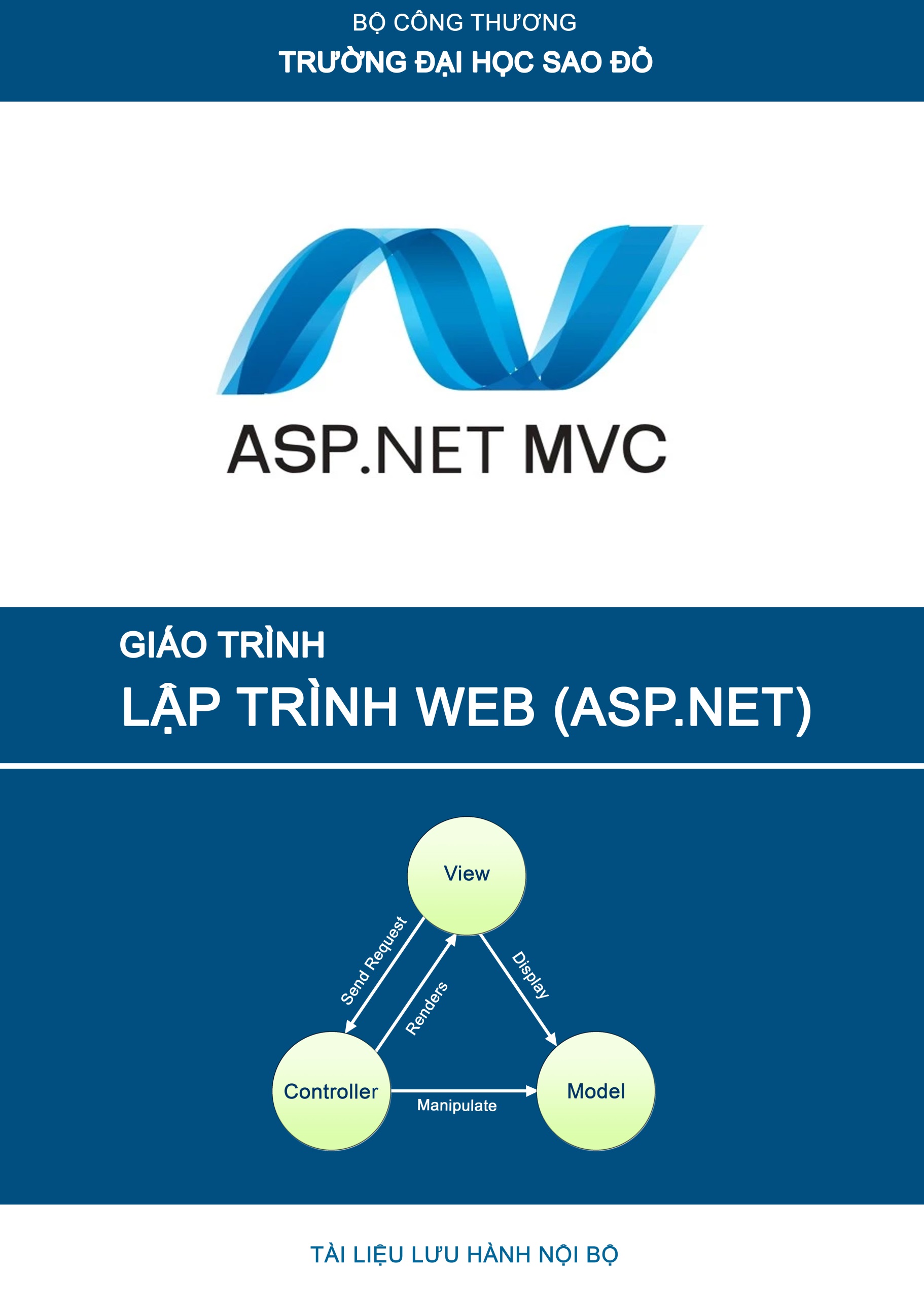
******

BỘ CÔNG THƯƠNG

**TRƯỜNG ĐẠI HỌC SAO ĐỎ**

**GIÁO TRÌNH**

**LẬP TRÌNH WEB (ASP.NET)**

**Dùng cho sinh viên Đại học chính quy**

**Ngành: Công nghệ thông tin**

*(Tài liệu lưu hành nội bộ)*

**NĂM 2021**

# MỤC LỤC

[MỤC LỤC 1](#_Toc109920483)

[LỜI NÓI ĐẦU 5](#_Toc109920484)

[Chương 1. TỔNG QUAN 7](#_Toc109920485)

[1.1. Giới thiệu ASP.NET MVC 7](#_Toc109920486)

[1.2. Tạo website ASP.NET MVC đầu tiên 9](#_Toc109920487)

[1.2.1. Tạo ứng dụng đầu tiên 10](#_Toc109920488)

[1.2.2. Cấu trúc thư mục web 11](#_Toc109920489)

[1.3. ASP.NET MVC Routing 12](#_Toc109920490)

[1.3.1. Giới Thiệu 12](#_Toc109920491)

[1.3.2. URL Patterns 14](#_Toc109920492)

[1.3.3. Thêm Routing vào MVC application 14](#_Toc109920493)

[1.3.4. Thiết lập giá trị mặc định cho URL parameters 15](#_Toc109920494)

[1.3.5. Khớp URL với Routes 15](#_Toc109920495)

[1.4. Controller 16](#_Toc109920496)

[1.5. ActionResult 18](#_Toc109920497)

[1.6. Action Selectors 19](#_Toc109920498)

[1.7. Model 22](#_Toc109920499)

[1.8. View 23](#_Toc109920500)

[1.8.1. Razor View Engine 24](#_Toc109920501)

[1.8.2. Thêm mới View 24](#_Toc109920502)

[1.8.3. Tạo Edit View 28](#_Toc109920503)

[1.9. Tích hợp Controller, View và Model 33](#_Toc109920504)

[1.10. Mô hình liên kết dữ liệu - Model Binding 36](#_Toc109920505)

[CÂU HỎI ÔN TẬP CHƯƠNG 1 41](#_Toc109920506)

[Chương 2. RAZOR VIEW ENGINE VÀ HTML HELPERS 42](#_Toc109920507)

[2.1. Razor 42](#_Toc109920508)

[2.1.1. Biểu thức nội tuyến 42](#_Toc109920509)

[2.1.2. Khai báo biến 42](#_Toc109920510)

[2.1.3. Các cấu trúc lệnh 43](#_Toc109920511)

[2.1.4. Mô hình – Model 43](#_Toc109920512)

[2.2. HTMLHelpers 43](#_Toc109920513)

[2.2.1. Giới thiệu 43](#_Toc109920514)

[2.2.2. HtmlHelper - TextBox 45](#_Toc109920515)

[2.2.3. HtmlHelper - TextArea 46](#_Toc109920516)

[2.2.4. HtmlHelper - CheckBox 47](#_Toc109920517)

[2.2.5. HtmlHelper - RadioButton 48](#_Toc109920518)

[2.2.6. HtmlHelper - DropDownList 49](#_Toc109920519)

[2.2.7. HtmlHelper - Hidden field 51](#_Toc109920520)

[2.2.8. HtmlHelper - Password 52](#_Toc109920521)

[2.2.9. HtmlHelper - Display 52](#_Toc109920522)

[2.2.10. HtmlHelper - Label 53](#_Toc109920523)

[2.2.11. HtmlHelper - Editor 53](#_Toc109920524)

[2.3. Biểu thức lambda (lambda expression) 55](#_Toc109920525)

[2.4. Xác thực dữ liệu - Data Validation 59](#_Toc109920526)

[CÂU HỎI ÔN TẬP CHƯƠNG 2 66](#_Toc109920527)

[Chương 3. GIAO DIỆN 67](#_Toc109920528)

[3.1. Layout view 67](#_Toc109920529)

[3.1.1. Tổng quan 67](#_Toc109920530)

[3.1.2. Tạo Layout View 72](#_Toc109920531)

[3.2. Partial View 74](#_Toc109920532)

[3.2.1. Tổng quan 74](#_Toc109920533)

[3.2.2. Tạo mới Partial View 75](#_Toc109920534)

[3.3. Thực thi các script 79](#_Toc109920535)

[3.3.1. Kỹ thuật đóng gói (Bundling) và Giảm thiểu (Minification) 79](#_Toc109920536)

[3.3.2. ScriptBundle 80](#_Toc109920537)

[3.3.3. StyleBundle 84](#_Toc109920538)

[3.4. Khu vực – Area 85](#_Toc109920539)

[CÂU HỎI ÔN TẬP CHƯƠNG 3 88](#_Toc109920540)

[Chương 4. TRUYỀN DỮ LIỆU TRONG ASP.NET MVC 89](#_Toc109920541)

[4.1. ViewBag 89](#_Toc109920542)

[4.2. ViewData 90](#_Toc109920543)

[4.3. TempData 92](#_Toc109920544)

[CÂU HỎI ÔN TẬP CHƯƠNG 4 95](#_Toc109920545)

[Chương 5. LẬP TRÌNH TƯƠNG TÁC CƠ SỞ DỮ LIỆU KHÔNG SỬ DỤNG ENTITY FRAMEWORK 96](#_Toc109920546)

[5.1. Tạo mới Project 96](#_Toc109920547)

[5.2. Tạo Model Class 96](#_Toc109920548)

[5.3. Tạo Controller 97](#_Toc109920549)

[5.4. Tạo Database, Table và Stored Procedure 97](#_Toc109920550)

[5.5. Tạo chuỗi kết nối trong Web.config 100](#_Toc109920551)

[5.6. Tạo lớp StudentDBHandle.cs 100](#_Toc109920552)

[5.7. Tạo phương thức hành động trong StudentController 103](#_Toc109920553)

[5.8. Tạo Partial View từ phương thức hành động 104](#_Toc109920554)

[5.9. Cấu hình RouteConfig.cs 108](#_Toc109920555)

[CÂU HỎI ÔN TẬP CHƯƠNG 5 110](#_Toc109920556)

[Chương 6. BỘ LỌC - FILTERS 111](#_Toc109920557)

[6.1. Filters 111](#_Toc109920558)

[6.2. Áp dụng bộ lọc 112](#_Toc109920559)

[6.3. Bộ lọc tùy chỉnh 114](#_Toc109920560)

[6.4. Bộ lọc hành động - Action Filters 115](#_Toc109920561)

[CÂU HỎI ÔN TẬP CHƯƠNG 6 119](#_Toc109920562)

[TÀI LIỆU THAM KHẢO 120](#_Toc109920563)

# LỜI NÓI ĐẦU

Trong giáo trình này, chúng tôi cung cấp cho bạn đọc về ASP.NET MVC5 với các kiến thức cơ bản để xây dựng và triển khai website ASP.NET [MVC](https://www.dammio.com/glossary/mvc). Đây là mô hình lập trình web được sử dụng rất phổ biến hiện nay. Tuy nhiên kiến thức về lập trình ASP.NET MVC rất phức tạp. Mặc dù nội dung giáo trình đều viết và mô tả khá tỉ mỉ, tuy nhiên đặc thù ASP.NET [MVC](https://www.dammio.com/glossary/mvc) để hiểu thật sâu không hề dễ, vì vậy chúng tôi mong bạn đọc cố gắng vừa đọc để hiểu, vừa thực hành theo các ví dụ minh họa và vừa tìm đọc thêm thông tin trên [Internet](https://www.dammio.com/glossary/internet).

Với những nội dung trên, chúng tôi hy vọng giáo trình sẽ hữu ích cho bạn đọc, đặc biệt là những người mới bắt đầu tiếp cận với ASP.NET MVC. Đây là tài liệu lưu hành nội bộ, trong quá trình biên soạn, chúng tôi có sử dụng tài liệu của các đồng nghiệp và bố cục lại cho phù hợp đối tượng đào tạo. Do kinh nghiệm còn nhiều hạn chế nên việc biên soạn không thể tránh khỏi những thiếu sót, rất mong nhận được những ý kiến đóng góp của bạn đọc để tài liệu hoàn chỉnh hơn.

**Tác giả**

# Chương 1. TỔNG QUAN

## 1.1. Giới thiệu ASP.NET MVC

ASP.NET MVC là một framework web được phát triển bởi Microsoft, thực thi mô hình MVC (model–view–controller). ASP.NET MVC là 1 phần mềm mã mở, tách rời với thành phần độc quyền ASP.NET Web Forms. ASP.NET MVC đang nổi lên là phương pháp phát triển web mạnh nhất và phổ biến nhất trên nền ASP.NET hiện nay.

Trong các phiên bản sau của ASP.NET, ASP.NET MVC, ASP.NET Web API, và ASP.NET Web Pages (nền tảng chỉ dùng trang Razor) sẽ được trộn chung vào MVC 6.

Các phiên bản ASP.NET có rất nhiều phiên bản từ khi ra đời năm 2007. Một số phiên bản đáng chú ý như MVC4, MVC5, MVC6 và phiên bản mới nhất là ASP.NET Core MVC 2.0.0, phát hành ngày 14 tháng 8 năm 2017.

Mô hình MVC là một chuẩn mô hình và đóng vai trò quan trọng trong quá trình xây dựng, phát triển, vận hành và bảo trì một hệ thống hay một ứng dụng, phần mềm. Nó tạo ra một mô hình 3 lớp Model – View – Controller tách biệt và tương tác nhau, giúp các chuyên gia có thể dễ dàng dựa vào mô hình để trao đổi và xử lý những nghiệp vụ một cách nhanh chóng. Đây là một mô hình đã xuất hiện từ những năm 70 của thế kỷ 20 tại phòng thí nghiệm Xerox PARC ở Palo Alto, nó không phụ thuộc vào môi trường, nền tảng xây dựng hay ngôn ngữ phát triển. Chúng ta có thể áp dụng mô hình MVC vào các dự án trong môi trường Windows, Linux… và sử dụng bất kỳ ngôn ngữ nào như PHP, ASP, JSP…

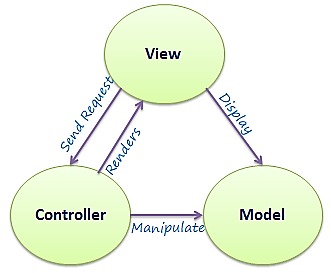
Dựa trên nền ASP.NET, ASP.NET MVC cho phép các nhà phát triển phần mềm xây dựng ứng dụng web như là một cấu thành của 3 vai trò: Model, View và Controller. Mô hình MVC định nghĩa ứng dụng web với 3 tầng logic:

Mô hình MVC được chia làm 3 lớp xử lý gồm Model – View – Controller:

**Model**: Nơi xử lý những nghiệp vụ tương tác với dữ liệu hoặc hệ quản trị cơ sở dữ liệu (mysql, mssql… ); nó sẽ bao gồm các class/function xử lý nhiều nghiệp vụ như kết nối database, truy vấn dữ liệu, thêm – xóa – sửa dữ liệu…

**View**: Nơi xử lý giao diện người dùng như một nút bấm, khung nhập, menu, hình ảnh… nó đảm nhiệm nhiệm vụ hiển thị dữ liệu và giúp người dùng tương tác với hệ thống.

**Controller**: Nơi tiếp nhận những yêu cầu xử lý được gửi từ người dùng, nó sẽ gồm những class/ function xử lý nhiều nghiệp vụ logic giúp lấy đúng dữ liệu thông tin cần thiết nhờ các nghiệp vụ lớp **Model** cung cấp và hiển thị dữ liệu đó ra cho người dùng nhờ lớp **View**



Hình 1.1. Sự tương tác giữa các lớp trong mô hình MVC

**Controller – View**sẽ lấy những hình ảnh, nút bấm … hoặc hiển thị dữ liệu được trả ra từ Controller để người dùng có thể quan sát và thao tác. Trong sự tương tác này cũng có thể không có dữ liệu được lấy từ Model và khi đó nó chỉ chịu trách nhiệm hiển thị đơn thuần như hình ảnh, nút bấm …

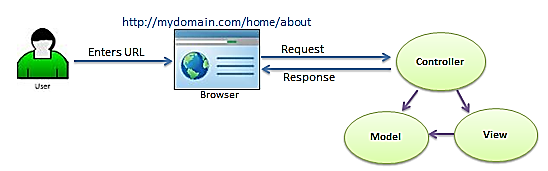
**Controller – Model** là luồng xử lý khi controller tiếp nhận yêu cầu và các tham số đầu vào từ người dùng, controller sẽ sử dụng các lớp/hàm trong Model cần thiết để lấy ra những dữ liệu chính xác.

**View – Model** có thể tương tác với nhau mà không qua Controller, nó chỉ đảm nhận hiển thị dữ liệu chứ không phải qua bất kỳ xử lý nghiệp vụ logics nào. Nó giống như các vùng dữ liệu hiển thị tĩnh trên các website như block slidebar…

Mô hình MVC có thể áp dụng cho nhiều loại dự án khác nhau: Dự án website, dự án ứng dụng – phần mềm … Trong phạm vi tài liệu chúng ta tìm hiểu mô hình này được áp dụng trong các dự án xây dựng và phát triển website.

Mô hình MVC trong các dự án website và sẽ hoạt động theo quy trình sau:

1. Người dùng sử dụng một trình duyệt web bất kỳ (Firefox, Chrome, IE,…) để có thể gửi những yêu cầu (HTTP Request) có thể kèm theo những dữ liệu nhập tới những Controller xử lý tương ứng. Việc xác định Controller xử lý sẽ dựa vào một bộ Routing điều hướng.
2. Khi Controller nhận được yêu cầu gửi tới, nó sẽ chịu trách nhiệm kiểm tra yêu cầu đó có cần dữ liệu từ Model hay không? Nếu có, nó sẽ sử dụng các class/function cần thiết trong Model và trả ra kết quả (Resulting Arrays), khi đó Controller sẽ xử lý giá trị đó và trả ra View để hiển thị. Controller sẽ xác định các View tương ứng để hiển thị đúng với yêu cầu.
3. Khi nhận được dữ liệu từ Controller, View sẽ chịu trách nhiệm xây dựng các thành phần hiển thị như hình ảnh, thông tin dữ liệu… và trả về GUI Content để Controller đưa ra kết quả lên trình duyệt.
4. Trình duyệt sẽ nhận giá trị trả về (HTTP Response) và sẽ hiển thị với người dùng. Kết thúc một quy trình hoạt động.



Hình 1.2. Mô hình tương tác trong MVC

**Ưu, nhược điểm của mô hình MVC**

**Ưu điểm:**

- Các dự án có thể áp dụng ngay mô hình MVC mà không phụ thuộc môi trường, nền tảng xây dựng hay ngôn ngữ lập trình phát triển.

- Quy hoạch các class/ function vào các thành phần riêng biệt Controller – Model – View, khi đó sẽ dễ dàng xây dựng, phát triển, quản lý, vận hành và bảo trì một dự án; tạo sự rõ ràng, logic trong quá trình phát triển dự án, kiểm soát được các luồng xử lý và tạo ra các thành phần xử lý nghiệp vụ chuyên biệt hóa.

- Tạo thành mô hình chuẩn cho nhiều dự án, từ đó dễ dàng tiếp cận, tìm hiểu các dự án một cách nhanh chóng và hiệu quả.

- Giúp các chuyên gia lập trình, nhà quản lý, nhà đầu tư … có thể hiểu được sự hoạt động của dự án hoặc giúp các lập trình viên dễ dàng quản lý, phát triển dự án.

- Đây là một mô hình chuẩn, là mô hình tối ưu nhất hiện nay so với nhiều mô hình khác và được sử dụng trong nhiều dự án và nhiều lĩnh vực, đặc biệt trong công nghệ phát triển phần mềm. Các lập trình viên sử dụng mô hình chuẩn MVC để có thể dễ dàng phân phối và chuyển giao công nghệ.

- Đây là mô hình đơn giản, xử lý những nghiệp vụ đơn giản, và dễ dàng triển khai với các dự án nhỏ.

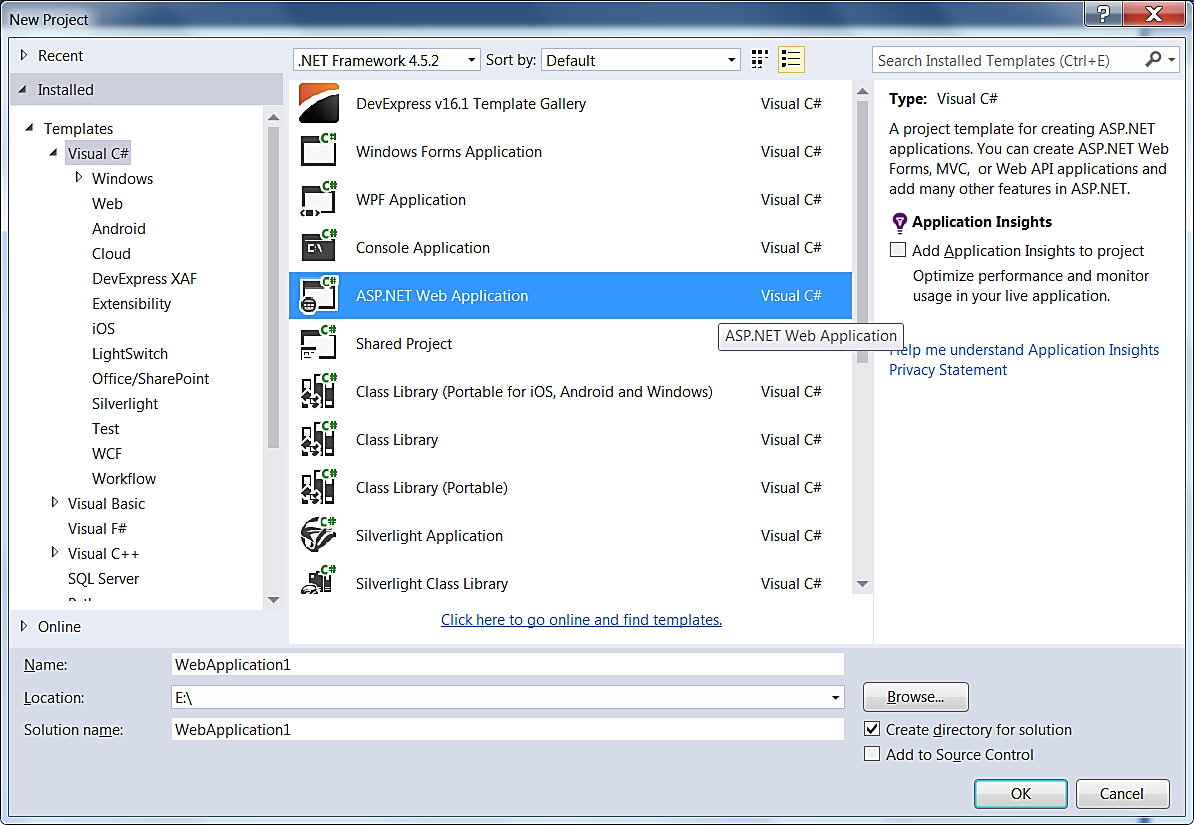
**Nhược điểm:** Yêu cầu về chuyên môn khá cao, có kiến thức vững về các mô hình chuẩn; khó triển khai với những dự án yêu cầu phúc tạp hơn. Hiện nay đang có một khái niệm mô hình mới đó là HMVC đang dần thay thế cho MVC

## 1.2. Tạo website ASP.NET MVC đầu tiên

Trước tiên phải cài đặt Visual Studio 2013 hoặc phiên bản trở về sau kèm theo SQL Server 2008 trở lên. Các phần hướng dẫn cho việc cài đặt có thể tìm thấy ở website của Microsoft hoặc trên mạng Internet.

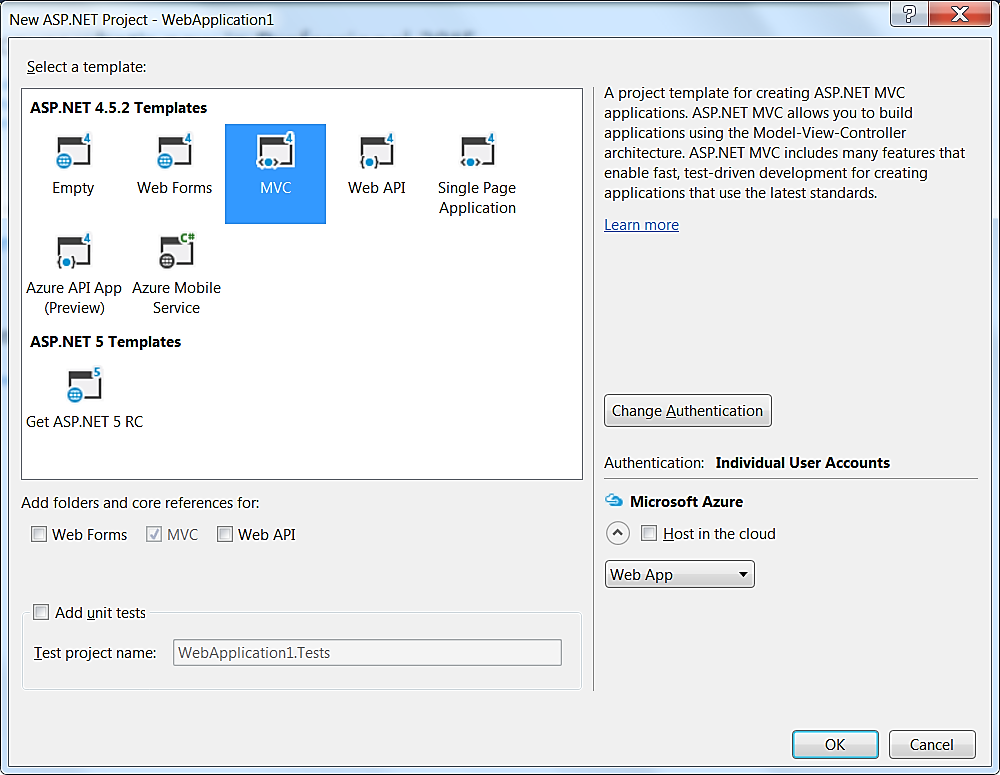
### 1.2.1. Tạo ứng dụng đầu tiên

Mở Visual Studio, chọn File\New\Project, một cửa sổ New Project sẽ hiện lên. Trong cửa sổ này, phần bên trái, chọn Visual C#, phần bên phải chọn ASP.NET Web Application, đặt tên dự án.



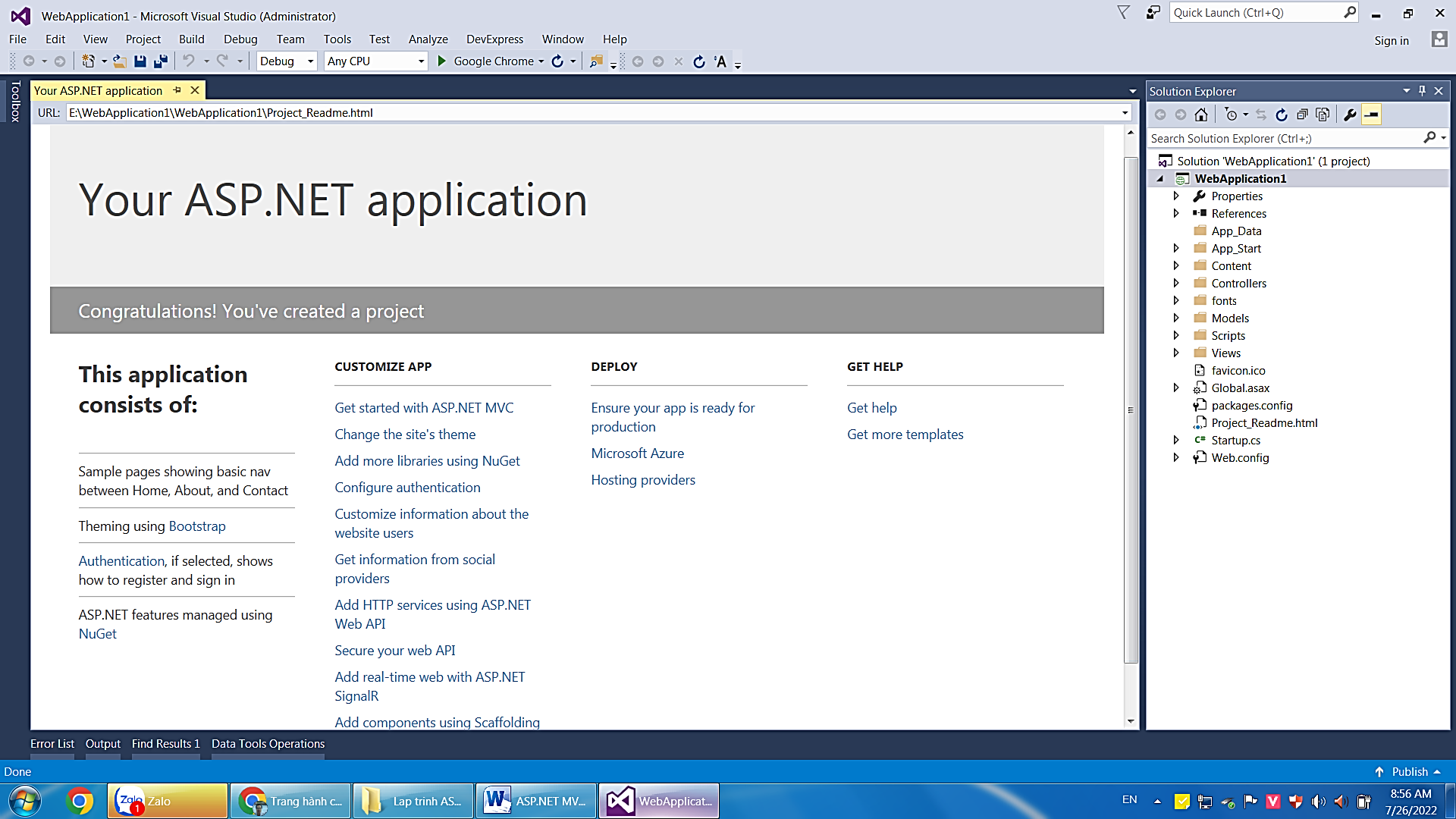
Hình 1.3. Tạo mới Project

Ở hộp thoại New ASP.NET Project, chọn dự án là kiểu MVC và nhấn OK.



Hình 1.4. Tạo mới Project MVC

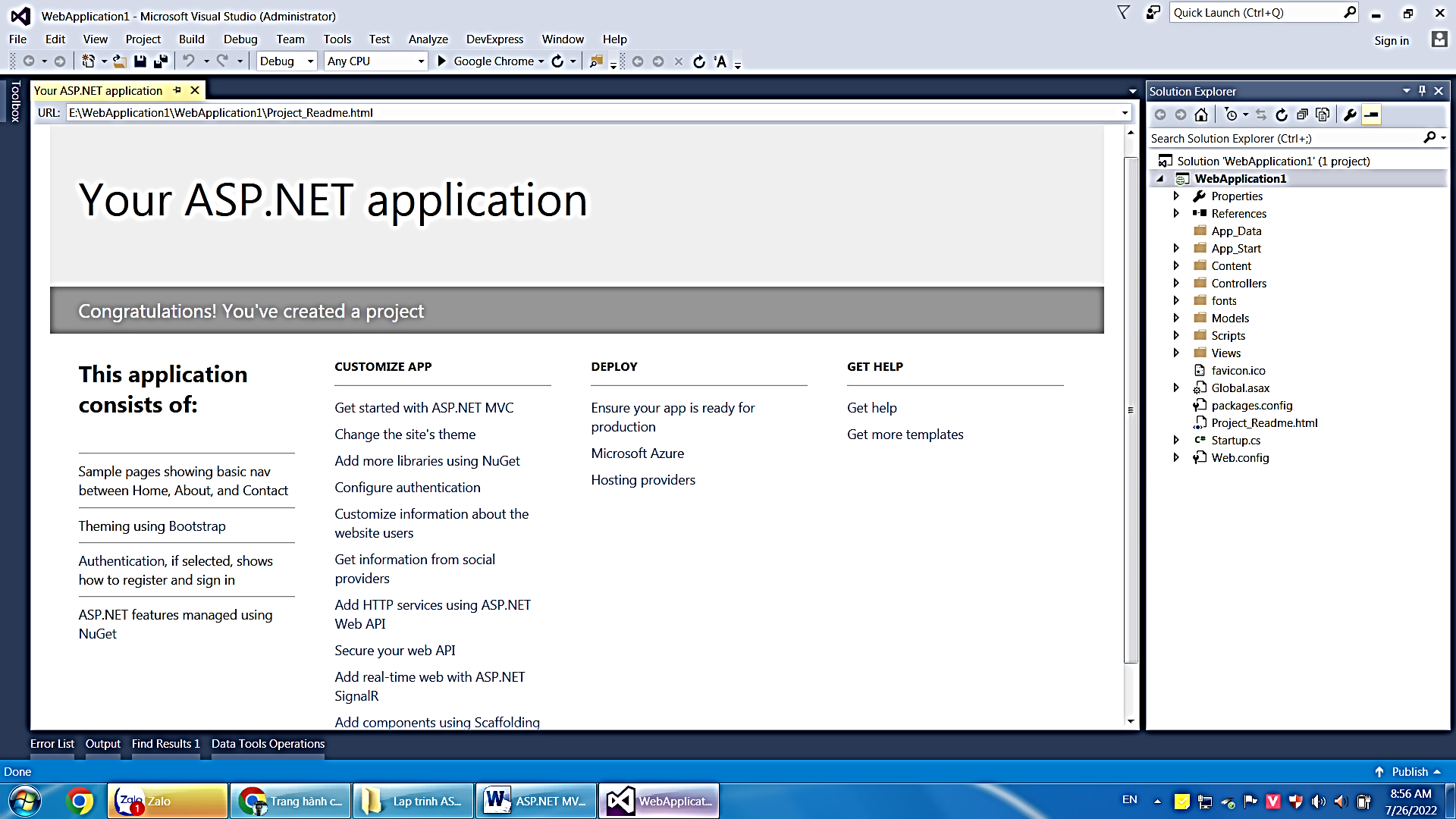
Visual Studio sẽ tự động tạo một template mặc định cho ứng ASP.NET MVC  vừa tạo ra, vì vậy có ngay 1 dự án đơn giản mà không cần làm gì hết.



1.5. Giao diện trang chủ

Trong ứng dụng mặc định, có 3 trang chính Home, About và Contact nằm thanh menu, có thể nhấn vào 3 liên kết để thử nghiệm. Ứng dụng mặc định cũng dùng giao diện Bootstrap, do đó khi phóng to, thu nhỏ hay chạy trên các màn hình có kích thước khác nhau, sẽ thấy giao diện thay đổi đáp ứng theo độ rộng màn hình.

### 1.2.2. Cấu trúc thư mục web



1.6. Cấu trúc thư mục web

Trong hình trên, một số giải thích cơ bản về cấu trúc thư mục, tập tin là:

- References: Nơi lưu trữ, thêm/bớt các thư viện DLL có liên quan đến website.

- App\_Data: Nơi lưu trữ cơ sở dữ liệu dạng tập tin .MDF, để làm dự án thuận tiện ta nên chọn làm cơ sở dữ liệu dạng này.

- App\_Start: Nơi lưu trữ các lớp cấu hình sẽ được kích hoạt chạy trước khi dự án web chạy.

- Content: Nơi lưu trữ các tập tin nội dung như css, image, … ta có thể thay đổi tập tin này nếu muốn.

- Controllers: Nơi lưu trữ các lớp điều khiển cho dự án, chính là Controller trong 3 thành phần MVC (Model - View - Controller)

- Models: Nơi lưu trữ các lớp mô hình của dự án, là thành phần Model của mô hình MVC.

- Views: Nơi lưu trữ các thành phần View của mô hình MVC (\*.cshtml) để hiển thị giao diện.

- Fonts, Script: Nơi lưu trữ dạng font và mã javascript giúp xây dựng giao diện website, ta có thể thay đổi tùy ý.

- Web.config: Chứa cấu hình web.

- Global.asax: Chứa lớp cấu hình Session, Cookies, Application cho dự án.

## 1.3. ASP.NET MVC Routing

Trong công nghệ làm web với mô hình MVC, dù là bất kỳ Framework nào như Ruby on Rails, laravel, hay ASP.NET MVC thì Routing luôn là một phần quan trọng và cần ưu tiên khi muốn làm quen với Framework đó.

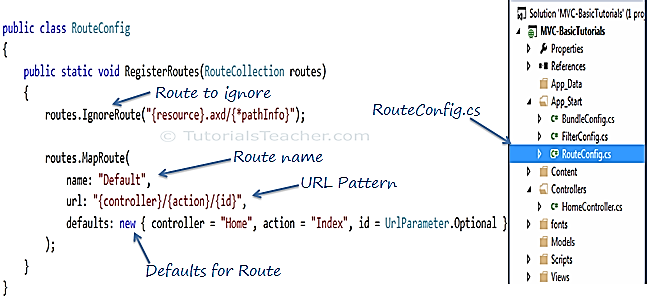
### 1.3.1. Giới Thiệu

[ASP.NET](http://asp.net/) Routing cho phép sử dụng các URL không dẫn đến một file cụ thể trong website. Bởi vì URL không dẫn đến một file cụ thể trong ứng dụng web, ta có thể làm cho URL trở nên thân thiện hơn với người dùng. Ở ứng dụng [ASP.NET](http://asp.net/) không sử dụng routing ([ASP.NET](http://asp.net/) webform - đã bị khai tử), một request từ một URL thường được dẫn đến một tập tin xử lý request đó. Chẳng hạn:

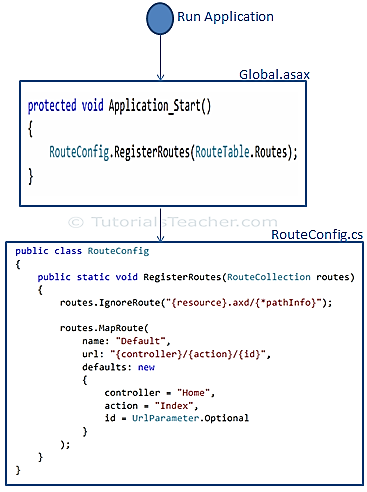
http://server/application/Products.aspx?id=4

sẽ dẫn đến file có tên Product.aspx có chứa code xử lý và render để trả lời trình duyệt. Web page này sử dụng query string id = 4 để xác định nội dung được hiển thị.

Với MVC (ví dụ: http://server/application/products/show/beverages), URL sẽ không truy vấn tới file /products/show/beverages mà sẽ được định nghĩa bởi Routing. Ví dụ, nếu Routing định nghĩa URL là server/application/{area}/{action}/{category} thì ở request URL trên, area = product, action = show và category = beverages.



Hình 1.7. Configure Route in MVC



Hình 1.8. Register Route

Trình duyệt yêu cầu một địa chỉ từ Controller Action trong [ASP.NET](http://asp.net/) MVC Framework được gọi là định tuyến URL (URL routing). URL routing sẽ chỉ định yêu cầu (request) tới Controller Action. URL routing sử dụng một bảng định tuyến để điều khiển các yêu cầu (request). Bảng định tuyến được tạo khi ứng dụng được chạy lần đầu tiên. Bảng định tuyến được thiết lập trong file Global.asax.

### 1.3.2. URL Patterns

Một URL pattern có thể chứa các parameters. các parameters ngăn cách nhau bởi dấu “/” (slash character). Khi một request URL được yêu cầu, URL được phân tách thành các đoạn như trong Routing đã định nghĩa, các giá trị được truyền vào từ URL sẽ được xử lý dưới dạng key - value. Trong URL pattern ta xác định bằng cách đặt key trong dấu {}, và định nghĩa dấu phân cách cho chúng (ví dụ: /, -).

Ví dụ: {language}-{country}/{action} là một URL pattern đúng còn {language}{country}/{action} là một URL pattern sai vì thiếu dấu phân cách.

Dưới đây là một vài ví dụ về URL pattern đúng và request URL tương ứng.

Bảng 1.1. Ví dụ về URL pattern.

| Định nghĩa Router | Ví dụ về URL đúng |
| --- | --- |
| {controller}/{action}/{id} | /Products/show/beverages |
| {table}/Details.aspx | /Products/Details.aspx |
| blog/{action}/{entry} | /blog/show/123 |
| {reporttype}/{year}/{month}/{day} | /sales/2008/1/5 |
| {locale}/{action} | /US/show |
| {language}-{country}/{action} | /en-US/show |

Với [ASP.NET](http://asp.net/) MVC, URL điển hình sẽ chứa {controller} và {action}.

### 1.3.3. Thêm Routing vào MVC application

Thông thường, nếu không cần các URL tương đối đặc biệt, ta có thể sử dụng Routing được định nghĩa sẵn khi tạo project MVC, nếu muốn chỉnh sửa routes trong ứng dụng MVC, ta có thể sử dụng MapRoute. Ví dụ dưới đây là đoạn code định nghĩa routes trong ứng dụng MVC.

public class MvcApplication: System.Web.HttpApplication

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

"Default", // Route name

"{controller}/{action}/{id}", // URL with parameters

new { controller = "Home", action = "Index", id = "" } // Parameter defaults

);

}

protected void Application\_Start()

{

RegisterRoutes(RouteTable.Routes);

}

}

### 1.3.4. Thiết lập giá trị mặc định cho URL parameters

Khi định nghĩa một route, ta có thể xác định giá trị mặc định (default value) cho parameter của route. Default value sẽ được sử dụng khi trong URL không chứa parameter. Ví dụ sau cho thấy cách thêm default values, sử dụng MapPageRoute.

void Application\_Start(object sender, EventArgs e)

{

RegisterRoutes(RouteTable.Routes);

}

public static void RegisterRoutes(RouteCollection routes)

{

routes.MapPageRoute("",

"Category/{action}/{categoryName}",

"~/categoriespage.aspx",

true,

new RouteValueDictionary

{{"categoryName", "food"}, {"action", "show"}});

}

Khi [ASP.NET](http://asp.net/) xử lý URL request, route sẽ được định nghĩa như bảng dưới đây:

Bảng 1.2. Ví dụ về giá trị Parameters.

| URL | Giá trị Parameter |
| --- | --- |
| /Category | action = "show" (default value) categoryName = "food" (default value) |
| /Category/add | action = "add" categoryName = "food" (default value |
| /Category/add/beverages | action = "add" categoryName= "beverages" |

### 1.3.5. Khớp URL với Routes

Khi Routing xử lý các URL request, nó sẽ cố gắng xử lý URL để phù hợp với một route nào đó, với các điều kiện sau đây:

* Các routes patterns định nghĩa hoặc các default route patterns, nếu có trong project.
* Thứ tự thêm chúng vào Routes collection.
* Bất kỳ default value nào định nghĩa cho một route.
* Bất kỳ hạn chế (validate) nào định nghĩa cho một route.
* Xác định một route để request tới một file vật lý.

Để tránh việc sai sót trong xử lý các request, phải xem xét tất cả các điều kiện trên khi định nghĩa route. Cũng nên tránh việc trùng lặp route không cần thiết như ví dụ sau: Ví dụ: chúng ta có các route như sau:

* Route 1 được định nghĩa là: {controller}/{action}/{id}
* Route 2 được định nghĩa là: products/show/{id}.

Route 2 sẽ không bao giờ phải xử lý bất kỳ một request nào từ URL vì Route 1 được định nghĩa trước, và các request phù hợp với route 2 sẽ luôn phù hợp với Route 1. Ví dụ: request http://server/application/products/show/bikes phù hợp với Route 2 nhưng nó sẽ được xử lý bởi route 1 với các giá trị:

* + controller là product
  + action là show
  + id là bikes

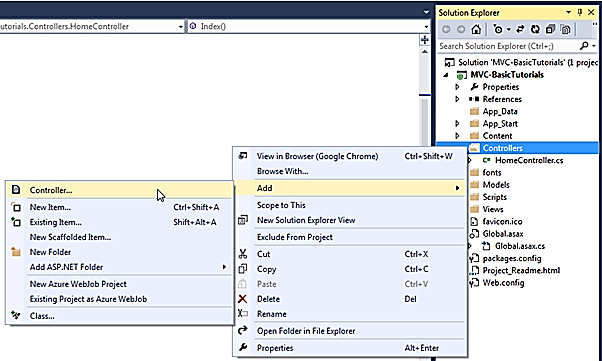
Default values sẽ được sử dụng nếu request không chứa parameter được định nghĩa. Xét ví dụ sau:

* Route 1: {report}/{year}/{month}, với default values là year và month.
* Route 2: {report}/{year}, với default value là year.

Route 2 sẽ không bao giờ được xử lý request, bởi vì với default là year, month thì request phù hợp với Route 2 sẽ luôn phù hợp với Route 1. Điều này có thể trái với mong muốn vì Route 1 là report theo tháng còn Route 2 là report theo năm. Có thể tránh được điều này bằng cách thêm vào một constant. ví dụ: annual/{report}/{year} và monthly/{report}/{year}/{month}.

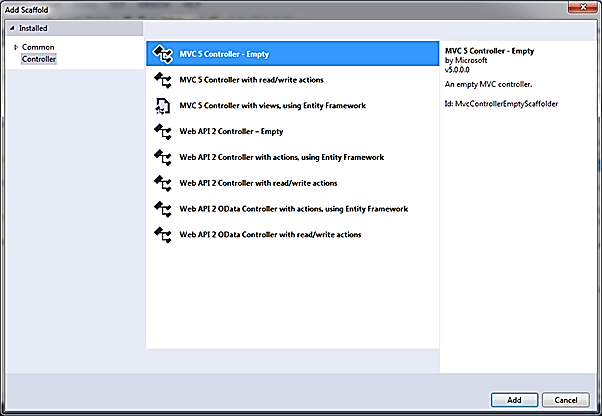
## 1.4. Controller

Trong cửa sổ Solution Explorer của dự án, click chuột phải lên thư mục Controllers\Add\Controller.



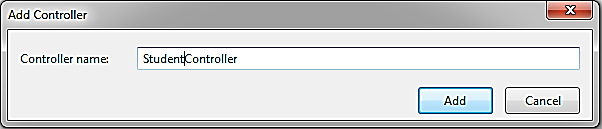
Hình 1.9. Tạo mới Controller

Trong hộp thoại Add Scaffold, chọn [MVC](https://www.dammio.com/glossary/mvc) 5 Controller – Empty\Add. Ở bước này, do chúng ta mới làm quen với Controller cho nên chúng ta sử dụng mẫu Controller dạng Empty, tức là nội dung trống. Từ đó, chúng ta sẽ viết 1 số đoạn code làm quen.



Hình 1.10. Tạo mới MVC Empty Controller

Đặt tên Controller là StudentController, lưu ý nên để tiền tố “Controller” cuối cùng khi đặt tên để dễ phân biệt lớp code nào là Controller.



Hình 1.11. Đặt tên Controller

Ở bước này, thư mục Controller, lớp StudentController.cs được tạo ra với nội dung mặc định như trong hình sau.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

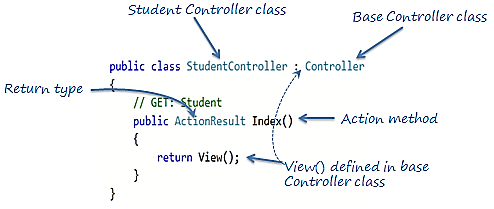
{

return View();

}

}

}



Hình 1.12. Hoạt động của phương thức trong Controllers

Sửa đoạn mã trên như sau:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public string Index()

{

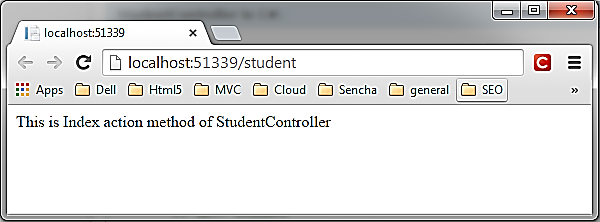
return "This is Index action method of StudentController";

}

}

}

Trong phần định tuyến, URL yêu cầu http://localhost/student hoặc http: //localhost/student/index. Nó được xử lý bởi phương thức Index() của lớp StudentController. Vì vậy, hãy gọi yêu cầu trên từ trình duyệt sẽ thấy kết quả hiển thị như sau.

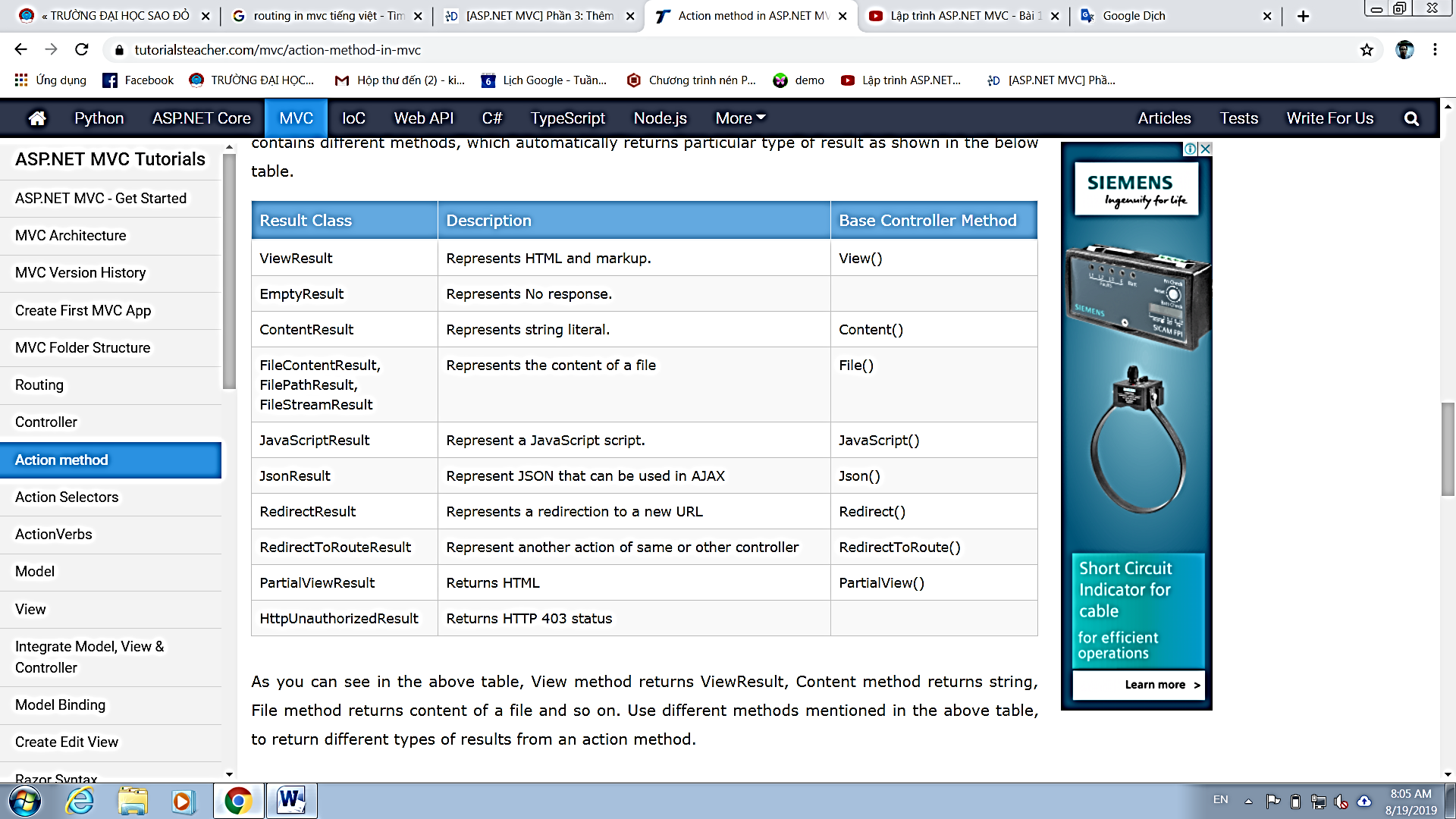


Hình 1.13. Kết quả hiển thị trên trình duyệt

## 1.5. ActionResult

MVC framework gồm các lớp kết quả khác nhau, có thể được trả về từ một phương thức hành động. Có các lớp kết quả đại diện cho các loại phản hồi khác nhau, chẳng hạn như html, tệp, chuỗi, json, javascript, … Bảng sau liệt kê tất cả các lớp kết quả (Result) có sẵn trong ASP.NET MVC.

Bảng 1.3. Các phương thức lớp Result



## 1.6. Action Selectors

Action Selectors là thuộc tính có thể được áp dụng cho các phương thức hành động. Nó giúp công cụ định tuyến chọn phương thức hành động chính xác để xử lý một yêu cầu cụ thể. MVC 5 bao gồm các thuộc tính hành động sau:

* ActionName,
* NonAction,
* ActionVerbs.

ActionName

Thuộc tính ActionName cho phép chúng ta chỉ định một tên hành động khác với tên phương thức. Ví dụ:

public class StudentController : Controller

{

public StudentController()

{

}

[ActionName("find")]

public ActionResult GetById(int id)

{

// get student from the database

return View();

}

}

Trong ví dụ trên, chúng ta đã áp dụng thuộc tính ActionName ("find") cho phương thức hành động GetById. Vì vậy, bây giờ, tên hành động sẽ là "find" thay vì "GetById". Phương thức hành động này được gọi trên trình duyệt là http://localhost/student/find/1 thay cho vì http://localhost/student/getbyid/1.

NonAction

Thuộc tính NonAction chỉ ra rằng phương thức chung của Controller không phải là phương thức hành động. Sử dụng thuộc tính NonAction khi muốn phương thức công khai trong Controller nhưng không muốn coi nó là phương thức hành động.

Ví dụ, phương thức công khai GetStudent() không thể được gọi giống như phương thức hành động trong ví dụ sau.

public class StudentController : Controller

{

public StudentController()

{

}

[NonAction]

public Student GetStudent(int id)

{

return studentList.Where(s => s.StudentId == id).FirstOrDefault();

}

}

ActionVerbs

ActionVerbs được sử dụng khi muốn kiểm soát việc lựa chọn phương thức hành động dựa trên các yêu cầu http. Ví dụ: có thể định nghĩa hai phương thức hành động khác nhau có cùng tên nhưng một phương thức hành động đáp ứng yêu cầu “Nhận bài” và phương thức hành động khác đáp ứng yêu cầu “Đăng bài”.

MVC hỗ trợ các ActionVerbs khác nhau, chẳng hạn như HttpGet, HttpPost, HttpPut, HttpDelete, HttpOptions & HttpPatch. Ta có thể áp dụng các thuộc tính này cho phương thức hành động để chỉ ra loại yêu cầu HTTP mà phương thức hành động hỗ trợ. Nếu không áp dụng bất kỳ thuộc tính nào thì nó sẽ coi đó là một yêu cầu GET theo mặc định.

Bảng 1.4. Các phương thức lớp HTTP

| Phương thức Http | Mô tả |
| --- | --- |
| GET | Để lấy thông tin từ máy chủ. Các tham số sẽ được thêm vào trong chuỗi truy vấn trên thanh địa chỉ trình duyệt. |
| POST | Để tạo một tài nguyên mới. |
| PUT | Để cập nhật tài nguyên hiện có. |
| HEAD | Tương tự với GET ngoại trừ máy chủ không trả về nội dung. |
| OPTIONS | Thể hiện một yêu cầu cung cấp thông tin về các tùy chọn giao tiếp được hỗ trợ bởi máy chủ web. |
| DELETE | Để xóa một tài nguyên hiện có. |
| PATCH | Để cập nhật toàn bộ hoặc một phần tài nguyên. |

Ví dụ sau đây cho thấy các phương thức hành động khác nhau hỗ trợ các ActionVerbs khác nhau:

Ví dụ: ActionVerbs

public class StudentController : Controller

{

public ActionResult Index()

{

return View();

}

[HttpPost]

public ActionResult PostAction()

{

return View("Index");

}

[HttpPut]

public ActionResult PutAction()

{

return View("Index");

}

[HttpDelete]

public ActionResult DeleteAction()

{

return View("Index");

}

[HttpHead]

public ActionResult HeadAction()

{

return View("Index");

}

[HttpOptions]

public ActionResult OptionsAction()

{

return View("Index");

}

[HttpPatch]

public ActionResult PatchAction()

{

return View("Index");

}

}

Cũng có thể áp dụng nhiều động từ HTTP bằng thuộc tính AcceptVerbs. Phương thức GetAndPostAction hỗ trợ cả hai GET và POST ActionVerbs trong ví dụ sau:

Ví dụ: AcceptVerbs

[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]

public ActionResult GetAndPostAction()

{

return RedirectToAction("Index");

}

## 1.7. Model

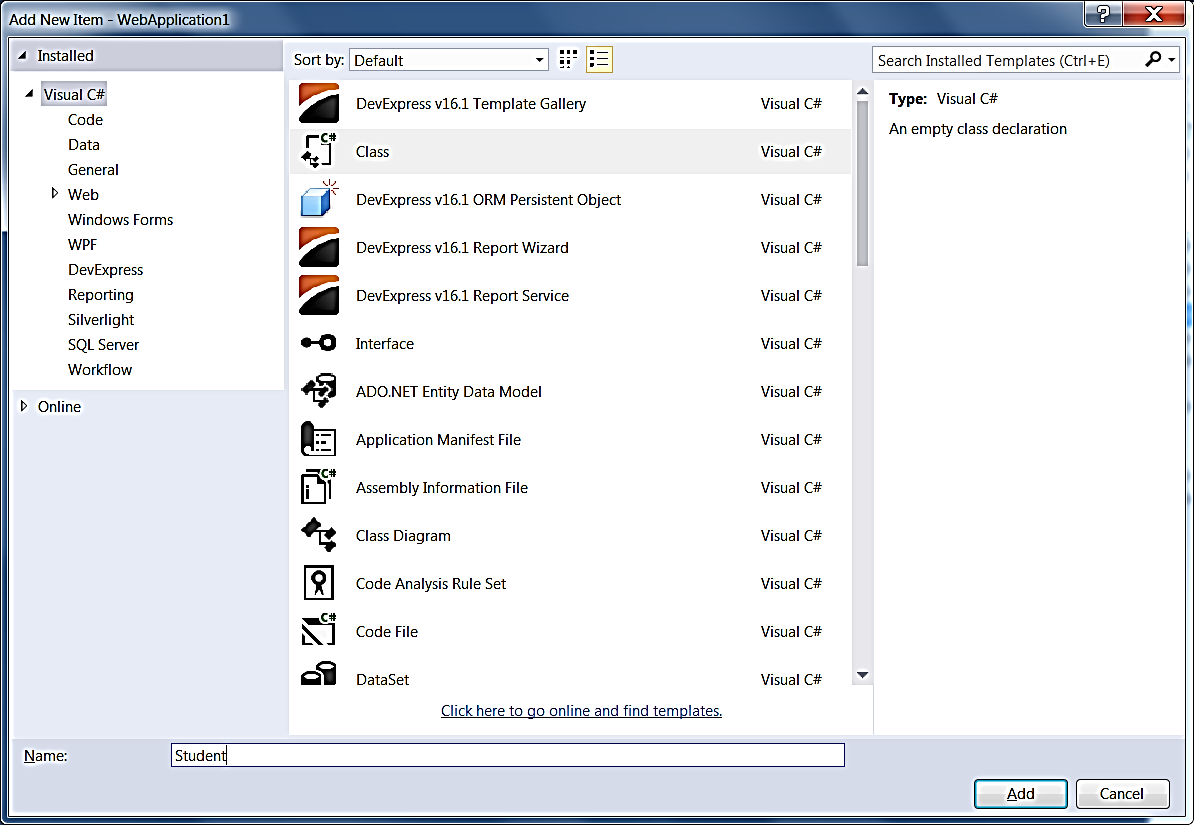
Model đại diện cho dữ liệu cụ thể của miền và logic nghiệp vụ trong kiến trúc MVC. Nó duy trì dữ liệu của ứng dụng. Các đối tượng Model lấy và lưu trữ trạng thái mô hình trong kho lưu trữ như cơ sở dữ liệu.

Lớp Model chứa dữ liệu trong các thuộc tính công cộng. Tất cả các lớp Model nằm trong thư mục Model trong cấu trúc thư mục MVC.

Thêm mới Model

- Click chuột phải vào thư mục Model\Add\Class…

- Trong hộp thoại Add new, nhập tên lớp 'Student' và chọn Add.



Hình 1.14. Thêm mới Model

Điều này sẽ thêm lớp Student mới trong thư mục Model. Bây giờ, thêm các thuộc tính Id, Name, Age như sau.

namespace MVC\_BasicTutorials.Models

{

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

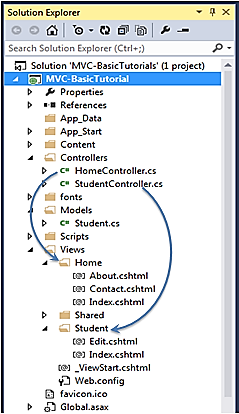
}

Theo cách này, ta có thể tạo một lớp Model và có thể sử dụng trong View.

## 1.8. View

View là một giao diện người dùng. View hiển thị dữ liệu từ Model cho người dùng và cũng cho phép người dùng sửa đổi dữ liệu.

Các View trong mô hình MVC được lưu trong thư mục Views. Các phương thức hành động khác nhau của một Controller có thể hiển thị các View khác nhau. Do đó thư mục Views chứa một thư mục riêng cho mỗi Controller tương ứng để phù hợp với các phương thức hành động của Controller. Ví dụ: các View tương ứng với HomeController nằm trong thư mục Views\Home. Theo cách tương tự, các View tương ứng với StudentController nằm trong Views\Student như hình sau:



Hình 1.15. Cấu trúc thư mục MVC

### 1.8.1. Razor View Engine

Microsoft đã giới thiệu công cụ Razor và được đóng gói từ MVC 3. Có thể viết hỗn hợp các thẻ html và mã phía máy chủ trong chế độ Razor View. Razor sử dụng ký tự @ cho mã phía máy chủ thay vì <%%> truyền thống. Ta có thể sử dụng cú pháp C# hoặc Visual Basic để viết mã phía máy chủ bên trong chế độ Razor view. Công cụ Razor tối đa hóa tốc độ viết mã bằng cách giảm thiểu số lượng ký tự và tổ hợp phím cần thiết khi viết code trong View. Các tệp Razor View có phần mở rộng .cshtml hoặc vbhtml. ASP.NET MVC hỗ trợ các loại tệp sau:

Bảng 1.5. Các loại tệp trong View

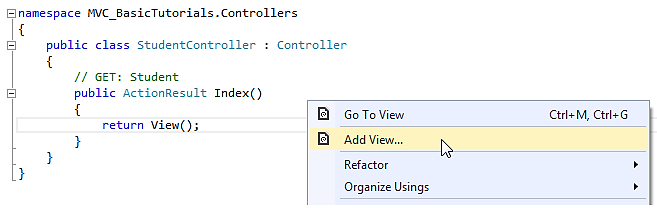
| Phần mở rộng | Mô tả |
| --- | --- |
| .cshtml | Razor View sử dụng C#. Hỗ trợ HTML. |
| .vbhtml | Razor View sử dụng Visual Basic. Hỗ trợ HTML. |
| .aspx | ASP.Net web form |
| .ascx | ASP.NET web control |

### 1.8.2. Thêm mới View

Chúng ta đã tạo mô hình StudentController và Student trong phần trước. Bây giờ, hãy tạo Student view và hiểu cách sử dụng Model vào View. Chúng ta sẽ tạo một View để hiển thị từ phương thức Index của StudentController.

- Mở lớp StudentController,

- Click chuột phải vào bên trong Phương thức Index\Add View …

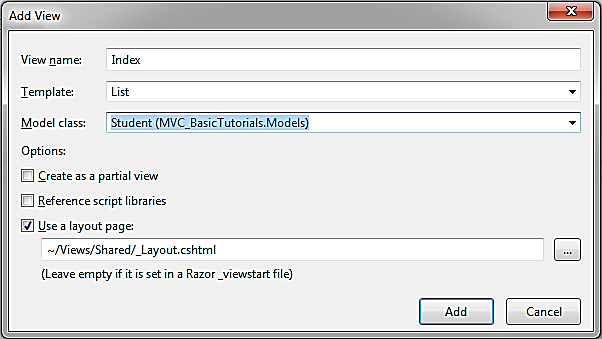


Hình 1.16. Add View

- Trong hộp thoại Add View, giữ tên View là Index. Tốt nhất là giữ tên View giống như tên phương thức hành động của Controller để không phải chỉ định rõ ràng tên View của phương thức hành động trong khi trả về dạng View.

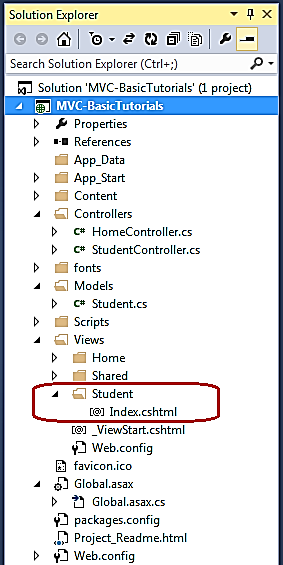
- Chọn mẫu cho các hành động ứng với View. Trình đơn thả xuống mẫu sẽ hiển thị các mẫu mặc định có sẵn cho Create, Delete, Details, Edit, List hoặc Empty view. Chọn mẫu "List" để hiển thị danh sách của Student (sinh viên) trong chế độ View.

- Chọn Student từ Model class. Trình đơn tự động hiển thị tên của tất cả các lớp trong thư mục Model. Chúng ta đã tạo lớp Student Model trong phần trước, vì vậy nó sẽ được đưa vào danh sách thả xuống.



Hình 1.17. Hộp thoại Add View

- Chọn hộp kiểm "Use a layout page" và chọn trang \_Layout.cshtml cho chế độ view này, sau đó click nút Add (Sau này chúng ta sẽ tìm hiểu trang bố cục - layout page là gì). Điều này sẽ tạo một View trong thư mục Views\Student như sau:



Hình 1.18. File View trong thư mục

Nội dung Index View (trang Index.cshtml) được tạo ở trên như sau.

@model IEnumerable<MVC\_BasicTutorials.Models.Student>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.StudentName)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.StudentName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |

@Html.ActionLink("Details", "Details", new { id=item.StudentId }) |

@Html.ActionLink("Delete", "Delete", new { id = item.StudentId })

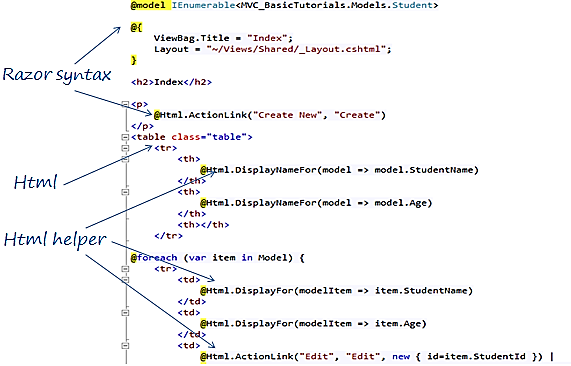
</td>

</tr>

}

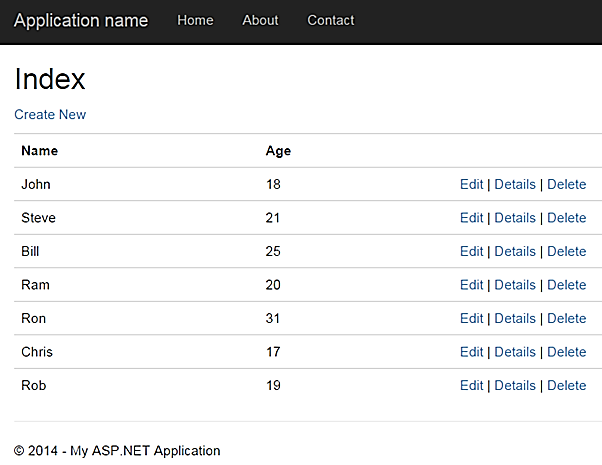
</table>

Như ta thấy trong Index View ở trên, nó chứa cả mã Html và Rator. Dấu hiệu Rator nội tuyến bắt đầu bằng ký hiệu @. @Html là một lớp trợ giúp (HTML Helpers) để tạo các điều khiển html. Chúng ta sẽ tìm hiểu cú pháp Razor và lớp HTML Helpers trong các phần sau.



Hình 1.19. Mô tả cấu trúc View

Kết quả khi chạy trang Index View ở trên sẽ có kết quả sau:



Hình 1.20. Index View

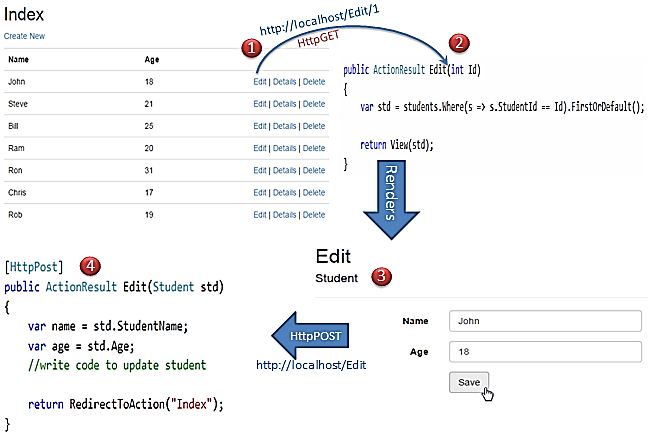
### 1.8.3. Tạo Edit View

Chúng ta đã tạo Index View trong phần trước. Trong phần này, chúng ta sẽ tạo Edit View bằng cách sử dụng mẫu phương thức hành động mặc định. Người dùng có thể cập nhật dữ liệu cho Student hiện có bằng cách sử dụng Edit View.



Hình 1.21. Edit view

Edit View sẽ được hiển thị khi nhấp vào nút Edit trong Index View. Hình sau mô tả các bước chỉnh sửa.



Hình 1.22. Mô tả các bước thực hiện chỉnh sửa

1. Người dùng nhấp vào liên kết Edit trong Index View, sẽ gửi yêu cầu httpGET (http://localhost/student/edit/{Id}) với tham số Id tương ứng trong chuỗi truy vấn. Yêu cầu này sẽ được xử lý bằng phương thức hành động Edit httpGET (theo phương thức hành động mặc định xử lý yêu cầu HttpGET nếu không có thuộc tính được chỉ định).

2. Phương thức hành động Edit httpGet sẽ lấy dữ liệu Student cụ thể từ cơ sở dữ liệu dựa trên tham số Id được cung cấp và hiển thị lên Edit View.

3. Người dùng có thể chỉnh sửa dữ liệu và nhấp vào nút Save trong chế độ Edit. Nút Save sẽ gửi yêu cầu httpPOST (http://localhost/Student/Edit) và truyền dữ liệu biểu mẫu (Model).

4. Phương thức hành động Edit httpPOST trong StudentController sẽ cập nhật dữ liệu vào cơ sở dữ liệu và hiển thị trang Index (Index View) với dữ liệu được làm mới bằng phương pháp RedirectToAction.

Trên đây là quá trình hoàn chỉnh để chỉnh sửa dữ liệu bằng cách sử dụng Edit View trong ASP.NET MVC. Để áp dụng vào ứng dụng, chúng ta cần thực hiện các công việc sau:

**Bước 1** - Tạo Student Model - C#: Bước này đã được thực hiện trong quá trình tạo Index view

public class Student

{

public int StudentId { get; set; }

[Display( Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

}

**Bước 2**: Tạo một phương thức hành động Edit httpGET trong StudentController. Khung nhìn Index hiển thị ở trên sẽ gửi tham số StudentId đến phương thức hành động Edit HTTPGet khi nhấp vào liên kết Edit.

Phương thức hành động Edit() của HTTPGet phải thực hiện hai tác vụ, trước tiên, nó sẽ tìm nạp thông tin Student từ studentList, có StudentId khớp với StudentId trong chuỗi truy vấn. Thứ hai, nó sẽ hiển thị thông tin lên Edit View để người dùng có thể cập nhật.

Vì vậy, phương thức hành động Edit() nên có tham số StudentId. MVC sẽ tự động liên kết một chuỗi truy vấn với các tham số của phương thức hành động nếu tên đó khớp với Routing, đảm bảo rằng tên tham số khớp với chuỗi truy vấn.

HttpGet Edit() Action method - C#

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentId=1, StudentName="John", Age = 18 },

new Student(){ StudentId=2, StudentName="Steve", Age = 21 },

new Student(){ StudentId=3, StudentName="Bill", Age = 25 },

new Student(){ StudentId=4, StudentName="Ram", Age = 20 },

new Student(){ StudentId=5, StudentName="Ron", Age = 31 },

new Student(){ StudentId=6, StudentName="Chris", Age = 17 },

new Student(){ StudentId=7, StudentName="Rob", Age = 19 }

};

public ActionResult Edit(int Id)

{

//Get the student from studentList sample collection for demo purpose.

//Get the student from the database in the real application

var std = studentList.Where(s => s.StudentId == Id).FirstOrDefault();

return View(std);

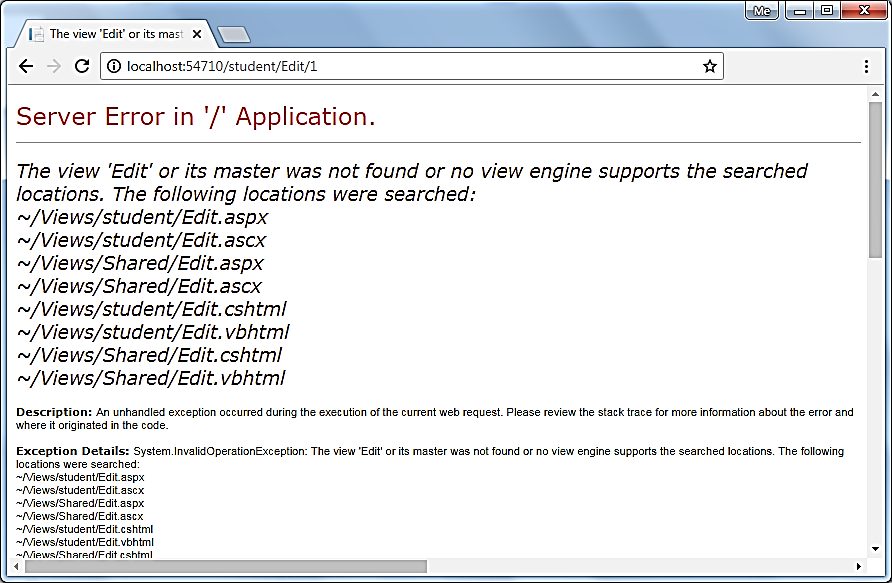
}

}

}

Có thể thấy trong phương thức Edit ở trên, chúng ta đã sử dụng truy vấn LINQ để đưa thông tin Student từ studentList có StudentId khớp với StudentId được cung cấp, sau đó đưa đối tượng Student đó vào View. Trong một ứng dụng thực tế, ta có thể lấy Student từ cơ sở dữ liệu thay vì tạo mẫu đặt trong studentList.

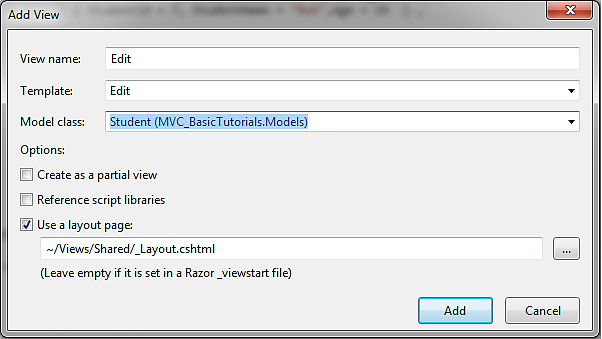
Bây giờ, nếu ta nhấp vào liên kết Edit từ trang Index View thì sẽ gặp lỗi sau.



Hình 1.23. Trang lỗi

Có lỗi ở trên vì chúng ta chưa tạo trang Edit View. Theo mặc định, MVC sẽ tìm tệp Edit.cshtml hoặc Edit.vbhtml hoặc Edit.aspx hoặc Edit.ascx trong thư mục View\Student hoặc Shared.

**Bước 3:** Để tạo trang Edit View, click chuột phải vào bên trong phương thức hành động Edit() chọn Add View ... Trong hộp thoại Add View, giữ tên View là Edit. Chọn Edit trong danh sách Template và cũng chọn class Student như hình dưới. Click nút Add.



Hình 1.24. Tạo view

Nội dung Edit View (trang Edit.cshtml) như sau:

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Student</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.StudentId)

<div class="form-group">

@Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger"< })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default"/>

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

Lưu ý: Edit.cshtml bao gồm phương thức HtmlHelper @USE (Html.BeginForm ()) để tạo một phần tử biểu mẫu html. Html.BeginForm gửi một yêu cầu HTTPPost theo mặc định.

Bây giờ, nhấp vào liên kết Edit của bất kỳ sinh viên nào trong danh sách. Edit View sẽ được hiển thị thông tin sinh viên có liên kết Edit được nhấp, như hiển thị bên dưới.

Ta có thể chỉnh sửa thông tin của sinh viên và nhấp vào Save. Phương thức Save() phải gửi yêu cầu HttpPOST vì yêu cầu POST gửi dữ liệu Form như một phần của yêu cầu, không phải trong chuỗi truy vấn. Do đó ta cần tạo một phương thức POST để xử lý hành động người dùng nhấn nút Save.

**Bước 4**: Viết phương thức hành động httpPOST Edit để lưu học sinh đã chỉnh sửa như sau:

[HttpPost]

public ActionResult Edit(Student std)

{

//write code to update student

return RedirectToAction("Index");

}

Trong đoạn mã trên, phương thức Edit() yêu cầu một đối tượng Student làm tham số đầu vào. Edit View sẽ tự động liên kết danh sách dữ liệu của Form với tham số Student Model. Tại đây, ta có thể cập nhật thông tin vào cơ sở dữ liệu và chuyển hướng nó sang hành động Index.

Bây giờ, nhấp vào nút Save trong trang Edit View sẽ lưu thông tin cập nhật và chuyển hướng nó sang phương thức hành động Index(). Theo cách này, ta có thể cung cấp chức năng chỉnh sửa bằng cách sử dụng mẫu Edit mặc định. Tuy nhiên, ta cũng có thể tạo Edit View riêng mà không cần sử dụng mẫu Edit có sẵn.

Sau đây là lớp StudentController với phương thức hành động Edit() đầy đủ.

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentId=1, StudentName="John", Age = 18 },

new Student(){ StudentId=2, StudentName="Steve", Age = 21 },

new Student(){ StudentId=3, StudentName="Bill", Age = 25 },

new Student(){ StudentId=4, StudentName="Ram", Age = 20 },

new Student(){ StudentId=5, StudentName="Ron", Age = 31 },

new Student(){ StudentId=6, StudentName="Chris", Age = 17 },

new Student(){ StudentId=7, StudentName="Rob", Age = 19 }

};

// GET: Student

public ActionResult Index()

{

return View(studentList);

}

public ActionResult Edit(int Id)

{

//Get the student from studentList sample collection for demo purpose.

//Get the student from the database in the real application

var std = studentList.Where(s => s.StudentId == Id).FirstOrDefault();

return View(std);

}

[HttpPost]

public ActionResult Edit(Student std)

{

//write code to update student

return RedirectToAction("Index");

}

}

}

## 1.9. Tích hợp Controller, View và Model

Chúng ta đã tạo StudentController, Model và View trong các phần trước, nhưng chúng ta chưa tích hợp tất cả các thành phần này với nhau để hoàn thành một chu trình. Dưới đây là đoạn mã của các thành phần StudentController, Student Model và Student View được tạo trong các phần trước.

StudentController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

return View();

}

}

}

Student Model class

namespace MVC\_BasicTutorials.Models

{

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

}

Student View (Index.cshtml hiển thị danh sách sinh viên)

@model IEnumerable<MVC\_BasicTutorials.Models.Student>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.StudentName)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.StudentName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |

@Html.ActionLink("Details", "Details", new { id=item.StudentId }) |

@Html.ActionLink("Delete", "Delete", new { id = item.StudentId })

</td>

</tr>

}

</table>

Bây giờ, để chạy nó thành công, chúng ta cần chuyển một đối tượng mô hình (Model) từ Controller sang View Index. Như chúng ta thấy trong Index.cshtml ở trên, nó sử dụng IEnumerable của lớp Student làm đối tượng mô hình. Vì vậy, chúng ta cần truyền dữ liệu qua Student Model từ phương thức hành động Index của lớp StudentController như sau.

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

var studentList = new List<Student>{

new Student() { StudentId = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentId = 2, StudentName = "Steve", Age = 21 } ,

new Student() { StudentId = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentId = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentId = 5, StudentName = "Ron" , Age = 31 } ,

new Student() { StudentId = 4, StudentName = "Chris" , Age = 17 } ,

new Student() { StudentId = 4, StudentName = "Rob" , Age = 19 }

};

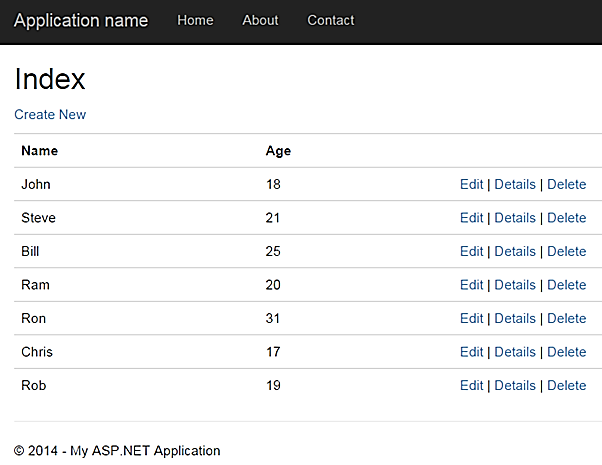
// Get the students from the database in the real application

return View(studentList);

}

}

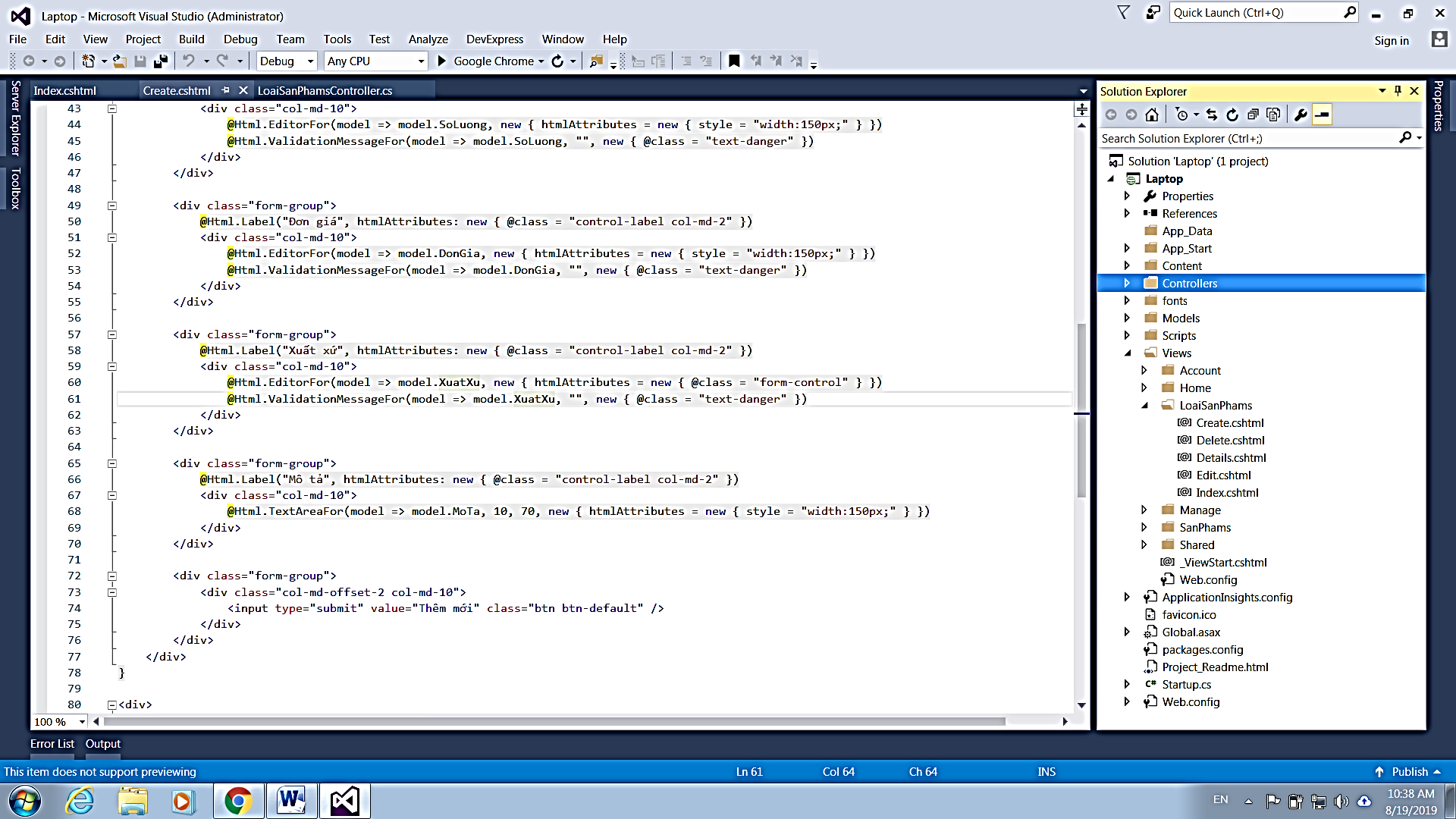
Có thể thấy trong đoạn mã trên, chúng ta đã tạo danh sách các đối tượng sinh viên studentList (trong ứng dụng thực tế, ta có thể lấy nó từ cơ sở dữ liệu). Sau đó chúng ta truyền đối tượng danh sách này làm tham số trong phương thức View(). Phương thức View() được định nghĩa trong lớp Controller, tự động liên kết đối tượng mô hình với khung nhìn. Bây giờ, có thể chạy dự án MVC bằng cách nhấn F5 và điều hướng đến http://localhost/Student. Ta sẽ thấy chế độ xem sau trong trình duyệt.

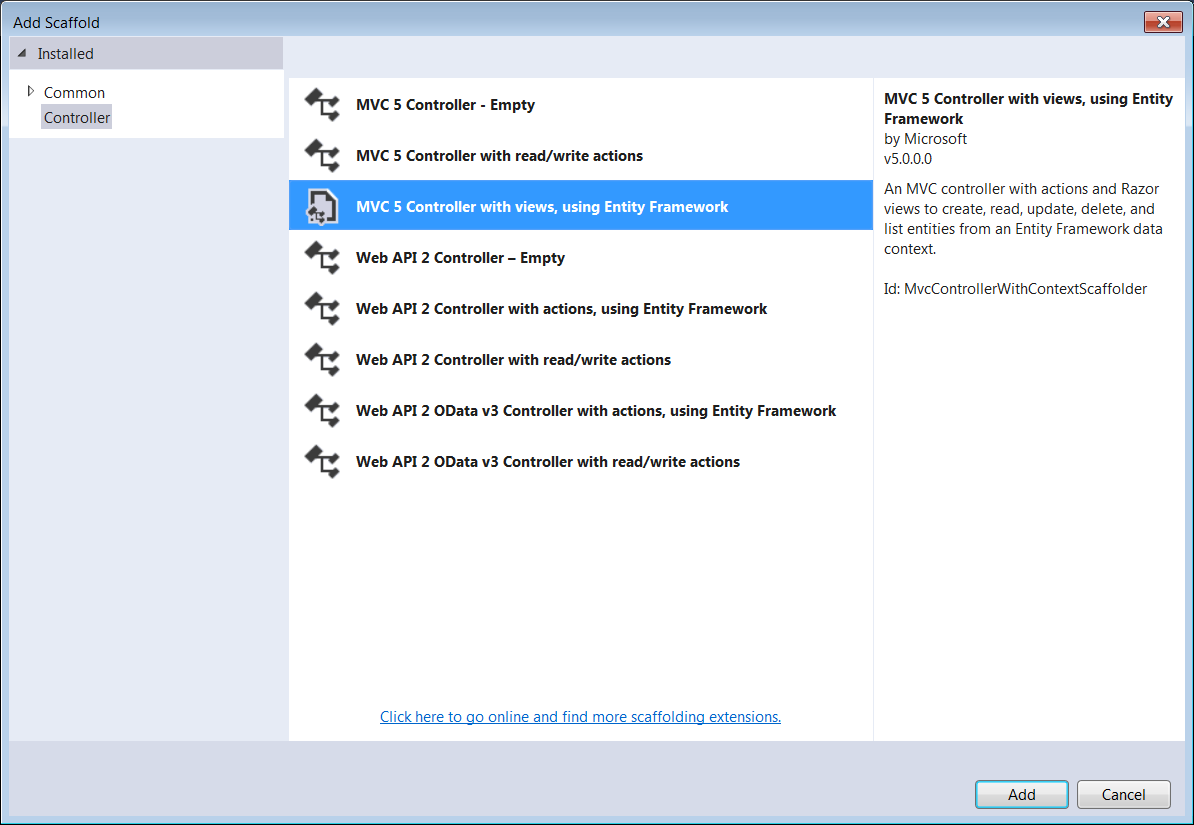


Hình 1.25. Trang Index

Tự động sinh các phương thức trong view

Khi chúng ta tạo Controller, chúng ta có thể chọn chế độ sinh các phương thức và View tự động bằng cách chọn MVC 5 Controller with views using Entity Framework. Visual Studio sẽ tự động tạo các View tương ứng với các phương thức cơ bản gồm: Create, Delete, Details, Edit và Index trong Controller.

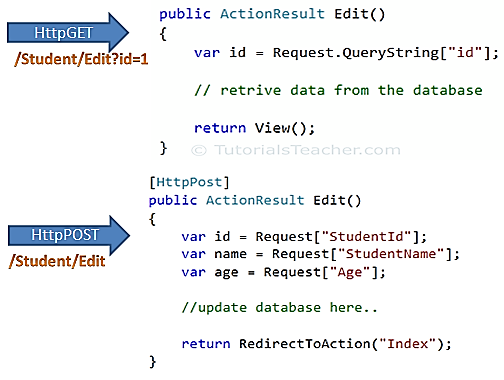
**



Hình 1.26. Tạo Controller with views

## 1.10. Mô hình liên kết dữ liệu - Model Binding

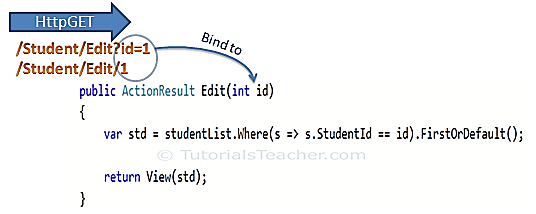
Để hiểu ràng buộc mô hình trong MVC, trước tiên hãy xem cách ta có thể nhận các giá trị yêu cầu http trong phương thức hành động bằng cách sử dụng kiểu ASP.NET truyền thống. Hình dưới mô tả cách ta có thể nhận các giá trị từ yêu cầu HttpGET và HttpPOST bằng cách sử dụng trực tiếp đối tượng Request trong phương thức hành động.



Hình 1.27. Mô tả cách thức nhận dữ liệu thông qua đối tượng Request

Trong hình trên, chúng ta sử dụng đối tượng Request.QueryString và Request (Request.Form) để nhận giá trị từ yêu cầu HttpGet và HttpPOST. Truy cập các giá trị yêu cầu bằng cách sử dụng đối tượng Request là một hoạt động cồng kềnh và lãng phí thời gian. Với liên kết mô hình, MVC chuyển đổi các giá trị yêu cầu http (từ chuỗi truy vấn hoặc bộ sưu tập biểu mẫu) thành các tham số phương thức hành động. Các tham số này có thể là loại nguyên thủy hoặc loại phức tạp.

Yêu cầu httpGET nhúng dữ liệu vào chuỗi truy vấn. MVC tự động chuyển đổi một chuỗi truy vấn thành các tham số của phương thức hành động. Ví dụ: chuỗi truy vấn "id" trong yêu cầu GET sau đây sẽ tự động được ánh xạ tới tham số id của phương thức hành động Edit().



Hình 1.28. Mô tả ánh xạ tham số của phương thức hành động

Có thể có nhiều tham số trong phương thức hành động với các loại dữ liệu khác nhau. Các giá trị chuỗi truy vấn sẽ được chuyển đổi thành tham số dựa trên tên phù hợp.

Ví dụ: http://localhost/Student/Edit?Id=1&name=John sẽ ánh xạ tới tham số id và tên của phương thức hành động Edit sau.

public ActionResult Edit(int id, string name)

{

// do something here

return View();

}

Có thể truyền các mô hình dữ liệu phức tạp. Liên kết mô hình trong MVC tự động chuyển đổi dữ liệu trường biểu mẫu của yêu cầu HttpPOST thành các thuộc tính của một tham số loại phức tạp của một phương thức hành động. Ví dụ chúng ta có Model classes sau:

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public Standard standard { get; set; }

}

public class Standard

{

public int StandardId { get; set; }

public string StandardName { get; set; }

}

Chúng ta có thể tạo phương thức hành động với tham số kiểu class để nhận giá trị từ form truyền sang như sau:

[HttpPost]

public ActionResult Edit(Student std)

{

var id = std.StudentId;

var name = std.StudentName;

var age = std.Age;

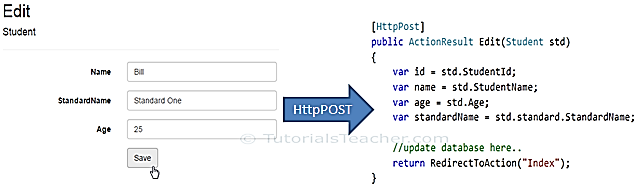
var standardName = std.standard.StandardName;

//update database here..

return RedirectToAction("Index");

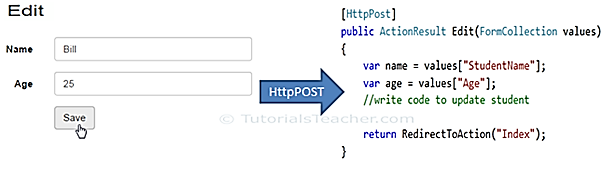
}

Bây giờ, MVC sẽ tự động ánh xạ các giá trị biểu mẫu sang tham số loại Student khi biểu mẫu gửi yêu cầu httpPOST đến phương thức hành động Edit như được mô tả như sau:

**

Hình 1.29. Phương thức hành động với tham số kiểu class

Ngoài việc sử dụng tham số kiểu class để nhận dữ liệu từ form, chúng ta cũng có thể tạo phương thức hành động sử dụng tham số của Form như cách truyền thống như sau:

**

Hình 1.30. Phương thức hành động với tham số kiểu Form

**Thuộc tính ràng buộc**

ASP.NET MVC Framework cho phép chỉ định các thuộc tính nào của lớp mô hình mà muốn liên kết. Thuộc tính [Bind] sẽ cho phép chỉ định các thuộc tính để truyền vào các phương thức.

Trong ví dụ sau, phương thức hành động Edit sẽ nhúng thuộc tính StudentId và StudentName của Student Model.

[HttpPost]

public ActionResult Edit([Bind(Include = "StudentId, StudentName")] Student std)

{

var name = std.StudentName;

//write code to update student

return RedirectToAction("Index");

}

Chúng ta cũng có thể sử dụng các thuộc tính loại trừ như dưới đây.

[HttpPost]

public ActionResult Edit([Bind(Exclude = "Age")] Student std)

{

var name = std.StudentName;

//write code to update student

return RedirectToAction("Index");

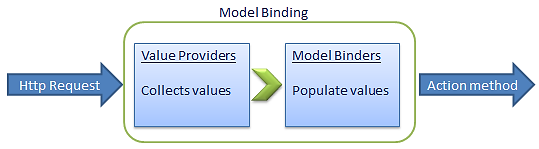
}

Thuộc tính Bind sẽ cải thiện hiệu suất bằng cách chỉ liên kết các thuộc tính được chỉ ra trong tham số [Bind] của phương thức.

Bên trong mô hình liên kết

Như trên đã trình bày, mô hình liên kết tự động chuyển đổi các giá trị yêu cầu thành một đối tượng kiểu nguyên thủy hoặc phức tạp. Mô hình liên kết là một quá trình hai bước. Đầu tiên, nó thu thập các giá trị từ yêu cầu http đến. Thứ hai, điền vào loại nguyên thủy hoặc loại phức tạp với các giá trị này.

Các lớp cung cấp giá trị chịu trách nhiệm thu thập các giá trị từ yêu cầu và mô hình liên kết chịu trách nhiệm trả các giá trị cho phương thức hành động.



Hình 1.31. Mô hình liên kết trong MVC

Các lớp cung cấp giá trị mặc định từ các nguồn sau:

- Các tham số hành động bị ràng buộc trước đó nếu là hành động con.

- Các trường mẫu (Request.Form)

- Các giá trị thuộc tính trong phần thân yêu cầu JSON hoặc AJAX (Request.InputStream).

- Dữ liệu tuyến đường (RouteData.Values).

- Các tham số chuỗi truy vấn (Request.QueryString).

- Các tập tin đã đăng (Request.Files).

# CÂU HỎI ÔN TẬP CHƯƠNG 1

1. Tạo ứng dụng ASP.NET MVC quản lý album ảnh với các yêu cầu sau:

- Tạo View để hiển thị các hình ảnh trong thư mục Uploaded của ứng dụng.

- Tạo Controller xử lý upload ảnh, xóa ảnh.

**

2. Tạo ứng dụng ASP.NET MVC quản lý nhà hàng với các yêu cầu sau:

- Tạo Controller gồm các Action để thực hiện: hiển thị danh sách món ăn, thêm, xóa, sửa thông tin món ăn.

- Tạo các View tương ứng cho Controller trên.

3. Tạo ứng dụng ASP.NET MVC quản lý hồ sơ sinh viên với các yêu cầu sau:

- Tạo Controller gồm các Action để thực hiện: hiển thị, thêm, xóa, sửa thông tin sinh viên.

- Tạo các View tương ứng cho Controller trên.

4. Tạo ứng dụng ASP.NET MVC quản lý lớp học với các yêu cầu sau:

- Tạo Controller gồm các Action để thực hiện: hiển thị, thêm, xóa, sửa thông tin lớp học.

- Tạo các View tương ứng cho Controller trên.

# Chương 2. RAZOR VIEW ENGINE VÀ HTML HELPERS

## 2.1. Razor

Razor là một trong những công cụ được hỗ trợ trong ASP.NET MVC. Razor cho phép viết hỗn hợp mã HTML và mã phía máy chủ bằng C# hoặc Visual Basic. Razor view với cú pháp cơ bản trực quan có phần mở rộng .vbhtml và cú pháp C# có phần mở rộng .cshtml. Trong Razor có phân biệt chữ hoa, chữ thường. Cú pháp Razor có các đặc điểm sau:

**- Compact**: Cú pháp Razor nhỏ gọn cho phép giảm thiểu số lượng ký tự và tổ hợp phím cần thiết để viết mã.

**- Easy to Learn**: Cú pháp Razor rất dễ học, có thể sử dụng ngôn ngữ quen thuộc C # hoặc Visual Basic.

**- Intellisense**: Cú pháp Razor hỗ trợ hoàn thành câu lệnh trong Visual Studio.

### 2.1.1. Biểu thức nội tuyến

Bắt đầu bằng ký hiệu @ để viết mã C# hoặc VB phía máy chủ trong mã html. Ví dụ: viết @Variable\_Name để hiển thị giá trị của biến phía máy chủ. Hoặc thay vì sử dụng hàm DateTime.Now trả về ngày và giờ hiện tại, ta dùng @DateTime. Biểu thức một dòng không yêu cầu dấu chấm phẩy ở cuối biểu thức.

<h1>Razor syntax demo</h1>

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

Output:

**Razor syntax demo**

08-09-2014

**Khối lệnh:** được đặt trong cặp @{….}

@{

var date = DateTime.Now.ToShortDateString();

var message = "Hello World";

}

<h2>Today's date is: @date </h2>

<h3>@message</h3>

Kết hợp với thẻ html

@{

var date = DateTime.Now.ToShortDateString();

string message = "Hello World!";

<text>Today's date is:</text> @date <br />

@message

}

### 2.1.2. Khai báo biến

Khai báo một biến trong một khối mã được đặt trong cặp 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>

### 2.1.3. Các cấu trúc lệnh

Cấu trúc if … else:

@if(DateTime.IsLeapYear(DateTime.Now.Year) )

{

@DateTime.Now.Year @:is a leap year.

}

else {

@DateTime.Now.Year @:is not a leap year.

}

Cấu trúc for:

@for (int i = 0; i < 5; i++) {

@i.ToString() <br />

}

### 2.1.4. Mô hình – Model

Sử dụng @model để sử dụng đối tượng mô hình 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>

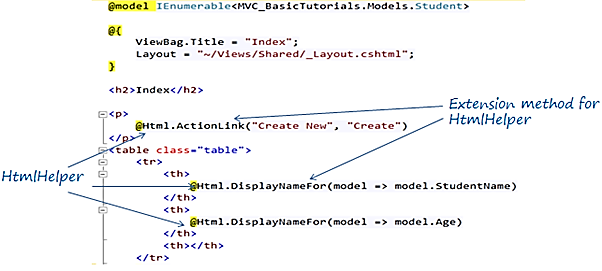
<li>Age: @Model.Age</li>

</ul>

## 2.2. HTMLHelpers

### 2.2.1. Giới thiệu

Lớp HtmlHelper tạo các phần tử html bằng cách sử dụng đối tượng lớp mô hình trong Razor view. Nó liên kết đối tượng mô hình với các phần tử html để hiển thị giá trị của các thuộc tính mô hình thành các phần tử html và cũng gán giá trị của các phần tử html cho các thuộc tính mô hình trong khi gửi từ web form. Vì vậy chúng ta nên luôn sử dụng lớp HtmlHelper trong chế độ Razor view thay vì viết các thẻ html thủ công. Hình dưới đây cho thấy việc sử dụng lớp HtmlHelper trong chế độ Razor view.

**

Hình 2.1. Mô tả một số thành phần lớp HtmlHelper

Trong hình trên, @Html là một đối tượng của lớp HtmlHelper. (Biểu tượng @ được sử dụng để truy cập đối tượng phía máy chủ theo cú pháp Razor). Html là một thuộc tính của HtmlHelper trong lớp cơ sở của chế độ Razor view. ActionLink() và DisplayNameFor() là các phương thức mở rộng trong lớp HtmlHelper.

Lớp HtmlHelper tạo các phần tử html. Ví dụ: @Html.ActionLink("Create New", "Create") sẽ tạo thẻ liên kết tương ứng <a href="/Student/Create">Create New</a>.

Có nhiều phương thức mở rộng của lớp HtmlHelper tạo ra các điều khiển html khác nhau. Bảng sau liệt kê các phương thức HtmlHelper và điều khiển html mỗi phương thức tạo ra.

Bảng 2.1. Các phương thức mở rộng của lớp HtmlHelper.

| Phương thức lớp HtmlHelper | | Điều khiển Html tương ứng |
| --- | --- | --- |
| Html.ActionLink |  | Thẻ liên kết |
| Html.TextBox | Html.TextBoxFor | Textbox |
| Html.TextArea | Html.TextAreaFor | TextArea |
| Html.CheckBox | Html.CheckBoxFor | Checkbox |
| Html.RadioButton | Html.RadioButtonFor | Radio button |
| Html.DropDownList | Html.DropDownListFor | Dropdown, combobox |
| Html.ListBox | Html.ListBoxFor | multi-select list box |
| Html.Hidden | Html.HiddenFor | Hidden field |
| Html.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 mô hình được chỉ định, ví dụ: hộp văn bản cho thuộc tính chuỗi, trường số cho kiểu int, double hoặc kiểu số khác. |

Sự khác biệt giữa việc gọi các phương thức HtmlHelper và sử dụng thẻ html là phương thức HtmlHelper được thiết kế để giúp dễ dàng liên kết để xem dữ liệu hoặc dữ liệu mô hình.

### 2.2.2. HtmlHelper - TextBox

Lớp HtmlHelper bao gồm hai phương thức mở rộng tạo ra một phần tử hộp văn bản (<input type="text">) trong chế độ Razor view: TextBox() và TextBoxFor(). Phương thức TextBox() là phương thức không kiểm tra dữ liệu nhập trong khi TextBoxFor() là phương thức có sử dụng kiểm tra dữ liệu.

Chúng ta sẽ sử dụng mô hình Student với phương thức TextBox() và TextBoxFor() như sau:

Student Model

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; }

}

**TextBox():**

TextBox() method

MvcHtmlString Html.TextBox(string name, string value, object htmlAttributes)

Html.TextBox() trong Razor view

@model Student

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

Html Result:

<input class="form-control"

id="StudentName"

name="StudentName"

type="text"

value="" />

**TextBoxFor():**

TextBoxFor() method

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

TextBoxFor() trong Razor view

@model Student

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

Html Result:

<input class="form-control"

id="StudentName"

name="StudentName"

type="text"

value="John" />

Trong ví dụ trên, tham số đầu tiên trong phương thức TextBoxFor() là biểu thức chỉ định thuộc tính StudentName để liên kết với hộp văn bản. Nó tạo ra một thành phần văn bản đầu vào với id và name được đặt thành tên các thuộc tính của textbox. Thuộc tính value sẽ được đặt thành giá trị của thuộc tính StudentName, ví dụ John.

Sự khác biệt giữa TextBox và TextBoxFor

- @Html.TextBox() là phương thức nhập dữ liệu lỏng lẻo trong khi @Html.TextBoxFor() là phương thức mở rộng được nhập dữ liệu mạnh (có kiểm tra dữ liệu nhập vào).

- TextBox() yêu cầu tên thuộc tính dưới dạng tham số chuỗi trong đó TextBoxFor() yêu cầu biểu thức **lambda** làm tham số.

- TextBox không cung cấp lỗi thời gian biên dịch nếu 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 lỗi biên dịch thời gian nếu chỉ định thay đổi tên thuộc tính hoặc tên thuộc tính sai (chế độ View không được biên dịch trong thời gian chạy).

### 2.2.3. HtmlHelper - TextArea

Lớp HtmlHelper bao gồm hai phương thức mở rộng để tạo phần tử <textarea> nhiều dòng trong chế độ Razor view: TextArea() và TextAreaFor(). Theo mặc định, nó tạo ra textarea với hàng = 2 và cột = 20.

Chúng ta sẽ sử dụng mô hình Student  ở trên với phương thức TextArea() và TextAreaFor().

**TextArea():**

TextArea() method

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

Html.TextArea() trong Razor view

@model Student

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

Html Result:

<textarea class="form-control"

id="Description"

name="Description"

rows="2"

cols="20">This is value</textarea>

**TextBoxFor():**

TextBoxFor() method

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

TextAreaFor() trong Razor view

@model Student

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

Html Result:

<textarea class="form-control"

cols="20"

id="Description"

name="Description"

rows="2"></textarea>

### 2.2.4. HtmlHelper - CheckBox

Lớp HtmlHelper bao gồm hai phương thức mở rộng để tạo phần tử <input type = "checkbox"> trong chế độ Razor view: CheckBox() và CheckBoxFor().

Xét mô hình Student sau:

Student Model

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; }

}

**CheckBox():**

CheckBox() method

MvcHtmlString CheckBox(string name, bool isChecked, object htmlAttributes)

Html.CheckBox() trong Razor view

@Html.CheckBox("isNewlyEnrolled", true)

Html Result:

<input checked="checked"

id="isNewlyEnrolled"

name="isNewlyEnrolled"

type="checkbox"

value="true" />

Trong ví dụ trên, tham số đầu tiên là thuộc tính "isNewlyEnrolled" của lớp Student Model sẽ được đặt làm name và id của hộp văn bản. Tham số thứ hai là giá trị boolean, chọn hoặc bỏ chọn hộp kiểm.

Phương thức CheckBoxFor là phương thức mở rộng. Nó tạo phần tử <input type = "checkbox"> cho thuộc tính mô hình được chỉ định bằng biểu thức lambda. Phương thức CheckBoxFor liên kết một thuộc tính mô hình đối tượng được chỉ định với thành phần hộp kiểm. Vì vậy, nó tự động chọn hoặc bỏ chọn một hộp kiểm dựa trên giá trị.

**CheckBoxFor():**

CheckBoxFor() method

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

Html.CheckBoxFor() trong Razor view

@model Student

@Html.CheckBoxFor(model => model.isNewlyEnrolled)

Html Result:

<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" />

### 2.2.5. HtmlHelper - RadioButton

Lớp HtmlHelper bao gồm hai phương thức mở rộng để tạo phần tử <input type = "radio"> trong chế độ Razor view: RadioButton() và RadioButtonFor().

Xét mô hình Student sau:

Student Model

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public string Gender { get; set; }

}

**RadioButton():**

RadioButton() method

MvcHtmlString RadioButton(string name, object value, bool isChecked, object htmlAttributes)

Html.RadioButton() trong Razor view

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

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

Html Result:

Male: <input checked="checked"

id="Gender"

name="Gender"

type="radio"

value="Male" />

Female: <input id="Gender"

name="Gender"

type="radio"

value="Female" />

**RadioButtonFor():**

RadioButtonFor() method

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

Html.RadioButtonFor() trong Razor view

@model Student

@Html.RadioButtonFor(model => model.Gender,"Male")

@Html.RadioButtonFor(model => model.Gender,"Female")

Html Result:

<input checked="checked"

id="Gender"

name="Gender"

type="radio"

value="Male" />

<input id="Gender"

name="Gender"

type="radio"

value="Female" />

### 2.2.6. HtmlHelper - DropDownList

Lớp HtmlHelper bao gồm hai phương thức mở rộng để tạo phần tử <select> trong chế độ Razor view: DropDownList() và DropDownListFor().

Xét mô hình Student sau:

Student Model

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

}

**DropDownList():**

DropDownList() method

MvcHtmlString Html.DropDownList(string name, IEnumerable<SelectLestItem> selectList, string optionLabel, object htmlAttributes)

Html.DropDownList() trong Razor view

@using MyMVCApp.Models

@model Student

@Html.DropDownList("StudentGender",

new SelectList(Enum.GetValues(typeof(Gender))),

"Select Gender",

new { @class = "form-control" })

Html Result:

<select class="form-control" id="StudentGender" name="StudentGender">

<option>Select Gender</option>

<option>Male</option>

<option>Female</option>

</select>

**DropDownListFor():**

DropDownListFor() method

MvcHtmlString Html.DropDownListFor(Expression<Func<dynamic,TProperty>> expression, IEnumerable<SelectLestItem> selectList, string optionLabel, object htmlAttributes)

Html.DropDownListFor() trong Razor view

@using MyMVCApp.Models

@model Student

@Html.DropDownListFor(model => model.StudentGender,

new SelectList(Enum.GetValues(typeof(Gender))),

"Select Gender")

Html Result:

<select class="form-control" id="StudentGender" name="StudentGender">

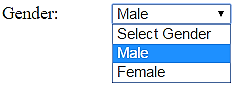
<option>Select Gender</option>

<option>Male</option>

<option>Female</option>

</select>

Trong ví dụ trên, tham số đầu tiên trong phương thức DropDownListFor() là biểu thức lambda chỉ định thuộc tính mô hình được liên kết với phần tử select. Chúng ta đã chỉ định thuộc tính StudentGender thuộc loại enum. Tham số thứ hai chỉ định các mục sẽ hiển thị trong danh sách thả xuống bằng cách sử dụng SelectList. Tham số thứ ba là tùy chọn Nhãn sẽ là mục đầu tiên của danh sách thả xuống. Vì vậy, bây giờ, nó tạo phần tử <select> với id và name được đặt thành tên thuộc tính - StudentGender và hai mục danh sách - Nam và Nữ như hiển thị bên dưới.



Hình 2.2. Ví dụ về DropDownListFor()

### 2.2.7. HtmlHelper - Hidden field

Tương tự như trên, chúng ta cũng có hai phương thức Hidden() và HiddenFor().

**Hidden():**

Hidden() method

MvcHtmlString Html.Hidden(string name, object value, object htmlAttributes)

Html.Hidden() trong Razor view

@model Student

@Html.Hidden("StudentId")

Html Result:

<input id="StudentId"

name="StudentId"

type="hidden"

value="1" />

**HiddenFor():**

HiddenFor() method

MvcHtmlString Html.HiddenFor(Expression<Func<dynamic,TProperty>> expression)

HiddenFor() trong Razor view

@model Student

@Html.HiddenFor(model => model.StudentId)

Html Result:

<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" />

HiddenFor() đã tạo thuộc tính dữ liệu của html5 được sử dụng để xác thực trong ASP.Net MVC.

### 2.2.8. HtmlHelper - Password

Tương tự như trên, chúng ta cũng có hai phương thức Password() và PasswordFor().

**Password():**

Password() method

MvcHtmlString Html.Password(string name, object value, object htmlAttributes)

Html.Password() trong Razor view

@model Student

@Html.Password("OnlinePassword")

Html Result:

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

**PasswordFor():**

PasswordFor() method

MvcHtmlString Html.PasswordFor(Expression<Func<dynamic,TProperty>> expression, object htmlAttributes)

PasswordFor() trong Razor view

@model Student

@Html.PasswordFor(model => model.Password)

Html Result:

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

### 2.2.9. HtmlHelper - Display

Display() và DisplayFor() sử dụng để hiển thị chuỗi ký tự.

**Display():**

Display() method: MvcHtmlString Display(string expression)

Html.Display() trong Razor view

@Html.Display("StudentName")

Html Result: "Steve"

**DisplayFor():**

DisplayFor() method:

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

DisplayFor() trong Razor view

@model Student

@Html.DisplayFor(model => model.StudentName)

### 2.2.10. HtmlHelper - Label

Label() và LabelFor() sử dụng để hiển thị chuỗi ký tự.

**Label():**

Label() method:

MvcHtmlString Label(string expression, string labelText, object htmlAttributes)

Html.Label() trong Razor view

@Html.Label("StudentName")

Html Result:

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

Html.Label() trong Razor view

@Html.Label("StudentName","Student-Name")

Html Result:

<label for="StudentName">Student-Name</label>

**LabelFor():**

LabelFor() method:

 MvcHtmlString LabelFor(<Expression<Func<TModel,TValue>> expression)

LabelFor() trong Razor view

@model Student

@Html.LabelFor(model => model.StudentName)

Html Result:

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

### 2.2.11. 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 bao gồm 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 của mô hình đối tượng. Nó cho phép nhập các loại dữ liệu và kiểm tra dữ liệu nhập vào có tương ứng với kiểu dữ liệu của đối tượng không.

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

Bảng 2.2. Các phần tử html trong phương thức Editor() hoặc EditorFor().

| Thuộc tính kiểu dữ liệu | Yếu tố Html tương ứng |
| --- | --- |
| 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" > |

Xét mô hình Student sau:

Student Model

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; }

}

**Editor():**

MvcHtmlString Editor(string propertyname)

Editor() trong Razor view

StudentId: @Html.Editor("StudentId")

Student Name: @Html.Editor("StudentName")

Age: @Html.Editor("Age")

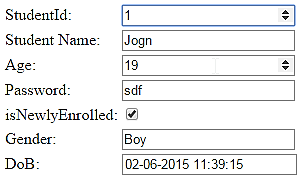
Password: @Html.Editor("Password")

isNewlyEnrolled: @Html.Editor("isNewlyEnrolled")

Gender: @Html.Editor("Gender")

DoB: @Html.Editor("DoB")

Result:

**

**EditorFor:**

EditorFor() signature:

MvcHtmlString EditorFor(<Expression<Func<TModel,TValue>> expression)

EditorFor() trong Razor view

StudentId: @Html.EditorFor(model => model.StudentId)

Student Name: @Html.EditorFor(model => model.StudentName)

Age: @Html.EditorFor(model => model.Age)

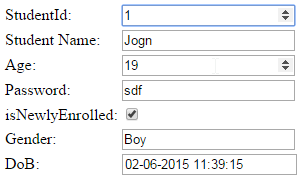
Password: @Html.EditorFor(model => model.Password)

isNewlyEnrolled: @Html.EditorFor(model => model.isNewlyEnrolled)

Gender: @Html.EditorFor(model => model.Gender)

DoB: @Html.EditorFor(model => model.DoB)

Result:

**

## 2.3. Biểu thức lambda (lambda expression)

Một biểu thức lambda là một hàm nặc danh (anonymous function) dùng để tạo các kiểu đại diện (delegates) hay cây biểu thức (expression tree). Bằng cách dùng biểu thức lambda, chúng ta có thể viết các hàm cục bộ có thể được chuyển như các tham số hay được trả về như giá trị của hàm gọi. Biểu thức lambda rất hữu ích cho việc viết các biểu thức truy vấn LINQ.

Cú pháp của biểu thức lambda:

(tham số đầu vào) => biểu thức hay khối lệnh

Ví dụ, biểu thức lambda y => y\*y xác định tham số tên y và trả về giá trị là y bình phương.

del myDelegate = y => y \* y;

int j = myDelegate(5);

Console.WriteLine(j); // 25

Console.ReadLine();

Có thể gán biểu thức lambda y => y\*y đến kiểu delegate.

**+ Expression tree**

Một expression tree thể hiện code trong một cấu trúc dữ liệu trông giống như dạng cây với bản thân các node là một biểu thức có thể là một phương thức hay một toán tử nhị phân như x>y. Biểu thức lambda được dùng trong việc xây dựng expression tree. Ví dụ dùng x => x\*x để tạo expression tree:

Expression<del> myET = x => x \* x;

**+ Expression lambda**

Trong ví dụ về biểu thức lambda y => y\*y, biểu thức y\*y còn được gọi expression lambda. Expression lambda được dùng trong xây dựng các cây biểu thức. expression lambda trả về kết quả của biểu thức và có cú pháp (trong C#) như sau:

(danh sách tham số đầu vào) => biểu thức

Dấu ngoặc đơn được yêu cầu khi có từ hai tham số đầu vào trở lên. Ví dụ (C#):

(x, y) => x == y

Trong trường hợp trình biên dịch gặp khó khăn trong việc suy diễn kiểu thì cần chỉ ra kiểu một cách tường minh cho các tham số. Ví dụ (C#):

(int x, string s) => s.Length > x

Thỉnh thoảng danh sách tham số là rỗng:

() => SomeMethod()

Lưu ý rằng, expression lambda trong ví dụ trên là một phương thức và ta chỉ có thể dùng các phương thức gọi trong biểu thức lambda trong phạm vi .NET Framework, vượt quá phạm vi, ví dụ dùng trong SQL Server, thì phương thức gọi không có ý nghĩa.

Một số quy tắc viết lambda expression:

1. Có thể bỏ qua kiểu dữ liệu của parameter truyền vào

(string qua) => {Console.WriteLine("Tặng quà: " + qua);}

(qua) => {Console.WriteLine("Tặng quà: " + qua);}

2. Nếu không có parameter, bỏ dấu () trống

() => {Console.WriteLine("Hello");}

3. Nếu chỉ có 1 parameter, có thể bỏ luôn dấu ()

(x) => {Console.WriteLine("Hello " + x);}

x => {Console.WriteLine("Hello " + x);}

4. Nếu có nhiều parameter, ngăn cách bằng dấu phẩy

(x, y) => {Console.WriteLine("Hello " + x + y);}

5. Nếu anonymous function chỉ có 1 câu lệnh, có thể bỏ dấu {}

x => { Console.WriteLine("Hello " + x); }

x => Console.WriteLine("Hello " + x)

6. Nếu chỉ return 1 giá trị, có thể bỏ chữ return.

//các lambda expression sau tương đương nhau

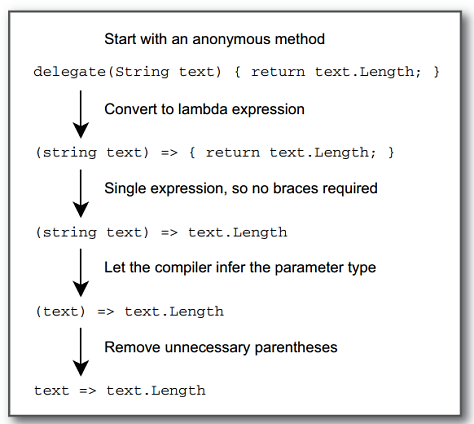
(x) => { return x > 4; }

x => { return x > 4; }

x => return x > 4

x => x > 4

Chúng ta có thể mình họa như sau:



Hình 2.3. Minh họa quá trình viết biểu thức lambda

**+ Statement lambda**

Một statement lambda tương tự như một biểu thức lambda ngoại trừ các lệnh sẽ được đặt trong dấu ngoặc nhọn {} như cú pháp sau:

(danh sách tham số đầu vào) => {statement;}

Ví dụ:

int[] source = new[] { 3, 8, 4, 6, 1, 7, 9, 2, 4, 8 };

foreach (int i in source.Where(x ⇒

    {

     if (x <= 3)

       return true;

     else (x >= 7)

       return true;

     return false;

   }

))

Console.WriteLine(i);

Console.ReadLine();

Kết quả:

3

8

1

7

9

2

8

**+ Async lambda**

Biểu thức lambda được tạo bởi quá trình không đồng bộ bằng cách dùng từ khóa async được gọi là async lambda. Ví dụ:

Func<Task<string>> getWordAsync = async() => "hello";

**+ Suy diễn kiểu trong lambda**

Trong C#, suy diễn kiểu là một chức năng hữu ích trong trường hợp kiểu không được xác định một cách tường minh. Với biểu thức lambda, suy diễn kiểu chỉ làm việc khi mỗi kiểu đã được xác định như trình biên dịch phải được thỏa mãn. Bản thân biểu thức lambda không có quan niệm “kiểu”.

**+ Phạm vi biến trong biểu thức lambda**

Phạm vi biến (variable scope) trong biểu thức lambda tuân theo luật sau:

- Một biến được nắm bắt sẽ không được đưa vào garbage collection (cơ chế quản lý bộ nhớ) cho đến khi delegate tham chiếu nó trở nên đủ điều kiện cho garbage collection.

- Một biến khai báo trong biểu thức lambda sẽ không được nhìn thấy bởi các phương thức bên ngoài.

- Một biểu thức lambda sẽ không thể nắm bắt trực tiếp một tham số ref hay out từ một phương thức kèm theo.

- Một lệnh trả về trong một biểu thức lambda sẽ không phát sinh phương thức kèm theo để trả về.

- Một biểu thức lambda sẽ không chứa lệnh goto, lệnh break, hay lệnh continue trong khối hàm lamda nếu đích đến của các lệnh ở bên ngoài khối hàm lambda. Cũng sẽ phát sinh lỗi nếu đích đến của một lệnh nhảy bên ngoài khối hàm lambda là bên trong khối hàm lambda.

Ví dụ (C#):

delegate bool D();

delegate bool D2(int i);

class Test

{

  D del;

  D2 del2;

  public void TestMethod(int input)

  {

   int j = 0;

   // khởi tạo các delegates với biểu thức lambda.

   // Chú ý truy cập đến hai biến ngoài.

   // del sẽ được gọi với phương thức này

   del = () => { j = 10;  return j > input; };

   // del2 sẽ được gọi khi TestMethod vượt khỏi phạm vi.

   del2 = (x) => {return x == j; };

   // Minh hoạ giá trị của j:

   // Kết quả: j = 0

   // Delegate chưa được gọi.

   Console.WriteLine("j = {0}", j);// gọi delegate.

   bool boolResult = del();

   // Kết quả: j = 10 b = True

   Console.WriteLine("j = {0}. b = {1}", j, boolResult);

  }

  static void Main()

  {

   Test test = new Test();

   test.TestMethod(5);

   // del2 có một bản sao của

   // biến j từ TestMethod.

   bool result = test.del2(10);

   // Kết quả: True

   Console.WriteLine(result);

   Console.ReadKey();

  }

}

**+ Biểu thức lambda với các toán tử truy vấn**

Một biểu thức lambda với một toán tử truy vấn được định giá bởi cùng yêu cầu và làm việc trên mỗi phần tử trong chuỗi đầu vào, không phải trên toàn bộ chuỗi. Các nhà phát triển được phép bằng biểu thức lambda để đưa logic của riêng họ vào trong các toán tử truy vấn chuẩn. Trong ví dụ sau đây chúng ta dùng toán tử ‘Where’ để lấy các giá trị lẻ từ danh sách bằng cách dùng biểu thức lambda:

int[] fibNum = { 1, 1, 2, 3, 5, 8, 13, 21, 34 };

double averageValue = fibNum.Where(num => num % 2 == 1).Average();

Console.WriteLine(averageValue);

Console.ReadLine();

**+ Lambda Expression và LINQ**

Ta sử dụng lambda expression để truyền 1 anonymous function vào hàm Where hoặc First. Ví dụ, khi dùng hàm Where của LINQ để tìm những phần tử trong 1 mảng:

var studentList = new List<Student>();

//Viết theo lambda expression

var students = studentList.Where(stu => stu.Age > 20);

//Viết thường

var student = studentList.Where(new delegate(Student stu) { return stu.Age > 20; });

//Hoặc

public bool FindStudentWithAge(Student stu) { return stu.Age > 20; }

var student = studentList.Where(FindStudentWithAge);

## 2.4. Xác thực dữ liệu - Data Validation

ASP.NET MVC sử dụng các thuộc tính DataAnnotations để thực hiện xác nhận. DataAnnotations bao gồm các thuộc tính xác thực tích hợp cho các quy tắc xác thực khác nhau, có thể được áp dụng cho các thuộc tính của lớp mô hình. ASP.NET MVC framework sẽ tự động thực thi các quy tắc xác thực này và hiển thị các thông báo xác thực trong view.

Các thuộc tính DataAnnotations có trong không gian tên System.ComponentModel.DataAnnotations. Bảng sau liệt kê các thuộc tính xác thực DataAnnotations.

Bảng 2.3. Các thuộc tính DataAnnotations.

| Thuộc tính | Mô tả |
| --- | --- |
| Required | Cho biết thuộc tính là trường bắt buộc. |
| StringLength | Xác định độ dài tối đa cho trường chuỗi. |
| Range | Xác định giá trị lớn nhất và nhỏ nhất cho một trường số. |
| RegularExpression | Chỉ định rằng giá trị trường phải khớp với Biểu thức chính quy được chỉ định. |
| CreditCard | Trường được chỉ định là số thẻ tín dụng. |
| CustomValidation | Phương pháp xác thực tùy chỉnh đã chỉ định để xác thực trường. |
| EmailAddress | Xác thực với định dạng địa chỉ email. |
| FileExtension | Xác thực với phần mở rộng tệp. |
| MaxLength | Chỉ định độ dài tối đa cho một trường chuỗi. |
| MinLength | Chỉ định độ dài tối thiểu cho một trường chuỗi. |
| Phone | Chỉ định rằng trường là số điện thoại sử dụng biểu thức chính quy cho số điện thoại. |

**ValidationMessage**

MvcHtmlString ValidateMessage(string modelName, string validationMessage, object htmlAttributes)

Ví dụ:

Student Model

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

public int Age { get; set; }

}

Razor view

@model Student

@Html.Editor("StudentName") <br />

@Html.ValidationMessage("StudentName", "", new { @class = "text-danger" })

Tùy chỉnh thông báo lỗi trong Model

public class Student

{

public int StudentId { get; set; }

[Required(ErrorMessage="Please enter student name.")]

public string StudentName { get; set; }

public int Age { get; set; }

}

Tùy chỉnh thông báo lỗi trong Razor view

@model Student

@Html.Editor("StudentName") <br />

@Html.ValidationMessage("StudentName", "Please enter student name.", new { @class = "text-danger" })

**ValidationMessageFor**

MvcHtmlString ValidateMessage(Expression<Func<dynamic,TProperty>> expression, string validationMessage, object htmlAttributes)

Ví dụ:

@model Student

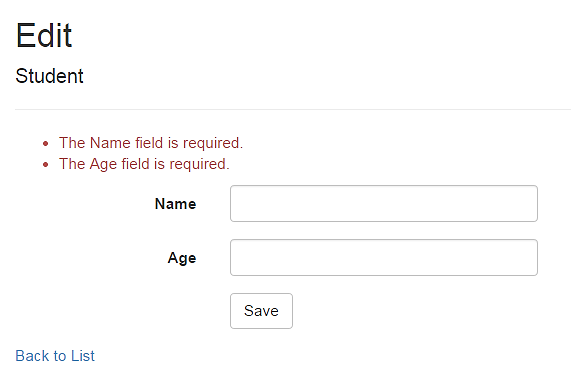
@Html.EditorFor(model => model.StudentName) <br />

@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })

**ValidationSummary**

Phương thức ValidationSummary tạo ra một danh sách không có thứ tự (phần tử ul) của các thông báo xác thực nằm trong đối tượng ModelStateDipedia.

Có thể sử dụng ValidationSummary để hiển thị tất cả các thông báo lỗi cho tất cả các trường. Nó cũng có thể được sử dụng để hiển thị các thông báo lỗi tùy chỉnh. Hình dưới đây cho thấy cách xác thực hiển thị các thông báo lỗi.



Hình 2.3. Trang Edit View

Cú pháp:

MvcHtmlString ValidateMessage(bool excludePropertyErrors, string message, object htmlAttributes)

Razor view:

@Html.ValidationSummary(false, "", new { @class = "text-danger" })

Tùy chỉnh thông báo lỗi:

Để hiển thị thông báo lỗi tùy chỉnh, trước hết, cần thêm lỗi tùy chỉnh vào ModelState trong phương thức hành động thích hợp.

Ví dụ:

if (ModelState.IsValid) {

//check whether name is already exists in the database or not

bool nameAlreadyExists = \* check database \*

if(nameAlreadyExists)

{

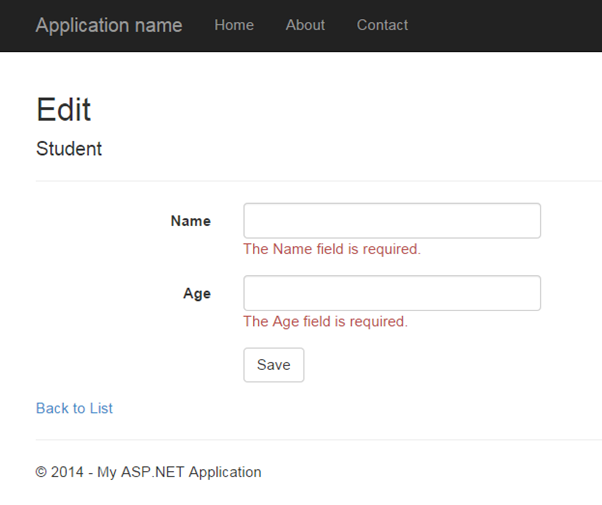
ModelState.AddModelError(string.Empty, "Student Name already exists.");

return View(std);

}

}

Chúng ta đã tạo chế độ Edit view cho Sinh viên trong phần trước. Bây giờ, chúng ta sẽ triển khai xác thực dữ liệu trong chế độ Edit, sẽ hiển thị các thông báo xác thực khi nhấp vào nút Save, như hiển thị bên dưới nếu Tên sinh viên hoặc Tuổi trống.



Hình 2.4. Trang Edit View

Bước 1: Trước hết, áp dụng thuộc tính DataAnnotation trên các thuộc tính của lớp mô hình Student. Chúng ta muốn xác thực rằng StudentName và Age không trống. Ngoài ra, Age nên nằm trong khoảng từ 5 đến 50.

Apply DataAnnotation Attributes

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

[Range(5,50)]

public int Age { get; set; }

}

Trong ví dụ trên, chúng ta đã áp dụng một thuộc tính bắt buộc cho thuộc tính StudentName. Vì vậy, bây giờ, MVC sẽ tự động hiển thị thông báo lỗi mặc định, nếu người dùng cố lưu biểu mẫu Edit mà không nhập Tên sinh viên. Theo cách tương tự, thuộc tính Range được áp dụng với giá trị tối thiểu và tối đa cho thuộc tính Age. Điều này sẽ xác thực và hiển thị thông báo lỗi nếu người dùng chưa nhập Tuổi hoặc nhập tuổi dưới 5 hoặc hơn 50.

Bước 2: Tạo phương thức GET Edit() và POST Edit() giống như phần trước. Phương thức hành động GET sẽ hiển thị chế độ Edit View để chỉnh sửa sinh viên đã chọn và phương pháp POST Edit sẽ lưu sinh viên đã chỉnh sửa như hiển thị bên dưới.

Edit Action methods:

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

public ActionResult Edit(int id)

{

var std = studentList.Where(s => s.StudentId == StudentId).FirstOrDefault();

return View(std);

}

[HttpPost]

public ActionResult Edit(Student std)

{

if (ModelState.IsValid) {

//write code to update student

return RedirectToAction("Index");

}

return View(std);

}

}

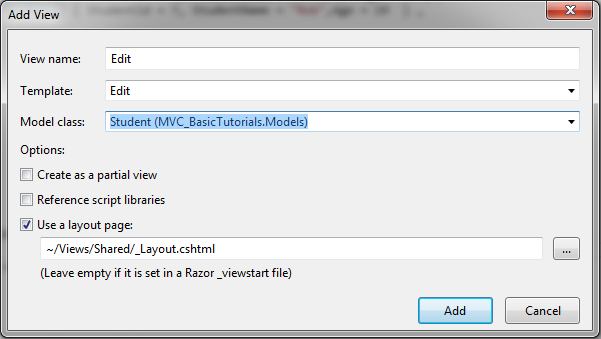
}

Như thấy trong phương thức POST Edit, trước tiên chúng ta kiểm tra xem ModelState có hợp lệ hay không. Nếu ModelState hợp lệ thì cập nhật sinh viên vào cơ sở dữ liệu, nếu không thì quay lại Edit() xem lại với cùng dữ liệu sinh viên.

ModelState.IsValid xác định rằng liệu các giá trị được gửi có đáp ứng tất cả các thuộc tính xác thực DataAnnotation được áp dụng cho các thuộc tính mô hình hay không.

Bước 3: Bây giờ, tạo chế độ xem Edit view cho Sinh viên.

Để tạo chế độ xem Edit view, nhấp chuột phải vào bên trong Edit\Add view.. và chọn các thông tin như hình dưới:



Hình 2.5. Tạo mới View

Click Add để tạo view trong thư mục View/Student. Edit.cshtml sẽ được tạo ra như hình dưới đây.

Edit.cshtml:

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Student</h4>

<hr />

**@Html.ValidationSummary(true, "", new { @class = "text-danger" })**

@Html.HiddenFor(model => model.StudentId)

<div class="form-group">

@Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" } })

**@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })**

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })

**@Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger" })**

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

Như ta có thể thấy trong Edit.cshtml ở trên, nó gọi phương thức ValidationMessageFor cho mọi trường và phương thức ValidationSummary ở trên cùng. ValidationMessageFor có trách nhiệm hiển thị thông báo lỗi cho trường được chỉ định. ValidationSummary hiển thị danh sách tất cả các thông báo lỗi cùng một lúc.

Vì vậy, bây giờ, nó sẽ hiển thị thông báo xác thực mặc định khi gửi biểu mẫu Edit mà không cần nhập Tên hoặc Tuổi.

**Lưu ý:**

- ASP.NET MVC sử dụng các thuộc tính DataAnnotations để xác nhận.

- Các thuộc tính DataAnnotations có thể được áp dụng cho các thuộc tính của lớp mô hình để chỉ ra loại giá trị mà thuộc tính sẽ giữ.

- Các thuộc tính xác thực sau đây có sẵn theo mặc định: Required, StringLength, Range, RegularExpression, CreditCard, CustomValidation, EmailAddress, FileExtension, MaxLength, MinLength, Phone.

- Sử dụng ValidationSummary để hiển thị tất cả thông báo lỗi trong view.

- Sử dụng phương thức ValidationMessageFor hoặc ValidationMessage để hiển thị các thông báo lỗi cấp trường trong view.

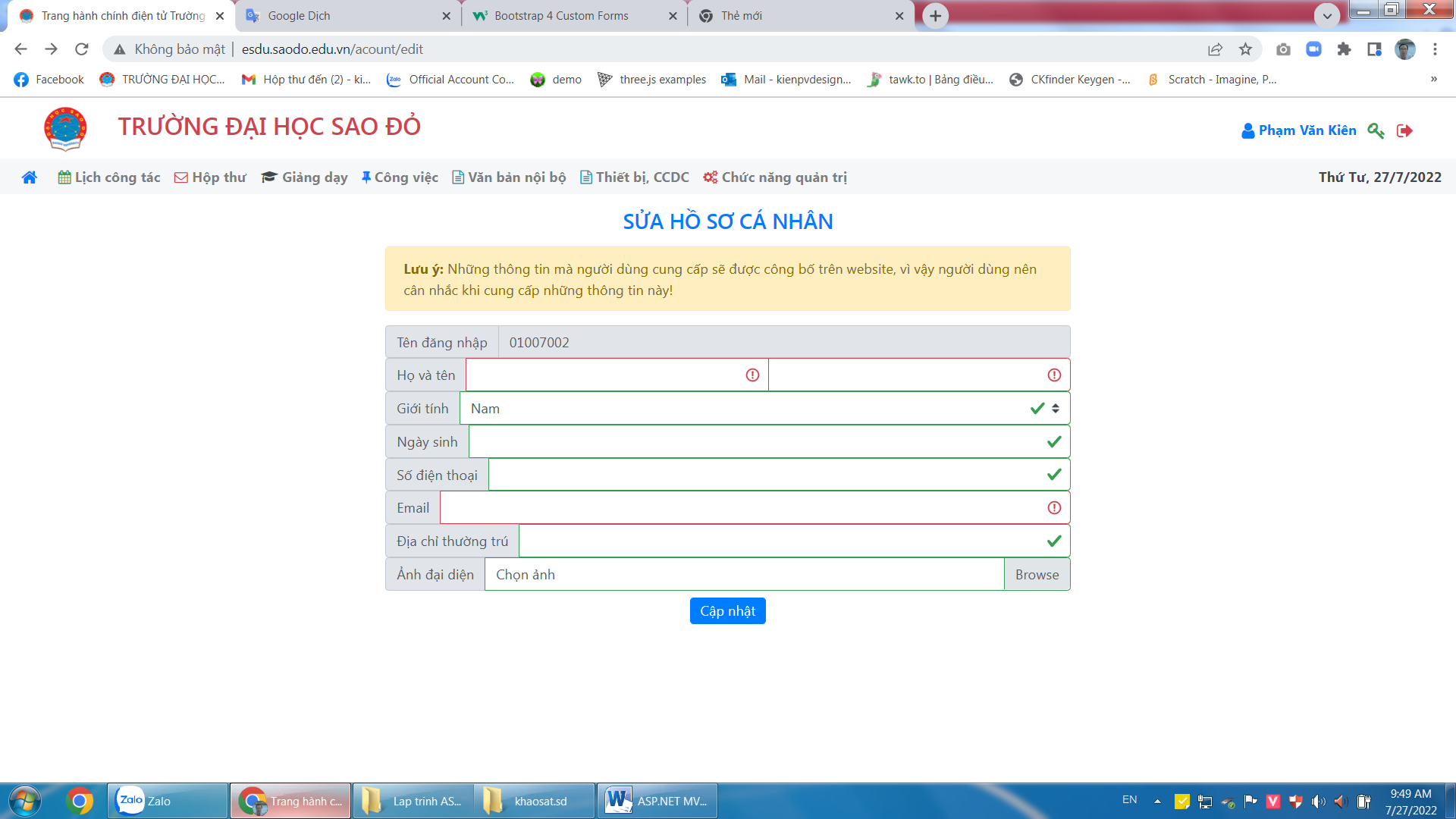
- Kiểm tra xem mô hình có hợp lệ không trước khi cập nhật trong phương thức hành động bằng ModelState.IsValid.

- Cho phép xác thực phía máy khách để hiển thị thông báo lỗi mà không có hiệu ứng gửi lại trong trình duyệt.

# CÂU HỎI ÔN TẬP CHƯƠNG 2

1. Tạo ứng dụng ASP.NET MVC thực hiện các yêu cầu sau:

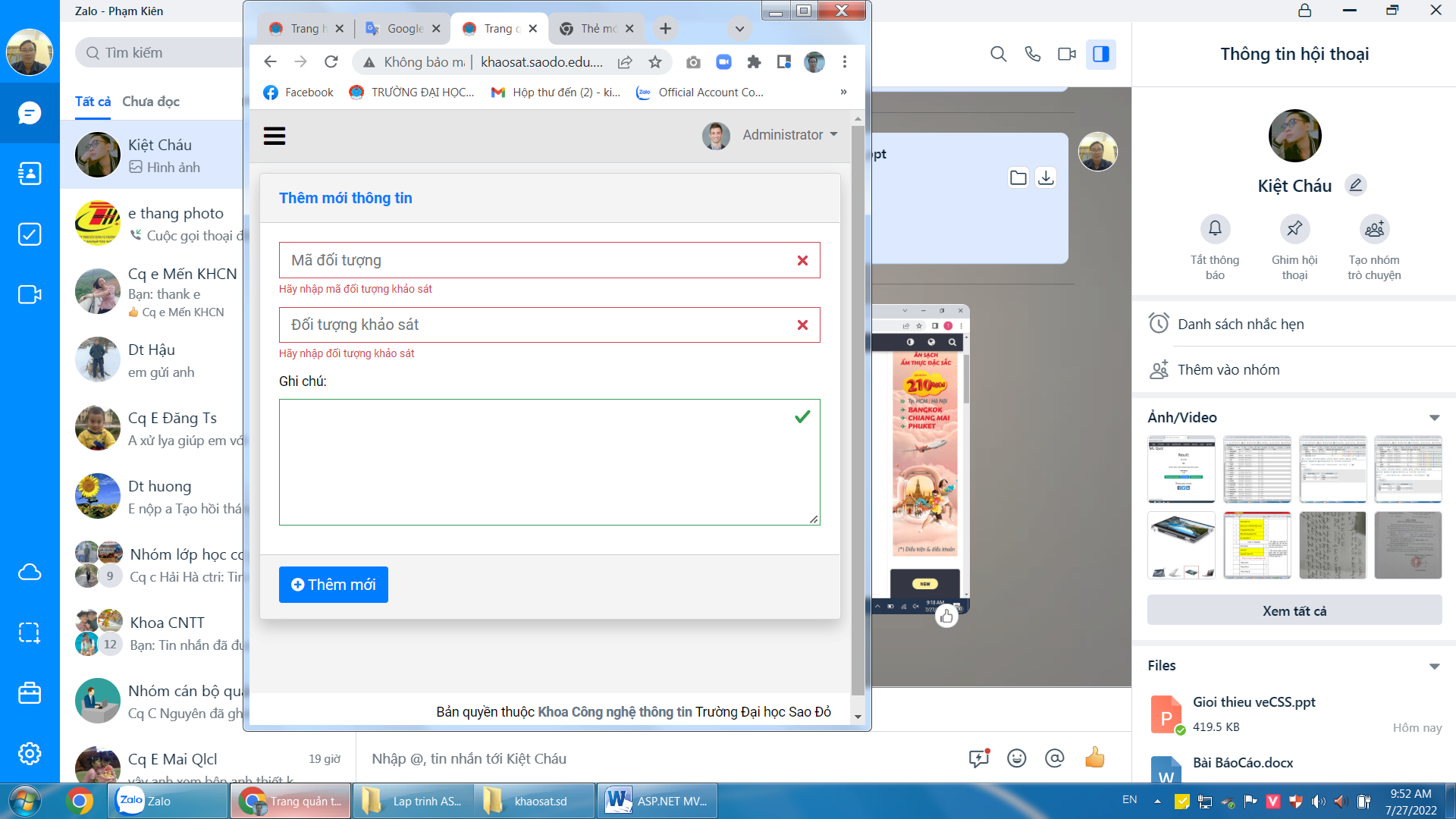
- Sử dụng HTML Helpers tạo cập nhập thông tin hồ sơ cá nhân theo mẫu sau:

**

- Tạo trang hiển thị thông tin sau khi người dùng cập nhật, yêu cầu có xác thực dữ liệu.

2. Tạo ứng dụng ASP.NET MVC thực hiện các yêu cầu sau:

- Sử dụng HTML Helpers tạo trang thêm thông tin đối tượng khảo sát trực tuyến theo mẫu sau:

**

- Tạo trang hiển thị thông tin sau khi người dùng thêm mới, yêu cầu có xác thực dữ liệu.

3. Tạo ứng dụng ASP.NET MVC thực hiện các yêu cầu sau:

- Tạo lớp Member bao gồm các trường thông tin: MemberId, Username, Password, Age, Email.

- Tạo Controller với phương thức AddMember().

- Tạo View cho phương thức AddMember() với form nhập liệu cho Member.

- Kiểm tra tính hợp lệ của dữ liệu nhập trên form (các trường không để trống, Age phải là số, Email đúng định dạng…). Có thể sử dụng nhiều cách khác nhau để kiểm tra dữ liệu nhập trên form.

- Hiển thị thông tin chi tiết khi nhập thành công.

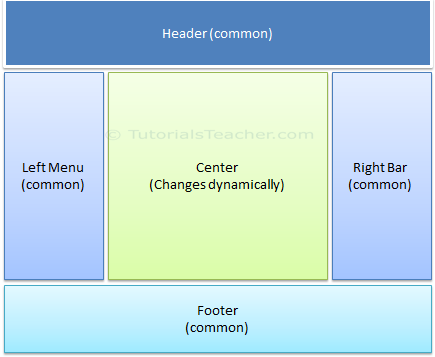
# Chương 3. GIAO DIỆN

## 3.1. Layout view

### 3.1.1. Tổng quan

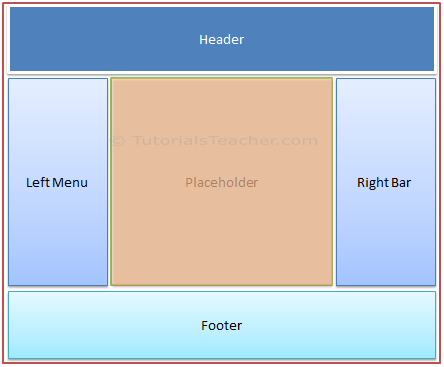
Một ứng dụng có thể chứa các phần chung trong giao diện người dùng như logo, tiêu đề, thanh điều hướng bên trái, thanh bên phải hoặc phần chân trang. ASP.NET MVC đã giới thiệu một giao diện bố cục (layout) chứa các phần UI chung này, do đó chúng ta không phải viết cùng một mã trong mỗi trang. Giao diện bố cục giống như trang chính của ứng dụng webform ASP.NET.

Ví dụ: Giao diện người dùng ứng dụng có thể chứa: phần đầu trang, thanh menu bên trái, thanh bên phải và phần chân trang vẫn giữ nguyên trong mỗi trang, chỉ phần giữa (center) thay đổi linh hoạt như hiển thị bên dưới.



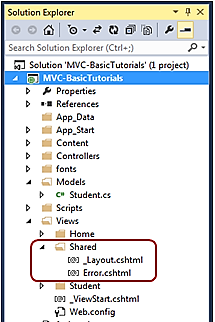
Hình 3.1. Layout UI

Layout View cho phép xác định một mẫu trang web chung, có thể được kế thừa trong nhiều View để cung cấp giao diện nhất quán trong nhiều trang của một ứng dụng. Layout giúp loại bỏ mã hóa trùng lặp và tăng cường tốc độ phát triển và bảo trì dễ dàng. Giao diện bố trí như hình trên sẽ chứa phần đầu trang, menu bên trái, thanh bên phải và phần chân trang. Nó chứa một trình giữ chỗ cho phần trung tâm (Placeholder) để thay đổi linh hoạt như hiển thị bên dưới.



Hình 3.2. Layout View

Bố cục trang có phần mở rộng tương tự như các View khác: .cshtml hoặc .vbhtml. Nó được chia sẻ với nhiều trang View, vì vậy nó phải được lưu trong thư mục Shared. Ví dụ: khi chúng ta tạo ứng dụng MVC đầu tiên của mình trong phần trước, nó cũng đã tạo sẵn file \_Layout.cshtml trong thư mục Shared như hiển thị bên dưới.



Hình 3.3. Thư mục Shared

\_Layout.cshtml:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

Như ta có thể thấy, Layout View chứa html Doctype, phần đầu và phần thân như mã html bình thường, sự khác biệt duy nhất là gọi các phương thức RenderBody() và RenderSection(). RenderBody hoạt động như một trình giữ chỗ cho các View khác. Ví dụ: Index.cshtml trong thư mục chính sẽ được chèn và hiển thị trong Layout View, trong đó phương thức RenderBody() đang được gọi.

**Sử dụng Layout View**

Có thể đặt Layout View theo nhiều cách, bằng cách sử dụng \_ViewStart.cshtml hoặc thiết lập đường dẫn của trang bố cục bằng thuộc tính Layout trong mỗi View riêng lẻ hoặc chỉ định tên Layout View trong phương thức hành động.

+ Sử dụng \_ViewStart.cshtml

Trang \_ViewStart.cshtml nằm trong thư mục View của ứng dụng, chúng ta thêm đoạn lệnh sau để áp dụng layout cho tất cả các view:

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

+ Đặt thuộc tính Layout trong các View riêng lẻ

Ví dụ trang Index View:

@{

ViewBag.Title = "Home Page";

Layout = "~/Views/Shared/\_myLayoutPage.cshtml";

}

+ Chỉ định trang bố cục 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();

}

}

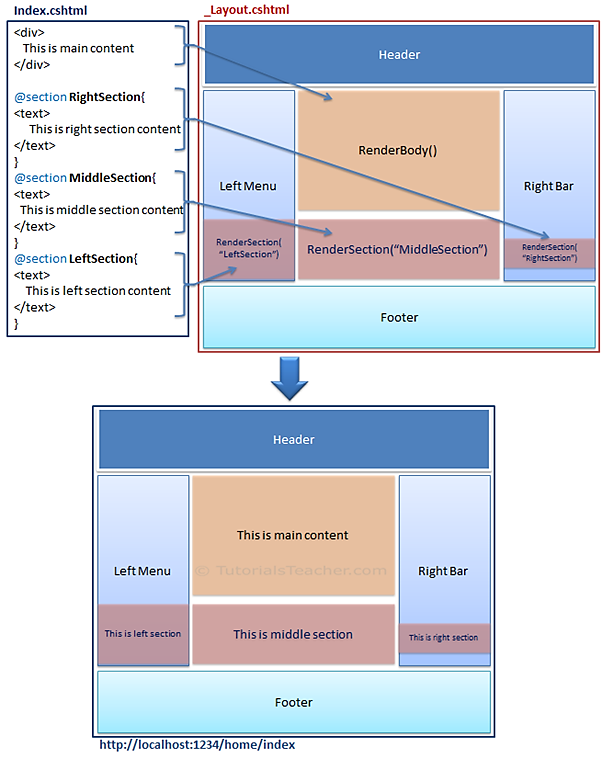
**Phương pháp kết xuất:**

ASP.NET MVC biểu diễn các khung nhìn con bằng các phương thức sau.

Bảng 3.1. Các phương thức biểu diễn các khung nhìn con.

| Phương thức | Mô tả |
| --- | --- |
| RenderBody() | Hiển thị phần của chế độ xem con không nằm trong phần được đặt tên. Giao diện bố trí phải bao gồm 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 dạng xem Bố cục. |

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



Hình 3.4. Minh họa RenderBody và RenderSection

Trong hình trên, \_Layout.cshtml bao gồm phương thức RenderBody() và phương thức RenderSection(). Các phương thức RenderSection chỉ định tên của một phần như LeftSection, MiddleSection và RightSection trong hình trên. Index.cshtml định nghĩa phần được đặt tên bằng cách sử dụng @section trong đó tên của từng phần khớp với tên được chỉ định trong phương thức RenderSection của \_Layout.cshtml, chẳng hạn như @Section RightSection, v.v. và MiddleSection sẽ được hiển thị tại vị trí thích hợp nơi phương thức RenderSection được gọi. Phần còn lại của Index View, không nằm trong bất kỳ phần nào được đặt tên sẽ hiển thị nơi phương thức RenderBody() đang được gọi.

**Lưu ý:**

- Giao diện Bố cục chứa các phần chung của Giao diện người dùng. Nó giống như trang chủ của các biểu mẫu web ASP.NET.

- Tệp \_ViewStart.cshtml có thể được sử dụng để chỉ định đường dẫn của trang bố cục, lần lượt sẽ được áp dụng cho tất cả các chế độ xem của thư mục và thư mục con của nó.

- Có thể đặt thuộc tính Bố cục trong chế độ xem riêng lẻ, để ghi đè cài đặt trang bố cục mặc định của \_ViewStart.cshtml

- Giao diện bố cục sử dụng hai phương thức kết xuất: RenderBody() và RenderSection().

- RenderBody chỉ có thể được sử dụng một lần trong chế độ xem bố cục, trong khi phương thức RenderSection có thể được gọi nhiều lần với tên khác nhau.

- Phương thức RenderBody hiển thị tất cả nội dung của chế độ xem không được gói trong phần được đặt tên.

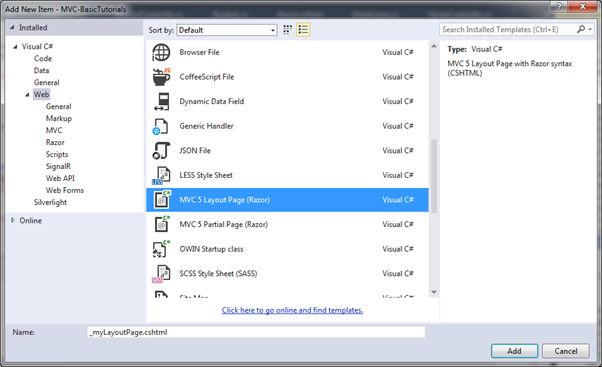
- Phương thức RenderSection hiển thị nội dung của chế độ xem được gói trong phần được đặt tên.

- RenderSection có thể được cấu hình theo yêu cầu hoặc tùy chọn. Nếu được yêu cầu, thì tất cả các khung nhìn con phải bao gồm phần được đặt tên đó.

### 3.1.2. Tạo Layout View

- Click chuột phải vào thư mục Shared\Add\New Item..

- Trong cửa sổ xuất hiện, chọn MVC 5 Layout Page (Razor), đặt tên cho Layout View là \_myLayoutPage.cshtml, sau đó nhấn nút Ok.



Hình 3.5. Add new layout

\_myLayoutPage.cshtml:

<!DOCTYPE html>

<html>

<head>

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

<title>@ViewBag.Title</title>

</head>

<body>

<div>

@RenderBody()

</div>

</body>

</html>

Bây giờ, thêm thẻ <footer> với phương thức RenderSection("footer", true) cùng với một số kiểu dáng như dưới đây. Xin lưu ý rằng bất kỳ View nào sử dụng \_myLayoutPage làm Layout View của nó phải bao gồm phần chân trang.

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>

Sau khi thiết lập Layout cho các View và sửa code:

Index View:

@{

ViewBag.Title = "Home Page";

Layout = "~/Views/Shared/\_myLayoutPage.cshtml";

}

<div class="jumbotron">

<h2>Index</h2>

</div>

<div class="row">

<div class="col-md-4">

<p>This is body.</p>

</div>

@section footer{

<p class="lead">

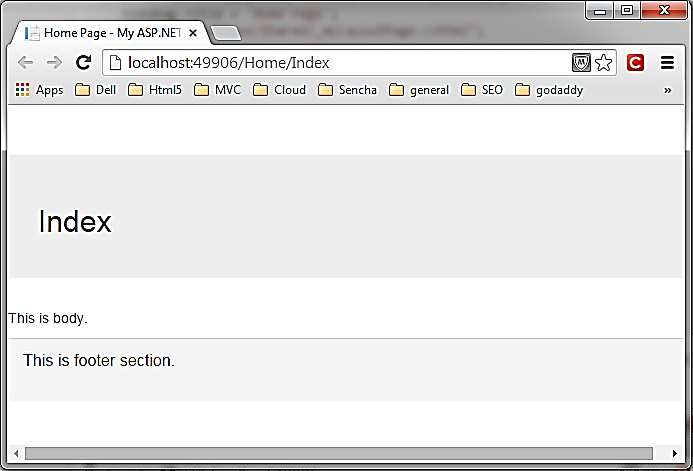
This is footer section.

</p>

}

</div>

Kết quả:

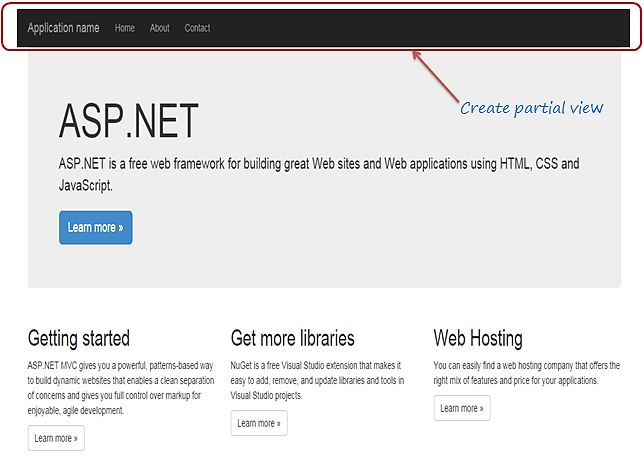


Hình 3.6. Thêm RenderSection

## 3.2. Partial View

### 3.2.1. Tổng quan

Partial view (chế độ xem một phần) là chế độ xem có thể sử dụng lại, có thể được sử dụng làm chế độ xem con trong nhiều chế độ xem khác. Nó loại bỏ mã hóa trùng lặp bằng cách sử dụng lại cùng một phần để hiển thị ở nhiều nơi. Có thể sử dụng chế độ xem một phần trong chế độ xem bố cục – Layout view, cũng như các chế độ xem nội dung khác.



3.7. Chế độ xem một phần

Để bắt đầu, hãy tạo chế độ xem một phần đơn giản cho thanh điều hướng cho mục đích demo. Chúng ta sẽ tạo chế độ xem một phần cho nó, để có thể sử dụng cùng một thanh điều hướng trong nhiều chế độ xem bố cục mà không cần viết lại cùng một mã ở mọi nơi.

Mã html cho thanh điều hướng ở trên như sau:

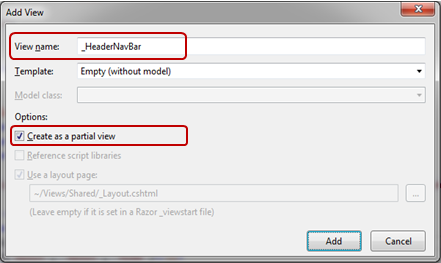


Hình 3.8. Mã html thanh điều hướng

### 3.2.2. Tạo mới Partial View

- Click chuột phải vào thư mục Shared\Add\View…

- Đặt tên và chọn "Create as a partial view" trong hộp thoại Add View.



3.9. Add view

- Tạo mã cho trang \_HeaderNavBar.cshtml như sau:

<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>

+ Hiển thị Partial View - Render Partial View

Phương thức @Html.Partial() hiển thị chế độ xem từng phần được chỉ định. Nó chấp nhận tên chế độ xem một phần dưới dạng tham số chuỗi và trả về MvcHtmlString. Nó trả về chuỗi html để ta có cơ hội sửa đổi html trước khi kết xuất. Bảng sau liệt kê các kiểu dữ liệu MvcHtmlString của phương thức Html.Partial.

Bảng 3.2. Các phương thức Partial.

| Phương thức Partial | Mô tả |
| --- | --- |
| MvcHtmlString Html.Partial(string partialViewName) | Hiển thị nội dung xem một phần nhất định trong chế độ xem được giới thiệu. |
| MvcHtmlString Html.Partial(string partialViewName,object model) | Hiển thị nội dung xem một phần trong chế độ xem được giới thiệu. Tham số mô hình sẽ chuyển đối tượng mô hình sang chế độ xem một phần. |
| MvcHtmlString Html.Partial(string partialViewName, ViewDataDictionary viewData) | Hiển thị nội dung xem một phần trong chế độ xem được giới thiệu. Tham số dữ liệu chuyển từ điển dữ liệu sang chế độ xem một phần. |
| MvcHtmlString Html.Partial(string partialViewName,object model, ViewDataDictionary viewData) | Hiển thị nội dung xem một phần trong chế độ xem được giới thiệu. Tham số mô hình sẽ chuyển đối tượng mô hình và từ điển dữ liệu xem sang chế độ xem một phần. |

+ Html.RenderPartial()

Phương thức RenderPartial giống như phương thức Partial ngoại trừ việc nó trả về void và ghi kết quả html của chế độ xem một phần được chỉ định vào luồng phản hồi http trực tiếp.

Bảng 3.3. Các phương thức RenderPartial.

| Phương thức RenderPartial lớp Helper | Mô tả |
| --- | --- |
| RenderPartial(String partialViewName) | Hiển thị chế độ xem một phần được chỉ định |
| RenderPartial(String partialViewName, Object model) | Hiển thị khung nhìn một phần được chỉ định và áp dụng cho đối tượng mô hình đã chỉ định. |
| RenderPartial(String partialViewName, ViewDataDictionary viewData) | Hiển thị chế độ xem một phần được chỉ định, thay thế thuộc tính ViewData của nó bằng đối tượng ViewDataDipedia đã chỉ định. |
| RenderPartial(String partialViewName, Object model, ViewDataDictionary viewData) | Hiển thị chế độ xem một phần được chỉ định, thay thế thuộc tính ViewData của chế độ xem một phần bằng đối tượng ViewDataDipedia đã chỉ định và áp dụng cho đối tượng mô hình đã chỉ định. |

+ Html.RenderAction()

Phương RenderAction gọi một bộ điều khiển hành động được chỉ định và hiển thị kết quả dưới dạng một phần của chế độ xem. Phương thức Action được chỉ định sẽ trả về PartialViewResult bằng phương thức Partial().

Bảng 3.4. Các phương thức RenderAction.

| Tên phương thức | Mô tả |
| --- | --- |
| RenderAction(String actionName) | Gọi phương thức hành động con đã chỉ định và hiển thị kết quả trong chế độ xem cha. |
| RenderAction(String actionName, Object routeValue) | 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 chế độ xem cha. |
| RenderAction(String actionName, String controllerName) | Gọi phương thức hành động con được chỉ định bằng cách sử dụng tên của bộ điều khiển và hiển thị nội tuyến kết quả trong chế độ xem cha. |
| RenderAction(String actionName, 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 chế độ xem cha. |
| RenderAction(String actionName, 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 chế độ xem cha. |
| RenderAction(String actionName, 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 chế độ xem cha. |

**Html.RenderPartial()**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

@{

**Html.RenderPartial("\_HeaderNavBar");**

}

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

RenderPartial trả về void, do đó có dấu chấm phẩy được yêu cầu ở cuối dòng lệnh và đặt trong khối lệnh.

**Html.Partial()**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

@Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")

</head>

<body>

@Html.Partial("\_HeaderNavBar")

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

</div>

@Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false)

</body>

</html>

Phương thức @ Html.Partial() không cần phải nằm trong khối mã vì nó trả về một chuỗi html.

**Lưu ý:**

- Chế độ xem một phần là chế độ xem có thể sử dụng lại, có thể được sử dụng làm chế độ xem con trong nhiều chế độ xem khác.

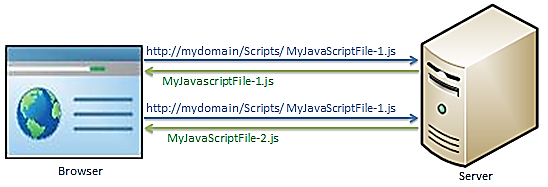
- Chế độ xem một phần có thể được hiển thị bằng phương thức Html.Partial (), Html.RenderPartial () hoặc Html.RenderAction ().

## 3.3. Thực thi các script

### 3.3.1. Kỹ thuật đóng gói (Bundling) và Giảm thiểu (Minification)

**Bundling**

Các kỹ thuật đóng gói (Bundling) đã được giới thiệu trong MVC 4 để cải thiện thời gian tải yêu cầu. Gói cho phép chúng ta tải một loạt các tệp tĩnh từ máy chủ vào một yêu cầu http. Hình dưới đây minh họa kỹ thuật Bundling:



Hình 3.10. Tải tập tin tập lệnh trong các yêu cầu

Trong hình trên, trình duyệt gửi hai yêu cầu riêng biệt để tải hai tệp JavaScript khác nhau MyJavaScriptFile-1.js và MyJavaScriptFile-2.js.

Kỹ thuật gói trong MVC 4 cho phép chúng ta tải nhiều hơn một tệp JavaScript, MyJavaScriptFile-1.js và MyJavaScriptFile-2.js trong một yêu cầu như dưới đây.



Hình 3.11. Load Bundle

**Minification**

Là kỹ thuật thu nhỏ tối ưu hóa kích thước tập tin hoặc tập tin css bằng cách loại bỏ khoảng trắng và nhận xét không cần thiết và rút ngắn tên biến thành một ký tự.

Ví dụ: JavaScript

sayHello = function(name){

//this is comment

var msg = "Hello" + name;

alert(msg);

}

Đoạn JavaScript ở trên sẽ được tối ưu hóa và tối thiểu hóa thành đoạn script sau. Ví dụ: Minified JavaScript

sayHello=function(n){var t="Hello"+n;alert(t)}

Như ta có thể thấy ở trên, nó đã loại bỏ khoảng trắng không cần thiết, các bình luận và cũng rút ngắn tên biến để giảm các ký tự sẽ lần lượt giảm kích thước tệp JavaScript.

Đóng gói và giảm thiểu tác động đến việc tải trang, nó tải trang nhanh hơn bằng cách giảm thiểu kích thước của tệp và số lượng yêu cầu.

Các loại gói

- MVC 5 bao gồm các lớp gói sau trong không gian tên System.web.Optimization:

- ScriptBundle: ScriptBundle chịu trách nhiệm thu nhỏ JavaScript của một hoặc nhiều tệp script.

- StyleBundle: StyleBundle chịu trách nhiệm thu nhỏ CSS các tệp trang tính đơn hoặc nhiều kiểu.

- DynamicFolderBundle: Đại diện cho một đối tượng Bundle mà ASP.NET tạo ra từ một thư mục chứa các tệp cùng loại.

- Tất cả các lớp gói ở trên được bao gồm trong không gian tên System.Web.Optimization.Bundle và xuất phát từ lớp Bundle

**Chú ý:**

- Đóng gói và Giảm thiểu tối ưu hóa tập lệnh tĩnh hoặc tập tin css làm giảm thời gian tải trang.

- Khung MVC cung cấp các lớp ScriptBundle, StyleBundle và DynamicFolderBundle. ScriptBundle thực hiện thu nhỏ các tệp JavaScript. StyleBundle thực hiện thu nhỏ các tệp CSS.

### 3.3.2. ScriptBundle

Chúng ta đã học được các hoạt động của kỹ thuật đóng gói trong ASP.NET MVC. Ở đây, chúng ta sẽ tìm hiểu cách tạo một gói nhiều tệp JavaScript trong một yêu cầu http. ASP.NET MVC API bao gồm lớp ScriptBundle thực hiện thu nhỏ và đóng gói JavaScript.

Mở tệp App\_Start\BundleConfig.cs trong các thư mục MVC. Tệp BundleConfig.cs được tạo bởi khung MVC theo mặc định. Chúng ta nên viết tất cả mã gói của mình trong phương thức BundleConfig.RegisterBundles(). Có thể tạo lớp tùy chỉnh của riêng thay vì sử dụng lớp BundleConfig, nhưng nên tuân theo thực tiễn tiêu chuẩn. Đoạn mã sau đây cho thấy một phần của phương thức RegisterBundles.

Ví dụ: BundleConfig.RegisterBundle()

using System.Web;

using System.Web.Optimization;

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

// create an object of ScriptBundle and

// specify bundle name (as virtual path) as constructor parameter

ScriptBundle scriptBndl = new ScriptBundle("~/bundles/bootstrap");

//use Include() method to add all the script files with their paths

scriptBndl.Include(

"~/Scripts/bootstrap.js",

"~/Scripts/respond.js"

);

//Add the bundle into BundleCollection

bundles.Add(scriptBndl);

BundleTable.EnableOptimizations = true;

}

}

Trong ví dụ trên, chúng ta đã tạo ra một gói gồm hai tệp JavaScript: bootstrap.js và respond.js bằng ScriptBundle qua các bước sau:

- Trước hết, tạo một thể hiện của lớp ScriptBundle bằng cách chỉ định tên gói làm tham số hàm tạo. Tên gói này là một đường dẫn ảo bắt đầu bằng ~ /. Có thể đưa ra bất cứ điều gì trong đường dẫn ảo nhưng nên đưa ra một đường dẫn sẽ dễ dàng xác định là một gói. Ở đây, chúng ta đã đưa ra đường dẫn "~/bundles/bootstrap", để có thể dễ dàng xác định rằng gói này bao gồm các tệp liên quan đến bootstrap.

- Sử dụng phương thức Include để thêm một hoặc nhiều tệp JS vào một gói với đường dẫn tương đối của nó sau đường dẫn gốc bằng dấu ~.

- Tiếp theo, thêm gói vào đối tượng BundleCollection, được chỉ định làm tham số trong phương thức RegisterBundle ().

- Cuối cùng, BundleTable.EnableOptimizes = true cho phép đóng gói và thu nhỏ trong chế độ gỡ lỗi. Nếu đặt nó thành false thì nó sẽ không thực hiện đóng gói và thu nhỏ.

Cũng có thể sử dụng phương thức lớp gói của lớp bao gồm tất cả các tệp trong thư mục cụ thể như dưới đây.

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/scripts").IncludeDirectory("~/Scripts/","\*.js",true));

}

Ta có thể tạo một gói tệp JavaScript bằng ScriptBundle. MVC gọi phương thức BundleConfig.RegisterBundle() từ sự kiện Application\_Start trong Global.asax.cs, để có thể thêm tất cả các gói vào BundleCollection khi bắt đầu ứng dụng.

Ví dụ: Invoke RegisterBundle() trong sự kiện Application\_Start

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

**Sử dụng ký tự đại diện - Wildcards**

Đôi khi các tệp tập lệnh bao gồm nhiều phiên bản trong tên của tệp tập lệnh. Vì vậy, không nên thay đổi mã bất cứ khi nào chúng ta nâng cấp phiên bản tệp tập lệnh. Với việc sử dụng ký tự đại diện, chúng ta không phải chỉ định phiên bản của tệp tập lệnh. Nó tự động bao gồm các tập tin với phiên bản có sẵn. Ví dụ: Các tệp jquery bao gồm phiên bản trong một tên. Vì vậy, ta có thể sử dụng ký tự đại diện {version} để chọn phiên bản dựa trên phiên bản có sẵn.

Ví dụ: Wildcard with bundle

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery")

.Include( "~/Scripts/jquery-{version}.js"));

}

}

Bây giờ, nó sẽ chọn tập tin jquery được thêm vào trong một dự án. Nếu đã bao gồm jquery-1.7.1.js thì nó sẽ kết xuất tệp này và khi ta nâng cấp tệp jquery lên jquery-1.10.2.js thì nó sẽ tự động kết xuất tệp phiên bản 1.10 mà không cần thay đổi hoặc biên dịch mã.

**Sử dụng CDN (Content Delivery Network)**

Có thể sử dụng Mạng phân phối nội dung (Content Delivery Network) để tải các tệp tập lệnh. Ví dụ: Có thể tải thư viện jquery từ CDN như bên dưới.

Ví dụ: tải files từ CDN

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

var cdnPath = "http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.1.min.js";

bundles.Add(new ScriptBundle("~/bundles/jquery", cdnPath)

.Include( "~/Scripts/jquery-{version}.js"));

}

}

Trong đoạn mã trên, jquery sẽ được yêu cầu từ CDN trong khi ở chế độ phát hành và ở chế độ gỡ lỗi, thư viện jquery sẽ được tải từ một nguồn cục bộ. Lưu ý chúng ta nên có cơ chế dự phòng để xử lý lỗi yêu cầu CDN.

**Nhúng ScriptBundle trong chế độ Razor View**

Chúng ta đã tạo ra một gói kịch bản ở trên. Bây giờ, chúng ta sẽ tìm hiểu làm thế nào để đưa gói vào chế độ Razor view. Các gói script có thể được bao gồm bằng cách sử dụng lớp Script tĩnh. Sử dụng phương thức Script.Render() để bao gồm gói tập lệnh được chỉ định khi chạy. Ví dụ: Scripts.Render():

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title</title>

**@Scripts.Render("~/bundles/bootstrap")**

</head>

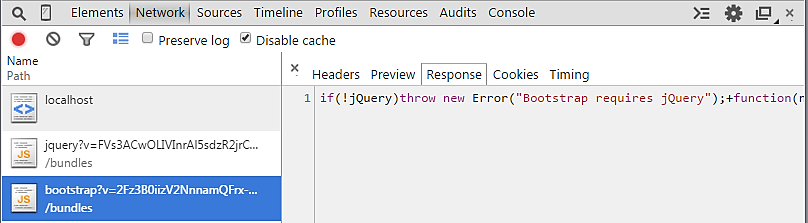
<body>

@\*html code removed for clarity \*@

</body>

</html>

Bây giờ, nếu chạy ví dụ trên thì sẽ thấy hai tệp script được kết hợp, thu nhỏ và tải trong một yêu cầu. Chú ý phải đảm bảo rằng chúng ta đã đặt debug = false trong web.config <compilation debug = "false" targetFramework = "4.5" />



Hình 3.12. Load Bundle trong trình duyệt

Như có thể thấy trong hình trên, gói bootstrap được tải trong một yêu cầu. Nó cũng đã kết hợp và thu nhỏ hai tệp JS cho bootstrap.

### 3.3.3. StyleBundle

ASP.NET MVC API bao gồm lớp StyleBundle thực hiện thu nhỏ và gói CSS. StyleBundle cũng có nguồn gốc từ một lớp Bundle vì vậy nó hỗ trợ các phương thức tương tự như ScriptBundle.

Như đã đề cập trong phần trước, chúng ta nên tạo các gói tập lệnh và tập tin css trong phương thức RegisterBundles() của lớp BundleConfig có trong tệp App\_Start -> BundleConfig.cs.

Đoạn mã sau đây cho thấy một phần của phương thức RegisterBundles(). Sử dụng phương pháp ScriptInclude hoặc includeDirectory để thêm các tệp css vào gói như dưới đây:

Ví dụ: StyleBundle

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new StyleBundle("~/bundles/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"

));

// add ScriptBundle here..

}

}

Trong ví dụ trên, chúng ta đã tạo cá thể StyleBundle với tên gói là đường dẫn ảo. Tên gói (đường dẫn ảo) phải bắt đầu bằng ~ /. Sử dụng phương thức Include() hoặc IncludeDirectory() với tên tệp css dưới dạng chuỗi.

Có thể sử dụng đường dẫn ký tự đại diện và CDN giống như ScriptBundle như trong phần trước.

**Nhúng Style Bundle trong chế độ Razor View**

Có thể sử dụng StyleBundle trong chế độ xem bố cục và hiển thị một loạt các tệp css trong một yêu cầu bằng cách sử dụng lớp Styles tĩnh. Styles là một lớp trợ giúp để kết xuất các gói css.

Ví dụ: Include Style Bundle trong 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("~/bundles/css")**

</head>

<body>

@\*html code removed for clarity \*@

</body>

</html>

Trong ví dụ trên, sử dụng phương thức Styles.Render() để bao gồm gói css được chỉ định khi chạy. Mở công cụ phát triển của trình duyệt và kiểm tra xem nó đã thu nhỏ và tải các tệp css như hiển thị bên dưới.



Hình 3.13. Load Bundle trong trình duyệt

**Chú ý:**

- Đóng gói và Giảm thiểu tối thiểu hóa tập lệnh tĩnh hoặc tập tin css giúp giảm thời gian tải trang.

- Khung MVC cung cấp các lớp ScriptBundle, StyleBundle và DynamicFolderBundle. StyleBundle thực hiện thu nhỏ các tệp CSS.

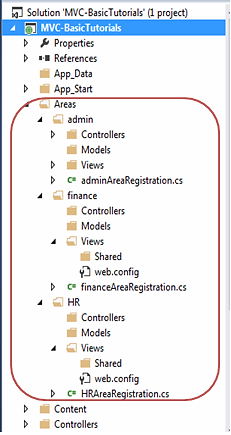
- Tạo các gói script hoặc css trong lớp BundleConfig có trong thư mục App\_Start.

- Sử dụng ký tự đại diện {version} để hiển thị tệp phiên bản có sẵn khi chạy.

- Sử dụng phương thức Styles.Render("tên gói") để nhúng các gói css trong chế độ Razor View.

## 3.4. Khu vực – Area

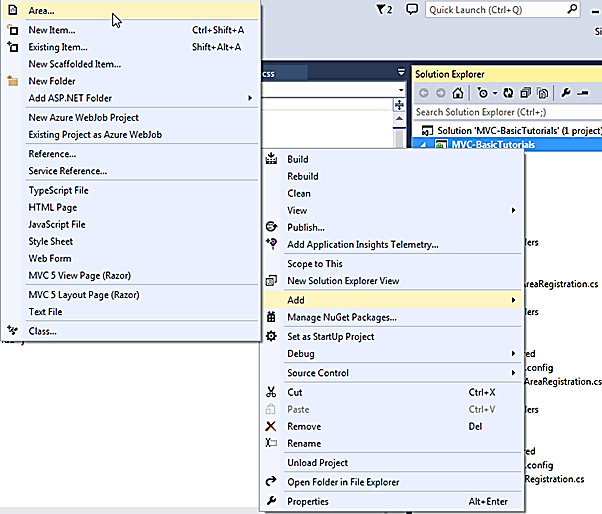
ASP.NET MVC Framework bao gồm các thư mục riêng cho Model, View và Controller. Tuy nhiên, ứng dụng lớn có thể bao gồm một số lượng lớn Controller, Model và View. Vì vậy, để duy trì số lượng lớn các đối tượng trên với cấu trúc dự án ASP.NET MVC mặc định có thể trở nên khó quản lý. ASP.NET MVC đưa ra khái niệm Area cho phép chúng ta phân vùng ứng dụng lớn thành các đơn vị nhỏ hơn, trong đó mỗi đơn vị chứa cấu trúc thư mục MVC riêng, giống như cấu trúc thư mục MVC mặc định. Ví dụ: ứng dụng doanh nghiệp lớn có thể có các mô-đun khác nhau như quản trị viên, tài chính, nhân sự, tiếp thị, v.v. Vì vậy, Area có thể chứa cấu trúc thư mục MVC riêng cho tất cả các mô-đun này như được hiển thị bên dưới.



Hình 3.14. Area

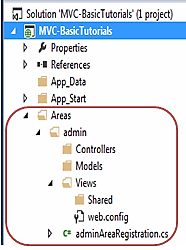
**Tạo Area**

- Click chuột phải vào Solution Explorer\Add\Area..



Hình 3.15. Tạo Area

- Đặt tên cho Area (ví dụ: admin) sẽ xuất hiện thư mục như hình sau:



Hình 3.16. Area admin

Mỗi khu vực bao gồm lớp AreaRegistration trong tệp {tên khu vực} + tệp AreaRegistration.cs. Dưới đây là mã nguồn lớp adminAreaRegistration được tạo với khu vực admin.

Area Registration:

public class adminAreaRegistration : AreaRegistration

{

public override string AreaName

{

get

{

return "admin";

}

}

public override void RegisterArea(AreaRegistrationContext context)

{

context.MapRoute(

"admin\_default",

"admin/{controller}/{action}/{id}",

new { action = "Index", id = UrlParameter.Optional }

);

}

}

Lớp AreaRegistration ghi đè phương thức RegisterArea để ánh xạ các định tuyến cho khu vực. Trong ví dụ trên, mọi URL bắt đầu với quản trị viên sẽ được xử lý bởi các bộ điều khiển có trong cấu trúc thư mục quản trị trong thư mục Area. Ví dụ: http://localhost/admin/[profile] sẽ được xử lý bởi bộ điều khiển có trong thư mục Areas/admin/controller/[Profile]Controller.

Cuối cùng, tất cả các khu vực phải được đăng ký trong sự kiện Application\_Start trong Global.asax.cs với tên AreaRegistration.Register ALLAreas();

# CÂU HỎI ÔN TẬP CHƯƠNG 3

1. Tạo ứng dụng ASP.NET MVC quản lý bán hàng và thực hiện các yêu cầu sau:

- Tải mẫu giao diện dành cho người dùng (tham khảo trên trang https://www.templatemonster.com).

- Tạo Layout View cho giao diện người dùng.

- Tạo trang View sử dụng Layout trên.

- Tạo Area dành cho quản trị.

- Tải mẫu giao diện dành cho quản trị (tham khảo trên trang https://startbootstrap.com/themes).

- Tạo Layout View cho giao diện quản trị.

- Tạo trang View sử dụng Layout trên.

2. Tạo ứng dụng ASP.NET MVC quản lý nhân sự và thực hiện các yêu cầu sau:

- Tải mẫu giao diện dành cho người dùng (tham khảo trên trang https://www.templatemonster.com).

- Tạo Layout View cho giao diện người dùng.

- Tạo trang View sử dụng Layout trên.

- Tạo Area dành cho quản trị.

- Tải mẫu giao diện dành cho quản trị (tham khảo trên trang https://startbootstrap.com/themes).

- Tạo Layout View cho giao diện quản trị.

- Tạo trang View sử dụng Layout trên.

3. Tạo ứng dụng ASP.NET MVC quản lý giáo trình điện tử và thực hiện các yêu cầu sau:

- Tải mẫu giao diện dành cho người dùng (tham khảo trên trang https://www.templatemonster.com).

- Tạo Layout View cho giao diện người dùng.

- Tạo trang View sử dụng Layout trên.

- Tạo Area dành cho quản trị.

- Tải mẫu giao diện dành cho quản trị (tham khảo trên trang https://startbootstrap.com/themes).

- Tạo Layout View cho giao diện quản trị.

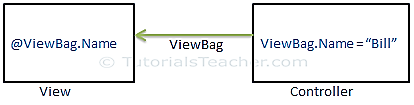
- Tạo trang View sử dụng Layout trên.

# Chương 4. TRUYỀN DỮ LIỆU TRONG ASP.NET MVC

## 4.1. ViewBag

Chúng ta đã nghiên cứu đối tượng mô hình được sử dụng để gửi dữ liệu trong chế độ Razor View. Tuy nhiên, có thể có một số tình huống mà ta muốn gửi một lượng nhỏ dữ liệu tạm thời để xem. Do đó, MVC bao gồm thành phần ViewBag.

ViewBag giúp chuyển dữ liệu tạm thời (không bao gồm trong Model) từ Controller sang View. ViewBag là một thuộc tính kiểu động của lớp ControllerBase, là lớp cơ sở của tất cả các Controller.



Hình 4.1. ViewBag

Trong hình trên, thuộc tính Name được đính kèm vào ViewBag với ký hiệu dấu chấm và gán giá trị là chuỗi "Bill" từ trong controller. Điều này có thể được truy cập trong View theo cú pháp @ViewBag.Name. (@ là cú pháp Razor để truy cập biến phía máy chủ).

Ta có thể gán một đối tượng kiểu nguyên thủy hoặc phức tạp làm giá trị cho thuộc tính ViewBag. Cũng có thể gán bất kỳ số lượng thuộc tính và giá trị nào cho ViewBag. Nếu gán cùng tên thuộc tính nhiều lần cho ViewBag, thì nó sẽ chỉ xem xét giá trị cuối cùng được gán cho thuộc tính.

**Chú ý:** ViewBag chỉ chuyển dữ liệu từ Controller sang View. Giá trị ViewBag = null nếu chuyển hướng xảy ra.

Ví dụ: Thiết lập ViewBag trong phương thức hành động.

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 }

};

public ActionResult Index()// GET: Student

{

ViewBag.TotalStudents = studentList.Count();

return View();

}

}}

Trong ví dụ trên, chúng ta muốn hiển thị tổng số sinh viên trong. Vì vậy, chúng ta đã đính kèm thuộc tính TotalStudents vào ViewBag và gán số lượng sinh viên bằng studentList.Count().

Bây giờ, trong chế độ xem Index.cshtml, ta có thể truy cập thuộc tính ViewBag.TotalStudents và hiển thị tất cả thông tin sinh viên như sau.

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

Trong nội bộ, ViewBag là một trình bao bọc xung quanh ViewData. Nó sẽ đưa ra một ngoại lệ nếu tên thuộc tính ViewBag khớp với khóa của ViewData.

**Chú ý:**

- ViewBag chuyển dữ liệu từ controller sang view, lý tưởng là dữ liệu tạm thời không có trong một mô hình.

- ViewBag là một thuộc tính động tận dụng các tính năng động trong C#.

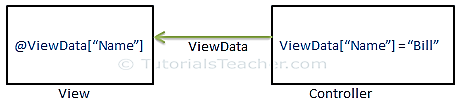
- Có thể gán bất kỳ số lượng thích hợp và giá trị nào cho ViewBag.

- Thời gian sống của ViewBag chỉ tồn tại trong yêu cầu http hiện tại. Giá trị ViewBag = null nếu chuyển hướng (chuyển trang) xảy ra.

- ViewBag là một trình bao bọc xung quanh ViewData.

## 4.2. ViewData

ViewData tương tự như ViewBag. Nó rất hữu ích trong việc chuyển dữ liệu từ Controller sang View. ViewData là một từ điển có thể chứa các cặp khóa-giá trị trong đó mỗi khóa phải là chuỗi.



Hình 4.2. ViewData

**Chú ý:** ViewData chỉ chuyển dữ liệu từ Controller sang View chứ không ngược lại. Nó chỉ có hiệu lực trong yêu cầu hiện tại.

Ví dụ sau minh họa cách truyền dữ liệu từ bộ điều khiển sang chế độ xem bằng ViewData.

**Thiết lập ViewData trong phương thức hành động**

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();

}

Truy cập ViewData trong Razor View

<ul>

@foreach (var std in ViewData["students"] as IList<Student>)

{

<li>

@std.StudentName

</li>

}

</ul>

Lưu ý rằng chúng ta phải truyền các giá trị ViewData cho kiểu dữ liệu phù hợp.

Cũng có thể thêm KeyValuePair vào ViewData như ví dụ sau:

**Thêm 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();

}

Cả ViewData và ViewBag đều sử dụng cùng một từ điển bên trong. Vì vậy, ta không thể sử dụng các tên thuộc tính của ViewData Key giống với của ViewBag, nếu không nó sẽ ném ngoại lệ khi chạy.

**Sử dụng ViewBag cùng ViewData**

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();

}

**Chú ý:**

- ViewData chuyển dữ liệu từ Controller sang View, chứ không ngược lại.

- ViewData có nguồn gốc từ ViewDataDipedia là một loại từ điển.

- Thời gian sống của ViewData chỉ kéo dài trong yêu cầu http hiện tại. Giá trị ViewData sẽ bị xóa nếu chuyển hướng xảy ra.

- Giá trị ViewData phải là được xác định trước khi sử dụng.

- ViewBag nội bộ chèn dữ liệu vào từ điển ViewData. Vì vậy, khóa của ViewData và thuộc tính của ViewBag phải KHÔNG giống nhau.

## 4.3. TempData

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 muốn chuyển dữ liệu không quan trọng từ một phương thức hành động này sang một phương thức hành động khác của cùng một Controller cũng như chuyển hướng. Đây là loại từ điển có nguồn gốc từ TempDataDipedia.

Có thể thêm một cặp khóa-giá trị trong TempData như trong ví dụ dưới đây:

public class HomeController : Controller

{

// GET: Student

public HomeController()

{

}

public ActionResult Index()

{

TempData["name"] = "Test data";

TempData["age"] = 30;

return View();

}

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());

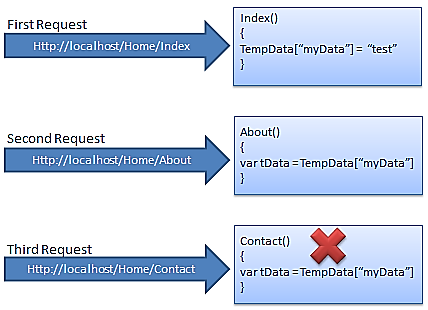
// do something with userName or userAge here

return View();

}

}

Trong ví dụ trên, chúng ta đã thêm dữ liệu vào TempData và truy cập cùng dữ liệu bằng một khóa bên trong phương thức hành động khác. Lưu ý rằng chúng ta đã chuyển đổi các giá trị thành loại thích hợp. Hình sau minh họa cách hoạt động của TempData.



Hình 4.3. TempData

Như ta có thể thấy trong hình trên, chúng ta thêm dữ liệu thử nghiệm trong TempData trong yêu cầu đầu tiên và trong yêu cầu thứ hai tiếp theo, chúng ta truy cập dữ liệu thử nghiệm từ TempData mà chúng ta đã lưu trữ trong yêu cầu đầu tiên. Tuy nhiên, ta không thể nhận được cùng một dữ liệu trong yêu cầu thứ ba vì TempData sẽ bị xóa sau yêu cầu thứ hai.

Để giữ lại các giá trị TempData trong yêu cầu thứ ba liên tiếp, chúng ta sử dụng phương thức TempData.Keep().

**TempData.Keep()**

public class HomeController : Controller

{

public HomeController()

{

}

public ActionResult Index()

{

TempData["myData"] = "Test data";

return View();

}

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();

}

}

**Chú ý:**

- TempData có thể được sử dụng để lưu trữ dữ liệu giữa hai yêu cầu liên tiếp. Giá trị TempData sẽ được giữ lại trong quá trình chuyển hướng.

- TempData là một loại TempDataDipedia.

- TempData sử dụng Phiên (Session) để lưu trữ dữ liệu. Vì vậy, hãy nghĩ về nó như một Phiên có thời gian sống ngắn.

- Giá trị TempData phải được xác định trước khi sử dụng. Kiểm tra giá trị null để tránh lỗi khi chạy.

- TempData chỉ có thể được sử dụng để lưu trữ các thông báo một lần như thông báo lỗi, thông báo xác thực.

- Sử dụng phương thức TempData.Keep() để giữ tất cả các giá trị của TempData trong yêu cầu thứ ba.

# CÂU HỎI ÔN TẬP CHƯƠNG 4

1. Sử dụng ứng dụng quản lý bán hàng MVC ở chương 3, tiếp tục thực hiện các yêu cầu sau:

- Sử dụng ViewBag để truyền, hiển thị tiêu đề trang từ Controller sang View.

- Sử dụng ViewData để truyền, hiển thị danh sách nhóm sản phẩm bàn hàng từ Controller sang View.

- Sử dụng TempData để truyền, hiển thị chi tiết nhóm sản phẩm bán hàng giữa các trang web.

2. Sử dụng ứng dụng quản lý nhân sự MVC ở chương 3, tiếp tục thực hiện các yêu cầu sau:

- Sử dụng ViewBag để truyền, hiển thị tiêu đề trang từ Controller sang View.

- Sử dụng ViewData để truyền, hiển thị danh sách đơn vị/phòng từ Controller sang View.

- Sử dụng TempData để truyền, hiển thị chi tiết nhóm đơn vị/phòng giữa các trang web.

3. Sử dụng ứng dụng quản lý giáo trình điện tử MVC ở chương 3, tiếp tục thực hiện các yêu cầu sau:

- Sử dụng ViewBag để truyền, hiển thị tiêu đề trang từ Controller sang View.

- Sử dụng ViewData để truyền, hiển thị danh sách nhóm giáo trình điện tử từ Controller sang View.

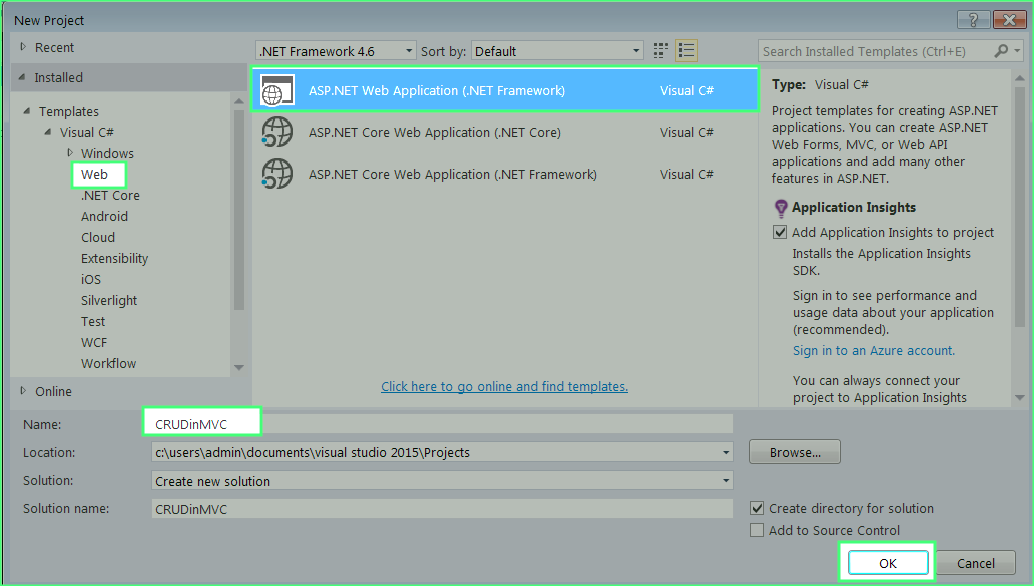
- Sử dụng TempData để truyền, hiển thị chi tiết nhóm giáo trình điện tử giữa các trang web.

# Chương 5. LẬP TRÌNH TƯƠNG TÁC CƠ SỞ DỮ LIỆU KHÔNG SỬ DỤNG ENTITY FRAMEWORK

## 5.1. Tạo mới Project

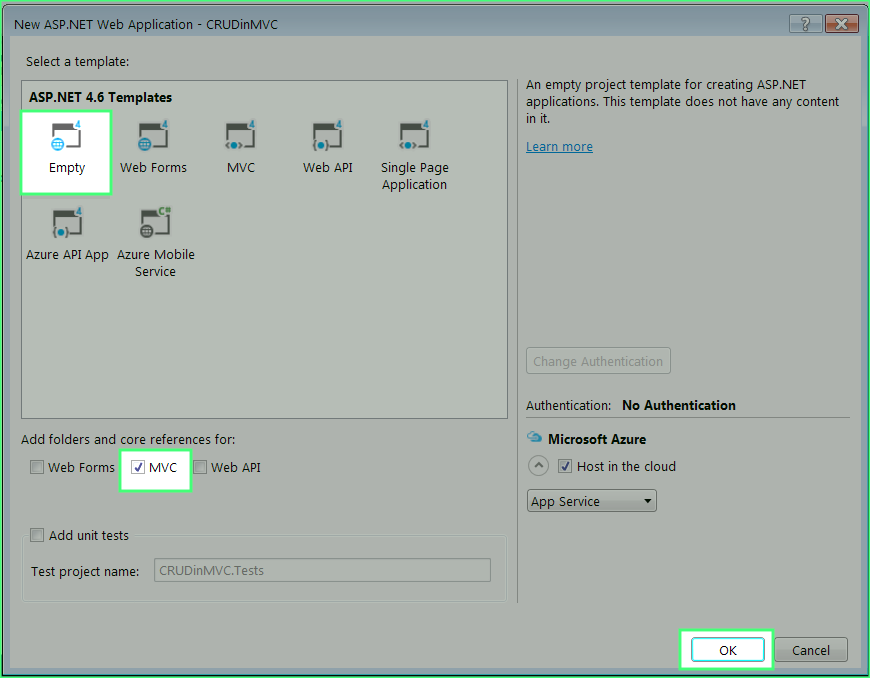
Mở Visual Studio. Chọn File\New\Project.

Chọn Web trong phần mẫu và chọn ASP.NET Web Application (Net.Framework). Đặt tên CRUDinMVC và nhấp vào OK.



Hình 5.1. Tạo mới Project

Một cửa sổ mẫu sẽ hiện ra. Chọn mẫu Empty và click chọn hộp kiểm MVC như được mô tả trong hình sau.



Hình 5.2. Empty project MVC

## 5.2. Tạo Model Class

Click chuột phải vào thư mục Model\Add\Class… Đặt tên lớp là StudentModel.cs.

Xây dựng các thuộc tính cho lớp trên như sau:

1. using **System**.**ComponentModel**.**DataAnnotations**;
3. namespace **CRUDinMVC**.**Models**
4. {
5. public class **StudentModel**
6. {
7. [**Display**(**Name** = "Id")]
8. public int **Id** { get; set; }
10. [**Required**(**ErrorMessage** = "First name is required.")]
11. public string **Name** { get; set; }
13. [**Required**(**ErrorMessage** = "City is required.")]
14. public string **City** { get; set; }
16. [**Required**(**ErrorMessage** = "Address is required.")]
17. public string **Address** { get; set; }
18. }
19. }

## 5.3. Tạo Controller

Click chuột phải vào thư mục Controllers\Add\Controller.

Trong cửa sổ xuất hiện, chọn MVC 5 Controller with read/write actions sau đó chọn Add.

Đặt tên cho Controller là StudentController sau đó chọn Add.

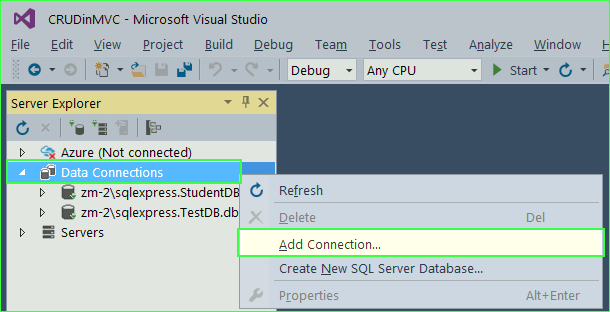
## 5.4. Tạo Database, Table và Stored Procedure

Mở panel Explorer.

Click chuột phải vào Data Connections\Create New SQL Server Database.

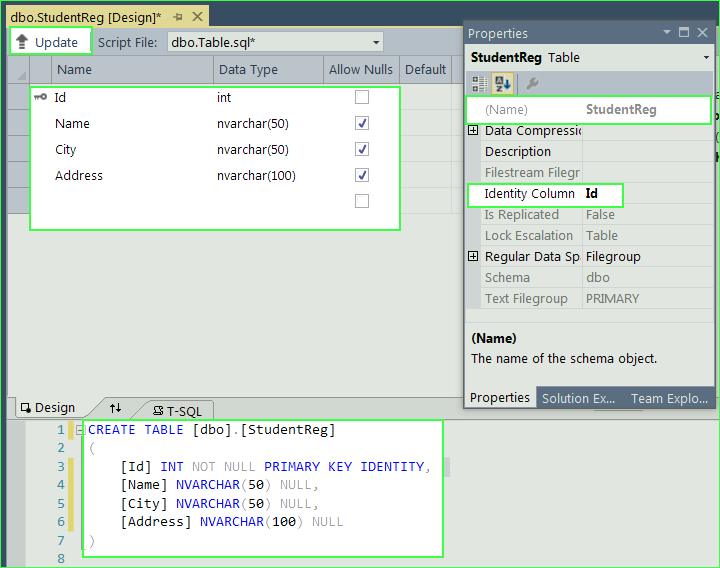
Đặt tên cơ sở dữ liệu là StudentDB.

Nếu đã có cơ sở dữ liệu thì có thể tạo kết nối với cơ sở dữ liệu hiện có. Click chuột phải vào Data Connections\Add Connection.



Hình 5.3. Add Connection

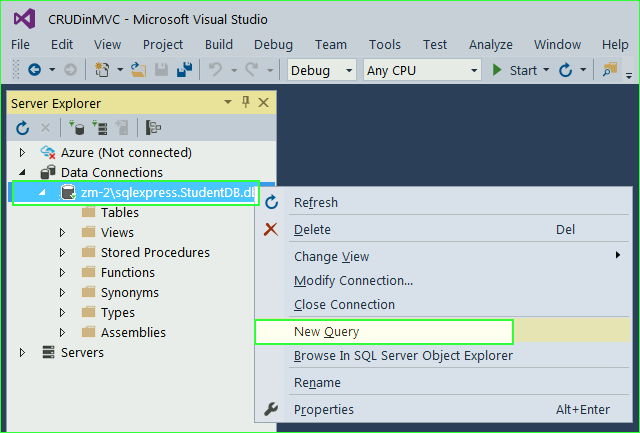
Tiếp theo ta tạo các bảng trong cơ sở dữ liệu. Để minh họa cho nội dung này chúng ta tạo bảng **StudentReg** với các trường thông tin như sau:



Hình 5.4. Bảng StudentReg

Tiếp theo chúng ta tạo các thủ tục trong SQL để thực hiện các thao tác **Create**, **Read**, **Update** và **Delete**.

Mở cửa sổ truy vấn mới. Click chuột phải vào tên cơ sở dữ liệu trong Server Explorer và chọn New Query. Bây giờ, có thể dán đoạn mã sau lần lượt từng đoạn một và thực thi mã bằng cách nhấn Ctrl + Shift + E hoặc nhấp vào biểu tượng Execute ở góc trên cùng bên trái của cửa sổ truy vấn.



Hình 5.5. Tạo mới truy vấn

**1. Tạo thủ tục thêm mới - AddNewStudent**

1. **Create** procedure [dbo].[**AddNewStudent**]
2. (
3. @Name nvarchar (50),
4. @City nvarchar (50),
5. @Address nvarchar (100)
6. )
7. as
8. begin
9. **Insert** into **StudentReg(Name,City,Address)** values(@Name,@City,@Address)
10. **End**

**2. Tạo thủ tục xem chi tiết - GetStudentDetails**

1. **Create** **Procedure** [dbo].[**GetStudentDetails**]
2. as
3. begin
4. select \* from **StudentReg**
5. **End**

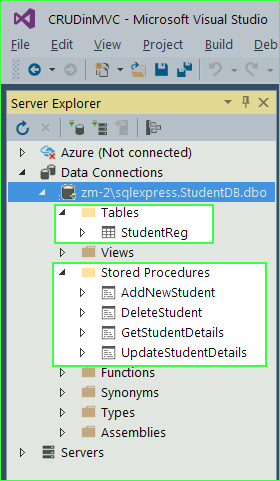
**3. Tạo thủ tục cập nhật - UpdateStudentDetails**

1. **Create** procedure [dbo].[**UpdateStudentDetails**]
2. (
3. @StdId int,
4. @Name nvarchar (50),
5. @City nvarchar (50),
6. @Address nvarchar (100)
7. )
8. as
9. begin
10. **Update** **StudentReg**
11. set **Name**=@Name,
12. **City**=@City,
13. **Address**=@Address
14. where **Id**=@StdId
15. **End**

**4. Tạo thủ tục xóa - DeleteStudent**

1. **Create** procedure [dbo].[**DeleteStudent**]
2. (
3. @StdId int
4. )
5. as
6. begin
7. **Delete** from **StudentReg** where **Id**=@StdId
8. **End**

Sau khi chúng ta thực hiện các bước trên sẽ có kết quả như trong hình sau:



Hình 5.6. Server Explorer

## 5.5. Tạo chuỗi kết nối trong Web.config

Thêm chuỗi kết nối trong tệp Web.config. Ta có thể tìm thấy chuỗi kết nối trong các thuộc tính cơ sở dữ liệu. Nhấp chuột phải vào tên cơ sở dữ liệu và chọn Properties. Sao chép chuỗi kết nối, mở file Web.config và dán chuỗi kết nối như sau.

1. <connectionStrings>
2. <add name="StudentConn" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=StudentDB;Integrated Security=True;Pooling=False"/>
3. </connectionStrings>

## 5.6. Tạo lớp StudentDBHandle.cs

Lớp StudentDBHandle.cs để xử lý tất cả các hoạt động liên quan đến cơ sở dữ liệu. Chúng ta có thể sử dụng các phương thức thao tác với cơ sở dữ liệu thông qua đối tượng SqlCommand như sau.

Click chuột phải vào thư mục Models\Add\Class. Đặt tên class là **StudentDBHandle.cs.** Xây dựng các phương thức cho lớp như sau:

1. using **System**;
2. using **System**.**Collections**.**Generic**;
3. using **System**.**Data**;
4. using **System**.**Data**.**SqlClient**;
5. using **System**.**Configuration**;
7. namespace **CRUDinMVC**.**Models**
8. {
9. public class **StudentDBHandle**
10. {
11. private **SqlConnection** con;
12. private void connection()
13. {
14. string constring = **ConfigurationManager**.**ConnectionStrings**["studentconn"].**ToString**();
15. con = new **SqlConnection**(constring);
16. }
18. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADD NEW STUDENT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
19. public bool **AddStudent**(**StudentModel** smodel)
20. {
21. connection();
22. **SqlCommand** cmd = new **SqlCommand**("AddNewStudent", con);
23. cmd.**CommandType** = **CommandType**.**StoredProcedure**;
25. cmd.**Parameters**.**AddWithValue**("@Name", smodel.**Name**);
26. cmd.**Parameters**.**AddWithValue**("@City", smodel.**City**);
27. cmd.**Parameters**.**AddWithValue**("@Address", smodel.**Address**);
29. con.**Open**();
30. int i = cmd.**ExecuteNonQuery**();
31. con.**Close**();
33. if (i >= 1)
34. return true;
35. else
36. return false;
37. }
38. // \*\*\*\*\*\*\*\*\*\* VIEW STUDENT DETAILS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
39. public **List**<**StudentModel**> **GetStudent**()
40. {
41. connection();
42. **List**<**StudentModel**> studentlist = new **List**<**StudentModel**>();
44. **SqlCommand** cmd = new **SqlCommand**("GetStudentDetails", con);
45. cmd.**CommandType** = **CommandType**.**StoredProcedure**;
46. **SqlDataAdapter** sd = new **SqlDataAdapter**(cmd);
47. **DataTable** dt = new **DataTable**();
49. con.**Open**();
50. sd.**Fill**(dt);
51. con.**Close**();
53. foreach(**DataRow** dr in dt.**Rows**)
54. {
55. studentlist.**Add**(
56. new **StudentModel**
57. {
58. **Id** = **Convert**.**ToInt32**(dr["Id"]),
59. **Name** = **Convert**.**ToString**(dr["Name"]),
60. **City** = **Convert**.**ToString**(dr["City"]),
61. **Address** = **Convert**.**ToString**(dr["Address"])
62. });
63. }
64. return studentlist;
65. }
67. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* UPDATE STUDENT DETAILS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
68. public bool **UpdateDetails**(**StudentModel** smodel)
69. {
70. connection();
71. **SqlCommand** cmd = new **SqlCommand**("UpdateStudentDetails", con);
72. cmd.**CommandType** = **CommandType**.**StoredProcedure**;
74. cmd.**Parameters**.**AddWithValue**("@StdId", smodel.**Id**);
75. cmd.**Parameters**.**AddWithValue**("@Name", smodel.**Name**);
76. cmd.**Parameters**.**AddWithValue**("@City", smodel.**City**);
77. cmd.**Parameters**.**AddWithValue**("@Address", smodel.**Address**);
79. con.**Open**();
80. int i = cmd.**ExecuteNonQuery**();
81. con.**Close**();
83. if (i >= 1)
84. return true;
85. else
86. return false;
87. }
88. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DELETE STUDENT DETAILS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
89. public bool **DeleteStudent**(int id)
90. {
91. connection();
92. **SqlCommand** cmd = new **SqlCommand**("DeleteStudent", con);
93. cmd.**CommandType** = **CommandType**.**StoredProcedure**;
95. cmd.**Parameters**.**AddWithValue**("@StdId", id);
97. con.**Open**();
98. int i = cmd.**ExecuteNonQuery**();
99. con.**Close**();
101. if (i >= 1)
102. return true;
103. else
104. return false;
105. }
106. }
107. }

Ngoài ra chúng ta có thể sử dụng thuộc tính CommandType để quy định cho SqlCommand thực thi các truy vấn SQL như sau.

Cách 1:

SqlCommand sqlcom;

sqlcom=new SqlCommand(ssql,sqlconn)

Cách 2:

SqlCommand sqlcom = new SqlCommand();

sqlcom.Connection = sqlconn;

sqlcom.CommandType = CommandType.Text;

sqlcom.CommandText = "select sTitle from tblHello where pkHelloID=1";

Phương thức:

* ExecuteReader: Dùng để thực thi đọc cơ sở dữ liệu từ bảng cơ sở dữ liệu.
* ExecuteNonQuery: Dùng để thực thi các phát biểu T-Sql như: Insert, Update, Delete, Create,…
* ExecuteScalar: Trả về từ phát biểu SQL dạng Select chỉ có một cột một hàng.

Ví dụ:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Web;

namespace BanjoOnMyKnee.Models

{

public class StudentDBHandle

{

public string connectionString = ".\\SQLEXPRESS; Initial-Catalog=StudentDB; Integrated-Security=true";

public void CreateStudent(StudentModel model)

{

using(SqlConnection connection = new SqlConnection(connectionString))

using (SqlCommand command = new SqlCommand("",connection))

{

command.CommandText = "insert into StudentTable values(@Name)";

command.Parameters.AddWithValue("@Name", model.Name);

command.ExecuteNonQuery();

}

}

public StudentModel FindStudent(int id)

{

var model = new StudentModel();

using (SqlConnection connection = new SqlConnection(connectionString))

using (SqlCommand command = new SqlCommand("", connection))

{

command.CommandText = "select \* from StudentTable where Id=@Id";

command.Parameters.AddWithValue("@Id",id);

SqlDataReader reader = command.ExecuteReader();

model.Id = id;

model.Name = reader["Name"].ToString();

}

return model;

}

public void DeleteStudent(StudentModel model)

{

using (SqlConnection connection = new SqlConnection(connectionString))

using (SqlCommand command = new SqlCommand("", connection))

{

command.CommandText = "delete from StudentTable where Id=@Id";

command.Parameters.AddWithValue("@Id", model.Id);

command.ExecuteNonQuery();

}

}

public void EditSomething(ViewModel model)

{

using (SqlConnection connection = new SqlConnection(connectionString))

using (SqlCommand command = new SqlCommand("", connection))

{

command.CommandText = "Update StudentTable set Name=@Name where Id=@Id";

command.Parameters.AddWithValue("@Name", model.Name);

command.Parameters.AddWithValue("@Id", model.Id);

command.ExecuteNonQuery();

}

}

}

}

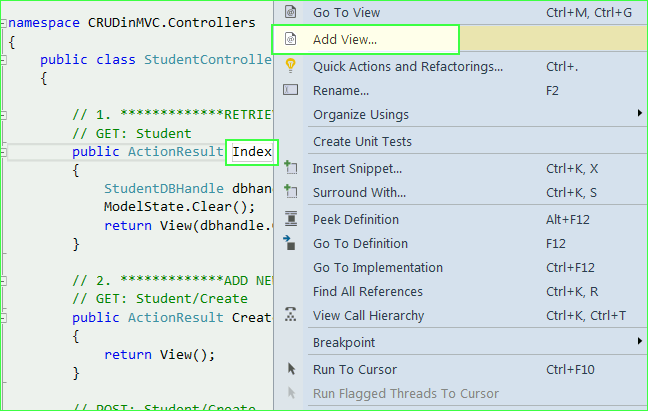
## 5.7. Tạo phương thức hành động trong StudentController

Các phương thức hành động gồm: **Index, Create()**,**Edit()**, **Delete().**

1. using **System**.**Web**.**Mvc**;
2. using **CRUDinMVC**.**Models**;
4. namespace **CRUDinMVC**.**Controllers**
5. {
6. public class **StudentController** : **Controller**
7. {
9. // 1. \*\*\*\*\*\*\*\*\*\*\*\*\*RETRIEVE ALL STUDENT DETAILS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
10. // GET: Student
11. public **ActionResult** **Index**()
12. {
13. **StudentDBHandle** dbhandle = new **StudentDBHandle**();
14. **ModelState**.**Clear**();
15. return **View**(dbhandle.**GetStudent**());
16. }
18. // 2. \*\*\*\*\*\*\*\*\*\*\*\*\*ADD NEW STUDENT \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
19. // GET: Student/Create
20. public **ActionResult** **Create**()
21. {
22. return **View**();
23. }
25. // POST: Student/Create
26. [**HttpPost**]
27. public **ActionResult** **Create**(**StudentModel** smodel)
28. {
29. try
30. {
31. if (**ModelState**.**IsValid**)
32. {
33. **StudentDBHandle** sdb = new **StudentDBHandle**();
34. if (sdb.**AddStudent**(smodel))
35. {
36. **ViewBag**.**Message** = "Student Details Added Successfully";
37. **ModelState**.**Clear**();
38. }
39. }
40. return **View**();
41. }
42. catch
43. {
44. return **View**();
45. }
46. }
48. // 3. \*\*\*\*\*\*\*\*\*\*\*\*\* EDIT STUDENT DETAILS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
49. // GET: Student/Edit/5
50. public **ActionResult** **Edit**(int id)
51. {
52. **StudentDBHandle** sdb = new **StudentDBHandle**();
53. return **View**(sdb.**GetStudent**().**Find**(smodel => smodel.**Id** == id));
54. }
56. // POST: Student/Edit/5
57. [**HttpPost**]
58. public **ActionResult** **Edit**(int id, **StudentModel** smodel)
59. {
60. try
61. {
62. **StudentDBHandle** sdb = new **StudentDBHandle**();
63. sdb.**UpdateDetails**(smodel);
64. return **RedirectToAction**("Index");
65. }
66. catch
67. {
68. return **View**();
69. }
70. }
72. // 4. \*\*\*\*\*\*\*\*\*\*\*\*\* DELETE STUDENT DETAILS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
73. // GET: Student/Delete/5
74. public **ActionResult** **Delete**(int id)
75. {
76. try
77. {
78. **StudentDBHandle** sdb = new **StudentDBHandle**();
79. if (sdb.**DeleteStudent**(id))
80. {
81. **ViewBag**.**AlertMsg** = "Student Deleted Successfully";
82. }
83. return **RedirectToAction**("Index");
84. }
85. catch
86. {
87. return **View**();
88. }
89. }
90. }
91. }

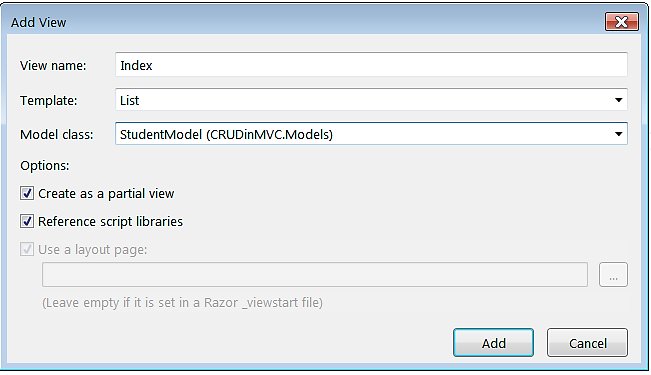
## 5.8. Tạo Partial View từ phương thức hành động

Click chuột phải vào phương thức hành động Index() chọn Add View.



Hình 5.7. Add View từ phương thức hành động

Trong cửa sổ xuất hiện, chọn các lựa chọn như hình sau:



Hình 5.8. Hộp thoại Add View

Tiếp theo chúng ta viết code cho Index View. Mở trang **Index()** và tạo code như sau:

1. @model **IEnumerable**<**CRUDinMVC**.**Models**.**StudentModel**>
3. <p>
4. @Html.**ActionLink**("Create New", "Create")
5. </p>
6. <table class="table">
7. <tr>
8. <th>
9. @Html.**DisplayNameFor**(model => model.**Name**)
10. </th>
11. <th>
12. @Html.**DisplayNameFor**(model => model.**City**)
13. </th>
14. <th>
15. @Html.**DisplayNameFor**(model => model.**Address**)
16. </th>
17. <th></th>
18. </tr>
20. @foreach (var item in **Model**) {
21. <tr>
22. <td>
23. @Html.**DisplayFor**(modelItem => item.**Name**)
24. </td>
25. <td>
26. @Html.**DisplayFor**(modelItem => item.**City**)
27. </td>
28. <td>
29. @Html.**DisplayFor**(modelItem => item.**Address**)
30. </td>
31. <td>
32. @Html.**ActionLink**("Edit", "Edit", new { id=item.**Id** }) |
33. @Html.**ActionLink**("Delete", "Delete", new { id=item.**Id** }, new { onclick = "return confirm('Are sure wants to delete?');" })
34. </td>
35. </tr>
36. }
38. </table>

Tương tự như trên, chúng ta tạo các View cho các hành động **Create()** và **Edit().**

Trang **Create():**

1. @model **CRUDinMVC**.**Models**.**StudentModel**

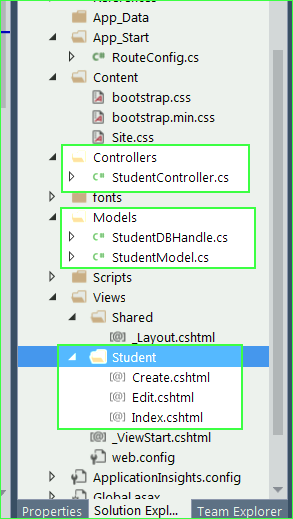
4. @using (**Html**.**BeginForm**())
5. {
6. @Html.**AntiForgeryToken**()
8. <div class="form-horizontal">
9. <h4>**StudentModel**</h4>
10. <hr />
11. @Html.**ValidationSummary**(true, "", new { @class = "text-danger" })
12. <div class="form-group">
13. @Html.**LabelFor**(model => model.**Name**, htmlAttributes: new { @class = "control-label col-md-2" })
14. <div class="col-md-10">
15. @Html.**EditorFor**(model => model.**Name**, new { htmlAttributes = new { @class = "form-control" } })
16. @Html.**ValidationMessageFor**(model => model.**Name**, "", new { @class = "text-danger" })
17. </div>
18. </div>
20. <div class="form-group">
21. @Html.**LabelFor**(model => model.**City**, htmlAttributes: new { @class = "control-label col-md-2" })
22. <div class="col-md-10">
23. @Html.**EditorFor**(model => model.**City**, new { htmlAttributes = new { @class = "form-control" } })
24. @Html.**ValidationMessageFor**(model => model.**City**, "", new { @class = "text-danger" })
25. </div>
26. </div>
28. <div class="form-group">
29. @Html.**LabelFor**(model => model.**Address**, htmlAttributes: new { @class = "control-label col-md-2" })
30. <div class="col-md-10">
31. @Html.**EditorFor**(model => model.**Address**, new { htmlAttributes = new { @class = "form-control" } })
32. @Html.**ValidationMessageFor**(model => model.**Address**, "", new { @class = "text-danger" })
33. </div>
34. </div>
36. <div class="form-group">
37. <div class="col-md-offset-2 col-md-10">
38. <input type="submit" value="Create" class="btn btn-default" />
39. </div>
40. </div>
41. <h3>@ViewBag.**Message**</h3>
42. </div>
43. }
45. <div>
46. @Html.**ActionLink**("Back to List", "Index")
47. </div>
49. <script src="~/Scripts/jquery-1.10.2.min.js"></script>
50. <script src="~/Scripts/jquery.validate.min.js"></script>
51. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>

Trang **Edit():**

1. @model **CRUDinMVC**.**Models**.**StudentModel**

4. @using (**Html**.**BeginForm**())
5. {
6. @Html.**AntiForgeryToken**()
8. <div class="form-horizontal">
9. <h4>**StudentModel**</h4>
10. <hr />
11. @Html.**ValidationSummary**(true, "", new { @class = "text-danger" })
12. @Html.**HiddenFor**(model => model.**Id**)
14. <div class="form-group">
15. @Html.**LabelFor**(model => model.**Name**, htmlAttributes: new { @class = "control-label col-md-2" })
16. <div class="col-md-10">
17. @Html.**EditorFor**(model => model.**Name**, new { htmlAttributes = new { @class = "form-control" } })
18. @Html.**ValidationMessageFor**(model => model.**Name**, "", new { @class = "text-danger" })
19. </div>
20. </div>
22. <div class="form-group">
23. @Html.**LabelFor**(model => model.**City**, htmlAttributes: new { @class = "control-label col-md-2" })
24. <div class="col-md-10">
25. @Html.**EditorFor**(model => model.**City**, new { htmlAttributes = new { @class = "form-control" } })
26. @Html.**ValidationMessageFor**(model => model.**City**, "", new { @class = "text-danger" })
27. </div>
28. </div>
30. <div class="form-group">
31. @Html.**LabelFor**(model => model.**Address**, htmlAttributes: new { @class = "control-label col-md-2" })
32. <div class="col-md-10">
33. @Html.**EditorFor**(model => model.**Address**, new { htmlAttributes = new { @class = "form-control" } })
34. @Html.**ValidationMessageFor**(model => model.**Address**, "", new { @class = "text-danger" })
35. </div>
36. </div>
38. <div class="form-group">
39. <div class="col-md-offset-2 col-md-10">
40. <input type="submit" value="Save" class="btn btn-default" />
41. </div>
42. </div>
43. </div>
44. }
46. <div>
47. @Html.**ActionLink**("Back to List", "Index")
48. </div>
50. <script src="~/Scripts/jquery-1.10.2.min.js"></script>
51. <script src="~/Scripts/jquery.validate.min.js"></script>
52. <script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>

Thực hiện các bước trên, chúng ta có kết quả như sau:



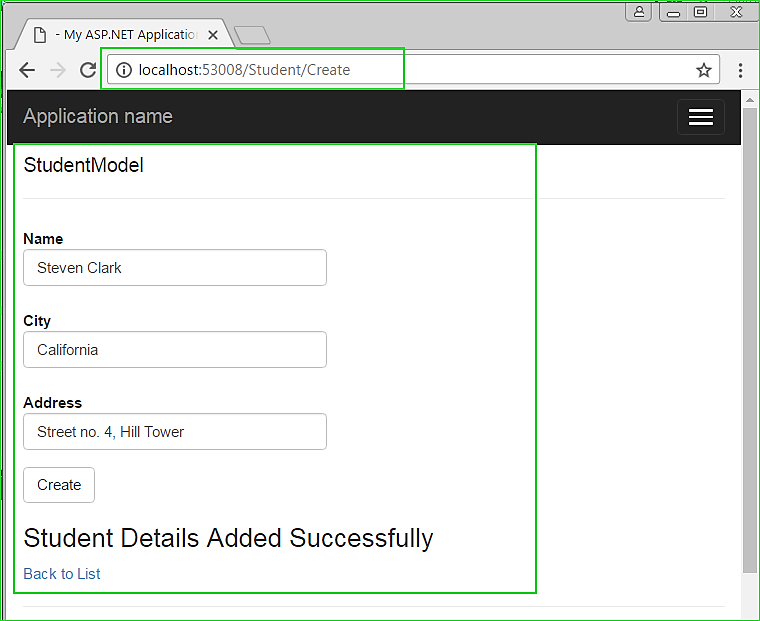
Hình 5.9. Panel Solution

## 5.9. Cấu hình RouteConfig.cs

Mở tệp RouteConfig.cs từ Solution Explorer, cấu hình như sau:

1. using **System**.**Web**.**Mvc**;
2. using **System**.**Web**.**Routing**;
4. namespace **CRUDinMVC**
5. {
6. public class **RouteConfig**
7. {
8. public static void **RegisterRoutes**(**RouteCollection** routes)
9. {
10. routes.**IgnoreRoute**("{resource}.axd/{\*pathInfo}");
12. routes.**MapRoute**(
13. name: "Default",
14. url: "{controller}/{action}/{id}",
15. defaults: new { controller = "Student", action = "Index",
16. id = **UrlParameter**.**Optional** }
17. );
18. }
19. }
20. }

Đến đây ta có thể chạy thử ứng dụng và test các hành động.



Hình 5.10. Hành động Create

# CÂU HỎI ÔN TẬP CHƯƠNG 5

1. Tạo ứng dụng MVC Quản lý bán hàng với các yêu cầu sau:

- Tạo cơ sở dữ liệu gồm các bảng: nhóm sản phẩm, sản phẩm, tài khoản.

- Tạo các thủ tục cơ sở dữ liệu để thực hiện các yêu cầu: lấy thông tin tất cả danh sách bản ghi, lấy thông tin chi tiết bản ghi, thêm mới, sửa, xóa bản ghi tương ứng với các bảng trong cơ sở dữ liệu (xem mục 5.4).

- Tạo các Model class cho các bảng trên.

- Tạo class tương tác cơ sở dữ liệu tương ứng với các bảng trong cơ sở dữ liệu (xem mục 5.6).

- Tạo Layout cho các trang của quản trị.

- Xây dựng các Controller, View thực hiện thêm mới, xem, sửa, xóa bản ghi tương ứng với các bảng trong cơ sở dữ liệu.

- Tạo Layout cho các trang người dùng.

- Tạo trang chủ website giới thiệu các sản phẩm.

2. Tạo ứng dụng MVC Quản lý nhân sự với các yêu cầu sau:

- Tạo cơ sở dữ liệu và các thủ tục cơ sở dữ liệu cho dự án quản lý nhân sự.

- Tạo Area cho quản trị

- Tạo Layout cho các trang của quản trị.

- Xây dựng các trang quản trị cơ sở dữ liệu.

- Tạo Layout cho các trang người dùng.

- Xây dựng các trang để người dùng thực hiện nhiệm vụ trong công tác quản lý nhân sự.

# Chương 6. BỘ LỌC - FILTERS

## 6.1. Filters

Trong ASP.NET MVC, một yêu cầu người dùng được chuyển đến Controller và phương thức hành động thích hợp. Tuy nhiên, có thể có trường hợp ta muốn thực thi một số logic trước hoặc sau khi phương thức hành động thực thi. ASP.NET MVC cung cấp các bộ lọc cho mục đích này.

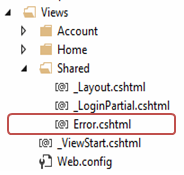
Filter là một lớp tùy chỉnh nơi ta có thể viết logic tùy chỉnh để thực thi trước hoặc sau khi phương thức hành động thực thi. Các bộ lọc có thể được áp dụng cho một phương thức hành động hoặc Controller theo cách khai báo hoặc lập trình. Khai báo có nghĩa là áp dụng một thuộc tính bộ lọc cho một phương thức hành động hoặc Controller và các phương tiện lập trình bằng cách thực hiện một giao diện tương ứng.

MVC cung cấp các loại bộ lọc khác nhau. Bảng sau liệt kê các loại bộ lọc, bộ lọc tích hợp cho loại và giao diện phải được triển khai để tạo lớp bộ lọc tùy chỉnh.

Bảng 6.1. Các loại bộ lọc trong MVC.

| Kiểu bộ lọc | Mô tả | Nhúng bộ lọc | Giao diện |
| --- | --- | --- | --- |
| Authorization filters | Thực hiện xác thực và ủy quyền trước khi thực hiện phương thức hành động. | [Authorize], [RequireHttps] | IAuthorizationFilter |
| Action filters | Thực hiện một số thao tác trước và sau khi phương thức hành động thực thi. |  | IActionFilter |
| Result filters | Thực hiện một số thao tác trước hoặc sau khi thực hiện xem kết quả. | [OutputCache] | IResultFilter |
| Exception filters | Thực hiện một số thao tác nếu có một ngoại lệ chưa được xử lý được ném trong quá trình thực thi đường dẫn ASP.NET MVC. | [HandleError] | IExceptionFilter |

Một bộ lọc ngoại lệ thực thi khi có một ngoại lệ chưa được xử lý xảy ra trong ứng dụng. Lớp HandleErrorAttribute ([Handler Error]) là lớp bộ lọc ngoại lệ tích hợp trong MVC. Lớp HandleErrorAttribute tích hợp trong file Error.cshtml được đặt trong thư mục Shared theo mặc định, khi xảy ra ngoại lệ chưa xử lý.



Hình 6.1. Đường dẫn file Error.cshtml

Ví dụ sau đây minh họa sử dụng bộ lọc ngoại lệ tích hợp Hard Error Attribution.

**Authorization Filter**

[HandleError]

public class HomeController : Controller

{

public ActionResult Index()

{

//throw exception for demo

throw new Exception("This is unhandled exception");

return View();

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

{

return View();

}

}

Trong ví dụ trên, chúng ta đã áp dụng thuộc tính [HandleError] cho HomeController. Vì vậy, bây giờ nó sẽ hiển thị trang Error.cshtml nếu bất kỳ phương thức hành động nào của HomeController bị ném ngoại lệ chưa được xử lý. Lưu ý rằng ngoại lệ chưa được xử lý là ngoại lệ không được xử lý bởi khối lệnh **try{} catch {}.**

Các bộ lọc được áp dụng cho Controller sẽ tự động được áp dụng cho tất cả các phương thức hành động của Controller. Hãy đảm bảo rằng chế độ CustomError được bật trong phần System.web của web.config, để HandleErrorAttribute hoạt động chính xác.

<customErrors mode="On" />

## 6.2. Áp dụng bộ lọc

Bộ lọc có thể được áp dụng ở ba cấp độ.

**+ Cấp độ toàn cục - Global Level**

Có thể áp dụng các bộ lọc ở cấp độ toàn cục trong sự kiện Application\_Start của tệp Global.asax.cs bằng cách sử dụng bộ lọc mặc định FilterConfig.RegisterGlobalFilters(). Các bộ lọc toàn cục sẽ được áp dụng cho tất cả các Controller và các phương thức hành động của nó.

Bộ lọc [HandleError] được áp dụng toàn cục trong ứng dụng MVC theo mặc định trong mọi ứng dụng MVC được tạo bằng Visual Studio như sau:

Ví dụ: Áp dụng bộ lọc toàn cục.

// MvcApplication class contains in Global.asax.cs file

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

}

}

// FilterConfig.cs located in App\_Start folder

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute());

}

}

**+ Cấp độ Controller - Controller level**

Bộ lọc cũng có thể được áp dụng cho lớp Controller. Vì vậy, các bộ lọc sẽ được áp dụng cho tất cả các phương thức hành động của lớp Controller đó.

Ví dụ: Bộ lọc trong Controller.

[HandleError]

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}

**+ Cấp độ phương thức hành động - Action method level**

Có thể áp dụng các bộ lọc cho một phương thức hành động riêng lẻ. Vì vậy, bộ lọc sẽ chỉ được áp dụng cho phương thức hành động cụ thể đó.

Ví dụ: Bộ lọc trong phương thức hành động.

public class HomeController : Controller

{

[HandleError]

public ActionResult Index()

{

return View();

}

}

Cùng một cách, ta có thể áp dụng nhiều bộ lọc tùy chỉnh hoặc tích hợp trên toàn cục hoặc ở cấp độ bộ điều khiển hoặc phương thức hành động cho các mục đích khác nhau, chẳng hạn như [Authorize], [RequireHttps], [ChildActionOnly], [OutputCache], [HandleError].

**Thứ tự bộ lọc:**

MVC bao gồm các loại bộ lọc khác nhau và nhiều bộ lọc có thể được áp dụng cho một lớp trình điều khiển hoặc phương thức hành động. Vì vậy, các bộ lọc chạy theo thứ tự sau đây.

1. Authorization filters (Bộ lọc ủy quyền)
2. Action filters (Bộ lọc hành động)
3. Response filters (Bộ lọc kết quả)
4. Exception filters (Bộ lọc ngoại lệ)

## 6.3. Bộ lọc tùy chỉnh

Có thể tạo các thuộc tính bộ lọc tùy chỉnh bằng cách triển khai giao diện bộ lọc phù hợp mà ta muốn tạo bộ lọc tùy chỉnh và cũng lấy ra một lớp FilterAttribute để có thể sử dụng lớp đó làm thuộc tính.

Ví dụ: Triển khai lớp IExceptionFilter và FilterAttribute để tạo bộ lọc ngoại lệ tùy chỉnh. Theo cách tương tự, triển khai giao diện IAuthorizationFilter và lớp FilterAttribute để tạo bộ lọc ủy quyền tùy chỉnh như sau:

class MyErrorHandler : FilterAttribute, IExceptionFilter

{

public override void IExceptionFilter.OnException(ExceptionContext filterContext)

{

Log(filterContext.Exception);

base.OnException(filterContext);

}

private void Log(Exception exception)

{

//log exception here..

}

}

Ngoài ra, ta cũng có thể lấy được một lớp bộ lọc tích hợp và ghi đè một phương thức thích hợp để mở rộng chức năng của các bộ lọc tích hợp.

Sau đây là bộ lọc ngoại lệ tùy chỉnh để ghi nhật ký mọi ngoại lệ chưa được xử lý bằng cách lấy lớp HandleErrorAttribute tích hợp và ghi đè phương thức OnException như sau:

class MyErrorHandler : HandleErrorAttribute

{

public override void OnException(ExceptionContext filterContext)

{

Log(filterContext.Exception);

base.OnException(filterContext);

}

private void Log(Exception exception)

{

//log exception here..

}

}

Bây giờ, có thể áp dụng thuộc tính MyErrorHandler ở cấp độ toàn cục hoặc Controller hoặc phương thức hành động, giống như cách chúng ta đã áp dụng thuộc tính HandleError.

[MyErrorHandler]

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

}

**Lưu ý:**

- Bộ lọc MVC được sử dụng để thực thi logic tùy chỉnh trước hoặc sau khi thực hiện phương thức hành động.

- Các loại bộ lọc MVC: Bộ lọc ủy quyền, bộ lọc hành động, bộ lọc kết quả, bộ lọc ngoại lệ.

- Các bộ lọc có thể được áp dụng trên toàn cục trong lớp FilterConfig hoặc ở cấp Controller hoặc cấp phương thức hành động.

- Lớp bộ lọc tùy chỉnh có thể được tạo bằng cách triển khai lớp FilterAttribute và giao diện tương ứng.

## 6.4. Bộ lọc hành động - Action Filters

Trong phần trước, chúng ta đã tìm hiểu về các bộ lọc trong MVC. Trong phần này, ta sẽ tìm hiểu về một loại bộ lọc cụ thể gọi là Bộ lọc hành động (Action Filters) trong ASP.NET MVC.

Bộ lọc hành động thực thi trước và sau khi phương thức hành động thực thi. Các thuộc tính bộ lọc hành động có thể được áp dụng cho một phương thức hành động riêng lẻ hoặc cho Controller. Khi bộ lọc hành động được áp dụng cho Controller thì nó sẽ được áp dụng cho tất cả các phương thức hành động trong Controller đó.

OutputCache là một thuộc tính của bộ lọc hành động tích hợp có thể được áp dụng cho một phương thức hành động mà chúng ta muốn lưu trữ đầu ra. Ví dụ: Đầu ra của phương thức hành động sau sẽ được lưu trong bộ nhớ cache trong 100 giây.

[OutputCache(Duration=100)]

public ActionResult Index()

{

return View();

}

**Bộ lọc hành động tùy chỉnh**

Có thể tạo bộ lọc hành động tùy chỉnh theo hai cách. Cách 1, triển khai giao diện IActionFilter và lớp FilterAttribute. Cách 2, lấy lớp trừu tượng ActionFilterAttribute.

Giao diện ActionFilter bao gồm các phương thức sau để thực hiện:

void OnActionExecuted (ActionExecutedContext filterContext)

void OnActionExecuting (ActionExecutingContext filterContext)

Lớp trừu tượng ActionFilterAttribute bao gồm các phương thức sau để ghi đè:

void OnActionExecuted(ActionExecutedContext filterContext)

void OnActionExecuting(ActionExecutingContext filterContext)

void OnResultExecuted(ResultExecutedContext filterContext)

void OnResultExecuting(ResultExecutingContext filterContext)

Có thể thấy rằng lớp ActionFilterAttribute có bốn phương thức để thực thi. Nó bao gồm các phương thức OnResultExecuted và OnResultExecuting, có thể được sử dụng để thực thi logic tùy chỉnh trước hoặc sau khi thực hiện kết quả. Các bộ lọc hành động thường được sử dụng để áp dụng các mối quan tâm xuyên suốt như ghi nhật ký, lưu trữ, ủy quyền, v.v.

Xét lớp bộ lọc Nhật ký tùy chỉnh sau đây để ghi nhật ký.

public class LogAttribute : ActionFilterAttribute

{

public override void OnActionExecuted(ActionExecutedContext filterContext)

{

Log("OnActionExecuted", filterContext.RouteData);

}

public override void OnActionExecuting(ActionExecutingContext filterContext)

{

Log("OnActionExecuting", filterContext.RouteData);

}

public override void OnResultExecuted(ResultExecutedContext filterContext)

{

Log("OnResultExecuted", filterContext.RouteData);

}

public override void OnResultExecuting(ResultExecutingContext filterContext)

{

Log("OnResultExecuting ", filterContext.RouteData);

}

private void Log(string methodName, RouteData routeData)

{

var controllerName = routeData.Values["controller"];

var actionName = routeData.Values["action"];

var message = String.Format("{0}- controller:{1} action:{2}", methodName,

controllerName,

actionName);

Debug.WriteLine(message);

}

}

Trong ví dụ trên, lớp Log dẫn xuất lớp ActionFilterAttribute. Nó ghi nhật ký trước và sau khi phương thức hành động hoặc kết quả thực thi. Ta có thể áp dụng thuộc tính Log cho bất kỳ Controller hoặc phương thức hành động nào mà ta muốn hành động đăng nhập. Bằng cách áp dụng thuộc tính Log cho Controller, nó sẽ ghi nhật ký từng phương thức hành động của Controller đó.

[Log]

public class StudentController : Controller

{

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

return View();

}

public ActionResult Contact()

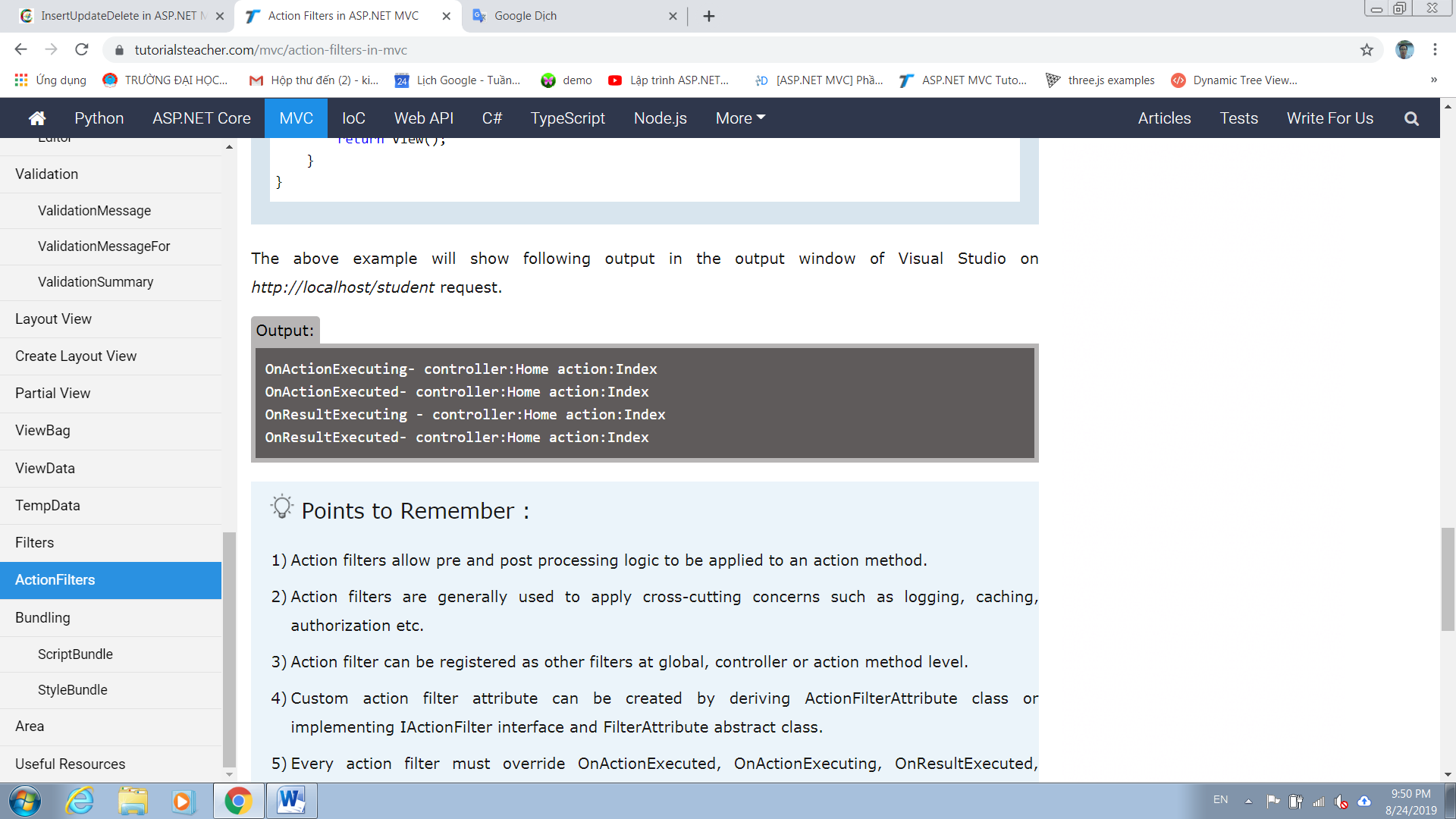
{

return View();

}

}

Ví dụ trên sẽ hiển thị kết quả trong cửa sổ Output của Visual Studio khi yêu cầu truy cập <http://localhost/student> như sau:

**

**Lưu ý:**

- Bộ lọc hành động cho phép logic xử lý trước và sau được áp dụng cho phương thức hành động.

- Các bộ lọc hành động thường được sử dụng để áp dụng các mối quan tâm xuyên suốt như ghi nhật ký, lưu trữ, ủy quyền, v.v.

- Bộ lọc hành động có thể được đăng ký như các bộ lọc khác ở cấp độ toàn cục, Controller hoặc phương thức hành động.

- Thuộc tính bộ lọc hành động tùy chỉnh có thể được tạo bằng cách lấy lớp ActionFilterAttribute hoặc thực hiện giao diện ActionFilter và lớp trừu tượng FilterAttribute.

- Mọi bộ lọc hành động phải ghi đè các phương thức OnActionExecuted, OnActionExecuting, OnResultExecuted, OnResultExecuting.

# CÂU HỎI ÔN TẬP CHƯƠNG 6

1. Sử dụng ứng dụng MVC quản lý bán hàng ở chương 5, tiếp tục thực hiện:

- Sử dụng bộ lọc để tối ưu hóa các phương thức hành động.

- Nghiên cứu về Session, xây dựng chức năng đăng nhập.

- Nghiên cứu và áp dụng cơ chế phân trang cho các trang web.

2. Sử dụng ứng dụng MVC quản lý nhân sự ở chương 5, tiếp tục thực hiện:

- Sử dụng bộ lọc để tối ưu hóa các Controller.

- Nghiên cứu về Session, xây dựng chức năng đăng nhập.

- Nghiên cứu và áp dụng cơ chế phân trang cho các trang web.

# TÀI LIỆU THAM KHẢO

1. Adam Freeman (2013), *Pro ASP.NET MVC 5 fifth edition*, ISBN-13 (electronic): 978-1-4302-6530-6.

2. https://www.tutorialsteacher.com/mvc.

3. https://viblo.asia/s/series-aspnet-mvc-bq5QL783lD8

4. https://howkteam.vn/

5. https://xuanthulab.net/

6. https://www.dammio.com/asp-net-mvc-co-ban