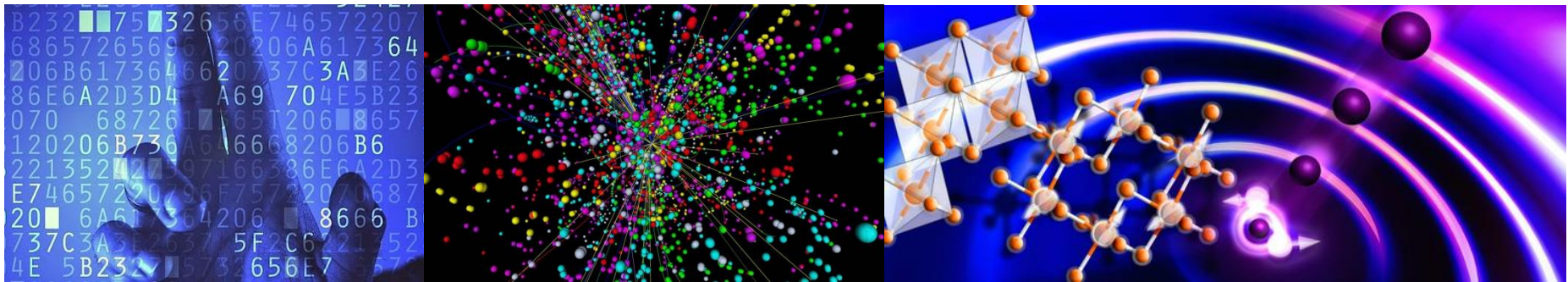


LẬP TRÌNH WEB

BỘ MÔN KHOA HỌC DỮ LIỆU & TRÍ TUỆ NHÂN TẠO



Chương 1

MÔ HÌNH MVC

1. Giới thiệu mô hình MVC
2. ViewData ViewBag in MVC
3. LayoutPage in MVC
4. Html Helper



1. Giới thiệu MVC

1.1. Giới thiệu mô hình MVC

1.2. So sánh MVC và WebForm

1.3. Tạo Web application với MVC4

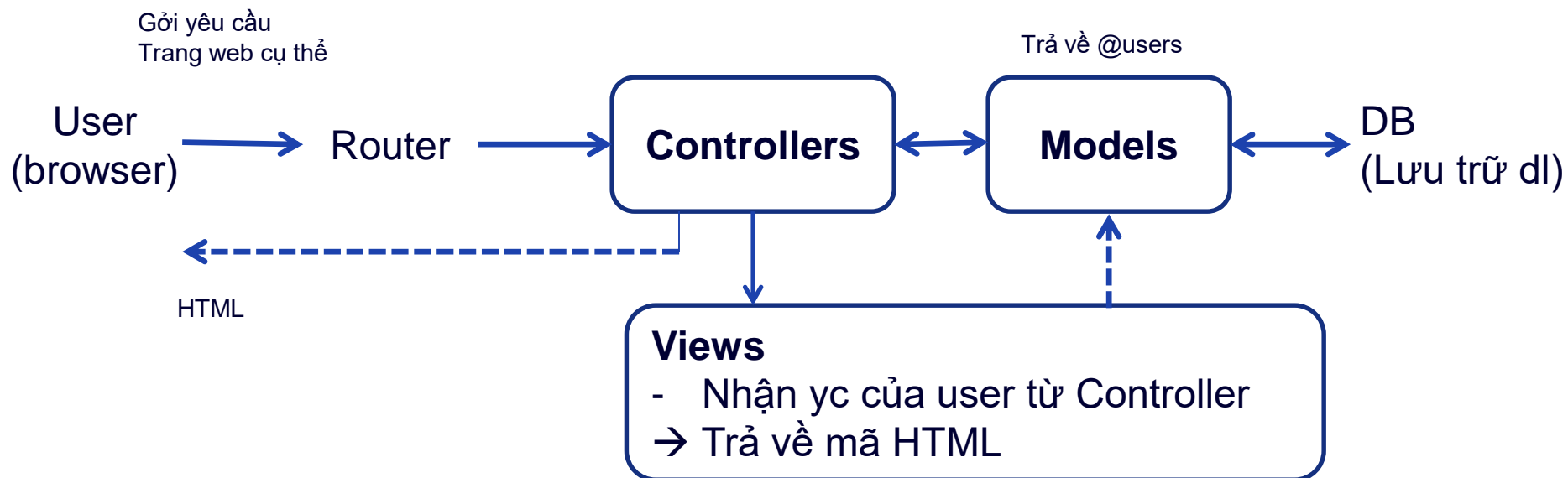
1. 1. Giới thiệu mô hình MVC

Mô hình MVC là một mô hình kiến trúc, theo hướng đối tượng, cho phép developer tách ứng dụng thành 3 phần chính

- **Model:** consist of the files that contain data access code, bussiness logic code and data validation code
- **View:** consist of the files that create the HTML for the user interface and return a response to the user.
- **Controller:** consist of the files that receive requests from the user, get the appropriate data from the model, and provide that data to the View

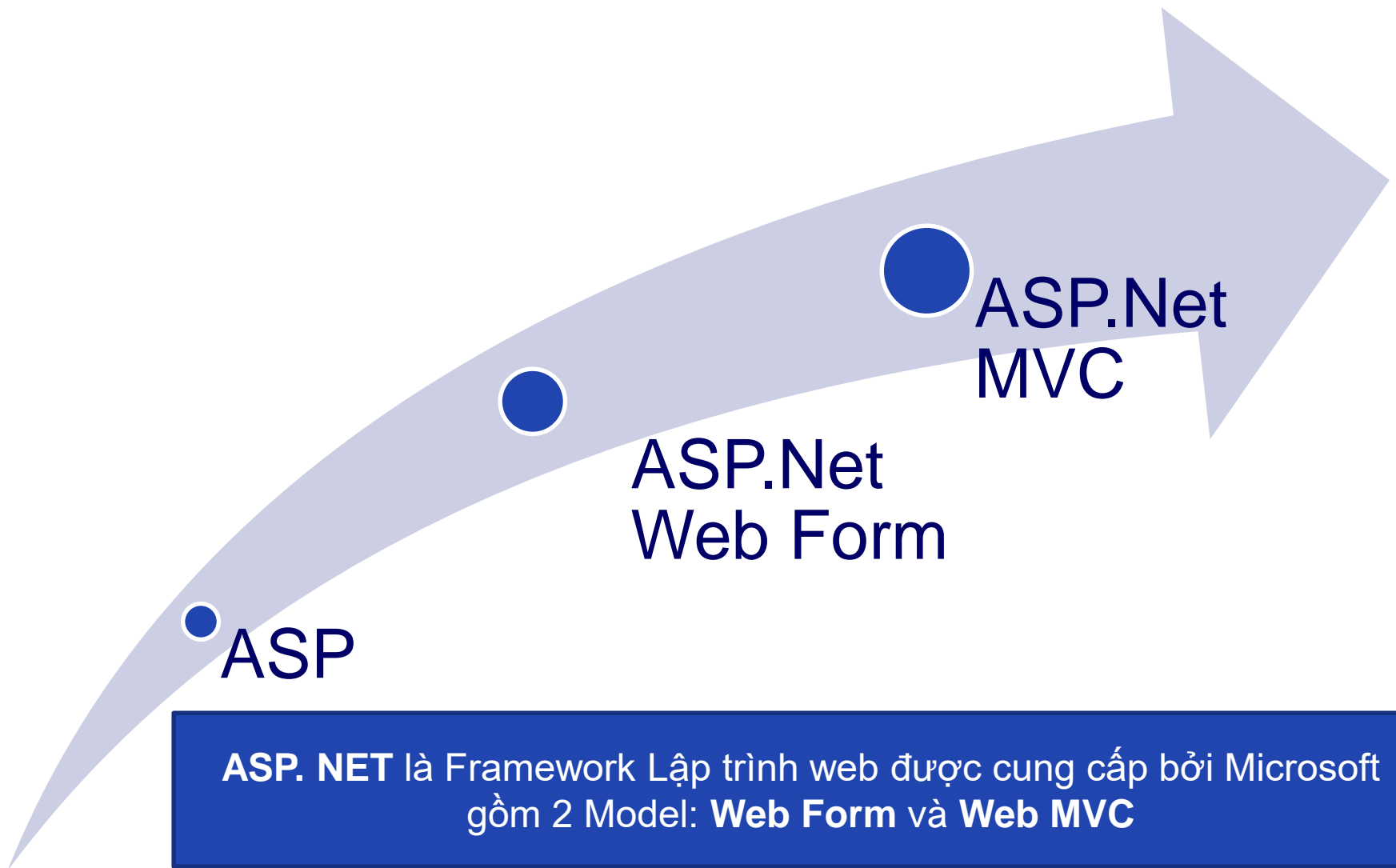
1. 1. Giới thiệu mô hình MVC

Web Forms applications often combine code that accesses databases with HTML code, as with SqlDataSource controls. Also, the code-behind files in Web Forms applications are tightly coupled with their aspx files, which makes it hard for one person to work on an aspx file while another works on its code-behind file. Shortcomings like that can make it difficult to code, test, debug, and maintain large applications. That's why the developers of large websites often use the *MVC (Model-View-Controller) design pattern*.



1. User gửi yêu cầu tới → Router sẽ phân loại và gửi y/c tới **Controller** tương ứng xử lý.
2. **Controller** nhận yêu cầu, xử lý yêu cầu, nếu yc cần truy xuất dữ liệu thì controller sẽ gửi yêu cầu đến tầng **model** để truy xuất dữ liệu.
3. Tầng **Model** sẽ lấy dữ liệu từ Database sau đó truyền dữ liệu qua tầng **View** thông qua tầng **Controller** để tầng **View** hiển thị dữ liệu cho User
4. User sẽ nhìn thấy thông tin hiển thị ở giao diện (tầng **View**)

Mô hình lập trình ASP.Net MVC



1.2. So sánh MVC và WebForm

- Code và (*.aspx và file.cs tương ứng) → khó chỉnh sửa
- Giao diện thiết kế sd hộp toolbox → không linh hoạt (trong thực tế dùng html, css, bootstrap)
- ASP.Net WebForm sd ViewState → tăng kích thước trang → giảm hiệu năng hoạt động

1.2. So sánh MVC và WebForm

Ưu điểm:

- Giúp phát triển phần mềm nhanh, một cách chuyên nghiệp, có thể chia công việc cho nhiều nhóm khác nhau (Designer, Developer, Database,..)
- Giao diện trong ASP.Net MVC sd công nghệ thiết kế web HTML, CSS → việc thiết kế giao diện dễ dàng, linh hoạt
- Dễ dàng thêm các chức năng cho từng lớp
- Dễ nâng cấp, bảo trì (có thể cô lập từng lớp, không làm ảnh hưởng đến cả toàn bộ chương trình)
- Asp.Net MVC không sd ViewState → trang web kg bị tăng kích thước → Hiệu năng hoạt động kg bị giảm

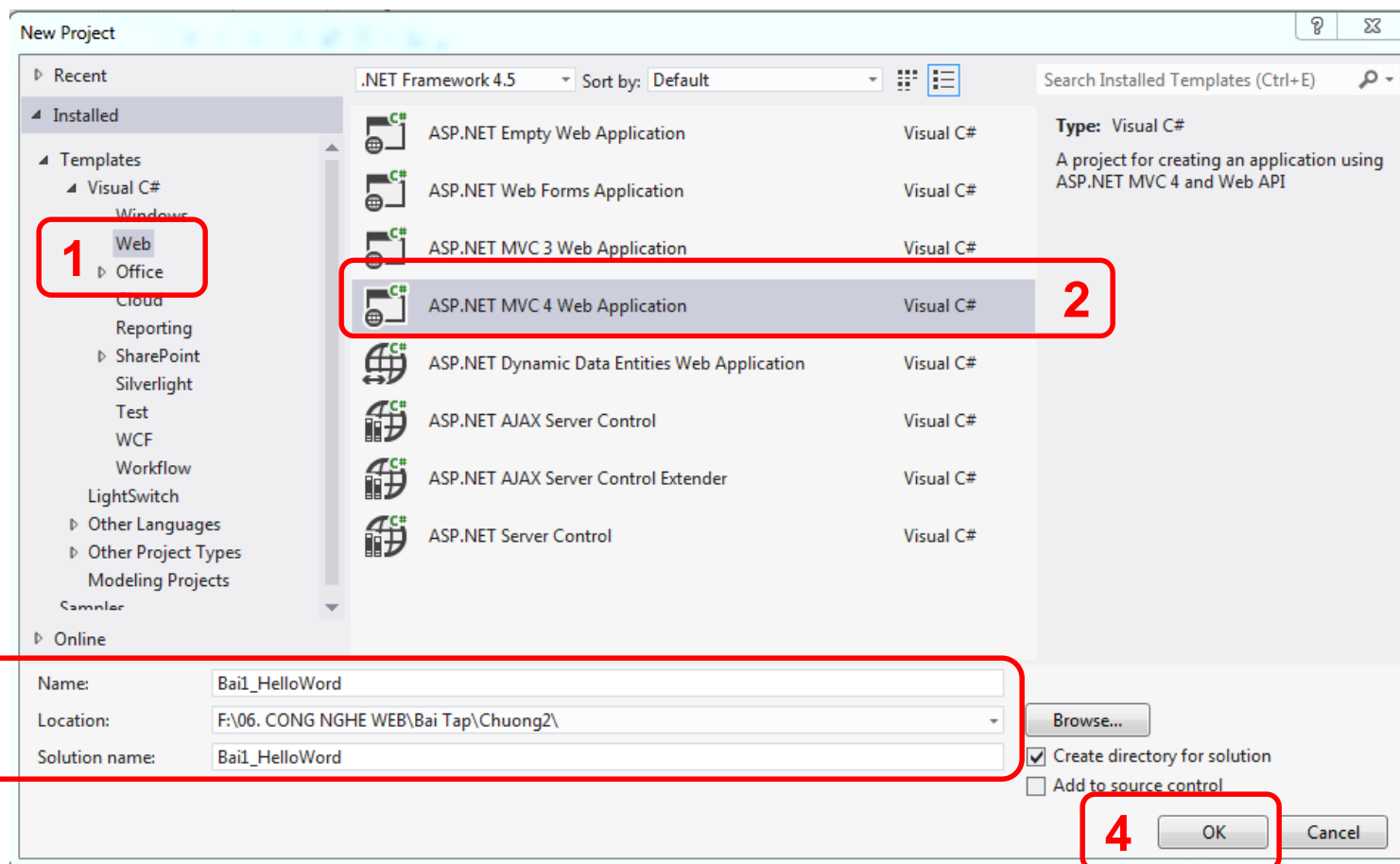
Nhược điểm:

- Thích hợp với dự án lớn, kg phù hợp với dự án nhỏ (sẽ gây công kênh)
- Tồn thời gian xử lý chuyển dữ liệu giữa các lớp, giữa các thành phần `<% %>` `@{}`

ASP.NET WEBFORM	ASP.NET MVC
1. Views tightly coupled to logic	1. View and logic separate
2. Pages	2. Controller (route-base URLs)
3. State management	3. No automatic state management
4. Web Form syntax only	4. Support multi syntaxes (Razor or default)
5. Master pages	5. Layouts
6. User controls	6. Partial Views
7. Server Controls	7. HTML Helper

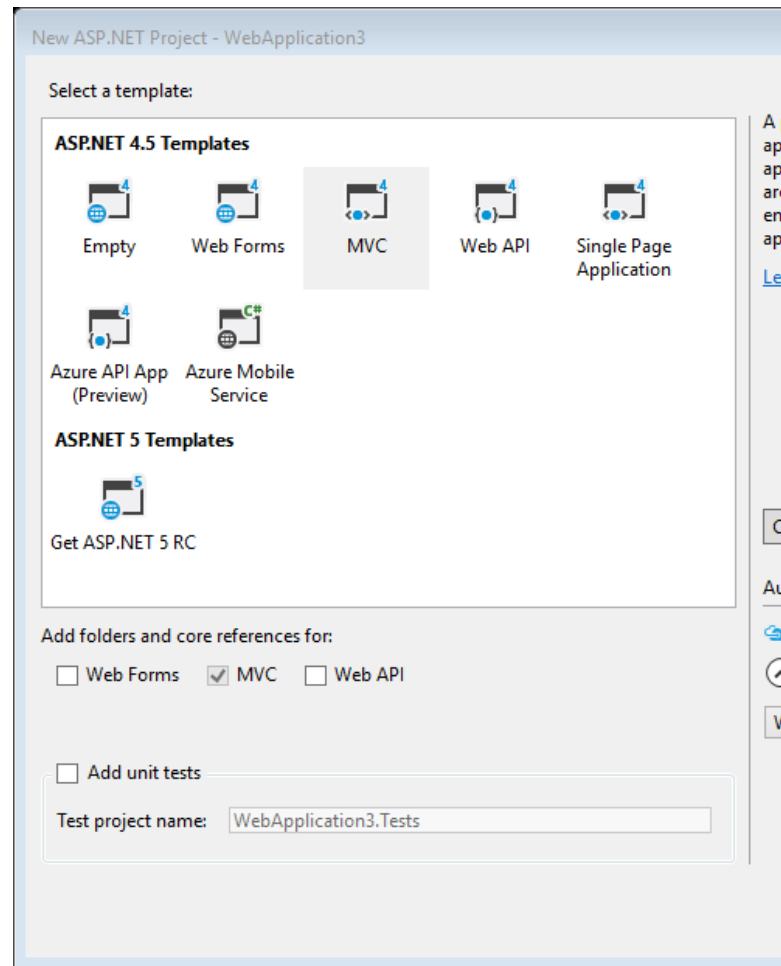
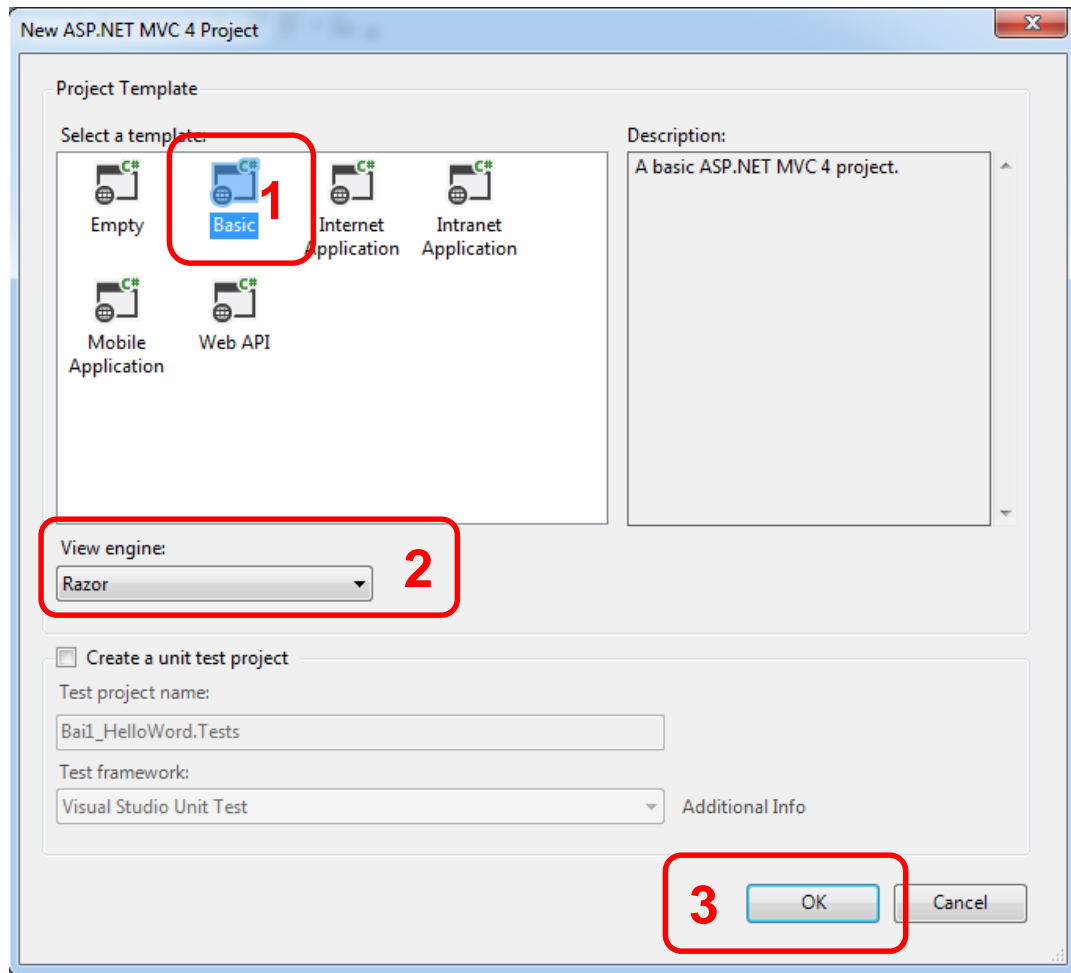
1.3. Tạo Web application với MVC4

- Bước 1:** Khởi động Visual Studio 2012, tạo project mới
File → New → Project ...



1.3. Tạo Web application với MVC4

- Chọn template



1.3. Tạo Web application với MVC4

Giới thiệu Solution Explorer

Controllers	Chứa các file xử lý đóng vai trò điều khiển
Models	Chứa các class
Views	Chứa giao diện (Code html, css)
App_Data	Lưu trữ các tập tin dữ liệu, tập tin XML hoặc Database
App_Start	Chứa các tập tin liên quan đến việc cấu hình cho các tính năng : routers, filters,...
Content	Chứa tập tin CSS, hình ảnh liên quan đến giao diện
Scripts	Chứa các thư viện JavaScript
Packages.config	Chứa tất cả các thư viện sd trong ứng dụng
Web.config	Cấu hình cho website
Global.asax	...

App_Start Chứa các tập tin liên quan đến việc cấu hình cho các tính năng : routers, filters,...

- App_Start
 - BundleConfig.cs
 - FilterConfig.cs
 - IdentityConfig.cs
 - RouteConfig.cs**
 - Startup.Auth.cs
- Content
- Controllers

```
namespace WebApplication3
{
    1 reference
    public class RouteConfig
    {
        1 reference
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index",
                               id = UrlParameter.Optional }
            );
        }
    }
}
```

App_Start

- **RouteConfig** qui định project thực thi là ActionResult **Index** được hiển thị mà không phải là **About** hay **Contact**.
- **Controller** được gọi lên là HomeController, trong đó ActionResult Index được thực thi.

Tạo _ViewHome.cshtml

Add View

View name:
_ViewHome

View engine:
Razor (CSHTML)

☐ Create a strongly-typed view
Model class:
[Empty]

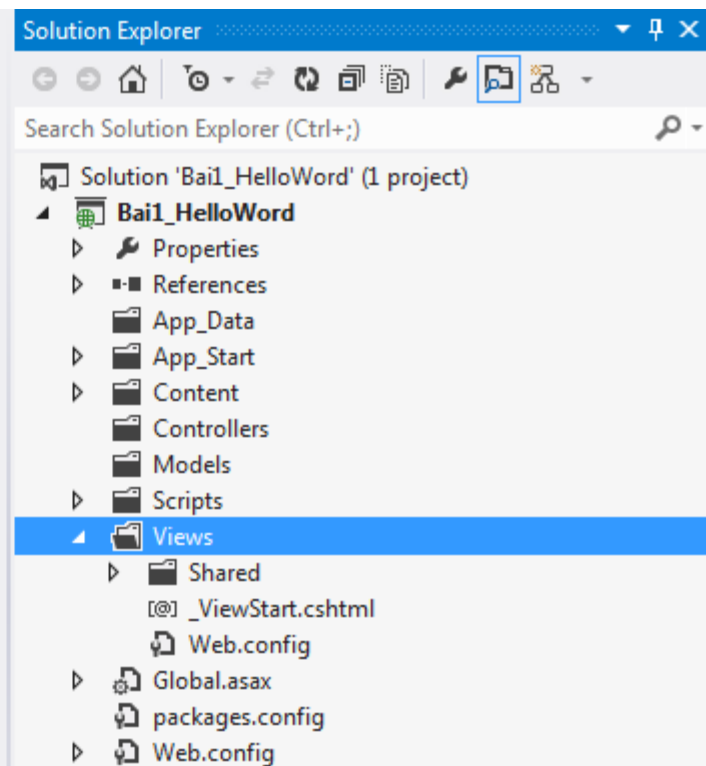
Scaffold template:
Empty ☒ Reference script libraries

☐ Create as a partial view

☐ Use a layout or master page:
[Empty] ...
(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add **Cancel**



_ViewHome.cshtml

```
@{
```

```
// Layout = null;
```

```
ViewBag.Title = "Home Page";
```

```
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta name="viewport" content="width=device-width" />
```

```
<title>_ViewHome</title>
```

```
</head>
```

```
<body>
```

```
<div>
```

```
<h1>This is Home Page </h1>
```

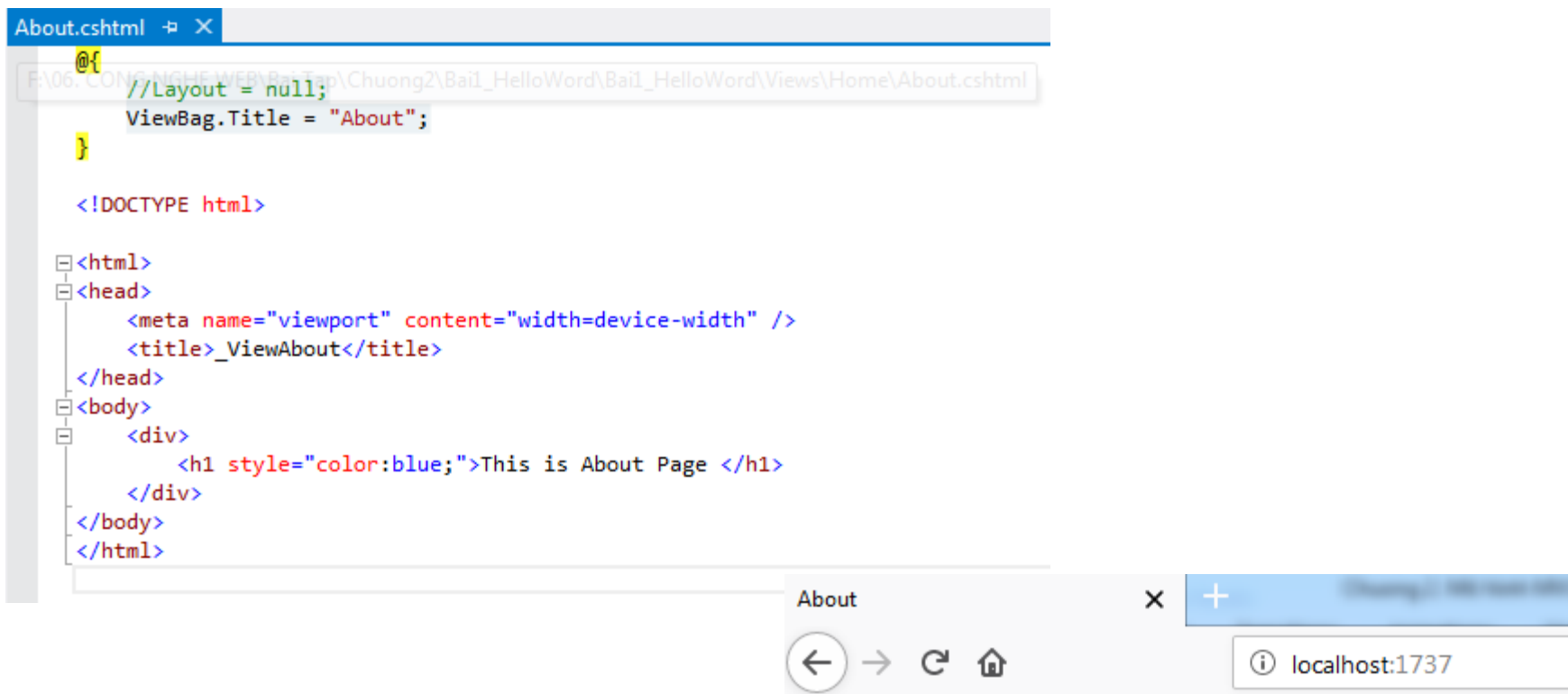
```
Hello Every body in class 08DHTH !
```

```
</div>
```

```
</body>
```

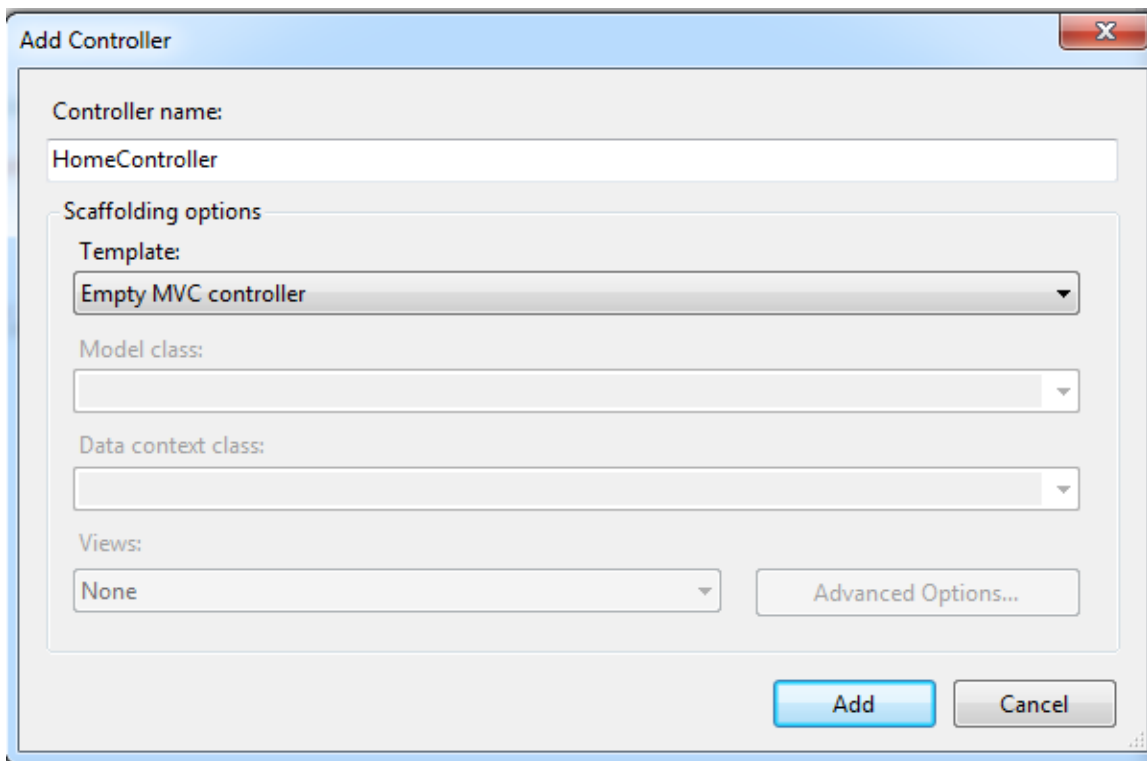
```
</html>
```

Tạo trang About.cshtml



This is About Page

Tạo HomeController.cs



Add Controller

Controller name:
HomeController

Scaffolding options

Template:
Empty MVC controller

Model class:
None

Data context class:
None

Views:
None

Advanced Options...

Add Cancel

- App_Start
 - BundleConfig.cs
 - FilterConfig.cs
 - RouteConfig.cs
 - WebApiConfig.cs
- Content
- Controllers**
- Models
- Scripts
- Views
 - Shared
 - _ViewAbout.cshtml
 - _ViewHome.cshtml
 - _ViewStart.cshtml
 - Web.config
- Global.asax
- packages.config
- Web.config

Điều hướng trang bằng RouteConfig.cs

```
namespace Bai1_HelloWord
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "About", id = UrlParameter.Optional }
            );
        }
    }
}
```

1.3. Tạo Web application với MVC4

Phân biệt Aspx và Razor View Engine

- Giống nhau là: đều được sử dụng để hiển thị giao diện lên web Browser
- Khác :

Aspx	Razor
Sử Dụng MasterPage, UserControl, subView	Sử dụng layoutPage, Partial View, subView
<% Các lệnh C# được viết %>	@ { Các lệnh C# được viết }

1.4. ACTION RESULT

ActionResult là lớp cha của tất cả các lớp Result khác.

Inheritance Hierarchy

System.Object

System.Web.Mvc.ActionResult

System.Web.Mvc.ContentResult

System.Web.Mvc.EmptyResult

System.Web.Mvc.FileResult

System.Web.Mvc.HttpStatusCodeResult

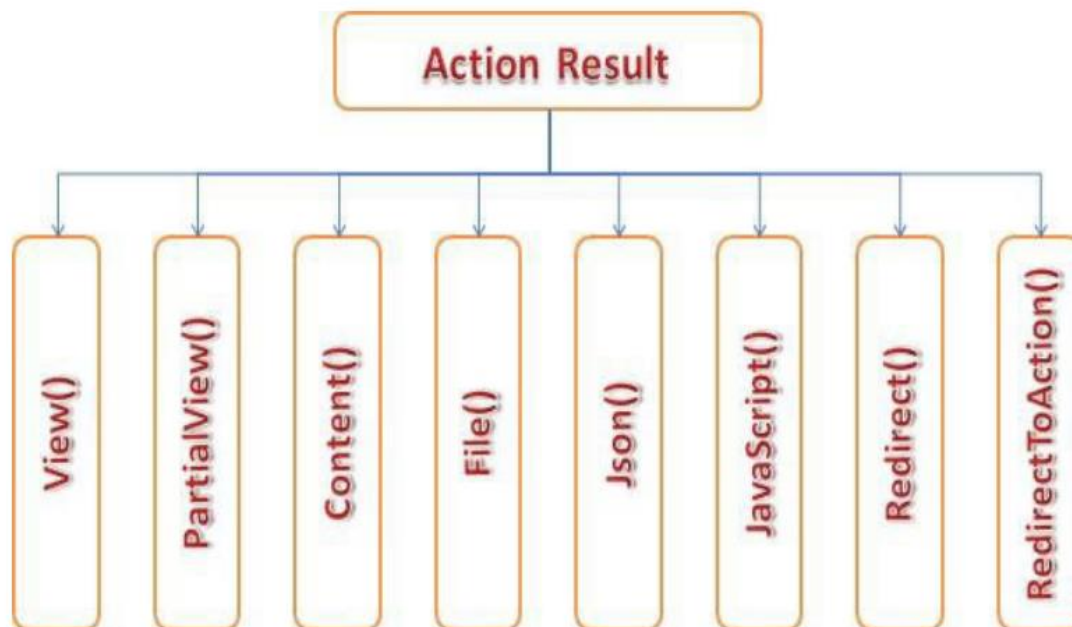
System.Web.Mvc.JavaScriptResult

System.Web.Mvc.JsonResult

System.Web.Mvc.RedirectResult

System.Web.Mvc.RedirectToRouteResult

System.Web.Mvc.ViewResultBase

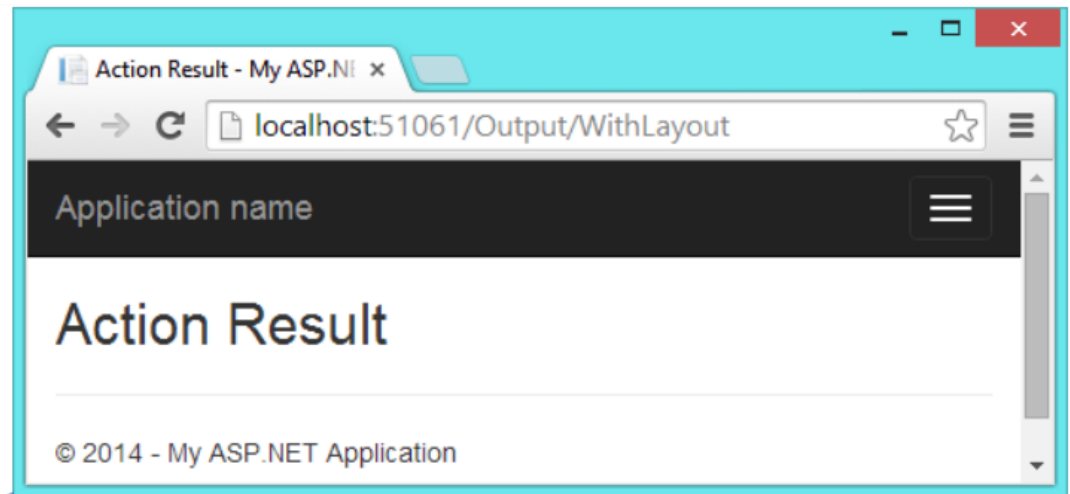


1.4. ACTION RESULT

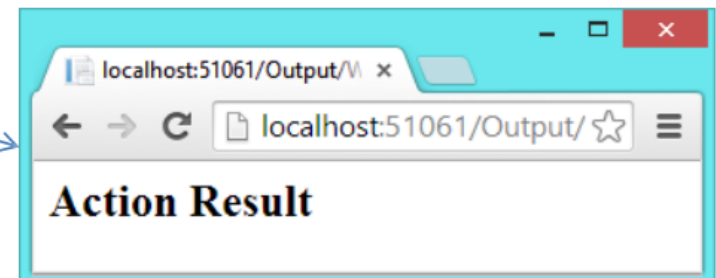
ViewResult trả về view. Sử dụng ViewResult khi ta cần render một View. → dùng ViewResult khi chắc chắn Action đây chỉ trả về View mà không trả về gì khác.

PatialView trả về một phần nhỏ của View. Hay nói cách khác, dùng PartialViewResult khi muốn render một phần nhỏ của trang web mà không phải là toàn bộ View. Giả sử trong View có một phần rất phức tạp cần xử lý riêng biệt, khi đó ta chỉ cần tạo ra một tệp cshtml View khác, một Action với kiểu PartialViewResult tương ứng, xử lý phần công việc đây và đối xử với PartialView như một View bình thường, và nhúng nó vào lại trang web thông qua phương thức :

Html.RenderPartial



```
public ActionResult WithLayout()  
{  
    return View("Index");  
}  
  
public ActionResult WithoutLayout()  
{  
    return PartialView("Index");  
}
```

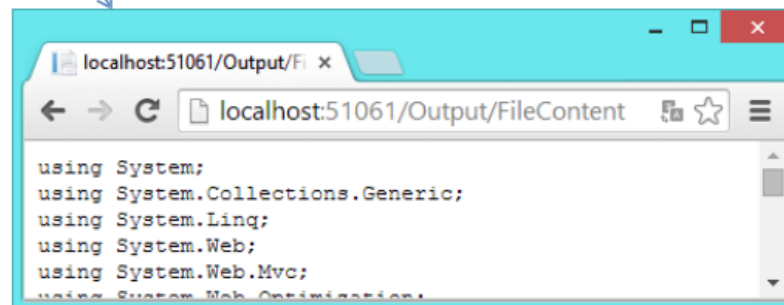
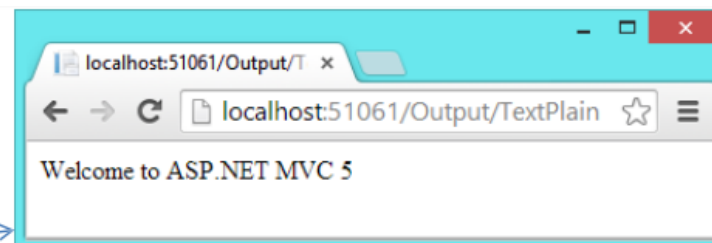


1.4. ACTION RESULT

ContentResult trả về loại phản hồi chỉ chứa một chuỗi ký tự.

FileResult trả về một file tĩnh trong phản hồi

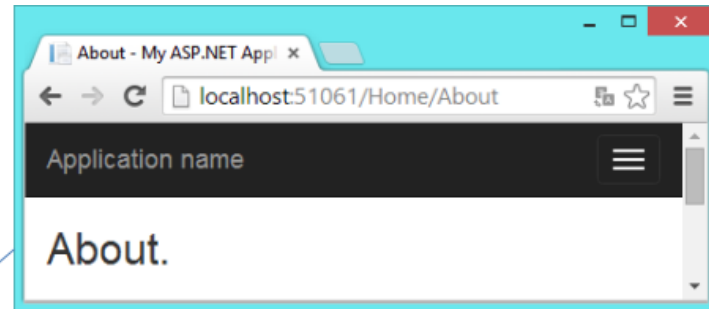
```
public ActionResult TextPlain()  
{  
    return Content("Welcome to ASP.NET MVC 5");  
}  
  
public ActionResult FileContent()  
{  
    return File("~/Global.asax.cs", "text/plain");  
}
```



1.4. ACTION RESULT

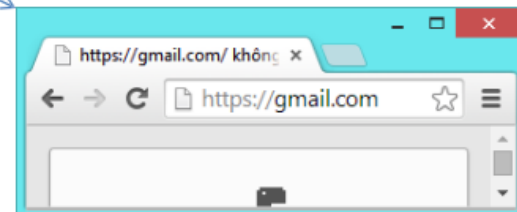
RedirectToAction chuyển sang một action, hoặc một action của một Controller khác.

Redirect chuyển sang một URL mới



```
public ActionResult RedirectToAction()  
{  
    return RedirectToAction("About", "Home");  
}
```

```
public ActionResult RedirectToUrl()  
{  
    return Redirect("http://gmail.com");  
}
```



2. View Data & ViewBag in MVC

ViewData & ViewBag được dùng để chuyển dữ liệu từ Controller đến một View.

Chỉ tồn tại trong một request và giá trị trả về null khi chuyển hướng (redirect)

- ViewData là một tập hợp các đối tượng được lưu trữ và truy xuất sử dụng string as key;

`ViewData["yourkey"] = "SomeData";`

- ViewBag dùng đặc tính động cho phép thêm vào các thuộc tính động cho đối tượng.

`ViewBag.YourProperty = "SomeData";`

2. View Data & ViewBag in MVC

Ví dụ: Sử dụng ViewBag và ViewData chuyển dữ liệu từ controller Index2 sang view Index2.cshtml

```
public ActionResult Index2()
{
    ViewBag.Fruits = new List<string>()
    {
        "Cam", "Xoài", "Cóc", "Ổi"
    };
    return View();
}
```

```
public ActionResult Index2()
{
    /* ViewBag.Fruits = new List<string>()
    {
        "Cam", "Xoài", "Cóc", "Ổi"
    };
    return View(); */

    ViewData["Fruits"] = new List<string>()
    {
        "Cam", "Xoài", "Cóc", "Ổi"
    };
    return View();
}
```

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index2</title>
</head>
<body>
    <div>
        <h3>DANH SÁCH TRÁI CÂY</h3>
        <ul>
            @foreach (string fru in ViewBag.Fruits)
            {
                <li>@fru</li>
            }
        </ul>
    </div>
</body>
```

Index2

localhost:1737

DANH SÁCH TRÁI CÂY

- Cam
- Xoài
- Cóc
- Ổi

```

public ActionResult Index2()
{
    //ViewBag.Fruits = new List<string>()
    //{
    //    "Cam", "Xoài", "Cóc", "Ổi"
    //};
    //return View();

    /*ViewData["Fruits"] = new List<string>
    {
        "Cam", "Xoài", "Cóc", "Ổi"
    };
    return View();*/

    List<string> Fruits = new List<string> { "Cam", "Xoài", "Cóc", "Ổi" };
    ViewData["Fruits"] = Fruits;
    return View();
}

```

```

@{
    var Fruits = ViewData["Fruits"] as List<string>;
}
@if (Fruits != null)
{
    <ul>
        @foreach (var fru in Fruits)
        {
            <li> @fru</li>
        }
    </ul>
}

```

3. Layout Page - RenderBody

Vì sao phải dùng LayoutPage ?

- Để tránh mất thời gian cho người thiết kế View
→ sử dụng layout dùng chung gọi là **LayoutPage**.
- LayoutPage được thiết kế một lần, khi tạo View mới ta chỉ cần kế thừa từ LayoutPage đã tạo → ta sẽ có sẵn một khung sườn.
- Nội dung tương ứng của từng trang sẽ được gọi thông qua **@RenderBody();**

- Để sử dụng các file CSS trong thư viện :

```
<!-- Custom styles for this template -->  
<link href="/Assets/css/agency.min.css" rel="stylesheet">
```

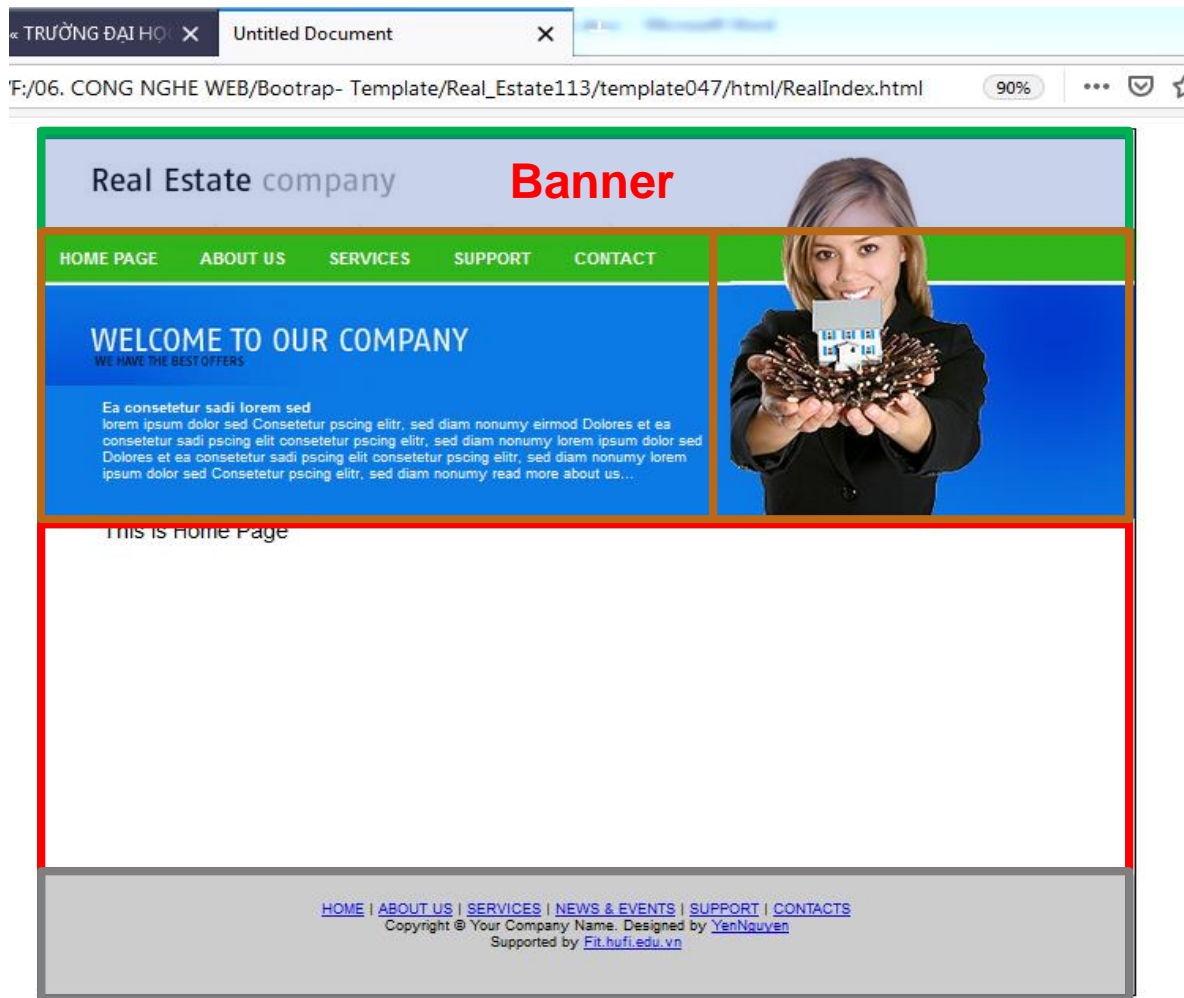
```
<!-- Custom styles for this template -->  
<link href="@Url.Content("/Assets/css/agency.min.css")" rel="stylesheet">
```

- Để sử dụng các file javascript trong thư viện :

```
<!-- Contact form JavaScript -->  
<script src="@Url.Content("/Assets/js/jqBootstrapValidation.js")"></script>  
<script src="@Url.Content("/Assets/js/contact_me.js")"></script>
```

```
<!-- Custom scripts for this template -->  
<script src="@Url.Content("/Assets/js/agency.min.js")"></script>
```

Ví dụ: Tạo một LayoutPage như sau:



wrapper

Header

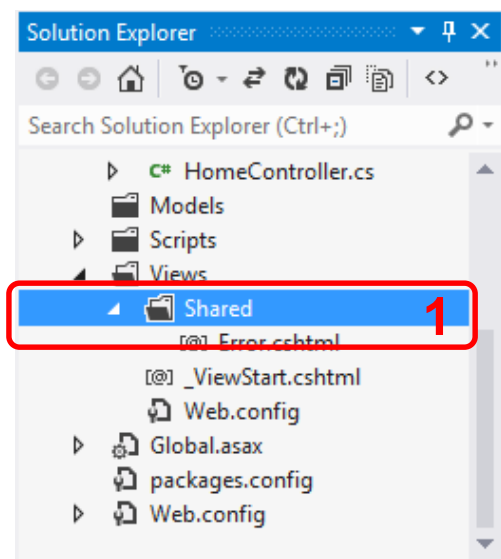
Content

Footer

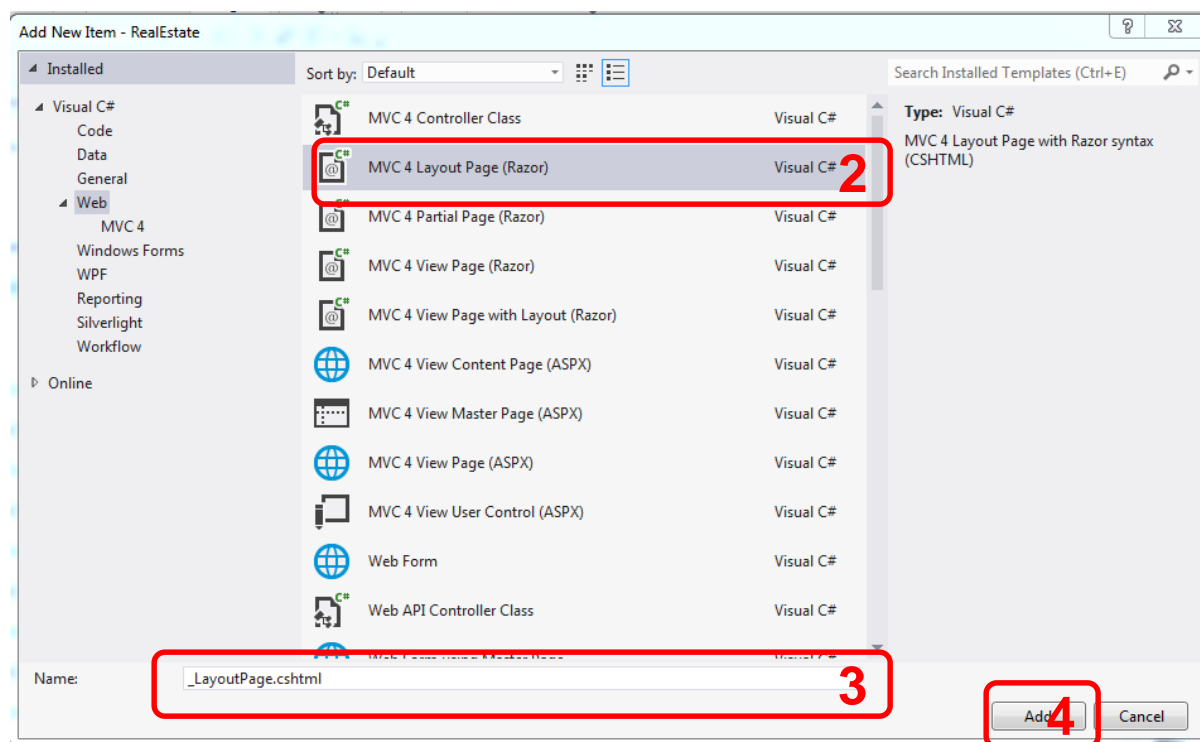
Yêu cầu:

- Tạo Controller home trong đó có các phương thức HomePage, AboutUs, Services, Support, Contact trả về các View tương ứng.
- Tạo các view HomePage.cshtml, ..., Contact.cshtml sử dụng layout dùng chung vừa tạo trong yc 1.
- Sử dụng **html.ActionLink** thực hiện chuyển giữa các View khi người dùng chọn menu tương ứng.

Bước 1: Rclick → Shared → Add → New Item →



Bước 2: Chọn MVC4 LayoutPage



About.cshtml Style1.css Home.cshtml **_LayoutPage.cshtml** HomeController.cs

```

<!DOCTYPE html>

<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <link rel="stylesheet" type="text/css" href="@Url.Content("/Assets/css/Style1.css")"/>
  <title>@ViewBag.Title</title>
</head>
<body>
  <div id="Wrapper">
    <div id="Header">...</div>
    <div id = "Content">
      @RenderBody()<br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br />
    </div>
    <div id="Footer">...</div>
  </div>

</body>
</html>
  
```

4. Html Helper

- **What is Html Helpers ?**

Html Helpers là một phương thức được sử dụng để render ra các nội dung html trong một giao diện View. Html Helper được bổ sung như là các phương thức mở rộng.

VD: để tạo một textbox trong ASP.Net MVC ta có thể dùng một trong 2 cách sau:

<input type = “text” name =“name” Id = “name” value=“Yen”>

@Html.Textbox (“name”) → biến name lưu DL của Textbox

@Html.Textbox (“name”, “Yến”) → biến name có gt = “Yến”

Định dạng style cho TextBox

4. Html Helper

- **The purpose of Html Helpers ?**

We can type the Html Tag , but using Html Helpers will greatly reduce the amount of Html that we have to write in a View. Views should be as simple as possible

- **Some of the Standard Html Helpers ?**

- a. Input Helpers**
- b. Html Form**
- c. Render Helpers**
- d. Liên kết**
- e. Kiểm lỗi**

a. Input Helpers

1. @Html.TextBox

```
Contact.cshtml*  About.cshtml  Style1.css  Home.cshtml  _LayoutPage.cshtml  HomeController

@{
    ViewBag.Title = "Contact";
    Layout = "~/Views/Shared/_LayoutPage.cshtml";
}

<h2>Please Contact us</h2>
@Html.TextBox("UserName", "yennh",
    new {
        Styles="background-color:grey;font-weight:bold;",
        title ="Enter your User Name, please ! "
    })
```

Please Contact us

yennh

2. @Html.Label

```
@Html.Label("UserName", "User Name")
```

Please Contact us

User Name

3. @Html.Password

```
@Html.Label("Password", "Password")  
@Html.Password("Password", "123456")
```

Password

4. @Html.TextArea

```
@Html.TextArea("Comments", "", 5, 15, null)
```

Please Contact us

User Name

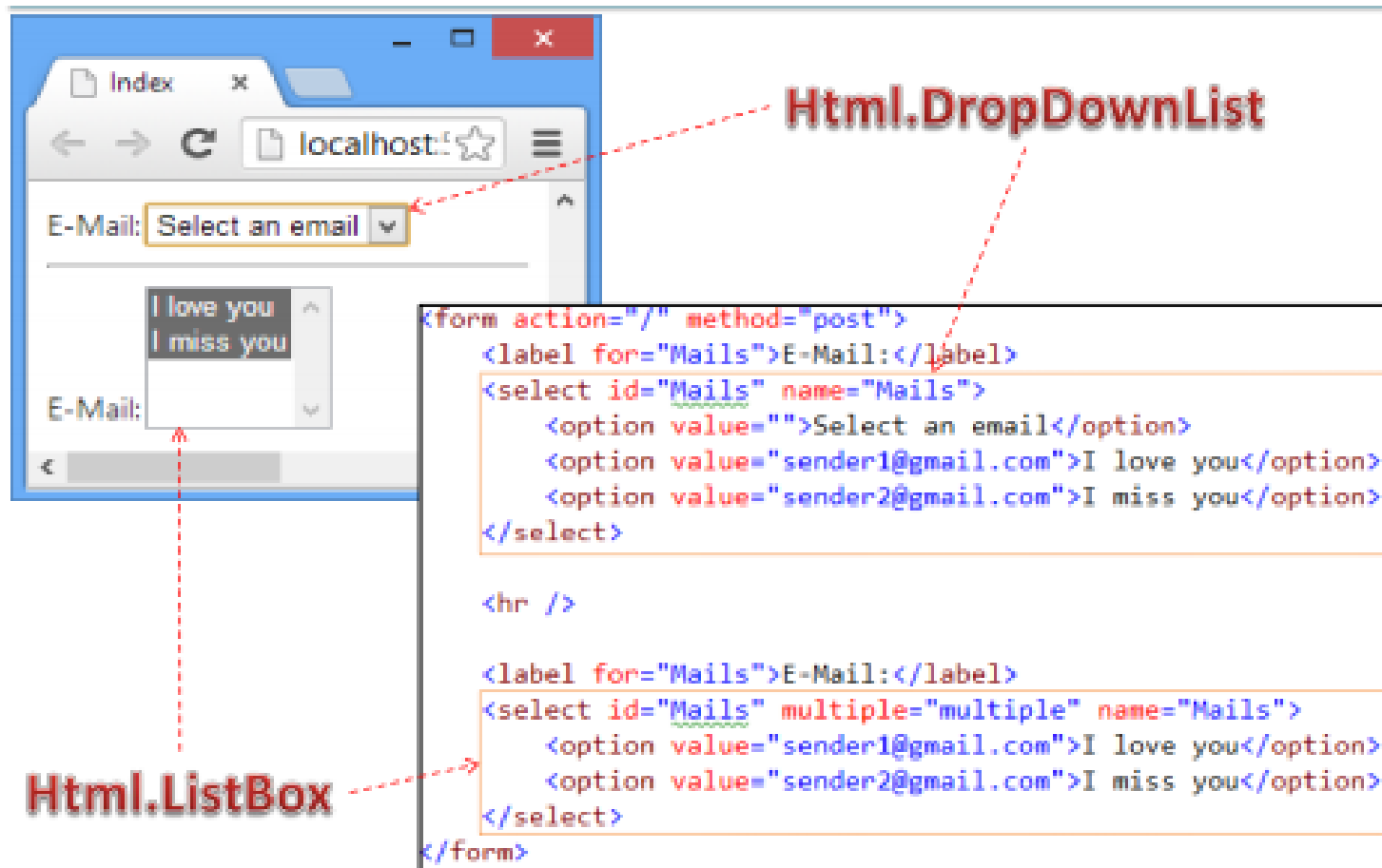
Password

5. @Html.DropDownList

```
List<Mail> Mails = new List<Mail>{  
    new Mail {  
        To = "sender1@gmail.com",  
        Subject = "I love you"  
    },  
    new Mail {  
        To = "sender2@gmail.com",  
        Subject = "I miss you"  
    }  
};  
ViewBag.Mails = new SelectList(Mails, "To", "Subject");
```

```
@using (Html.BeginForm()){  
    @Html.Label("Mails", "E-Mail:");  
    @Html.DropDownList("Mails", "Select an email")  
    <hr />  
    @Html.Label("Mails", "E-Mail:");  
    @Html.ListBox("Mails")  
}
```


5. @Html. DropDownList



The image shows a web browser window with two email selection controls. The top control is a single-select dropdown menu labeled "E-Mail:" with the text "Select an email" and a downward arrow. The bottom control is a multi-select list box labeled "E-Mail:" with two visible options: "I love you" and "I miss you".

Red dashed arrows point from the controls to their respective code snippets:

- The top arrow points from the single-select dropdown to the `Html.DropDownList` code snippet.
- The bottom arrow points from the multi-select list box to the `Html.ListBox` code snippet.

Html.DropDownList

```
<form action="/" method="post">
  <label for="Mails">E-Mail:</label>
  <select id="Mails" name="Mails">
    <option value="">Select an email</option>
    <option value="sender1@gmail.com">I love you</option>
    <option value="sender2@gmail.com">I miss you</option>
  </select>

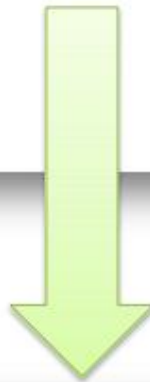
  <hr />

  <label for="Mails">E-Mail:</label>
  <select id="Mails" multiple="multiple" name="Mails">
    <option value="sender1@gmail.com">I love you</option>
    <option value="sender2@gmail.com">I miss you</option>
  </select>
</form>
```

Html.ListBox

b. Html Form

```
@using (Html.BeginForm("Register", "Member")) {  
    ... nội dung form ...  
}
```



```
<form action="/Member/Register" method="post">  
    ... nội dung ...  
</form>
```

b. Html Form

Duy trì các trường trong Form

Helper	HTML
@Html.BeginForm()	Sinh thẻ <form> bắt đầu
@Html.EndForm()	Sinh thẻ </form> kết thúc
@Html.CheckBox()	Sinh thẻ <input type="checkbox" >
@Html.Hidden()	Sinh thẻ <input type="hidden" >
@Html.Password()	Sinh thẻ <input type="password" >
@Html.RadioButton()	Sinh thẻ <input type="radio" >
@Html.TextArea()	Sinh thẻ <textarea></textarea>
@Html.TextBox()	Sinh thẻ <input type="text" >
@Html.DropDownList()	Sinh thẻ <select><option></select>
@Html.ListBox()	Sinh thẻ <select multiple><option></select>

b. Html Form

Ví dụ

Full Name	<code>@{Html.BeginForm("Action", "Controller");}</code>
<input type="text"/>	<code><div>Full Name</div></code>
	<code>@Html.TextBox("FullName")</code>
Password	<code><div>Password</div></code>
<input type="password"/>	<code>@Html.Password("Password")</code>
Photo	<code><div>Photo</div></code>
<input type="button" value="Chọn tệp"/> Không có tệp	<code><input name="Photo" type="file" /></code>
Married Status	<code><div>Married Status</div></code>
<input type="checkbox"/> Single	<code><label>@Html.CheckBox("Status") Single</label></code>
Gender	<code><div>Gender</div></code>
<input type="radio"/> Male <input type="radio"/> Female	<code><label>@Html.RadioButton("Gender", true) Male</label></code>
	<code><label>@Html.RadioButton("Gender", false) Female</label></code>
Description	<code><div>Description</div></code>
<input type="text"/>	<code>@Html.TextArea("Description")</code>
	<code>@Html.Hidden("Active")</code>
	<code><hr /></code>
<input type="button" value="Submit"/>	<code><input type="submit" value="Submit" /></code>
	<code>@{Html.EndForm();}</code>

c. Định dạng

Helper	Mô tả
@Html.FormatValue (value, format)	Định dạng một giá trị số, chuỗi hoặc thời gian
@String.Format(format, value1, value2...)	Định dạng nhiều giá trị hỗn hợp
@Html.Raw (html)	Giải mã chuỗi đã mã hóa HTML

- Số bình thường: 12345.8765
- Phân nhóm: 12,345.877
- Tiền tệ: \$12,345.88
- Phần trăm: 72.00 %

- Ngày bình thường: 5/27/2014 9:26:09 PM
- Định dạng D: Tuesday, May 27, 2014
- Định dạng ISO: 2014-05-27
- Định dạng English: 05/27/2014
- Định dạng 24 giờ: 21:26:09
- Định dạng 12 giờ: 09:26:09 PM

- Có mã hóa HTML: Hello
- Không mã hóa HTML: **Hello**

c. Định dạng

Định dạng số

Ký hiệu	Mô tả
{0:C}	Currency – tiền tệ theo ngôn ngữ
{0:P}	Percent – số phần trăm
{0:#,###.##0}	Number – số phân nhóm và 3 số lẻ

```
@{  
    var number1 = 12345.8765;  
    var number2 = 0.72;  
}  
<ul>  
    <li>Số bình thường: @number1</li>  
    <li>Phân nhóm: @Html.FormatValue(number1, "{0:#,###.##0}")</li>  
    <li>Tiền tệ: @Html.FormatValue(number1, "{0:c}")</li>  
    <li>Phần trăm: @Html.FormatValue(number2, "{0:p}")</li>  
</ul>
```



- Số bình thường: 12345.8765
- Phân nhóm: 12,345.877
- Tiền tệ: \$12,345.88
- Phần trăm: 72.00 %

c. Định dạng

Định dạng thời gian

Ký hiệu	Mô tả
{0:D}	Date – theo ngôn ngữ được chọn
{0:MMMM-dd-yyyy hh:mm:ss tt}	<ul style="list-style-type: none">✓ M,MM,MMM,MMMM: tháng 1, 2 ký tự số, 3 ký tự viết tắt, tên tháng đầy đủ✓ d,dd: ngày 1, 2 ký tự✓ yy,yyyy: năm 2, 4 ký tự số✓ H, HH, h, hh: 1,2 ký tự giờ 24 hoặc 12 giờ mỗi ngày✓ m,mm: 1,2 ký tự số phút✓ s,ss: 1,2 ký tự số giây✓ tt: 2 ký tự sáng/chiều

- Ngày bình thường: 5/27/2014 9:26:09 PM
- Định dạng D: Tuesday, May 27, 2014
- Định dạng ISO: 2014-05-27
- Định dạng English: 05/27/2014
- Định dạng 24 giờ: 21:26:09
- Định dạng 12 giờ: 09:26:09 PM

```
@{  
    var now = DateTime.Now;  
}  
<ul>  
    <li>Ngày bình thường: @now</li>  
    <li>Định dạng D: @Html.FormatValue(now, "{0:D}")</li>  
    <li>Định dạng ISO: @Html.FormatValue(now, "{0:yyyy-MM-dd}")</li>  
    <li>Định dạng English: @Html.FormatValue(now, "{0:MM/dd/yyyy}")</li>  
    <li>Định dạng 24 giờ: @Html.FormatValue(now, "{0:HH:mm:ss}")</li>  
    <li>Định dạng 12 giờ: @Html.FormatValue(now, "{0:hh:mm:ss tt}")</li>  
</ul>
```

c. Định dạng

Mã hóa Html

- Có mã hóa HTML: `Hello`
- Không mã hóa HTML : **Hello**

```
@{  
    var chuoi = "<strong>Hello</strong>";  
}  
<ul>  
    <li>Có mã hóa HTML: @chuoi</li>  
    <li>Không mã hóa HTML : @Html.Raw(chuoi)</li>  
</ul>
```


d. Liên kết

@Html.ActionLink() : được sử dụng để sinh liên kết

```
@Html.ActionLink("Giới thiệu", "About" )  
<a href="/Home/About">Giới thiệu</a>
```

@Html.ActionLink() : nhận một số các tham số

linkText – nhãn của liên kết

actionName – tên action

routeValues – tập các giá trị truyền đến action.

controllerName – tên controller

htmlAttributes – tập thuộc tính HTML của thẻ <a>

```
@Html.ActionLink("Edit Record", "Edit", new {Id=3})  
<a href="/Store/Edit/3">Edit Record</a>
```

Liên kết chứa ảnh

```
<a href="@Url.Action("Delete")">  
    </a>
```

Trong trường hợp muốn sử dụng thêm các thuộc tính của html (style, title, bootstrap,...), sử dụng như sau

```
@Html.EditorFor(model => model.FullName, new
{
    @class = "form-control",
    style = "background-color: red; color:white; font-weight:bold;",
    title = "Please Enter Your FullName !"
})

<div class="form-group">
    <label>Selects an course </label>
    @Html.DropDownListFor(model => model.SelectCourse, new List<SelectListItem>
    {
        new SelectListItem { Text = "1. Html CSS Bootstrap", Value = "1", Selected = true },
        new SelectListItem { Text = "2. JavaScript", Value = "2"},
        new SelectListItem { Text = "3. WebForm", Value = "3"},
        new SelectListItem { Text = "4. ASP.Net MVC", Value = "4"},
        new SelectListItem { Text = "5. Web Services", Value = "5"}
    }, "Select an course")
    @Html.ValidationMessageFor(model => model.SelectCourse)
</div>
```

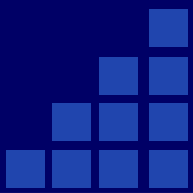
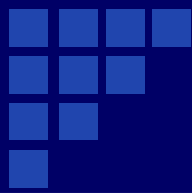
e. DisplayTemplated

@Html.Display(EmployeeData) used with a view that is not strongly typed.

Ex. If you have stored data in ViewData, then we can use this template helper using the key that was used to store data in ViewData.

@Html.DisplayFor(model=>model) used with strongly typed views. If your model have properties that return complex objects, then this template helper is very useful.

@Html.DisplayForModel() used with strongly typed views. Walks thru each property, in the model to display the object



e. Kiểm lỗi

Kiểm tra tính hợp lệ model

Kiểm lỗi bằng tay phía server

Định nghĩa Anotation kiểm lỗi tùy biến

Kiểm lỗi bằng tay phía Client với jQuery

e. Kiểm lỗi

Kiểm tra tính hợp lệ model

Kiểm lỗi bằng tay phía server

Định nghĩa Anotation kiểm lỗi tùy biến

Kiểm lỗi bằng tay phía Client với jQuery


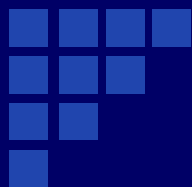
5. HTTP Post / HTTP Get

Được sử dụng trên các function của Controller

Attribute [HttpGet]: Các function mà trả về dữ liệu và được gọi như page load. → mặc định là get (gọi view lần đầu)

Attribute [HttpPost]: Các function mà dữ liệu trả về sau khi form hoặc ajax gọi phương thức Post. Nếu không dùng cách này, có thể dùng phương thức IsPost để xác định khi nào thì phương thức Post được gọi.

Sự khác biệt chính giữa GET và POST request là: với HTTP GET, tất cả các tham số hay dữ liệu được chuyển đến server được chứa trong chính địa chỉ URL. Điều này có nghĩa là ta có thể gọi trực tiếp các thủ tục của server từ xa thông qua URL và các tham số của nó. Tuy nhiên, để chuyển các tham số, ta bị giới hạn bằng định dạng văn bản đơn giản với độ dài các tham số bị giới hạn bởi kích thước lớn nhất của chiều dài dòng request của máy chủ Web. Ví dụ, trên server Web Tomcat, kích thước tối đa mặc định của dòng request được đặt là 8190 bytes



HTTP POST thích hợp hơn đối với việc truyền lượng dữ liệu lớn hay dữ liệu nhị phân, bởi vì dữ liệu được gửi đến server độc lập với URL. Việc này có ưu điểm là lượng dữ liệu không bị hạn chế, như trong trường hợp của phương thức GET.

Q & A