


CÔNG NGHỆ WEB

VIEWS

NỘI DUNG

- Giới thiệu
- Một số quy tắc trong view
- Một số kiểu view
- Tạo view
- Razor Engine
- HTML Helpers
- Dynamic content

GIỚI THIỆU VIEWS

- Views cung cấp giao diện (UI) cho người dùng.
- Khác với một số Framwork khác như ASP.NET Web Forms và PHP, views không tự truy cập trực tiếp mà view được rendered bởi controller.
- Controller thường cung cấp thông tin cho view bằng cách truyền “data transfer object” gọi là model. View chuyển model này thành định dạng biểu diễn được cho người dùng.

MỘT SỐ QUY TẮC TRONG VIEWS

- Trong controller chỉ cần gọi return View() để render the *view* mà không cần chỉ định tên view? Folder View được cấu tạo:
  - Mỗi controller có một thư mục trùng tên trong thư mục view /Views/ControllerName (Không chứa hậu tố controller).
  - Mỗi controller folder trên chứa một view file cho mỗi action method có tên trùng với tên action method.

MỘT SỐ QUY TẮC TRONG VIEWS

- Ta có thể "override" các quy tắc này bằng cách chỉ định tên view muốn được render.
  - Chỉ định view khác nhưng trong cùng một thư mục:

```
public ActionResult Index(){
    return View("NotIndex");
}
```

  - Chỉ định View khác khác cả thư mục:

```
public ActionResult Index(){
    return
        View("~/Views/Example/Index.cshtml");
}
```

MỘT SỐ QUY TẮC TRONG VIEWS

- Ta có thể "override" các quy tắc này bằng cách chỉ định tên view muốn được render.
  - Chỉ định view khác nhưng trong cùng một thư mục:

```
public ActionResult Index(){
    return View("NotIndex");
}
```

  - Chỉ định View khác khác cả thư mục:

```
public ActionResult Index(){
    return
        View("~/Views/Example/Index.cshtml");
}
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Dùng ViewBag chứa Album trong controller:

```
public ActionResult List(){
    var albums = new List<Album>();
    for (int i = 0; i < 10; i++){
        albums.Add(new Album {Title = "Product " + i });
    }
    ViewBag.Albums = albums;
    return View();


}
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Giả sử ta có ViewBag chứa Album trong controller:

Biểu diễn trên View:

```
@using MyMvcProject.Models
<h2>List</h2>
<ul>
@foreach (Album a in (ViewBag.Albums as
IEnumerable<Album>)) {
      <li>@a.Title</li>
}
</ul>
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Giả sử ta có ViewBag chứa Album trong controller:

Hoặc:

```
@using MyMvcProject.Models
<h2>List</h2>
<ul>
@foreach (dynamic p in ViewBag.Albums) {
 <li>@p.Title</li>
}
</ul>
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Dùng model:

```
public ActionResult List(){
    var albums = new List<Album>();
    for (int i = 0; i < 10; i++){
        albums.Add(new Album{Title= "Product" + i });
        }
        return View(albums);
    }
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Dùng model:

```
@using MyMvcProject.Models
@model IEnumerable<Album>
<ul>
@foreach (Album p in Model) {
 <li>@p.Title</li>
}
</ul>
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Có thể khai báo namespace trong tệp web config trong view:

```
<pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Routing" />
      <add namespace="MyMvcProject.Models" />
    </namespaces>
  </pages>
```

MỘT SỐ KIỂU VIEWS

- ViewBag Falls Short:
  - Có thể khai báo namespace trong tệp web config

```
@model IEnumerable<Album>
<ul>
@foreach (Album p in Model) {
 <li>@p.Title</li>
}
</ul>
```

MỘT SỐ KIỂU VIEWS

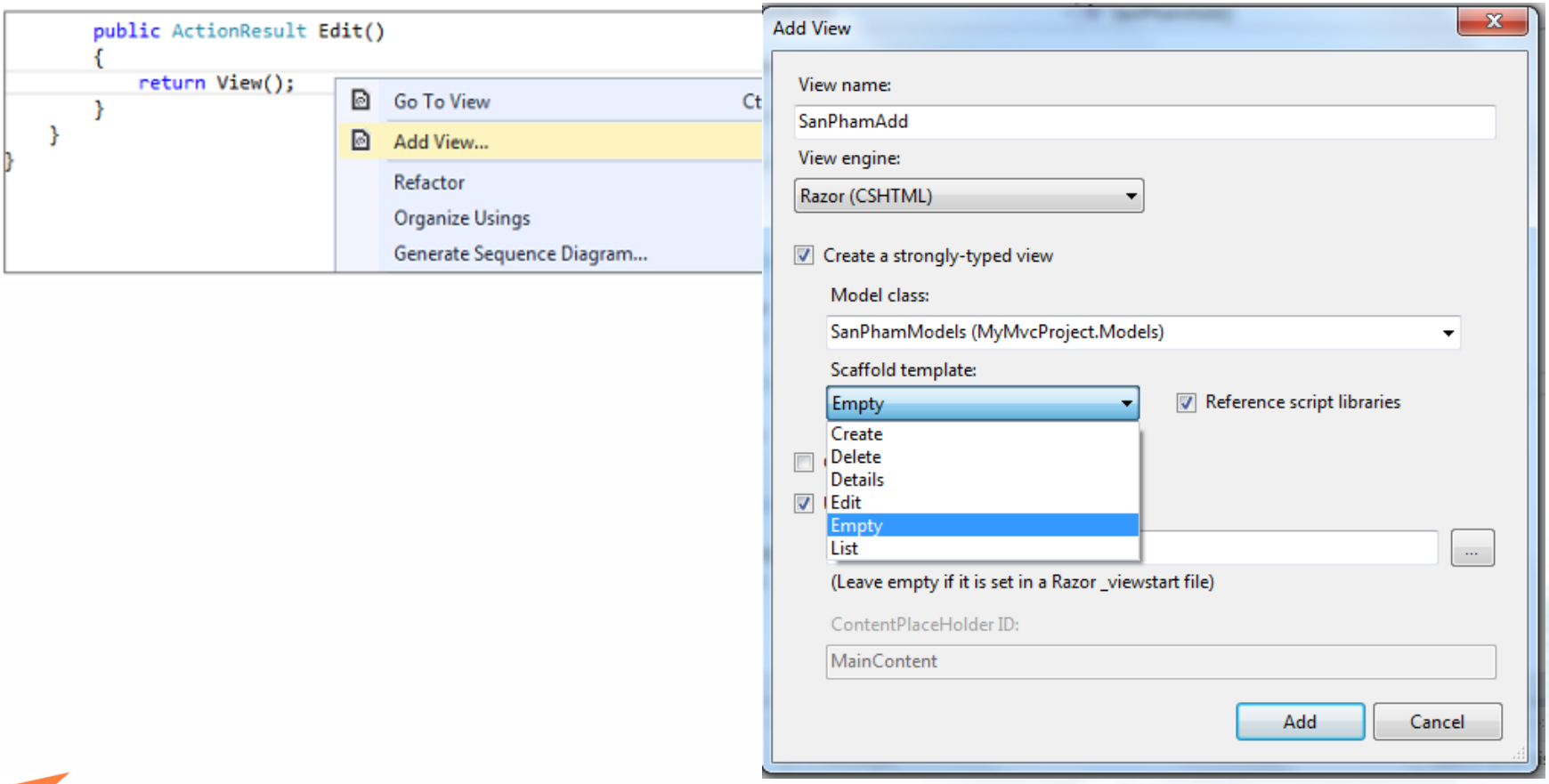
- ViewBag, ViewData, và ViewDataDictionary:
  - Giống nhau giữa ViewBag & ViewData:
    - Giúp duy trì data khi chuyển từ controller đến view.
    - Sử dụng truyền data từ controller đến view tương ứng.
    - "Short life" nghĩa là giá trị thành null khi redirection xuất hiện.

MỘT SỐ KIỂU VIEWS

- ViewBag, ViewData, và ViewDataDictionary:
  - Khác nhau ViewBag & ViewData:
    - ViewData là dictionary của các đối tượng được dẫn xuất từ class ViewDataDictionary và được truy cập bằng cách sử dụng strings như là keys.
    - ViewBag là một dynamic property và là ưu điểm của các đặc tính động mới trong C# 4.0.
    - ViewData yêu cầu chuyển đổi kiểu dữ liệu (typecasting) và kiểm tra null values để tránh lỗi.
    - ViewBag không yêu cầu typecasting cho complex data type.

TẠO VIEWS

- Right click vào Action method và chọn AddView:



TẠO VIEWS

- Right click vào Action method và chọn AddView:
  - View Name: Đặt tên view, thường trùng tên Action
  - View Engine: Chọn mã code Razor hoặc ASPX
  - Chọn Create a strongly typed view:
    - Chọn Model class
    - Chọn scaffold template, các kiểu cho trong bảng sau:

SCAFFOLD	Mô tả
Create	Tạo view với form cho việc sinh thể hiện mới của model. Sinh ra các một label và input field cho mỗi thuộc tính của model type.
Delete	Tạo view với form cho việc xóa các thể hiện của model. Hiển thị label và current value mỗi thuộc tính của model.

TẠO VIEWS

- Right click vào Action method và chọn AddView:

SCAFFOLD	Mô tả
Details	Tạo view hiển thị một label và giá trị cho mỗi thuộc tính của model type.
Edit	Tạo view với form cho việc sửa các thể hiện của model. Sinh một label và input field cho mỗi thuộc tính của model type.
Empty	Tạo empty view. Chỉ model type được chỉ định sử dụng cú pháp @model

TẠO VIEWS

- Right click vào Action method và chọn AddView:
  - Create view as Partial View: Tạo view không đầy đủ, view là một thành phần của view khác.
  - Uses a layout or master pages: Chọn Layout hoặc Master pape cho view.
- Layout/Master Page:
  - Là thành phần giao diện dùng chung cho nhiều View.
  - Layout Page thường dùng cho Razor engine
  - Master Pape dùng cho ASPX engine.

- Partial View:
  - Là view cho phép sử dụng lại trong web application giống như user control.
  - Partial view giống như view có đuôi mở rộng trong Razor Engine là *.cshtml*.
  - Class Helper cung cấp một số phương thức cho view gọi partial view:
    - Html.Partial()
    - Html.RenderPartial()

- Partial View:
  - Ví dụ: file _PartialView.chtml

```
@model IEnumerable<MyMvcProject.Models.SanPhamModels>
@if(Model!=null)
{   <table>
        <tr> <th> Mã Sản phẩm </th>
            <th>Tên Sản phẩm </th>
            <th> Giá sản phẩm</th> </tr>
    @foreach (var item in Model) {
        <tr><td>  @Html.DisplayFor(modelItem => item.MaSP) </td>
            <td>  @Html.DisplayFor(modelItem => item.TenSP) </td>
            <td>  @Html.DisplayFor(modelItem => item.GiaSP)</td></tr>
    }
    </table>
}
```

- Partial View:
  - Ví dụ: file _UsePartialView.chtml

```
@{ Layout = null;}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>UserPartialView</title>
</head>
<body>
    <div>
        @Html.Partial("_PartialView")
    </div>
</body>
</html>}
```

TẠO VIEWS

- Partial View:
  - Trong Controller ta có thể thay thế `return View()` thành `return PartialView()`

```
public ActionResult PartialViewExaple(){
     return PartialView(_sanpham._listSanPham);
}
```

    - return View(); //Trả về trang HTML đầy đủ, render ra normal view cũng như Partial view.
    - return PartialView(); //Trả về một phần của HTML có thể được gọi thông qua AJAX requests.

TẠO VIEWS

- ViewStart:
  - Trong mỗi view để chỉ định Layout ta sử dụng thuộc tính Layout:

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

  - Ta có thể sử dụng _ViewStart.cshtml để thay thế cho chỉ định này.

TẠO VIEWS

- ViewStart:
  - Code của _ViewStart.cshtml được thực thi trước tất cả các view nằm trong cùng thư mục. File này cũng áp dụng với tất cả các view nằm bên trong thư mục con.
  - Chỉ định thuộc tính Layout nằm trong _ViewStart.cshtml

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

RAZOR ENGINE

- Giới thiệu:
  - ASPX và Razor
  - Razor có các tính năng như ASP.NET markup truyền thống nhưng dễ học, dễ sử dụng.
  - Razor là server side markup syntax như ASP và PHP
  - Razor cung cấp ngôn ngữ C# và Visual Basic

RAZOR ENGINE

- Các quy tắc cú pháp Razor chung cho C#
  - Khối code được bao trong cặp `@{...}`
  - Inline expressions (variables và functions) bắt đầu từ ký tự `@`
  - Các câu lệnh kết bở dấu ";"
  - Chuỗi hằng ký tự nằm trong cặp `"..."`
  - Các biến được khai báo bằng từ khóa `var`
  - C# files có đuôi mở rộng .cshtml

RAZOR ENGINE

- Các quy tắc Razor Syntax cho C#

```
<h2>(1+2)=@(1+2)</h2>
<h4>@Model.Title</h4>
<span>@Html.Raw(@Model.Title)</span>
@{   int x = 123;
     string y = "because."; }
@if (showMessage) {
    <text>This is plain text</text> }
```

RAZOR ENGINE

- Code Blocks:

```
@foreach(dynamic a in ViewBag.ListSP ) {
    <tr><td>@a.MaSP</td>
    <td>@a.TenSP</td>
    <td>@a.GiaSP</td></tr> }
```

- Razor - C# Variables:
  - Biến dùng để lưu trữ dữ liệu
  - Tên biến được đặt bắt đầu từ chữ cái
  - Biến được khai báo bắt đầu từ khóa var (ASP.NET tự động xác định kiểu dữ liệu) hoặc kiểu dữ liệu.

```
@{var greeting = "Welcome to W3Schools";
  var counter = 103;
  var today = DateTime.Today;
// Using data types:
  string greeting = "Welcome to W3Schools";
  int counter = 103;
  DateTime today = DateTime.Today;}
```

RAZOR ENGINE

- Razor - C# Loops và Arrays
  - For Loops:

```
@for(var i = 10; i < 21; i++){
        <p>Line @i</p>}
```

  - For Each Loops:

```
@foreach(dynamic a in ViewBag.ListSP){
  <tr><td>@a.MaSP</td>
  <td>@a.TenSP</td>
  <td>@a.GiaSP</td></tr> }
```

- Razor - C# Loops và Arrays
  - While Loops:

```
@{   var i = 0;
     while (i < 5){
          i+= 1;
          <p>Line @i</p>
     }
}
```

- Razor - C# Loops và Arrays
  - Arrays:

```
@{    string[] members = {"Jani", "Hege", "Kai", "Jim"};
      int i = Array.IndexOf(members, "Kai")+1;
      int len = members.Length;
      string x = members[2-1]; }
         <h3>Members</h3>
   @foreach (var person in members){
       <p>@person</p>  }
    <p>The number of names in Members are @len</p>
    <p>The person at position 2 is @x</p>
    <p>Kai is now in position @i</p>
```

- Razor - C# Logic Conditions
  - Switch Conditions:

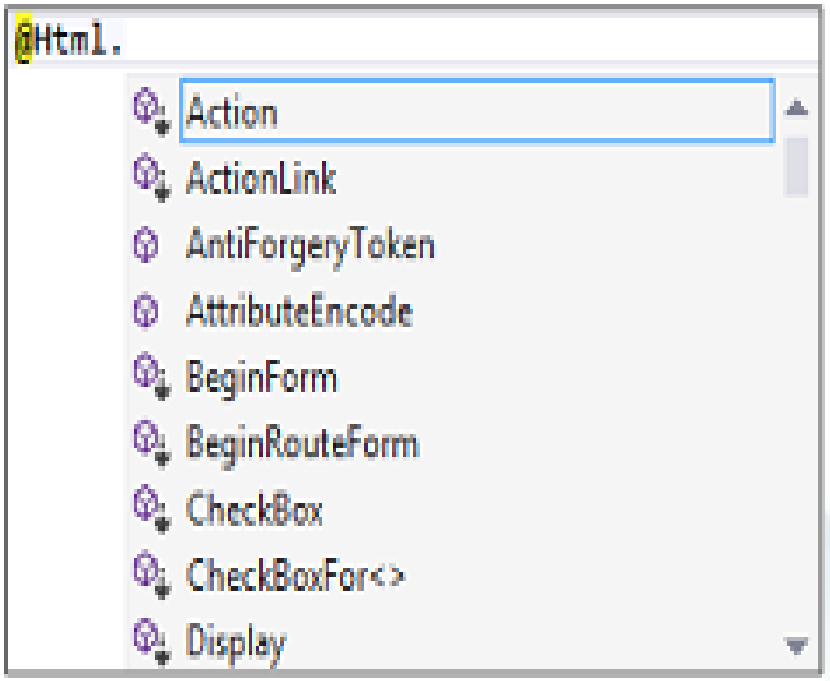
```
@{  var weekday=DateTime.Now.DayOfWeek;
    var day=weekday.ToString();
    var message="";}
@switch(day){
    case "Monday":
        message="This is the first weekday.";break;
    case "Thursday":
        message="Only one day before weekend.";break;
    case "Friday":
        message="Tomorrow is weekend!"; break;
    default:
        message="Today is " + day; break; }
 <p>@message</p>
```

- Razor - C# Logic Conditions
  - If Condition:

```
@{var price=25;}
 @if (price>=30) {
          <p>The price is high.</p> }
   else if (price>20 && price<30) {
          <p>The price is OK.</p> }
   else {<p>The price is low.</p> }
```

HTML HELPERS

- Giới thiệu:
  - HTML Helper bao gồm các phương thức giúp tạo các thuộc tính HTML trên view.
  - Tự động mã hóa
  - Thuộc tính Html là kiểu của System.Web.Mvc. HtmlHelper<T>.



HTML HELPERS

- Custom Helper Methods
  - Inline Html Helpers

```
@helper ListingItems(string[] items){
    <ol> @foreach (string item in items){
                    <li>@item</li>
         }
     </ol>
}
    <h3>Programming Languages:</h3>
 @ListingItems(new string[] { "C", "C++", "C#" })
 <h3>Book List:</h3>
 @ListingItems(new string[] { "How to C", "how to C++",
"how to C#" })
```

HTML HELPERS

- Built-In Html Helpers
  - Standard Html Helpers
    - @Html.TextBox() :
    - @Html.TextArea()
    - @Html.Password()
    - @Html.Hidden()
    - @Html.CheckBox()
    - @Html.RadioButton()
    - @Html.DropDownList ()
    - @Html.ListBox()
    - @Html.BeginForm()
    - @Html.EndForm(), ...

HTML HELPERS

- Built-In Html Helpers:

HTML Element	Example

Checkbox	`Html.CheckBox("myCheckbox", false)`
	Output:
	`<input id="myCheckbox" name="myCheckbox" type="checkbox" value="true" />`
	`<input name="myCheckbox" type="hidden" value="false" />`

Hidden field	`Html.Hidden("myHidden", "val")`
	Output:
	`<input id="myHidden" name="myHidden" type="hidden" value="val" />`

Radio button	`Html.RadioButton("myRadiobutton", "val", true)`
	Output:
	`<input checked="checked" id="myRadiobutton" name="myRadiobutton" type="radio" value="val" />`

Password	`Html.Password("myPassword", "val")`
	Output:
	`<input id="myPassword" name="myPassword" type="password" value="val" />`

Text area	`Html.TextArea("myTextarea", "val", 5, 20, null)`
	Output:
	`<textarea cols="20" id="myTextarea" name="myTextarea" rows="5">val</textarea>`

HTML HELPERS

HTML Element	Example

Checkbox	`Html.CheckBox("myCheckbox", false)`
	Output: `<input id="myCheckbox" name="myCheckbox" type="checkbox" value="true" />`
	`<input name="myCheckbox" type="hidden" value="false" />`

Hidden field	`Html.Hidden("myHidden", "val")` Output: `<input id="myHidden" name="myHidden" type="hidden" value="val" />`

Radio button	`Html.RadioButton("myRadiobutton", "val", true)`
	Output: `<input checked="checked" id="myRadiobutton" name="myRadiobutton"`
	`type="radio" value="val" />`

Password	`Html.Password("myPassword", "val")` Output: `<input id="myPassword" name="myPassword" type="password" value="val" />`

Text area	`Html.TextArea("myTextarea", "val", 5, 20, null)`
	Output: `<textarea cols="20" id="myTextarea" name="myTextarea" rows="5">`
	`val</textarea>`

Text box	`Html.TextBox("myTextbox", "val")` Output: `<input id="myTextbox" name="myTextbox" type="text" value="val" />`

Drop-down list	`Html.DropDownList("myList", new SelectList(new [] {"A", "B"}), "Choose")`
	Output: `<select id="myList" name="myList">`
	`<option value="">Choose</option>`
	`<option>A</option>`
	`<option>B</option>`
	`</select>`

Multiple-select	`Html.ListBox("myList", new MultiSelectList(new [] {"A", "B"}))`
	Output: `<select id="myList" multiple="multiple" name="myList">`
	`<option>A</option>`
	`<option>B</option>`
	`</select>`

HTML HELPERS

- Built-In Html Helpers
  - Standard Html Helpers

```
@using (Html.BeginForm("Search", "Home", FormMethod.Get)) {
        @Html.TextBox("txtTen", "Text")
        @Html.DropDownList("DropDownList1", new
SelectList(new [] {"Male", "Female"})) }
```

Hoặc:

```
@{Html.BeginForm("Search", "Home", FormMethod.Get)}
        @Html.TextBox("txtTen", "Text")
        @Html.DropDownList("DropDownList1", new
SelectList(new [] {"Male", "Female"}))
@{Html.EndForm();}
```

HTML HELPERS

- Built-In Html Helpers
  - Strongly Typed HTML Helpers: Sinh ra các phần tử HTML cơ bản dựa trên các thuộc tính của model.
    - @Html.TextBoxFor
    - @Html.TextAreaFor
    - @Html.PasswordFor
    - @Html.HiddenFor
    - @Html.CheckBoxFor
    - @Html.RadioButtonFor
    - @Html.DropDownListFor
    - @Html.ListBoxFor,...

HTML HELPERS

HTML Element	Example

Checkbox	`Html.CheckBoxFor(x => x.IsApproved)`
	Output: `<input id="IsApproved" name="IsApproved" type="checkbox" value="true" />`
	`<input name="IsApproved" type="hidden" value="false" />`

Hidden field	`Html.HiddenFor(x => x.FirstName)` Output: `<input id="FirstName" name="FirstName" type="hidden" value="" />`
Radio button	`Html.RadioButtonFor(x => x.IsApproved, "val")` Output: `<input id="IsApproved" name="IsApproved" type="radio" value="val" />`
Password	`Html.PasswordFor(x => x.Password)` Output: `<input id="Password" name="Password" type="password" />`
Text area	`Html.TextAreaFor(x => x.Bio, 5, 20, new{})` Output: `<textarea cols="20" id="Bio" name="Bio" rows="5">Bio value</textarea>`
Text box	`Html.TextBoxFor(x => x.FirstName)` Output: `<input id="FirstName" name="FirstName" type="text" value="" />`

Drop-down list	`Html.DropDownListFor(x => x.Gender, new SelectList(new [] {"M", "F"}))`
	Output:
	`<select id="Gender" name="Gender">`
	`<option>M</option>`
	`<option>F</option>`
	`</select>`

Multiple-select	`Html.ListBoxFor(x => x.Vals, new MultiSelectList(new [] {"A", "B"}))`
	Output: `<select id="Vals" multiple="multiple" name="Vals">`
	`<option>A</option>`
	`<option>B</option>`
	`</select>`

HTML HELPERS

- Built-In Html Helpers
  - Templated Helpers: Sử dụng metadata và template. Metadata bao gồm thông tin về giá trị của model như tên, kiểu dữ liệu
    - Html.Display
    - Html.Editor
    - Html.DisplayFor
    - Html.EditorFor,
    - Html.DisplayForModel
    - Html.EditorForModel.

- Built-In Html Helpers
  - Templated Helpers:

Helper	Example	Description
DisplayFor	Html.DisplayFor(x => x.FirstName)	Strongly typed của helper version trước
Editor	Html.Editor("FirstName")	Sinh ra editor cho thuộc tỉnh chỉ định trong model, lựa chọn phần tử HTML trương ứng kiểu thuộc tính và metadata.
EditorFor	Html.EditorFor(x => x.FirstName)	Strongly typed của helper version trước
Label	Html.Label("FirstName")	Sinh ra phần từ HTML <label> tham chiếu đến thuộc tính của model.
LabelFor	Html.LabelFor(x => x.FirstName)	Strongly typed của helper version trước
DisplayForModel	Html.DisplayForModel()	Sinh ra read only view cho cả đối tượng model.
EditorForModel	Html.EditorForModel()	Sinh ra các phần tử editor cho cả đối tượng model.
LabelForModel	Html.LabelForModel()	Sinh ra HTML <label> tham chiếu đến cả đối tượng model.

HTML HELPERS

- Built-In Html Helpers
  - Ví dụ:

```
<div class="editor-label">
      @Html.LabelFor(m=>m.MaSP)
      @Html.TextBoxFor(m=>m.MaSP)
</div>
<div class="editor-field">
      @Html.EditorFor(m=> m.MaSP)
      @Html.ValidationMessageFor(m=>m.MaSP)
</div>
```

- RENDERING HELPERS
  - Html.ActionLink and Html.RouteLink
    - ActionLink method: Sinh ra hyperlink đến một controller action khác. Sau đây là một số kiểu link.
      - √ Liên kết đến Action cùng controller:

```
@Html.ActionLink("Link Text", "AnotherAction")
```

      - √ Liên kết đến controller khác:

```
@Html.ActionLink("Link Text", "Action“, "AnotherController")
```

      - √ Liên kết và truyền dữ liệu đến Action:

```
@Html.ActionLink("Edit link text", "Edit", new {Id=3})
@Html.ActionLink("Edit link text","Edit","StoreManager",new {id=10720})
```

- RENDERING HELPERS
  - Html.ActionLink and Html.RouteLink
    - RouteLink method: có cùng pattern như ActionLink helper, cũng có khác khái niệm route name nhưng không có các tham số cho controller name và action name.

```
@Html.RouteLink("Link Text", new
    {action="AnotherAction"})
```

- RENDERING HELPERS
  - URL Helpers: URL helpers tương tự như HTML ActionLink và RouteLink helpers, nhưng thay vì sinh ra HTML thì chúng sinh ra URLs dưới dạng các chuỗi. Có 3 helpers:
    - Action
    - Content
    - RouteUrl

- RENDERING HELPERS
  - URL Helpers:

```
<span>
    @Url.Action("Browse", "Store", new { genre = "Jazz" })
</span>
```

➔ `<span> /Store/Browse?genre=Jazz </span>`

```
 <script src="@Url.Content("~/Scripts/jquery-1.10.2.min.js")"
type="text/javascript">
  </script>
```

➔ `<script src="~/Scripts/jquery-1.5.1.min.js" type="text/javascript"></script>`

- RENDERING HELPERS
  - URL Helpers:

```
<span>
    @Url.Action("Browse", "Store", new { genre = "Jazz" })
</span>
```

➔ `<span> /Store/Browse?genre=Jazz </span>`

```
 <script src="@Url.Content("~/Scripts/jquery-1.10.2.min.js")"
type="text/javascript">
  </script>
```

➔ `<script src="~/Scripts/jquery-1.5.1.min.js" type="text/javascript"></script>`

- RENDERING HELPERS
  - Html.Partial and Html.RenderPartial:
    - Partial helper sinh partial view thành string, các dạng:

```
public void Partial(string partialViewName);
public void Partial(string partialViewName, object model);
public void Partial(string partialViewName, ViewDataDictionary
viewData);
public void Partial(string partialViewName, object
model,ViewDataDictionary viewData);
```

- RENDERING HELPERS
  - Html.Partial and Html.RenderPartial:
    - RenderPartial helper tương tự như Partial, nhưng RenderPartial viết trực tiếp response output stream thay ch việc trả về string. Đây cũng là lý do ta phải đặt RenderPartial trong khối code.

```
@{Html.RenderPartial("AlbumDisplay "); }
@Html.Partial("AlbumDisplay ")
```

- RENDERING HELPERS
  - Html.Action and Html.RenderAction:
    - Action và RenderActionthực thi controller action riêng biệt và hiển thị results. Action cung cáp tính linh hoạt và khả năng sử dụng lại vì controller action xây dựng trên các model khác nhau.
    - Ví dụ

```
public ActionResult Index(){ return View();}
[ChildActionOnly]
public ActionResult Menu() {
      var menu = GetMenuFromSomewhere();
      return PartialView(menu);
 }
```

RENDERING HELPERS

- Partial view Menu:

```
@model Menu
<ul>
@foreach (var item in Model.MenuItem) {
  <li>@item.Text</li>
}
</ul>
```

- View Index:

```
<html>
<head><title>Index with Menu</title></head>
<body>
    @Html.Action("Menu")
   <h1>Welcome to the Index View</h1>
    @{Html.RenderAction("HeaderCart", "Home");}
</body>
</html>
```

DYNAMIC CONTENT

Công nghệ	Mô tả
Inline code	Sử dụng các câu lệnh như if, foreach...
HTML helper methods	Dùng ddeersinh ra các phần tử HTML, thường dùng trong view model hoặc view data values.
Sections	Sử dụng tạo các sections mà nội dung sẽ được chèn vào layout tại các vị trí chỉ định.
Partial views	Sử dụng cho sharing subsections của các view. Partial views không gọi một action method nào, do đó không thể sử dụng để thực thi business logic.
Child actions	Sử dụng cho việc tạo reusable UI controls hoặc vấn đề chứa business logic. Khi sử dụng child action, nó gọi một action method, sinh ra view, và bơm (inject) kết quả vào response stream.

- Sections:
  - + Tạo các Section trong Layout:

```
<body>
    @RenderSection("Header")
    <div class="layout">
        This is part of the layout
    </div>
    @RenderBody()
     <div class="layout">
        This is part of the layout
    </div>
     @RenderSection("Footer")
</body>
```

- Sections:
  - Sử dụng Sections trong View:

```
@section Header {
    <div class="view">
   @foreach (string str in new [] {"Home", "List", "Edit"}){
       @Html.ActionLink(str, str, null, new { style = "margin: 5px" })
    }
    </div>
}
<div class="view">
    This is a list of fruit names:
    @foreach (string name in Model) {
        <span><b>@name</b></span>
    }
</div>
 @section Footer {
      <div class="view">
        This is the footer
    </div>
}
```

DYNAMIC CONTENT

- Sections:
  - RenderSection (“Name”) helper method chỉ định vị trí sẽ được chèn vào trên view.
  - @section Name: Sử dụng trên mỗi view để định Section chỉ định
  - Phần của view không chứa trong mỗi section được chèn vào trong tại vị trí RenderBody helper.

- Sections:
  - Kiểm tra Section:

```
@if (IsSectionDefined("Footer")) {
    @RenderSection("Footer")
} else {
    <h4>This is the default footer</h4>
}
```

  - Tạo Section dạng lựa chọn:

```
@RenderSection("scripts", required:false)
```

- Child Actions:
  - Là các action methods được gọi trong view. Nó giúp tránh lặp lại controller logic mà muốn lặp lại nhiều chỗ trên ứng dụng.
  - Child actions là các actions như partial views để view (`return PartialView();`).
  - Tạo Child Action:

```
[ChildActionOnly]
public ActionResult Time(){
    return PartialView(DateTime.Now);
}
```

- Child Actions:
  - Sử dụng Child Action:
    - Cùng controller

```
@{Html.RenderAction("HeaderCart");}
@Html.Action("HeaderCart")
```

    - Khác controller”

```
@{Html.RenderAction("HeaderCart", "Home");}
@Html.Action("HeaderCart", "Home")
```

THẢO LUẬN – CÂU HỎI