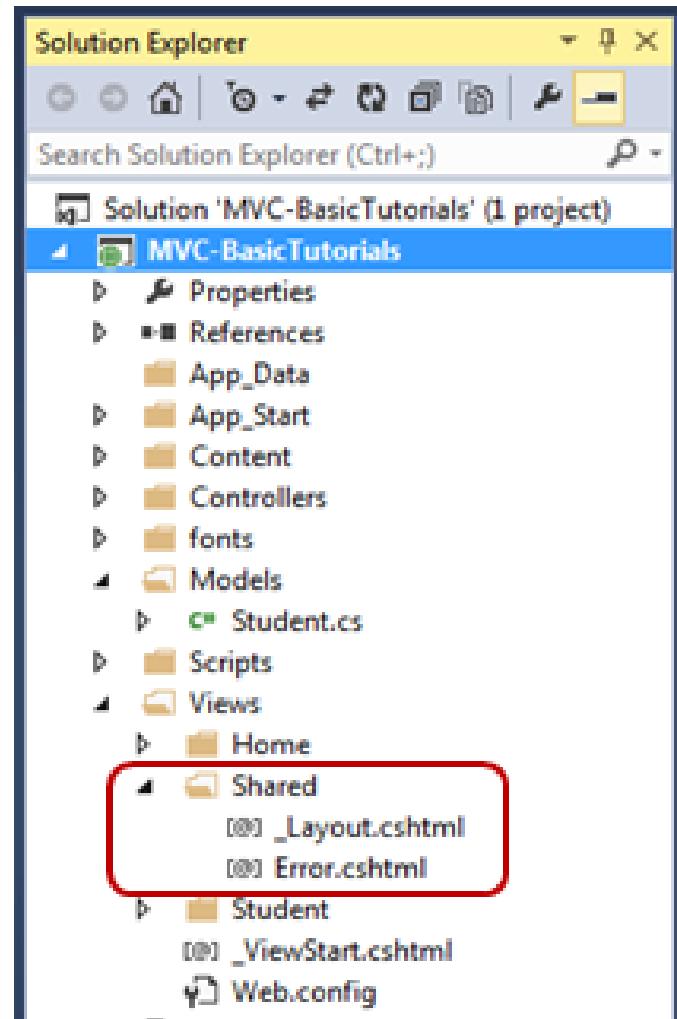
- ❖ Layout cho phép bạn xác định một mẫu trang web chung, có thể được kế thừa trong nhiều view để cung cấp giao diện nhất quán nhiều trang trong một ứng dụng. Layout bố trí giúp loại bỏ mã trùng lặp.
- ❖ Giao diện bố trí cho giao diện người dùng ở trên sẽ chứa phần Header, Menu bên trái, Thanh bên phải và Phần Footer. Nó chứa một placeholder nằm ở Center để dễ dàng thay đổi nội dung.

6. LAYOUT VIEW



6. LAYOUT VIEW

- ❖ Trong Razor các layout có phần mở rộng giống các view khác **.cshtml** hoặc **.vbhtml**. Các layout này được chia sẻ với nhiều views vì vậy nó phải được lưu trong thư mục **Shared**.



6. LAYOUT VIEW

The screenshot shows a code editor interface with two tabs: '_Layout.cshtml*' and 'TinTuc.cshtml'. The '_Layout.cshtml*' tab is active, displaying the following ASP.NET MVC layout code:

```
1 <html>
2   <head>
3     <meta charset="utf-8" />
4     <meta name="viewport" content="width=device-width, initial-scale=1.0">
5     <title>@ViewBag.Title - My ASP.NET Application</title>
6     @Styles.Render("~/Content/css")
7     @Scripts.Render("~/bundles/modernizr")
8   </head>
9   <body>
10    <div class="navbar navbar-inverse navbar-fixed-top">
11      <div class="container">
12        <div class="navbar-header">
13          <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
14            <span class="icon-bar"></span>
15            <span class="icon-bar"></span>
16            <span class="icon-bar"></span>
17          </button>
18          @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
19        </div>
20        <div class="navbar-collapse collapse">
21          <ul class="nav navbar-nav">
22            <li>@Html.ActionLink("Home", "Index", "Home")</li>
23            <li>@Html.ActionLink("About", "About", "Home")</li>
24            <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
25          </ul>
26        </div>
27      </div>
28    </div>
29    <div class="container body-content">
30      @RenderBody()
31      <hr />
32      <footer> <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p> </footer>
33    </div>
34    @Scripts.Render("~/bundles/jquery")
35    @Scripts.Render("~/bundles/bootstrap")
36    @RenderSection("scripts", required: false)
37  </body>
38 </html>
```

The 'TinTuc.cshtml' tab is visible in the background, indicating it is the current view being edited.

6. LAYOUT VIEW

❖ Sử dụng Layout:

- Làm thế nào để View biết nên sử dụng Layout nào?
- Bạn có thể đặt layout theo nhiều cách:
 - ✓ Sử dụng `_ViewStart.cshtml`
 - ✓ Hoặc thiết lập đường dẫn của layout bằng thuộc tính `layout` trong view riêng lẻ
 - ✓ Hoặc gọi layout trong phương thức.

6. LAYOUT VIEW

❖ Sử dụng _ViewStart.cshtml

- Theo mặc định, ViewStart.cshtml nằm trong thư mục Views. Nó thiết lập trang layout mặc định cho tất cả các view trong thư mục và các thư mục con bằng thuộc tính Layout. Bạn có thể chỉ định đường dẫn hợp lệ của bất kỳ trang Layout nào cho thuộc tính Layout.

```
_ViewStart.cshtml*  X _Layout.cshtml      TinTuc.cshtml      HomeController.cs  
D:\ĐH GTVT\1\WebApplication2\WebApplication2\Views\_ViewStart.cshtml*  
1 @{}  
2     Layout = "~/Views/Shared/_Layout.cshtml";  
3 }
```

6. LAYOUT VIEW

❖ Cài đặt thuộc tính Layout cho từng views:

- Bạn cũng có thể ghi đè trang layout mặc định được đặt bởi _ViewStart.cshtml bằng cách đặt thuộc tính Layout trong mỗi view .cshtml riêng lẻ. Ví dụ: view Index sử dụng _myLayoutPage.cshtml khi đó ta đổi _ViewStart.cshtml thành _Layout.cshtml.

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

6. LAYOUT VIEW

❖ Gọi trang Layout trong phương thức ActionResult:

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

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

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

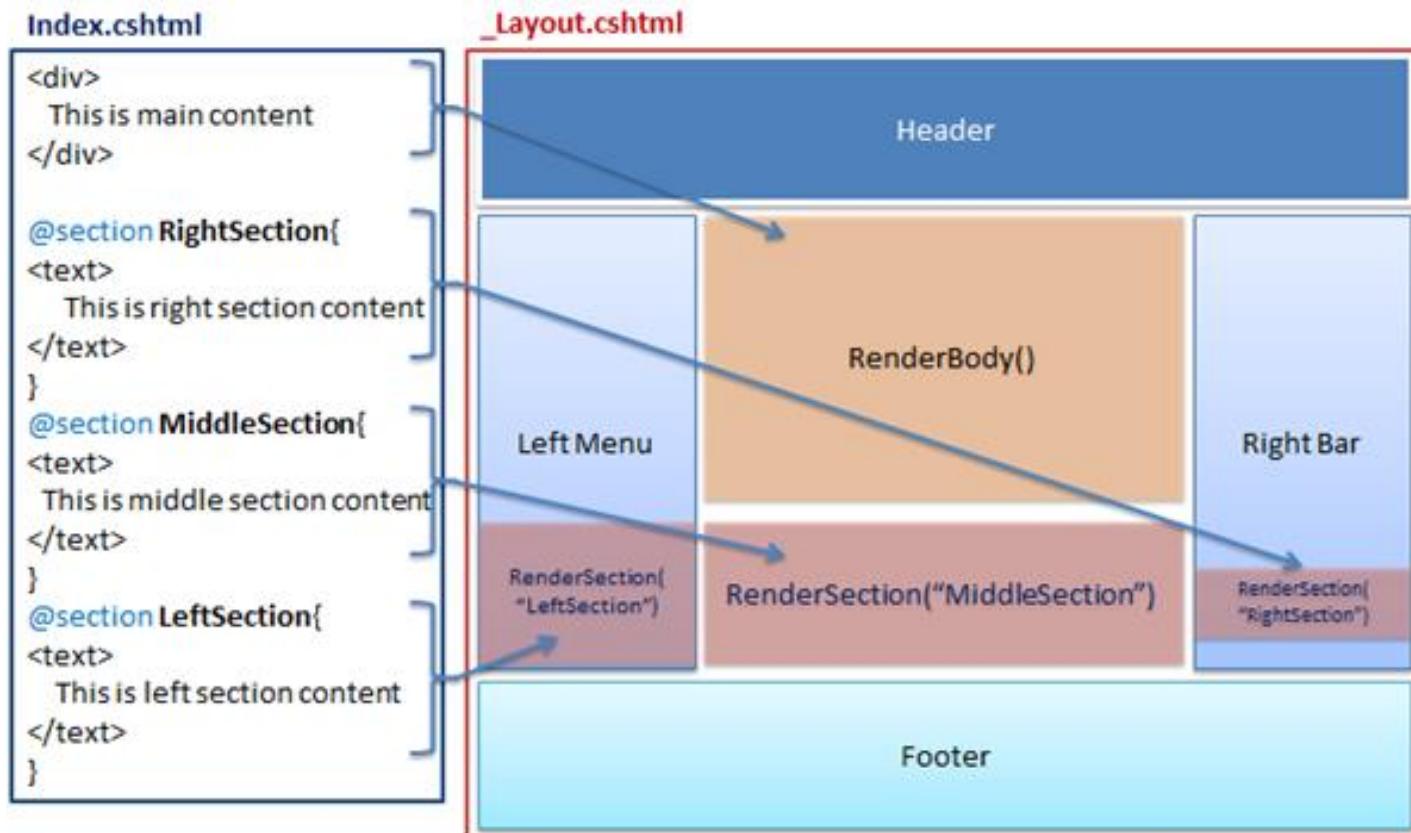
6. LAYOUT VIEW

- ❖ Các phương thức hiển thị, ASP.NET MVC hiển thị các Views sử dụng các phương thức sau:

Phương thức	Giải thích
RenderBody()	Hiển thị phần của view con không nằm trong phần được đặt tên. Layout phải có phương thức RenderBody () .
RenderSection(string name)	Hiển thị nội dung của phần được đặt tên và chỉ định xem phần đó có bắt buộc không. RenderSection () là tùy chọn trong Layout.

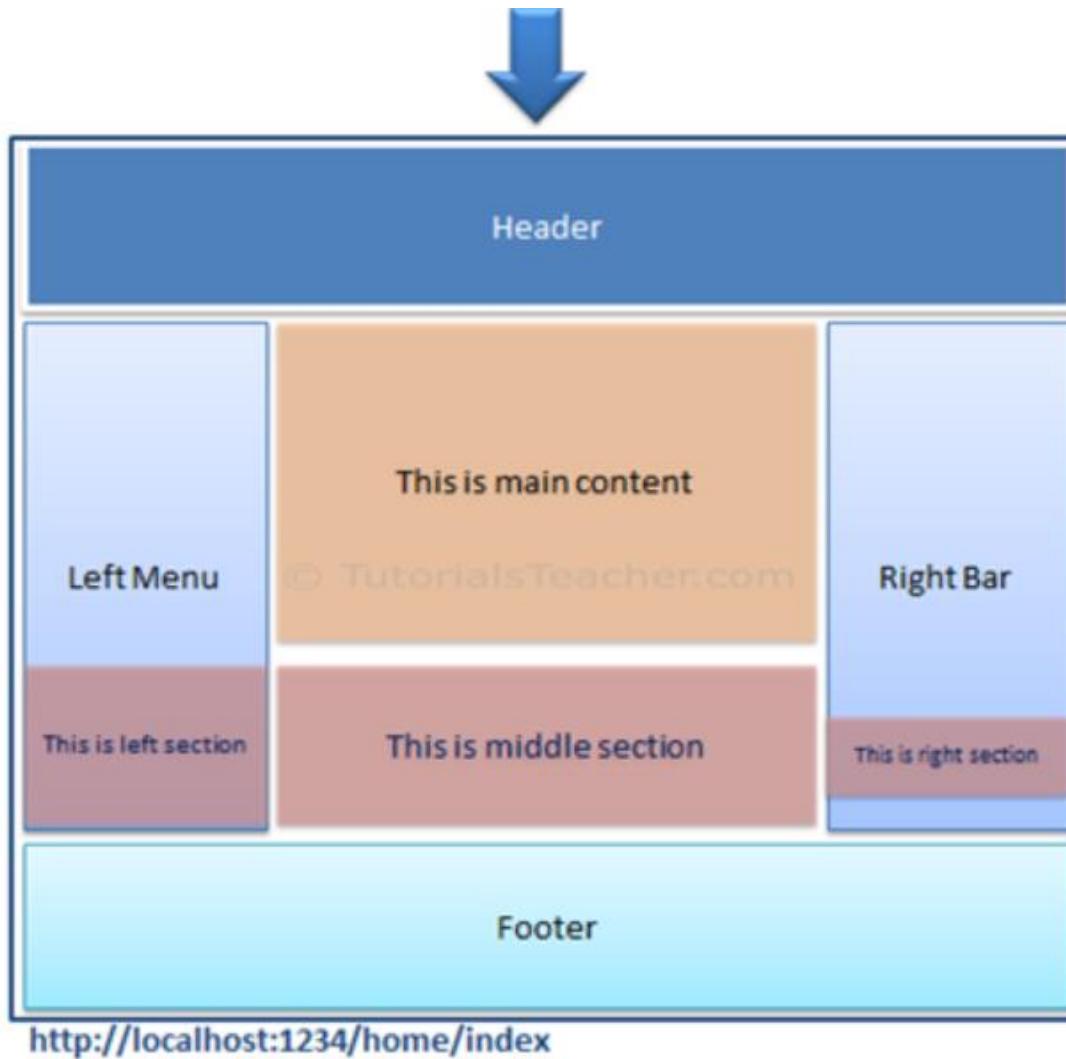
6. LAYOUT VIEW

- ❖ Hình dưới đây minh họa việc sử dụng các phương thức **RenderBody** và **RenderSection**.



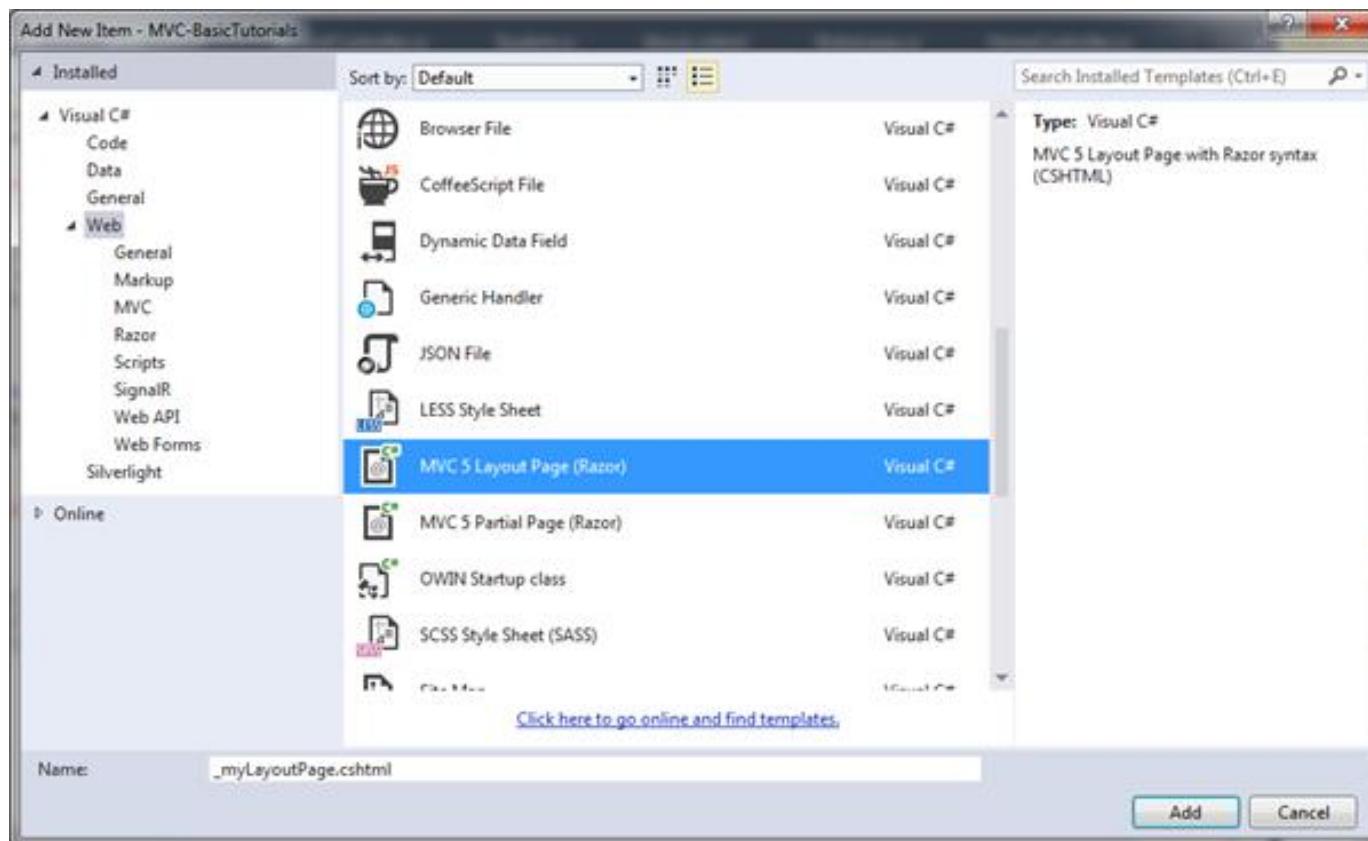
6. LAYOUT VIEW

❖ Kết quả:



7. CÁCH TẠO LAYOUT VIEW TRONG MVC

- ❖ Để tạo một layout mới trong Visual Studio, click phải chuột trên thư mục shared --> Add --> New Item..



7. CÁCH TẠO LAYOUT VIEW TRONG MVC

- ❖ Bạn sẽ thấy `_myLayoutPage.cshtml` như sau:

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

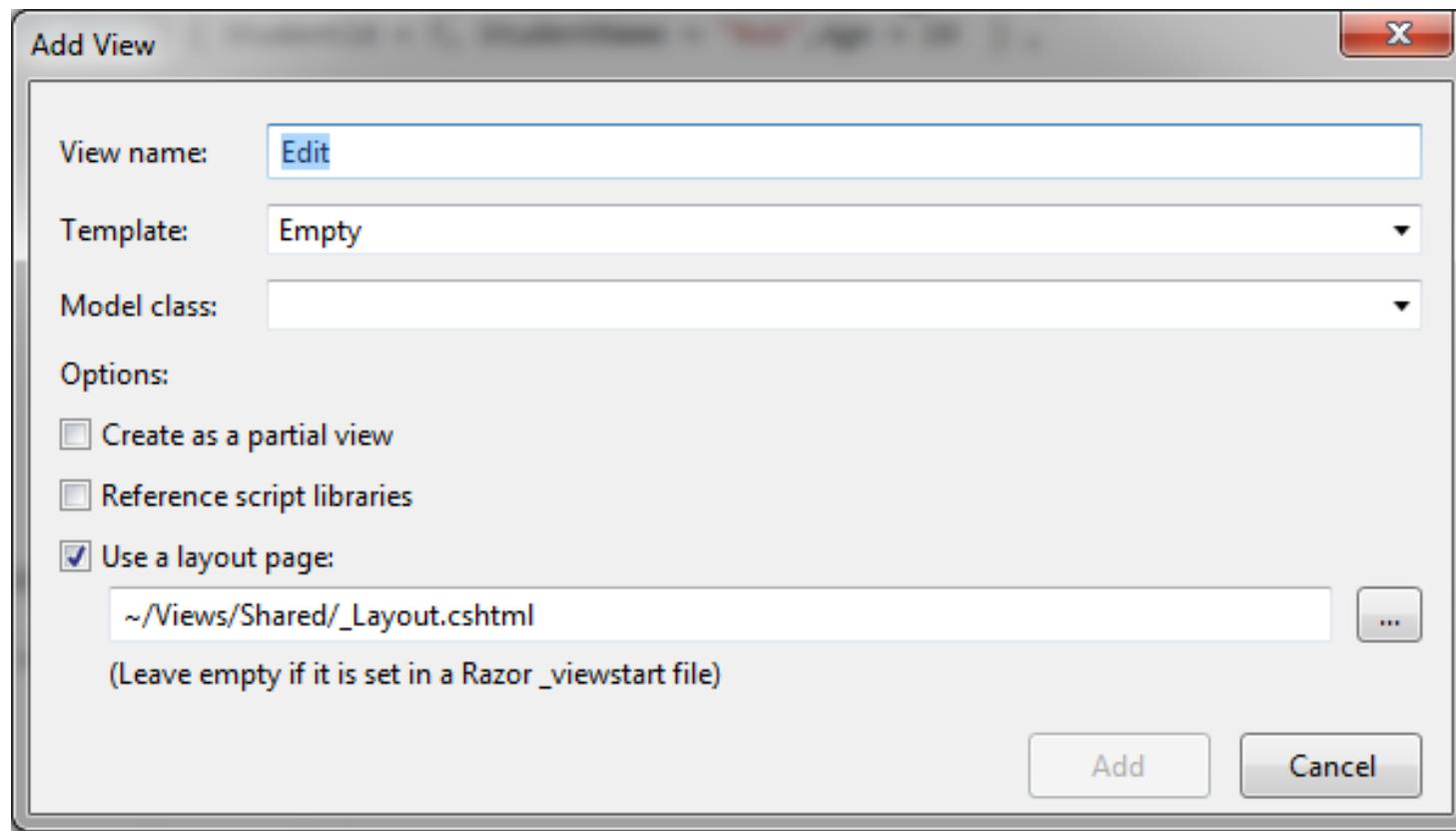
7. CÁCH TẠO LAYOUT VIEW TRONG MVC

❖ Ví dụ thêm 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>
```

7. CÁCH TẠO LAYOUT VIEW TRONG MVC

- ❖ Bây giờ, chúng ta sử dụng `_myLayoutPage.cshtml` này với View Index của HomeController.



7. CÁCH TẠO LAYOUT VIEW TRONG MVC

- ❖ Tập tin **Index.cshtml** như hiển thị bên dưới.

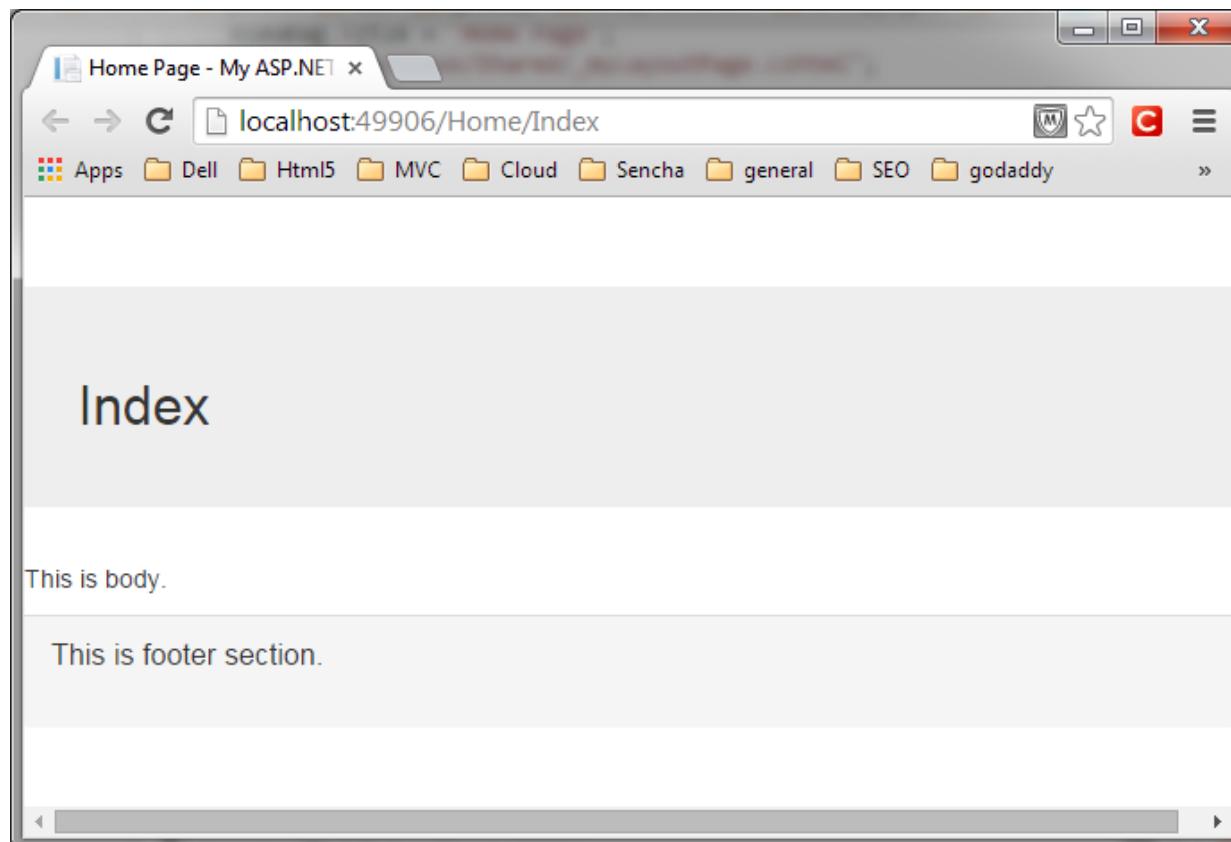
```
@{  
    ViewBag.Title = "Home Page";  
    Layout = "~/Views/Shared/_myLayoutPage.cshtml";  
}  
  
<h2>Index</h2>
```

7. CÁCH TẠO LAYOUT VIEW TRONG MVC

```
@{  
    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>
```

7. CÁCH TẠO LAYOUT VIEW TRONG MVC

- ❖ Chạy ứng dụng và ta sẽ thấy view Index sẽ chứa body và footer như hình bên dưới.



8. SỬ DỤNG VIEW RAZOR

- ❖ Razor View Engine là 1 ngôn ngữ cho phép bạn tạo ra các giao diện cho ứng dụng ASP.NET MVC trong khi vẫn giữ được sự phân chia rõ ràng, khả năng có thể kiểm tra, và sự phát triển dựa trên pattern.
- ❖ Các lập trình viên ASP.NET MVC đang tìm kiếm cho mình 1 ngôn ngữ có cú pháp rõ ràng, ngắn gọn, và bây giờ nó đã được xây dựng sẵn (dĩ nhiên là có rất nhiều các View Engine của hàng thứ 3 khác) với ngôn ngữ quen thuộc là C#.

8. SỬ DỤNG VIEW RAZOR

❖ Cú pháp Razor có các đặc điểm sau:

- **Compact**: Razor nhỏ gọn cho phép bạn giảm thiểu số lượng ký tự và tổ hợp phím cần thiết để viết mã.
- **Easy to Learn**: Razor dễ học, bạn có thể dùng các ngôn ngữ lập trình: C#, Visual Basic.
- **Intellisense**: Razor hỗ trợ các câu lệnh trong Visual Studio.
- **Unit Testable**: Razor hỗ trợ khả năng unit test cho các view mà không cần các controller hoặc web-server.

8. SỬ DỤNG VIEW RAZOR

- ❖ Cú pháp Razor: Razor sử dụng ký tự `@` để chuyển HTML sang C#. Có 2 cách để khai báo:
 - Sử dụng Razor expression
 - Sử dụng khối lệnh Razor

Các biểu thức này được xử lý bởi Razor View Engine và được gán vào response.

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

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

Kết quả:

```
Razor syntax demo
08-09-2014
```

8. SỬ DỤNG VIEW RAZOR

❖ Khối lệnh

- Chúng ta có thể viết nhiều dòng lệnh phía máy chủ được đặt trong dấu ngoặc nhọn @ {...}. Mỗi dòng phải kết thúc bằng dấu chấm phẩy giống như C#.

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

8. SỬ DỤNG VIEW RAZOR

- ❖ Hiển thị văn bản từ khối lệnh
- ❖ Sử dụng @: hoặc <text> / <text> để hiển thị văn bản trong khối lệnh.

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

8. SỬ DỤNG VIEW RAZOR

- ❖ Hiển thị văn bản bằng `<text>` trong một khối mã như được hiển thị bên dưới.

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

8. SỬ DỤNG VIEW RAZOR

❖ Câu lệnh điều kiện (if.. else)

- Viết điều kiện if-else bắt đầu bằng ký hiệu @. Khối mã if-else phải được đặt trong dấu ngoặc {}, ngay cả đối với câu lệnh đơn.

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

8. SỬ DỤNG VIEW RAZOR

❖ Vòng lặp:

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

❖ Kết quả:

```
0  
1  
2  
3  
4
```

8. SỬ DỤNG VIEW RAZOR

❖ Model:

- Sử dụng `@model` để sử dụng đối tượng model ở bất cứ đâu trong 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>
```

8. SỬ DỤNG VIEW RAZOR

❖ Khai báo biến:

- Khai báo một biến trong một khôi mã được đặt trong ngoặc và sau đó sử dụng các biến đó trong html với ký hiệu @.

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

8. SỬ DỤNG VIEW RAZOR

- ❖ Xem thêm cú pháp:

<https://learn.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c>

9. SỬ DỤNG HTML HELPER

- ❖ **HMTLHelper** giúp chúng ta viết phần tử HTML trong **Razor** sử dụng cú pháp thân thiện với HTML.
- ❖ **Razor** được tạo sử dụng **HMTLHelper** nhìn như phần tử HTML thuần. Nó thao tác với các phần tử HTML như thêm mới phần tử HTML hay thay thế các nội dung có sẵn bằng một cái mới.

9. SỬ DỤNG HTML HELPER

- ❖ Ví dụ, sử dụng thẻ **Form HMTLHelper**, chúng ta có thể tạo ra thẻ `<form>` như dưới đây.



9. SỬ DỤNG HTML HELPER

- ❖ Lớp **HtmlHelper** tạo các phần tử html. Ví dụ:
 - HtmlHelper class generates html elements. For example,
 @Html.ActionLink("Create New", "Create")
- ❖ Sẽ tạo ra thẻ HTML
 - Create New
- ❖ Có nhiều phương thức mở rộng cho lớp **HtmlHelper**,
tạo ra các điều khiển html khác nhau.

9. SỬ DỤNG HTML HELPER

- ❖ Bảng sau liệt kê các phương thức **HtmlHelper** và điều khiển html mỗi phương thức tạo ra.

HtmlHelper	Strongly Typed	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

9. SỬ DỤNG HTML HELPER

Strongly Typed		
HtmlHelper	HtmlHelpers	Html Control
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.Editor	Html.EditorFor	Tạo các điều khiển Html dựa trên loại dữ liệu của thuộc tính .

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextBox:

- Lớp **HtmlHelper** có hai phương thức mở rộng để tạo **Textbox(<input type="text'>)** trong Razor là : **TextBox()** và **TextBoxFor()**.
- Phương thức **Html.TextBox()** tạo phần tử **<input type="text" />** với tên được chỉ định, giá trị và thuộc tính **html.Html**.
- Phương thức **Html.TextBoxFor()** tạo ra một phần tử **<input type="text'>** với thuộc tính model xác định sử dụng một biểu thức lambda.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextBox:

➤ Ví dụ TextBox() và TextBoxFor():

```
public class Student
{
    public int StudentId { get; set; }
    [Display(Name="Name")]
    public string StudentName { get; set; }
    public int Age { get; set; }
    public bool isNewlyEnrolled { get; set; }
    public string Password { get; set; }
}
```

Kết quả:

```
<input class="form-control"
      id="StudentName"
      name="StudentName"
      type="text"
      value="" />
```

```
@model Student
```

```
@Html.TextBox("StudentName", null, new { @class = "form-control" })
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextBox:

- Bạn cũng có thể chỉ định bất kỳ tên cho hộp văn bản. Tuy nhiên, nó sẽ không được liên kết với Model nào cả.

```
@Html.TextBox("myTextBox", "This is value", new {  
    @class = "form-control" })
```

Kết quả:

```
<input class="form-control"  
      id="myTextBox"  
      name="myTextBox"  
      type="text"  
      value="This is value" />
```



This is value

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextBox:

- **TextBoxFor** tạo ra một phần tử `<input type = "text">` với thuộc tính model xác định sử dụng một biểu thức lambda.
- Cú pháp phương thức **TextBoxFor**:

```
MvcHtmlString TextBoxFor(Expression<Func<TModel, TValue>>  
expression, object htmlAttributes)
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextBox:

```
@model Student  
  
@Html.TextBoxFor(m => m.StudentName, new { @class = "form-control" })
```

Kết quả:

```
<input class="form-control"  
      id="StudentName"  
      name="StudentName"  
      type="text"  
      value="John" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextBox:

➤ Sự khác biệt giữa TextBox và TextBoxFor:

- ✓ @Html.TextBox() là phương thức không chăt chẽ trong khi @Html.TextBoxFor() là phương thức mở rộng nên chăt chẽ hơn.
- ✓ TextBox() yêu cầu tên thuộc tính dưới dạng tham số chuỗi trong khi đó TextBoxFor() yêu cầu biểu thức lambda tham số.
- ✓ TextBox không cung cấp cho bạn lỗi thời gian biên dịch nếu bạn đã chỉ định tên thuộc tính sai. Nó sẽ ném ngoại lệ thời gian chạy.
- ✓ TextBoxFor là phương thức chung vì vậy nó sẽ cung cấp cho bạn lỗi biên dịch thời gian nếu bạn đã chỉ định thay đổi tên thuộc tính hoặc tên thuộc tính sai.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextArea

- Lớp **HtmlHelper** có hai phương thức mở rộng để tạo phần tử `<textarea>` nhiều dòng trong Razor: **TextArea()** và **TextAreaFor()**.
- Chúng ta sẽ sử dụng Model Student với phương thức **TextArea()** và **TextAreaFor()**.

```
public class Student
{
    public int StudentId { get; set; }
    [Display(Name="Name")]
    public string StudentName { get; set; }
    public string Description { get; set; }
}
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextArea

➤ Cú pháp phương thức TextArea():

MvcHtmlString **Html.TextArea(string name, string value, object htmlAttributes)**

```
@model Student  
  
@Html.TextArea("Description", null, new { @class = "form-control" })
```

Kết quả:

```
<textarea class="form-control"  
         id="Description"  
         name="Description"  
         rows="2"  
         cols="20">This is value</textarea>
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextArea

- Bạn cũng có thể chỉ định bất kỳ tên nào cho textarea. Tuy nhiên, nó sẽ không bị ràng buộc với một model.

```
@Html.TextArea("myTextArea", "This is value", new {  
    @class = "form-control" })
```

Kết quả:

```
<textarea class="form-control"  
         cols="20"  
         id="myTextArea"  
         name="myTextArea"  
         rows="2">This is value</textarea>
```



This is value

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextArea

- TextAreaFor tạo ra một phần tử textarea với thuộc tính model xác định sử dụng một biểu thức lambda.
- Cú pháp phương thức TextAreaFor:

MvcHtmlString TextAreaFor(<Expression<Func<TModel, TValue>>expression,
object htmlAttributes)

```
@model Student
```

```
@Html.TextAreaFor(m => m.Description, new { @class = "form-control" })
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – TextArea

```
@model Student  
  
@Html.TextAreaFor(m => m.Description, new { @class = "form-control" })
```

Kết quả:

```
<textarea class="form-control"  
         cols="20"  
         id="Description"  
         name="Description"  
         rows="2"></textarea>
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – CheckBox

- Lớp HtmlHelper có hai phương thức mở rộng để tạo CheckBox(<input type="checkbox">) trong Razor là: CheckBox() và CheckBoxFor().
- **Html.CheckBox()** là một phương thức tạo ra <input type = "checkbox"> với tên được chỉ định, các thuộc tính boolean và html.
- **Html.CheckBoxFor()** tạo ra một phần tử <input type="checkbox"> với thuộc tính model xác định sử dụng một biểu thức lambda.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – CheckBox

➤ Html.CheckBox()

```
@Html.CheckBox("isNewlyEnrolled", true)
```

Kết quả:

```
<input checked="checked"  
       id="isNewlyEnrolled"  
       name="isNewlyEnrolled"  
       type="checkbox"  
       value="true" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – CheckBox

➤ Html.CheckBoxFor()

```
@model Student  
  
@Html.CheckBoxFor(m => m.isNewlyEnrolled)
```

Kết quả:

```
<input data-val="true"  
       data-val-required="The isNewlyEnrolled field is required."  
       id="isNewlyEnrolled"  
       name="isNewlyEnrolled"  
       type="checkbox"  
       value="true" />  
  
<input name="isNewlyEnrolled" type="hidden" value="false" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – RadioButton

- Lớp **HtmlHelper** có hai phương thức mở rộng để tạo **RadioButton(<input type="text'>)** trong Razor là: **RadioButton()** và **RadioButtonFor()**.
- Phương thức **Html.RadioButton()** tạo phần tử **<input type="text" />** với tên được chỉ định, giá trị và thuộc tính **html**.
- **Html.RadioButtonFor()** tạo ra một phần tử **input text** với thuộc tính **model** xác định sử dụng một biểu thức lambda.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – RadioButton

Male:
Female:

➤ Html.RadioButton()

```
Male: @Html.RadioButton("Gender", "Male")
Female: @Html.RadioButton("Gender", "Female")
```

Kết quả:

```
Male: <input checked="checked"
           id="Gender"
           name="Gender"
           type="radio"
           value="Male" />
```

```
Female: <input id="Gender"
            name="Gender"
            type="radio"
            value="Female" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – RadioButton

➤ Html.RadioButtonFor()

```
@model Student  
@Html.RadioButtonFor(m => m.Gender, "Male")  
@Html.RadioButtonFor(m => m.Gender, "Female")
```

Kết quả:

```
<input checked="checked"  
       id="Gender"  
       name="Gender"  
       type="radio"  
       value="Male" />  
  
<input id="Gender"  
       name="Gender"  
       type="radio"  
       value="Female" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – DropDownList

- Lớp HtmlHelper có hai phương thức mở rộng để tạo <Select> trong Razor là: **DropDownList()** và **DropDownListFor()**.

```
public class Student
{
    public int StudentId { get; set; }
    [Display(Name="Name")]
    public string StudentName { get; set; }
    public Gender StudentGender { get; set; }
}

public enum Gender
{
    Male,
    Female
}
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – DropDownList

➤ Html.DropDownList()

```
using MyMVCApp.Models

@model Student

@Html.DropDownList("StudentGender",
                    new SelectList(Enum.GetValues(typeof(Gender))),
                    "Select Gender",
                    new { @class = "form-control" })
```

Kết quả:

```
<select class="form-control" id="StudentGender" name="StudentGender">
    <option>Select Gender</option>
    <option>Male</option>
    <option>Female</option>
</select>
```

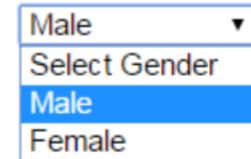
9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – DropDownList

➤ Html.DropDownListFor()

```
@using MyMVCApp.Models  
  
@model Student  
  
@Html.DropDownListFor(m => m.StudentGender,  
                      new SelectList(Enum.GetValues(typeof(Gender))),  
                      "Select Gender")
```

Gender:



Kết quả HTML

```
<select class="form-control" id="StudentGender" name="StudentGender">  
    <option>Select Gender</option>  
    <option>Male</option>  
    <option>Female</option>  
</select>
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Hidden field

- Lớp HtmlHelper có hai phương thức mở rộng để tạo Textbox(<input type="hidden">) trong Razor là: **Hidden()** và **HiddenFor()**.
- Phương thức **Html.Hidden()** tạo phần tử <input type="hidden" /> với tên được chỉ định, giá trị và thuộc tính html.
- **Html.HiddenFor()** tạo ra một phần tử input text với thuộc tính model xác định sử dụng một biểu thức lambda.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Hidden field

```
@model Student
```

```
@Html.Hidden("StudentId")
```

Kết quả HTML:

```
<input id="StudentId"
       name="StudentId"
       type="hidden"
       value="1" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Hidden field

```
@model Student
```

```
@Html.HiddenFor(m => m.StudentId)
```

Kết quả HTML:

```
<input data-val="true"
       data-val-number="The field StudentId must be a number."
       data-val-required="The StudentId field is required."
       id="StudentId"
       name="StudentId"
       type="hidden"
       value="1" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Password

- Lớp HtmlHelper có hai phương thức mở rộng để tạo (<input type="password">) trong Razor là: **Password()** và **PasswordFor()**.
- Phương thức **Html.Password()** tạo phần tử <input type="Password" /> với tên được chỉ định, giá trị và thuộc tính html.
- **Html.PasswordFor()** tạo ra một phần tử <input type="Password" /> với thuộc tính model xác định sử dụng một biểu thức lambda.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Password

Ví dụ **Html.Password()**:

```
@model Student  
  
@Html.Password("OnlinePassword")
```

Kết quả HTML:

Password:

...

```
<input  
    id="OnlinePassword"  
    name="OnlinePassword"  
    type="password"  
    value="" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Password

Ví dụ: PasswordFor ()

```
@model Student  
  
@Html.PasswordFor(m => m.Password)
```

Kết quả HTML:

```
<input id="Password" name="Password" type="password" value="mypassword" />
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Display

- HtmlHelper có hai phương thức mở rộng để tạo ra chuỗi html: **Display()** và **DisplayFor()**.
- Chúng ta sẽ sử dụng Model Student với 2 phương thức Display() và DisplayFor().

Ví dụ: Model Student

```
public class Student
{
    public int StudentId { get; set; }
    public string StudentName { get; set; }
    public int Age { get; set; }
}
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Display

Cú pháp phương thức Display()

```
MvcHtmlString Display(string expression)
```

Ví dụ : Html.Display() trong chế độ Razor

```
@Html.Display("StudentName")
```

Kết quả HTML:

```
"Steve"
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Display

Cú pháp phương thức `DisplayFor`

```
MvcHtmlString DisplayFor(<Expression<Func<TModel, TValue>> expression)
```

Ví dụ : `DisplayFor()` trong chế độ Razor.

```
@model Student  
  
@Html.DisplayFor(m => m.StudentName)
```

Kết quả HTML:

```
" Steve "
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Label

- HtmlHelper có hai phương thức mở rộng để tạo ra chuỗi html: **Label()** và **LabelFor()**.

Ví dụ **Html.Label()** trong chế độ Razor

```
@Html.Label("StudentName")
```

Kết quả HTML:

```
<label for="StudentName">Name</label>
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Label

Ví dụ LabelFor() trong chế độ Razor

```
@model Student
```

```
@Html.LabelFor(m => m.StudentName)
```

Kết quả HTML:

```
<label for="StudentName">Name</label>
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Editor

- Chúng ta đã thấy các phương thức HtmlHelper khác nhau được sử dụng để tạo các phần tử html khác nhau trong các phần trước.
- ASP.NET MVC cũng có một phương thức tạo các phần tử đầu vào html dựa trên kiểu dữ liệu. Phương thức mở rộng **Editor()** hoặc **EditorFor()** tạo các phần tử html dựa trên kiểu dữ liệu của thuộc tính đối tượng.

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Editor

- Bảng sau liệt kê phần tử html được tạo cho từng loại dữ liệu theo phương thức Editor () hoặc EditorFor ().

Kiểu dữ liệu thuộc tính	Thành phần Html
string	<input type="text" >
int	<input type="number" >
decimal, float	<input type="text" >
boolean	<input type="checkbox" >
Enum	<input type="text" >
DateTime	<input type="datetime" >

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Editor

- Chúng ta sẽ sử dụng lớp **Model** sau với phương thức **Editor** và **EditorFor**.

Ví dụ Model Student

```
public class Student
{
    public int StudentId { get; set; }
    [Display(Name="Name")]
    public string StudentName { get; set; }
    public int Age { get; set; }
    public bool isNewlyEnrolled { get; set; }
    public string Password { get; set; }
    public DateTime DoB { get; set; }
}
```

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Editor

Ví dụ Editor() trong chế độ Razor

```
StudentId:      @Html.Editor("StudentId")
Student Name:   @Html.Editor("StudentName")
Age:           @Html.Editor("Age")
Password:       @Html.Editor("Password")
isNewlyEnrolled: @Html.Editor("isNewlyEnrolled")
Gender:         @Html.Editor("Gender")
DoB:            @Html.Editor("DoB")
```

Kết quả:

StudentId:	<input type="text" value="1"/>
Student Name:	<input type="text" value="Jogn"/>
Age:	<input type="text" value="19"/>
Password:	<input type="text" value="sdf"/>
isNewlyEnrolled:	<input checked="" type="checkbox"/>
Gender:	<input type="text" value="Boy"/>
DoB:	<input type="text" value="02-06-2015 11:39:15"/>

9. SỬ DỤNG HTML HELPER

❖ HtmlHelper – Editor

Ví dụ EditorFor() trong chế độ Razor

```
StudentId:      @Html.EditorFor(m => m.StudentId)  
Student Name:   @Html.EditorFor(m => m.StudentName)  
Age:           @Html.EditorFor(m => m.Age)  
Password:       @Html.EditorFor(m => m.Password)  
isNewlyEnrolled: @Html.EditorFor(m => m.isNewlyEnrolled)  
Gender:         @Html.EditorFor(m => m.Gender)  
DoB:            @Html.EditorFor(m => m.DoB)
```

StudentId:	<input type="text" value="1"/>
Student Name:	<input type="text" value="Jogn"/>
Age:	<input type="text" value="19"/>
Password:	<input type="text" value="sdf"/>
isNewlyEnrolled:	<input checked="" type="checkbox"/>
Gender:	<input type="text" value="Boy"/>
DoB:	<input type="text" value="02-06-2015 11:39:15"/>

