

# *Lập trình Web ASP.NET Core 2.0*



## MỤC LỤC

Chương 1: TỔNG QUAN NGÔN NGỮ C# .....	9
1.1 Giới thiệu C# .....	9
1.2 Giới thiệu .NET Core.....	9
1.3 Các khái niệm cơ bản .....	10
1.3.1 Tạo ứng dụng đầu tiên .....	10
1.3.2 Cấu trúc chương trình.....	10
1.3.3 Định danh (identifier) .....	11
1.3.4 Không gian tên (namespace).....	12
1.3.5 Toán tử `.`.....	13
1.3.6 Từ khóa using .....	13
1.3.7 Từ khóa static.....	13
1.3.8 Biến (variable) .....	15
1.3.9 Từ khóa var .....	17
1.3.10 Lệnh và khối lệnh.....	17
1.3.11 Biểu thức.....	18
1.3.12 Chú thích (comment).....	19
1.4 Cấu trúc điều kiện, lựa chọn .....	20
1.4.1 Câu lệnh điều kiện if.....	20
1.4.2 Câu lệnh lựa chọn switch .....	21
1.5 Cấu trúc Lặp.....	23
1.5.1 Câu lệnh for.....	23
1.5.2 Câu lệnh while .....	24
1.5.3 Câu lệnh do while.....	26
1.5.4 Câu lệnh For Each .....	27
1.6 Return, Break, Continue .....	27
1.7 Xử lý ngoại lệ (Exception).....	28
1.7.1 Ngoại lệ là gì? .....	28

1.7.2	Cấu trúc câu lệnh try... catch... finally.....	29
1.7.3	Phát sinh và bắt giữ ngoại lệ .....	29
1.7.4	Câu lệnh <i>throw</i> .....	30
1.7.5	Dẫn xuất một ngoại lệ .....	30
1.8	Các bước Debug chương trình.....	31
1.9	Hàm (Function) .....	32
1.9.1	Định nghĩa.....	32
1.9.2	Khai báo:.....	33
1.9.3	Lời gọi hàm.....	33
1.9.4	Truyền tham số cho hàm .....	33
1.10	Mảng .....	37
1.10.1	Giới thiệu mảng trong C# .....	37
1.10.2	Mảng 1 chiều .....	37
1.10.3	Mảng nhiều chiều .....	39
1.10.4	Mảng răng cưa.....	40
1.11	Tập hợp - Collections .....	41
1.11.1	List.....	41
1.11.2	ArrayList.....	42
1.11.3	Hashtable .....	43
1.12	Khái niệm lớp (Class) & Đối tượng (Object).....	44
1.12.1	Lớp .....	44
1.12.2	Đối tượng .....	44
1.12.3	Thuộc tính (Field).....	44
1.12.4	Phương thức (Method).....	45
1.12.5	Bảng tầm vực thuộc tính truy cập .....	45
1.12.6	Constructor.....	45
1.12.7	Object Initialize.....	46
1.12.8	Properties.....	46
1.12.9	Automatic Properties .....	46

1.12.10	Từ khóa static.....	47
1.12.11	Phương thức mở rộng.....	47
1.12.12	Kiểu Anonymous Type .....	48
1.13	Bài tập.....	48
Chương 2:	THIẾT KẾ GIAO DIỆN WEB.....	54
2.1	Ngôn ngữ HTML .....	54
2.1.1	Một số khái niệm.....	54
2.1.2	Giới thiệu HTML .....	55
2.1.3	Cấu trúc của 1 trang web.....	55
2.1.4	Các tag HTML căn bản.....	56
2.1.5	Các tag mới trong HTML5 .....	57
2.1.6	Cấu trúc 1 trang web.....	58
2.2	Bảng định kiểu – CASCADING STYLE SHEET (CSS) .....	59
2.2.1	Giới thiệu.....	59
2.2.2	Khởi động nhanh.....	59
2.2.3	Tạo style định dạng .....	60
2.2.4	Các thuộc tính CSS.....	61
2.2.5	Bộ chọn (Selector).....	63
2.2.6	Qui tắc nạp chồng .....	70
2.2.7	CSS3 .....	71
2.3	JAVASCRIPT VÀ JQUERY .....	74
2.3.1	Javascript .....	74
2.3.2	jQuery.....	75
2.3.3	Các thành phần giao diện jQueryUI.....	80
2.4	BootStrap.....	87
2.4.1	Giới thiệu.....	87
2.4.2	Hệ thống lưới – Grid System.....	87
2.4.3	Định dạng cơ bản.....	88
2.4.4	Form.....	91

Chương 3: TỔNG QUAN VỀ ASP.NET CORE 2.2 .....	94
3.1 Giới thiệu Ứng dụng Web .....	94
3.1.1 Giới thiệu.....	94
3.1.2 Nguyên lý hoạt động .....	94
3.1.3 Các khái niệm .....	95
3.1.4 Kiến trúc công nghệ ứng dụng web.....	96
3.2 Tổng quan về ASP.NET Core MVC .....	97
3.2.1 Giới thiệu về ASP.NET.....	97
3.2.2 ASP.NET Core là gì?.....	97
3.2.3 Tạo ứng dụng ASP.NET Core MVC .....	99
3.2.4 Application Startup .....	101
3.2.5 Dependency Injection .....	104
3.2.6 Middleware .....	104
3.2.7 Mô hình Model – View – Controller.....	105
3.2.8 Thêm mới Controller.....	108
3.2.9 Thêm mới View.....	110
Chương 4: CONTROLLER.....	112
4.1 Cấu trúc Controller.....	112
4.2 Action Method .....	112
4.3 Tiếp nhận tham số.....	113
4.4 ActionResult .....	113
4.5 Routing.....	116
4.5.1 Routing .....	116
4.5.2 Attribute Routing.....	119
4.6 Action Selector .....	121
4.7 Bài tập Ứng dụng.....	122
4.7.1 Máy tính cá nhân.....	122
4.7.2 Đọc ghi file .....	124
4.7.3 Upload file .....	128
Chương 5: TỔ CHỨC WEBSITE .....	133

5.1	Các thành phần layout .....	133
5.1.1	Giới thiệu.....	133
5.1.2	Đánh dấu vùng động.....	135
5.1.3	Tập tin _ViewStart.cshtml, _ViewImports.cshtml .....	136
5.2	Bundles – đóng gói tài nguyên css và javascript .....	137
5.3	Module hóa giao diện .....	139
5.3.1	Sử dụng Partial View .....	139
5.3.2	Truyền dữ liệu cho PartialView .....	141
5.4	Phân vùng ứng dụng (Areas) .....	141
5.4.1	Tạo phân vùng.....	141
5.4.2	Định tuyến.....	143
Chương 6:	CHIA SẺ DỮ LIỆU .....	145
6.1	Dẫn nhập.....	145
6.2	Truyền từ Controller qua View .....	146
6.2.1	Sử dụng ViewBag và ViewData .....	146
6.2.2	Sử dụng model.....	149
6.3	Session .....	151
6.3.1	Cài đặt & Cấu hình .....	152
6.3.2	Sử dụng Session .....	152
6.3.3	Ví dụ áp dụng .....	155
Chương 7:	Razor & Helper.....	158
7.1	Razor.....	158
7.1.1	Giới thiệu.....	158
7.1.2	Làm thế nào nó làm việc? .....	158
7.1.3	Làm việc với các đối tượng.....	159
7.1.4	Câu lệnh điều khiển.....	159
7.1.5	Bảng tham khảo lệnh Razor .....	161
7.2	Tag Helper .....	161
7.2.1	Anchor Tag Helper .....	162

7.2.2	Các Model Helper .....	163
7.2.3	Form Tag Helper .....	163
7.2.4	Tag Helper tùy biến .....	165
7.3	HTML Helper .....	168
7.3.1	HTML Links .....	168
7.3.2	Các phần tử HTML Form .....	169
7.3.3	DropDownList và ListBox .....	169
7.3.4	Custom HTML Helper .....	170
Chương 8:	Kiểm lỗi dữ liệu vào .....	172
8.1	Giới thiệu .....	172
8.2	Mô hình lập trình kiểm lỗi .....	172
8.3	Annotation kiểm lỗi .....	176
8.4	Kiểm lỗi tùy biến .....	179
8.4.1	Kiểm lỗi phía Server .....	179
8.4.2	Kiểm lỗi phía client .....	181
8.4.3	Regular Expression .....	186
Chương 9:	Database & EntityFramework .....	188
9.1	SQL và cơ sở dữ liệu quan hệ .....	188
9.1.1	Khái niệm SQL .....	188
9.1.2	Vai trò của SQL .....	188
9.1.3	Mô hình dữ liệu quan hệ .....	189
9.1.4	Bảng (Table) .....	189
9.1.5	Khóa chính của bảng (Primary Key) .....	189
9.1.6	Mối quan hệ (Relationship) và khóa ngoại (Foreign Key) .....	189
9.2	Sơ lược về câu lệnh SQL .....	190
9.2.1	Các câu lệnh .....	190
9.2.2	Quy tắc sử dụng tên trong SQL .....	190
9.2.3	Kiểu dữ liệu .....	191
9.2.4	Toán tử .....	191

9.3	View, Stored Procedure, Trigger, Function .....	192
9.3.1	Bảng ảo – View .....	192
9.3.2	Stored Procedure .....	192
9.3.3	Trigger .....	193
9.3.4	Function .....	193
9.4	Giới thiệu Entity Framework Core .....	194
9.5	Làm việc với CSDL theo mô hình Database First .....	195
9.6	Mô hình Code First của EF Core.....	197
9.6.1	Entity .....	197
9.6.2	Tạo lớp DbContext: Xây dựng lớp ngữ cảnh CSDL.....	199
9.6.3	Thực hiện Migration CSDL.....	199
9.7	Lập trình Entity Framework.....	200
9.7.1	Tóm tắt .....	200
9.7.2	Màn hình hiển thị Loại .....	201
9.7.3	Màn hình hiển thị chi tiết Loại (Detail).....	203
9.7.4	Màn hình chỉnh sửa Loại .....	204
9.7.5	Màn hình thêm mới (Create) .....	206
9.7.6	Màn hình xóa Loại .....	209
9.8	LINQ.....	210
9.8.1	Giới thiệu.....	210
9.8.2	Kỹ thuật truy vấn dữ liệu .....	210
9.8.3	Truy vấn đối tượng.....	212
9.8.4	Truy vấn phân trang.....	213
9.8.5	Truy vấn 1 thực thể.....	213
9.8.6	Tổng hợp số liệu .....	213
9.8.7	Phương thức kiểm tra .....	213
9.8.8	Ứng dụng LINQ.....	213
Chương 10:	Kỹ thuật AJAX .....	216
10.1	Giới thiệu Ajax.....	216



10.2	Cơ chế làm việc của ajax .....	216
10.2.1	Cơ chế truyền thông đồng bộ .....	217
10.2.2	Cơ chế truyền thông bất đồng bộ .....	217
10.3	jQuery Ajax .....	218
Chương 11:	Web API .....	224
11.1	Giới thiệu về ASP.NET Core Web API .....	224
11.2	Các loại API Action: .....	225
11.3	Xây dựng Web API với Entity Framework .....	226
11.3.1	Xây dựng API dùng data local .....	226
11.3.2	Xây dựng API dùng EF kết nối SQL Server .....	236
Chương 12:	Bảo mật .....	240
12.1	Identity .....	240
12.2	Authentication .....	240
12.2.1	Tạo dự án có hỗ trợ security .....	240
12.2.2	Cấu hình Cookie authentication .....	245
Chương 13:	UPLOAD FILE LÊN HOST .....	249
Chương 14:	Thực hiện dự án .....	258
TÀI LIỆU THAM KHẢO	.....	259

## Chương 1: TỔNG QUAN NGÔN NGỮ C#

**Sau khi học xong chương này, học viên có khả năng:**

- Mô tả được các kiểu dữ liệu cơ bản trong C#.
- Sử dụng được các cấu trúc lệnh.
- Xử lý ngoại lệ và biết Debug chương trình.

### 1.1 Giới thiệu C#

C# (đọc là Cee Sharp) là ngôn ngữ lập trình cho nền tảng *.Net platform*. Phiên bản đầu tiên năm 2002, C# trải qua các phiên bản 1.0, 2.0, 3.0, ... và hiện nay là 7.3. C# được phát triển bởi đội ngũ kỹ sư của Microsoft, trong đó người dẫn đầu là Anders Hejlsberg và Scott Wiltamuth.

Ngôn ngữ C# khá đơn giản, nhưng nó có ý nghĩa cao khi thực thi những khái niệm lập trình hiện đại. C# bao gồm tất cả những hỗ trợ cho cấu trúc, thành phần component, lập trình hướng đối tượng. Những tính chất đó hiện diện trong một ngôn ngữ lập trình hiện đại. Và C# hội đủ những điều kiện như vậy, hơn nữa nó được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.

Trong ngôn ngữ C#, mọi thứ liên quan đến khai báo lớp đều được tìm thấy trong phần khai báo của nó. Định nghĩa một lớp trong ngôn ngữ C# không đòi hỏi phải chia ra tập tin header và tập tin nguồn giống như trong ngôn ngữ C++. Hơn thế nữa, ngôn ngữ C# hỗ trợ kiểu XML, cho phép chèn các tag XML để phát sinh tự động các document cho lớp.

C# cũng hỗ trợ giao diện interface. Một lớp chỉ có thể kế thừa duy nhất từ một lớp cha (tức là không cho đa kế thừa như trong ngôn ngữ C++), tuy nhiên một lớp có thể thực thi nhiều giao diện. Khi một lớp thực thi một giao diện thì nó sẽ cung cấp chức năng thực thi giao diện. C# cũng hỗ trợ cấu trúc, nhưng khái niệm về ngữ nghĩa của nó thay đổi khác với C++. Trong C#, một cấu trúc được giới hạn, là kiểu dữ liệu nhỏ gọn, và khi tạo thể hiện thì nó yêu cầu ít hơn về hệ điều hành và bộ nhớ so với một lớp. Một cấu trúc không thể kế thừa từ một lớp (hoặc kế thừa một cấu trúc khác), nhưng một cấu trúc có thể thực thi một giao diện.

C# cung cấp những đặc tính hướng thành phần (component-oriented): những thuộc tính, những sự kiện. Lập trình hướng thành phần được hỗ trợ bởi CLR cho phép lưu trữ metadata với mã nguồn cho một lớp. Metadata mô tả cho một lớp, bao gồm những phương thức và những thuộc tính của nó, cũng như những bảo mật cần thiết và những thuộc tính khác. Mã nguồn chứa đựng những logic cần thiết để thực hiện những chức năng của nó... Do vậy, một lớp được biên dịch như là một khối self-contained, môi trường hosting biết được cách đọc metadata của một lớp và mã nguồn cần thiết mà không cần những thông tin khác để sử dụng nó.

### 1.2 Giới thiệu .NET Core

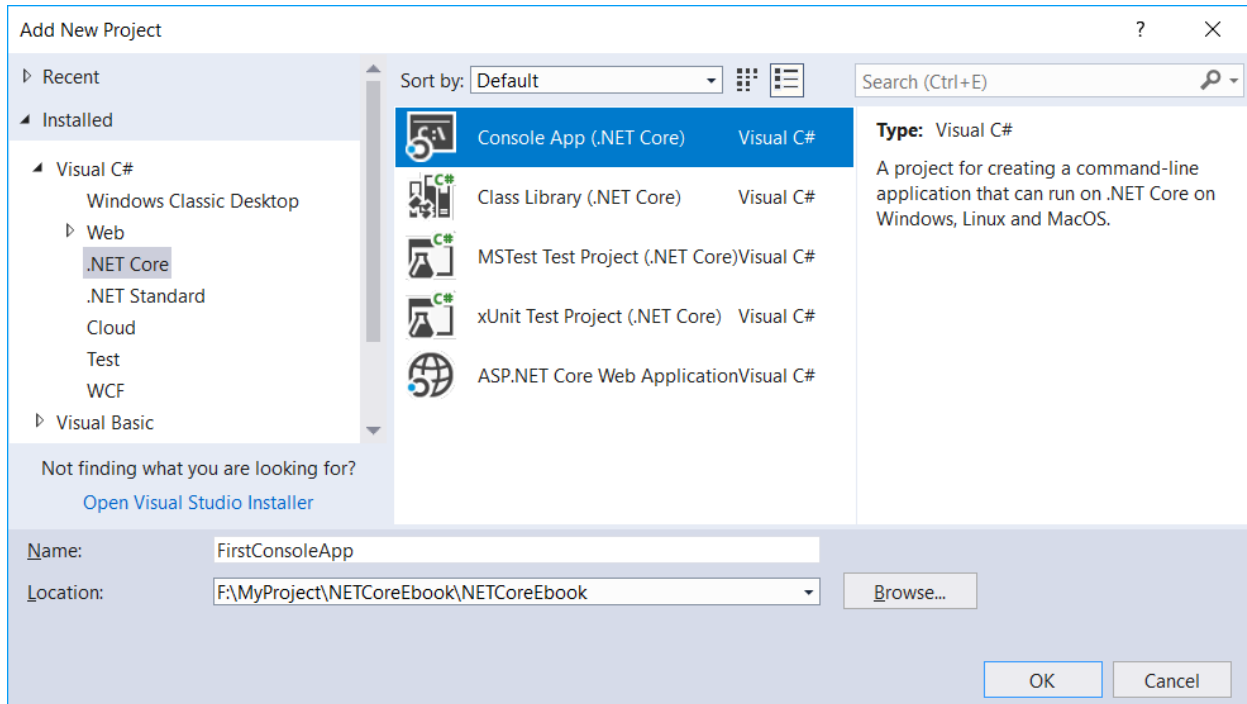
Microsoft .NET Core là một framework miễn phí, mã nguồn mở được phát triển dựa vào .NET Framework, đa nền tảng (cross-platform – có thể chạy trên Windows, Linux, MacOS), nhanh, nhẹ và hiện đại dùng để xây dựng ứng dụng di động, web, Windows desktop, Mac, gaming, machine learning & AI, IoT chạy trên được nhiều hệ điều hành Windows, Linux.

Trước khi bắt đầu viết code, bạn phải cài đặt .NET Core và các công cụ liên quan trên máy. Truy cập vào trang web: <https://www.microsoft.com/net/download/>. Có 3 cách để xây dựng ứng dụng .NET Core: sử dụng command line (CLI), Visual Studio Code và Visual Studio. Trong giáo trình này, chúng tôi sử dụng Visual Studio 2017 phiên bản 15.5.

## 1.3 Các khái niệm cơ bản

### 1.3.1 Tạo ứng dụng đầu tiên

Mở Visual Studio lên, tạo mới Project, chọn template là **.NET Core** bên trái, chọn ứng dụng dạng **Console App** và đặt tên project.



Hình 1-1: Tạo project đầu tiên

### 1.3.2 Cấu trúc chương trình

Bắt đầu từ chương trình "*FirstConsoleApp*" đơn giản trong C#:

```
using System;
```

```
namespace FirstConsoleApp  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

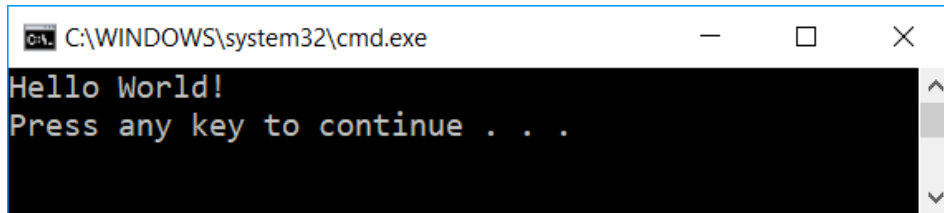
#### Giải thích:

- Phần đầu của chương trình là các khai báo thư viện với từ khóa **using**, theo sau là tên của thư viện cần khai báo.
- Toàn bộ chương trình được "đóng gói" trong một **namespace**. Bạn sẽ rõ hơn về **namespace** trong các phần sau.
- Bản thân chương trình trong C# là một lớp (*class*), như bạn thấy, có tên là **Program**. Lớp này chứa hàm **Main** – điểm bắt đầu của chương trình.

- Hàm Main ở trên chỉ chứa duy nhất một câu lệnh: `Console.WriteLine("Hello World!");`

Để viết ra màn hình dòng chữ: Hello World

Gõ tổ hợp phím `CTRL+F5` để chạy chương trình, bạn sẽ được kết quả:



**Hình 1-2: Kết quả chạy chương trình**

### Hàm Main:

Trong C#, hàm `Main()` được viết ký tự hoa đầu, và có thể trả về giá trị `void` hay `int`. Khi chương trình thực thi, CLR gọi hàm `Main()` đầu tiên, hàm `Main()` là đầu vào của chương trình, và mỗi chương trình phải có một hàm `Main()`. Đôi khi chương trình có nhiều hàm `Main()` nhưng lúc này ta phải xác định các chỉ dẫn biên dịch để CLR biết đâu là hàm `Main()` đầu vào duy nhất trong chương trình.

### 1.3.3 Định danh (identifier)

Định danh được sử dụng để đặt cho các đối tượng trong chương trình như: tên biến, tên kiểu dữ liệu, tên hàm, tên lớp, tên thuộc tính,...

Ngôn ngữ lập trình cũng giống như ngôn ngữ tự nhiên, chúng đều có cú pháp và ngữ nghĩa. Do đó việc đặt tên cho các đối tượng trong chương trình là rất quan trọng, làm sao phải đảm bảo được hai yếu tố: *đúng cú pháp* và *dễ đọc*.

### Quy tắc đặt tên:

- Tên (định danh – identifier) là một chuỗi ký tự bắt đầu bằng một chữ cái hoặc dấu gạch nối “\_”, được dùng để đặt cho các đối tượng trong chương trình (như lớp, thuộc tính, phương thức, biến, kiểu dữ liệu, ..)
- C# phân biệt chữ in hoa và in thường (case sensitive).

*Chú ý:*

- Tên không được bắt đầu bằng một chữ số.
- Tên không được trùng với từ khóa.
- Tên không được chứa khoảng trắng

*Ví dụ:* đặt tên như sau là **sai** cú pháp:

```
int class = 3;           //tên trùng với từ khóa (class)
double 1abc = 12.3;     //tên bắt đầu bằng chữ số 1
```

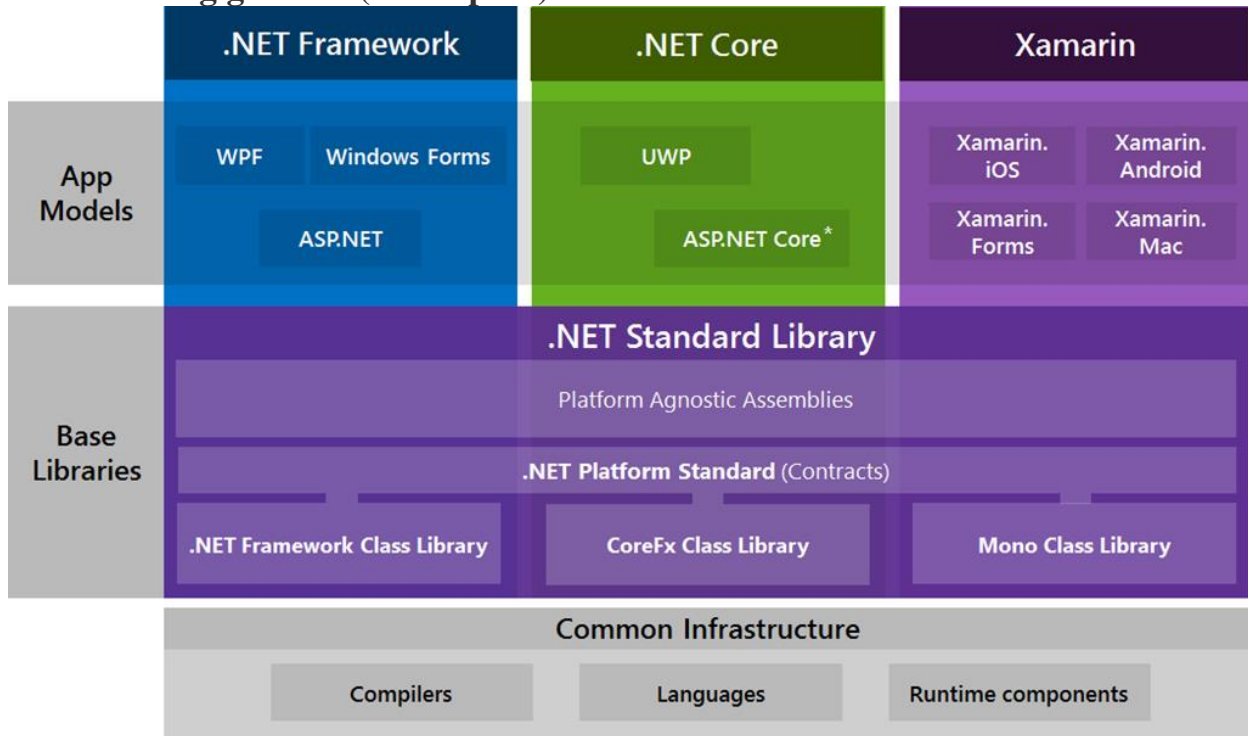
Trong việc đặt tên ngoài những quy tắc bắt buộc, các lập trình viên thường tìm cách đặt tên sao cho dễ đọc.

Ví dụ:

Tên của biến (**myDictionary**) thường được đặt theo cách đặt tên cú pháp lạc đà.

Tên của hàm (**DrawLine**) và thuộc tính (**ColorBackground**) đặt theo cú pháp Pascal.

### 1.3.4 Không gian tên (namespace)



Hình 1-3: Kiến trúc .NET Core

.NET Framework cung cấp một thư viện các lớp đồ sộ, có tên là FCL (Framework Class Library). Trong đó Console chỉ là một lớp nhỏ trong hàng ngàn lớp trong thư viện. Mỗi lớp có một tên riêng, vì vậy FCL có hàng ngàn tên như *ArrayList*, *Dictionary*, *FileSelector*,...

Điều này làm nảy sinh vấn đề, người lập trình không thể nào nhớ hết được tên của các lớp trong .NET Framework. Tệ hơn nữa là sau này có thể ta tạo lại một lớp trùng tên với lớp đã có chẳng hạn. Ví dụ trong quá trình phát triển một ứng dụng ta cần xây dựng một lớp từ điển và lấy tên là *Dictionary*, và điều này dẫn đến sự tranh chấp khi biên dịch vì C# chỉ cho phép một tên duy nhất. Khi đó chúng ta phải đổi tên của lớp từ điển mà ta vừa tạo thành một cái tên khác chẳng hạn như *myDictionary*. Do đó sẽ làm cho việc phát triển các ứng dụng trở nên phức tạp, cồng kềnh. Đến một sự phát triển nhất định nào đó thì chính là cơn ác mộng cho nhà phát triển.

Để giải quyết vấn đề này là ta tạo ra một *namespace*. *Namespace* sẽ hạn chế phạm vi của một tên, làm cho tên này chỉ có ý nghĩa trong vùng đã định nghĩa. Các *namespace* để phân thành các vùng cho các lớp trùng tên không tranh chấp với nhau.

Như vậy nếu .NET framework có xây dựng một lớp *Dictionary* bên trong *namespace System.Collections*, và tương ứng ta có thể tạo một lớp *Dictionary* khác nằm trong *namespace Lab2*, điều này hoàn toàn không dẫn đến sự tranh chấp với nhau.

#### Một ví dụ về namespace:

```
namespace Lab2
{
    namespace NS1
    {
        class class1
        {
```

```
        static public void method1()
        {
            //làm công việc gì đó
        }
    }
}
namespace NS2
{
    class class1
    {
        static public void method1()
        {
            //làm công việc gì đó
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Lab2.NS1.class1.method1();//gọi phương thức method1 trong namespace NS1
            Lab2.NS2.class1.method1();//gọi phương thức method1 trong namespace NS2
        }
    }
}
}
```

### 1.3.5 Toán tử ‘.’

Trong ví dụ trên dấu ‘.’ được sử dụng để truy cập đến phương thức hay dữ liệu trong một lớp (trong trường hợp này phương thức là `method1()`), và ngăn cách giữa tên lớp đến một *namespace* xác định (namespace `Lab2.NS1` và lớp `class1`). Việc thực hiện này theo hướng từ trên xuống, trong đó mức đầu tiên *namespace* là `Lab2`, tiếp theo là namespace `NS1`, tên lớp `class1` và cuối cùng là truy cập đến các phương thức hay thuộc tính của lớp.

### 1.3.6 Từ khóa using

Để làm cho chương trình gọn hơn, và không cần phải viết từng *namespace* cho từng đối tượng, C# cung cấp từ khóa là *using*, sau từ khóa này là một *namespace* hay subnamespace với mô tả đầy đủ trong cấu trúc phân cấp của nó.

Ví dụ: bằng việc khai báo

```
using Lab2.MyFolder;
```

Ta có thể viết gọn hơn:

```
StatementDemo test = new StatementDemo();
```

Thay vì:

```
MyFolder.StatementDemo test = new MyFolder.StatementDemo();
```

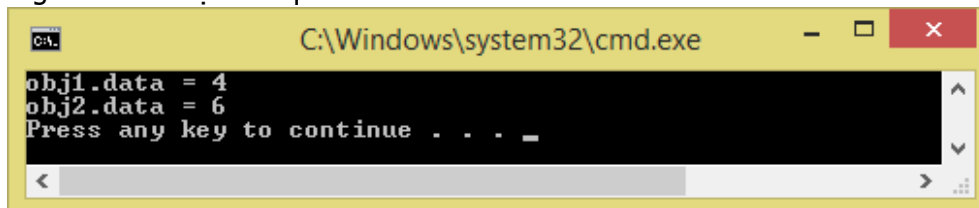
### 1.3.7 Từ khóa static

Theo mặc định, các thành phần trong một lớp là *non static*, có nghĩa là khi một đối tượng thuộc lớp này được tạo (khi ta gọi hàm dựng – *constructor*) thì một vùng nhớ riêng được cấp phát để lưu trữ các thành phần của đối tượng này. Ví dụ:

```
class MyClass
{
    public int data;
    public void Method()
    {
```

```
        //làm công việc gì đó
    }
}
class Program
{
    static void Main()
    {
        MyClass obj1 = new MyClass();
        MyClass obj2 = new MyClass();
        obj1.data = 4;
        obj2.data = 6;
        Console.WriteLine("obj1.data = " + obj1.data.ToString());
        Console.WriteLine("obj2.data = " + obj2.data.ToString());
        obj1.Method();
        obj2.Method();
    }
}
```

Chạy chương trình ta được kết quả:



Điều này khẳng định: mỗi đối tượng của `MyClass` lưu dữ liệu của mình một cách riêng biệt. Tương tự như vậy cho các thành phần khác (phương thức, thuộc tính,...).

Trong một số trường hợp, chúng ta cần các đối tượng sử dụng chung một thành phần nào đó. C# giải quyết vấn đề này bằng cách "chuyển" thành phần này cho lớp thay vì "để" ở đối tượng.

Ví dụ:

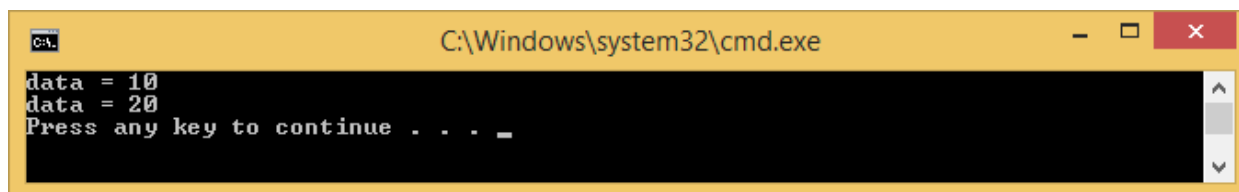
```
class OtherClass
{
    static public int data;
    public void DisplayData()
    {
        Console.WriteLine("data = " + data.ToString());
    }
    static public void Method()
    {
        //làm công việc gì đó
    }
}
class Program
{
    static void Main()
    {
        OtherClass.data = 10;
        OtherClass obj1 = new OtherClass();
        obj1.DisplayData();
        OtherClass obj2 = new OtherClass();
    }
}
```

```

        OtherClass.data = 20;
        obj2.DisplayData();
        OtherClass.Method();//phương thức của lớp!
    }
}

```

Ta có kết quả:



Kết quả này chứng tỏ dòng lệnh `OtherClass.data = 20;` đã tác động đến “thành phần” dữ liệu của đối tượng `obj2`. Hay chính xác hơn là chỉ có một vùng dữ liệu được tạo ra cho mọi đối tượng của lớp `OtherClass`.

### 1.3.8 Biến (variable)

#### Khái niệm:

Biến là khái niệm rất quan trọng trong các ngôn ngữ lập trình. Biến là một vùng nhớ (trong bộ nhớ sơ cấp – RAM) được đặt tên, dùng để lưu trữ dữ liệu. Mỗi biến có một kiểu dữ liệu xác định.

#### Cú pháp Khai báo:

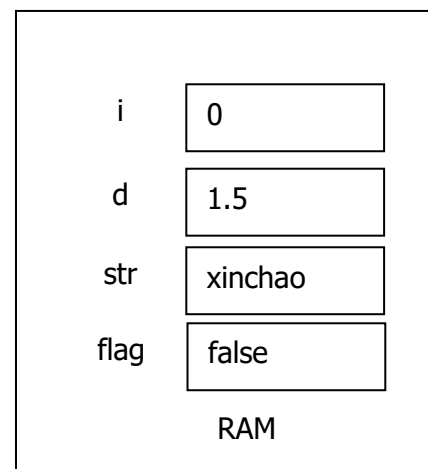
**<Tên kiểu> Tên\_biến;**

Ví dụ sau đây khai báo và khởi tạo giá trị ban đầu cho các biến:

```

static void Main(string[] args)
{
    int i = 0;
    double d = 1.5;
    string str = "xin chào";
    bool flag = false;
}

```



Hình 1-4: Khai báo biến

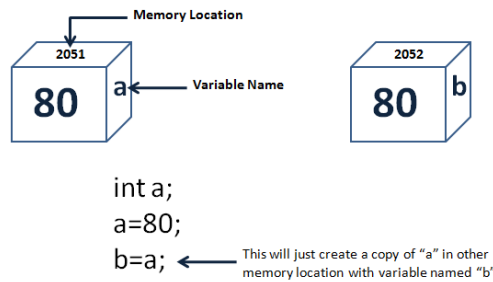
#### Biến được lưu trữ như thế nào?

Tùy thuộc vào kiểu dữ liệu mà mối quan hệ giữa tên biến và vùng dữ liệu chứa giá trị của biến sẽ khác nhau. Cụ thể:

- Các biến thuộc kiểu giá trị - value type (như int, double, enum, ..) thì tên biến là tên vùng nhớ trực tiếp chứa giá trị của biến.

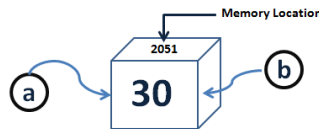


**VALUE TYPE - Example**

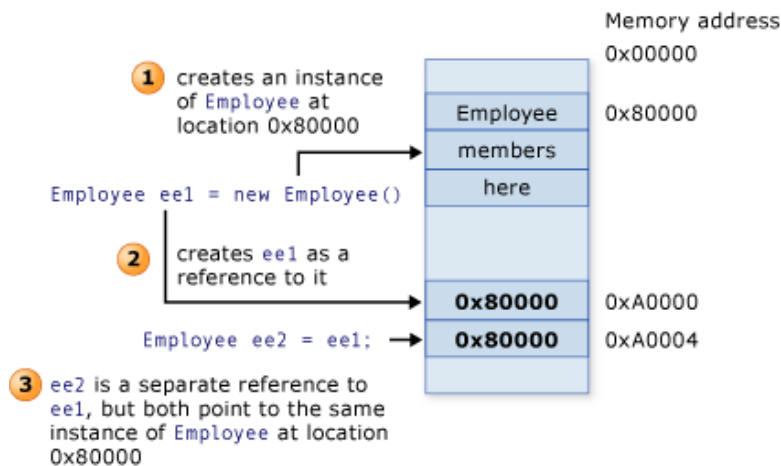


- Còn các biến thuộc kiểu tham chiếu - reference type (như class, string, ..) thì tên biến là tên vùng nhớ chứa giá trị tham chiếu đến giá trị thực của biến (giá trị thực của biến được lưu trong một vùng nhớ khác).

**Reference Type - Example**



```
Employee a=new Employee();
a.age = 26;
Employee b=a; ← This will just create a new variable which will
b.age = 30      point to same location as "a" points.
```



**Tầm vực của biến (variable scope):** cho biết biến có hiệu lực ở đâu và khi nào.

*Biến mức lớp:* là các biến được khai báo như một *fields non static* của một lớp. Biến này có tầm vực hoạt động trong toàn bộ lớp mà nó được khai báo. Ví dụ:

```
class VariableScope
{
    int data;//biến mức lớp.
}
```

**Biến mức phương thức (hàm):** là các biến được khai báo (cục bộ) trong một phương thức hoặc một hàm. Chúng chỉ có tầm vực hoạt động trong phương thức hoặc hàm này. Ví dụ:

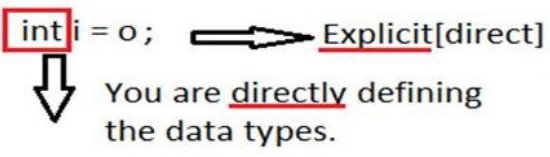
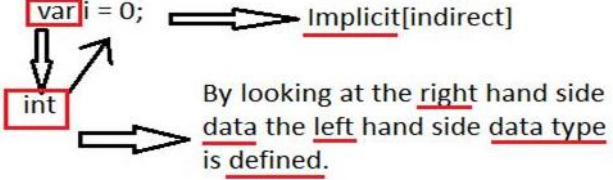
```
class VariableScope
{
    int data; //biến có tầm vực lớp.
    public int TinhTong(int n)
    {
        int i, t = 0; //biến mức phương thức, hàm.
        for (i = 1; i <= n; i++) t += i;
        return t;
    }
    //truy xuất biến t và i ở đây là không hợp lệ
}
```

### 1.3.9 Từ khóa var

**Mục đích:** để khai báo biến (trong phạm vi hàm, phương thức)

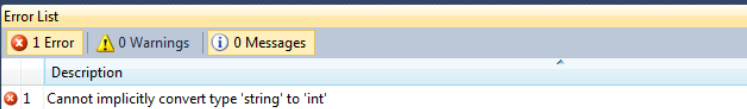
**Yêu cầu:** phải khởi tạo giá trị khi khai báo.

**Kiểu dữ liệu:** ngầm định kiểu dữ liệu của biến là kiểu của biểu thức bên phải phép gán.

Khai báo tường minh	Sử dụng từ khóa var
	

Ví dụ:

```
10 static void Main(string[] args)
11 {
12     var i = 0;
13     Console.WriteLine(i.GetType().ToString());
14     i = "abc";
15 }
```



### 1.3.10 Lệnh và khối lệnh

**Lệnh** là một khối (*block*) mã thực hiện một công việc xác định. Lệnh trong C# kết thúc bằng dấu ";"

C# phân chia các lệnh theo nhóm, như sau:

Loại	Từ khóa
Lệnh lựa chọn (rẽ nhánh)	if, else, switch, case
Lệnh lặp	do, for, foreach, in, while
Lệnh nhảy (jump)	break, continue, default, goto, return
Lệnh điều khiển ngoại lệ	throw, try, catch, finally
Đã / chưa kiểm tra	checked, unchecked
Lệnh fixed	fixed
Lệnh khóa	lock

**Lệnh gán** (assignment statement): sử dụng để gán giá trị cho một biến.

Ví dụ: `var i = 0;`

### Các lệnh xuất nhập console

C# cung cấp các phương thức *static* gắn với lớp *Console* để nhập và xuất dữ liệu bàn phím – màn hình.

- Nhập dữ liệu từ bàn phím:  
`Console.Read();` //đọc 1 kí tự  
`Console.ReadLine();` //đọc 1 dòng  
`Console.ReadKey();` //đọc 1 phím
- Xuất dữ liệu ra màn hình:  
`Console.Write();`  
`Console.WriteLine();` //viết xong xuống dòng mới

Ví dụ:

```
class CommandDemo
{
    double diem;
    string hoTen;
    public void NhapGiaTriChoBien()
    {
        Console.Write("nhap ho ten = ");
        hoTen = Console.ReadLine();
        Console.Write("nhap diem = ");
        diem = double.Parse(Console.ReadLine());
        Console.WriteLine("xin chao {0}, diem của bạn = {1}", hoTen, diem);
    }
}
```

**Khối lệnh:** là một nhóm các câu lệnh đặt trong cặp dấu { và }. Toàn bộ khối lệnh được xem như một lệnh (đơn).

Ví dụ:

```
if (a > b)
{
    temp = a;
    a = b;
    b = temp;
}
```

### 1.3.11 Biểu thức

Tương tự như trong toán học, biểu thức bao gồm các toán hạng và toán tử (phép toán). Ví dụ:

`i*d + Math.Sqrt(Math.Sin(Math.PI)) + str.Length`

Là một biểu thức gồm các toán hạng, ở đây là các biến (`i`, `d`, `str`, ..) và các toán tử (`*`, `+`, ..)

Mỗi biểu thức có một giá trị (khi biết giá trị của các toán hạng trong biểu thức đó).

Các loại phép toán:

- Phép toán số học: `+`, `-`, `*`, `/`
- Phép toán logic: `&&`, `||`, `!`
- Phép toán quan hệ: `>`, `<`, `>=`, `<=`, `!=`
- Phép toán tăng giảm: `++`, `--`

- Phép gán: =, +=, -=, \*=, /=

Trong một biểu thức, toán hạng có thể là hằng, biến, hàm, hay một biểu thức con.

### 1.3.12 Chú thích (comment)

Chú thích chỉ đơn thuần là các văn bản được sử dụng để giải thích, ghi chú trong chương trình. Chúng là công cụ để các nhà phát triển “giao tiếp” với nhau (có khi là với chính họ) chứ không phải với máy tính!

Có nhiều cách để chú thích trong C#. Ví dụ:

```
/*  
* chú thích trên nhiều dòng  
*  
*/  
//chú thích trên một dòng
```

```
///chú thích lặp lại  
///chú thích lặp lại  
///chú thích lặp lại
```

Đặc biệt là chú thích dạng XML, thường được sử dụng cho các phương thức, các hàm. Ví dụ:

```
///<summary>  
///Returns the Square of the specified number  
</summary>  
///<param name = "x">The number to square</param>  
///<returns>The squared value.</returns>  
static public double Square(double x)  
{  
    return x * x;  
}
```

Vì C# hỗ trợ cơ chế “gợi ý” khi gọi các phương thức được chú thích theo dạng này:

```
double result = NS1.class1.Square(  
double class1.Square(double x)  
Returns the Square of the specified number  
x: The number to square
```

Ví dụ: Viết *comment* cho hàm.

Bước 1: Viết hàm

```
static public dynamic CodeSnippetDemo(dynamic param1, dynamic param2)  
{  
    return param1 + param2;  
}
```

Bước 2: Đặt con trỏ tại dòng phía trên tiêu đề của hàm, gõ `///`. Net sẽ tự động sinh comment:

```
/// <summary>  
///  
/// </summary>  
/// <param name="param1"></param>  
/// <param name="param2"></param>
```

```
/// <returns></returns>  
static public dynamic CodeSnippetDemo(dynamic param1, dynamic param2)  
{  
    return param1 + param2;  
}
```

Bước 3: Viết nội dung cho comment

```
/// <summary>  
/// Demo CodeSnippet  
/// </summary>  
/// <param name="param1">Tham số thứ 1</param>  
/// <param name="param2">Tham số thứ 2</param>  
/// <returns>Trả về tổng 2 tham số</returns>  
static public dynamic CodeSnippetDemo(dynamic param1, dynamic param2)  
{  
    return param1 + param2;  
}
```

Bước 4: Gọi hàm

```
CodeSnippetDemo (  
dynamic Program.CodeSnippetDemo(dynamic param1, dynamic param2)  
Demo CodeSnippet  
param1: Tham số thứ 1
```

## 1.4 Cấu trúc điều kiện, lựa chọn

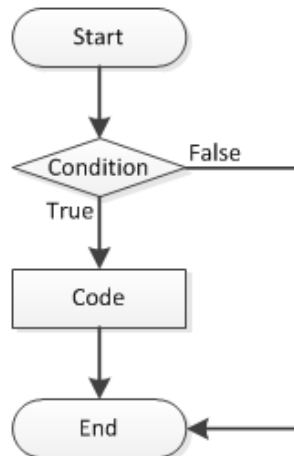
Trong quá trình xử lý, đôi khi chúng ta phải chọn 1 trong 2 hoặc nhiều công việc. Tùy thuộc vào một điều kiện nào đó, C# cung cấp cho chúng ta các câu lệnh *if* và *switch* để thực hiện điều này.

### 1.4.1 Câu lệnh điều kiện if

\* Cú pháp:

```
if(conditional expression)  
    <statement1>  
else  
    <statement2>
```

\* Sơ đồ khối:



\* Các ví dụ:

Ví dụ 1: Hàm sau đây trả về số lớn nhất trong 3 số mà nó nhận vào.

```
///////////////static public double SoLonNhat(double x, double y, double z)  
{  
    double max = x;  
    if (max > y) max = y;  
    if (max > z) max = z;  
    return max;  
}
```

\* *Chú ý:* Các câu lệnh if có thể lồng nhau nhiều cấp.

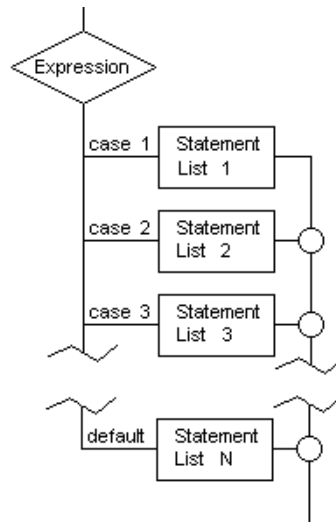
### 1.4.2 Câu lệnh lựa chọn switch

Câu lệnh *switch* cung cấp một cấu trúc điều khiển lựa chọn để thực hiện một công việc nào đó dựa trên giá trị của biểu thức tại thời điểm run-time.

\* Cú pháp:

```
switch(Expression)  
{  
    case lable_1: <statement_1>  
    case lable_2: <statement_2>  
    case lable_3: <statement_3>  
    ...  
    case lable_k: <statement_k>  
    [default: <statement_k+1>]  
}
```

\* Sơ đồ khối:



**\* Các ví dụ:**

Ví dụ 1: Hàm sau đây sẽ tính số tiền khách hàng phải trả khi thuê xe. Dữ liệu đầu vào là số ngày thuê *n* và loại xe. Mỗi loại xe có một giá thuê riêng, sử dụng cấu trúc *switch* để xác định giá thuê xe dựa trên loại xe. Nếu số ngày thuê lớn hơn 10 thì giảm giá thuê xe 10%.

```

///<summary>Tính tiền thuê xe</summary>
///<param name = "n">Số ngày thuê</param>
///<param name = "loai">Loại xe</param>
///<returns>Số tiền phải trả</returns>
static public double TienThueXe(int n, string loai)
{
    double gia = 0;
    switch (loai.ToUpper())
    {
        case "A": gia = 1000000; break;
        case "B": gia = 700000; break;
        case "C": gia = 500000; break;
    }
    if (n > 10) gia = gia * 0.9;
    return gia * n;
}

```

Ví dụ 2: Đoạn mã sau xác định số ngày của tháng bất kỳ trong năm 2000

```

int thang;
int soNgay;
Console.Write("thang = ");
thang = int.Parse(Console.ReadLine());
switch (thang)
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: soNgay = 31; break;
    case 4:
    case 6:
    case 9:

```

```

    case 11: soNgay = 30; break;
    case 2: soNgay = 29; break;
    default: soNgay = 0; break;
}
Console.WriteLine("so ngay = " + soNgay.ToString());
* Chú ý:

```

- Nhãn *default* trong cấu trúc *switch* là tùy chọn. Nếu có không có nhãn nào trước đó có giá trị bằng với giá trị của biểu thức thì *<statement\_k+1>* sau nhãn *default* sẽ được thực hiện.
- Có thể sử dụng *break* để thoát khỏi cấu trúc *switch* khi cần thiết (Xem ví dụ 2 ở trên).

## 1.5 Cấu trúc Lặp

### 1.5.1 Câu lệnh for

\* Cú pháp:

```
for(<init>;<condition>;<increment/decrement>)  
<statement(s)>
```

\* Sơ đồ khối:

\* Các ví dụ:

Ví dụ 1: Tính tổng  $S = 1 + 2 + \dots + n$

```

class Program
{
    static void Main(string[] args){
        int tong = 0;
        Console.Write("n = ");
        int n = int.Parse(Console.ReadLine());
        for (int i = 1; i <= n; i++) tong += i;
        //tong = tong + i;
        Console.WriteLine("tong = {0}", tong);
    }
}

```

Ví dụ 2: Tính tổng các phần tử dương trong mảng a.

```

class Program
{
    static void Main(string[] args){
        int tong = 0;
        //khai báo và khởi tạo mảng a:
        int[] a = { 1, -23, 6, 7, 3, -4, 8 };

        for (int i = 0; i < a.Length; i++)
            if(a[i] > 0) tong += a[i]; //tong = tong + a[i];
        Console.WriteLine("Tong cac phan tu duong = {0}", tong);
    }
}

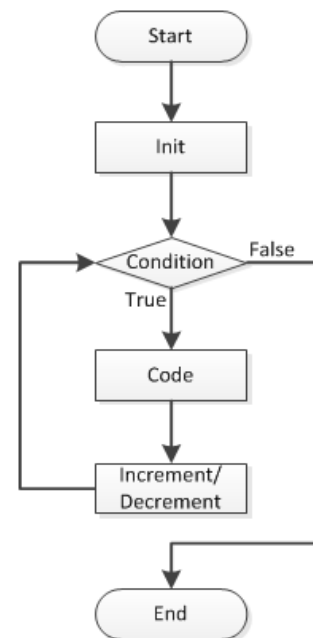
```

Tương tự Ví dụ 2, chúng ta có thể sử dụng *break* và *continue* thay cho lệnh *if*:

```

for (int i = 0; true; i++)
{
    if (i == a.Length) break; //thoát for gần nhất
    if (a[i] <= 0) continue;
}

```





```
        tong = tong + a[i];  
    }  
}
```

Ví dụ 3: Tính tổng  $S = 1! + 2! + \dots + k!$

Cách 1: Sử dụng *for* lồng nhau

```
class Program  
{  
    static void Main(string[] args)  
    {  
        int tong = 0;  
        Console.Write("n = ");  
        int n = int.Parse(Console.ReadLine());  
        for (int i = 1; i <= n; i++)  
        {  
            //tính i giai thừa  
            int giaiThua = 1;  
            for (int j = 1; j <= i; j++)  
                giaiThua *= j;  
            //cộng vào tổng:  
            tong += giaiThua;  
        }  
        Console.WriteLine("Tong = {0}", tong);  
    }  
}
```

Cách 2: Cải tiến cách 1

Nhận xét rằng tại bước lặp thứ  $i$  ở cách 1 chúng ta đã tính lại  $i$  giai thừa bằng cách nhân từ 1 đến  $i$ . Tuy nhiên, trước đó, tại bước  $i-1$  chúng ta đã tính  $i-1$  giai thừa rồi. Do vậy có thể tính  $i$  giai thừa ngay bằng lệnh:

```
giaiThua = giaiThua * i;
```

Chương trình được sửa lại chỉ dùng 1 vòng lặp *for* như sau:

```
int giaiThua = 1;  
int tong = 0;  
for (int i = 1; i <= n; i++)  
{  
    giaiThua *= i;  
    tong += giaiThua;  
}
```

\* Chú ý:

- Có thể sử dụng *break* và *continue* trong *for* (xem ví dụ 3)
- Các câu lệnh *for* có thể lồng nhau nhiều cấp.

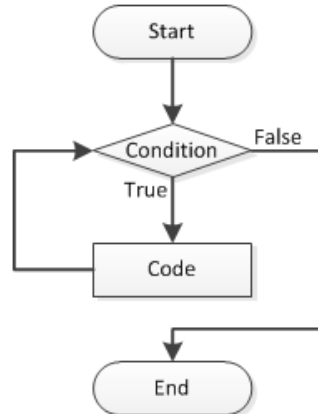
### 1.5.2 Câu lệnh *while*

Trong các ngôn ngữ lập trình *while* được xem là câu lệnh lặp điều kiện trước. Nghĩa là trong khi điều kiện còn thỏa mãn thì khối lệnh trong thân *while* còn tiếp tục được thực hiện.

\* Cú pháp:

```
while(conditional expression)  
{  
    <statement(s)>  
}
```

\* Sơ đồ khối:



\* Các ví dụ:

Ví dụ 1: Hàm sau đây sẽ đếm số chữ số của số nguyên n (với n là tham số đầu vào của hàm)

```
public static int DemSoChuSo(int n)  
{  
    int dem = 0;  
    while (n > 0)  
    {  
        ++dem;  
        n = n / 10;  
    }  
    return dem;  
}
```

Ví dụ 2: Hàm sau đây tính tổng các số nguyên tố trong một mảng (số nguyên tố là số chỉ chia hết cho 1 và chính nó).

```
public static int TongCacSoNguyenTo(int[] a)  
{  
    int tong = 0;  
    int so;  
    int i = -1;  
    while (true)  
    {  
        ++i;  
        if (i == a.Length) break;  
        //kiểm tra xem a[i] có nguyên tố hay không:  
        so = 2;  
        while (a[i] % so != 0) ++so;  
        //nếu a[i] không nguyên tố thì bỏ qua  
        if (so < a[i]) continue;  
        //ngược lại thì cộng vào tổng:  
        tong += a[i];  
    }  
}
```

```
    return tong;  
}
```

\* Chú ý: Chúng ta cũng có thể sử dụng *break* và *continue* trong lệnh *while* tương tự như ở trong lệnh *for*.

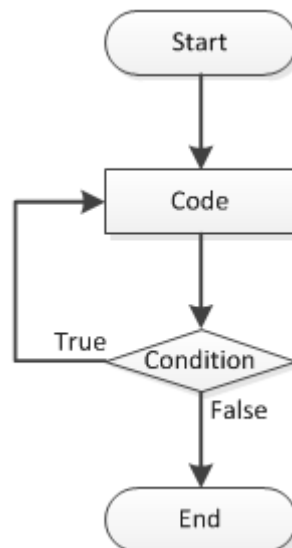
### 1.5.3 Câu lệnh do while

*Do while* được gọi là câu lệnh lặp điều kiện sau. Nghĩa là việc kiểm tra điều kiện chỉ được thực hiện sau khi đã thực hiện khối lệnh <statement(s)> sau *do*

\* Cú pháp:

```
do  
{  
    <statement(s)>  
}  
While(conditional expression);
```

\* Sơ đồ khối:



\* Các ví dụ:

Ví dụ 1: Chương trình sau yêu cầu người sử dụng nhập vào một giá trị số. Nếu không đúng là số thì yêu cầu nhập lại.

```
static void Main(string[] args)  
{  
    int so;  
    bool thanhCong;  
    do  
    {  
        Console.WriteLine("Hay nhap vao mot so = ");  
        thanhCong = int.TryParse(Console.ReadLine(), out so);  
        if (!thanhCong) Console.WriteLine("Gia tri khong hop le, vui long  
        nhap lai!");  
    }while (!thanhCong);  
}
```

```
Console.WriteLine("so ban da nhap = {0}", so);
```

```
}
```

\* *Chú ý:* Tương tự như các lệnh for và while, chúng ta cũng có thể sử dụng break và continue trong lệnh do while.

### 1.5.4 Câu lệnh For Each

For Each là một cấu trúc lặp trong C#. For Each khác với tất cả các vòng lặp khác: For Each không có điểm bắt đầu, không có điểm kết thúc, cũng như không có bước nhảy giữa các lần lặp. For Each đơn giản là duyệt từng phần tử có bên trong mảng.

Cấu trúc:

```
foreach (string name in arr)
{
    //to do here
}
```

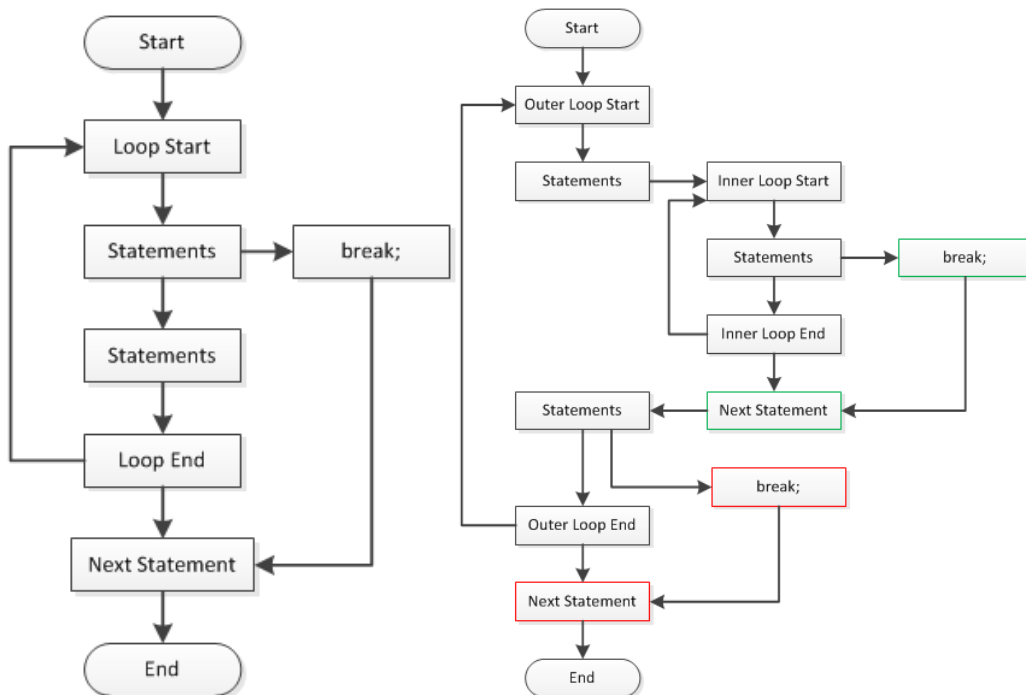
Ví dụ:

```
int[] fibarray = new int[] { 0, 1, 1, 2, 3, 5, 8, 13 };
foreach (int element in fibarray)
{
    Console.WriteLine(element);
}
Console.WriteLine();
```

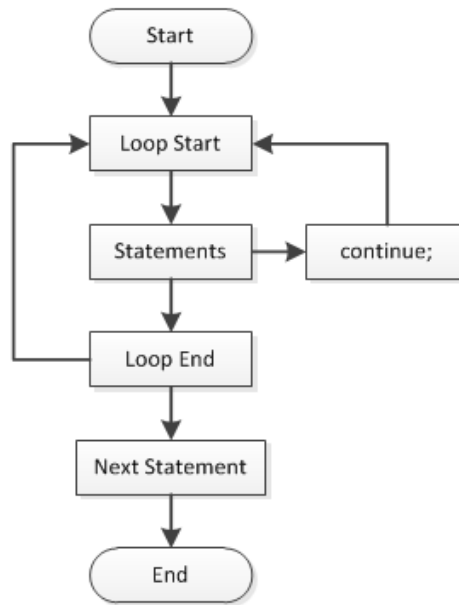
### 1.6 Return, Break, Continue

Trong phần này chúng ta trình bày rõ thêm về các lệnh *return*, *break* và *continue*

**Break:** cho phép thoát khỏi các lệnh *switch*, *for*, *while* và *do while* gần nhất.



**Continue:** bỏ qua việc thực hiện <statement(s)> và tiếp tục lặp với giá trị mới của biến lặp.



**Return:** lệnh này sử dụng để trả về giá trị cho hàm (trong trường hợp hàm có kiểu *void* thì không cần lệnh *return*) và kết thúc hàm đó.

Ví dụ: Hàm Main sau đây sẽ tính và in ra căn bậc 2 của một số thực.

```
static void Main(string[] args)
{
    double so;
    Console.Write("Hay nhap vao mot so = ");
    if (double.TryParse(Console.ReadLine(), out so) == false)
    {
        Console.WriteLine("Gia tri ban nhap khong phai la so");
        return;
    }
    if (so < 0)
    {
        Console.WriteLine("Gia tri ban nhap < 0");
        return;
    }
    Console.WriteLine("Can bac 2 cua {0} = {1}",so,Math.Sqrt(so));
    return;
}
```

## 1.7 Xử lý ngoại lệ (Exception)

### 1.7.1 Ngoại lệ là gì?

Ngôn ngữ C# cũng cho phép xử lý những lỗi và các điều kiện không bình thường với những ngoại lệ. Ngoại lệ là một đối tượng đóng gói những thông tin về sự cố của một chương trình không bình thường. Chúng ta phân chia các "sự cố" thành bug, lỗi, và ngoại lệ. Một bug là một lỗi lập trình có thể được sửa chữa trước khi mã nguồn được chuyển giao. Những ngoại lệ thì không được bảo vệ và tương phản với những bug. Mặc dù một bug có thể là nguyên nhân sinh ra ngoại lệ, chúng ta cũng không dựa vào những ngoại lệ để xử lý những bug trong chương trình, tốt hơn là chúng ta nên sửa chữa những bug này.

Lỗi gây ra có thể do lỗi của người sử dụng. Ví dụ, người sử dụng nhập vào một số nhưng họ lại nhập vào ký tự chữ cái. Một lần nữa, lỗi có thể làm xuất hiện ngoại lệ, nhưng chúng ta có thể ngăn ngừa điều này bằng cách bắt giữ lỗi với mã hợp lệ. Những lỗi có thể được đoán trước và được ngăn ngừa. Thậm chí nếu chúng ta xóa tất cả những bug và dự đoán tất cả các lỗi của người dùng, chúng ta cũng có thể gặp phải những vấn đề không mong đợi, như là xuất hiện trạng thái thiếu bộ nhớ (out of memory), thiếu tài nguyên hệ thống,... Những nguyên nhân này có thể do các chương trình khác cùng hoạt động ảnh hưởng đến, chúng ta không thể ngăn ngừa các ngoại lệ này, nhưng có thể xử lý chúng để chúng không thể làm tổn hại đến chương trình.

Khi một chương trình gặp một tình huống ngoại lệ, như là thiếu bộ nhớ thì nó sẽ tạo một ngoại lệ. Khi một ngoại lệ được tạo ra, việc thực thi của các chức năng hiện hành sẽ bị treo cho đến khi nào việc xử lý ngoại lệ tương ứng được tìm thấy. Điều này có nghĩa rằng nếu chức năng hoạt động hiện hành không thực hiện việc xử lý ngoại lệ, thì chức năng này sẽ bị chấm dứt và hàm gọi sẽ nhận sự thay đổi đến việc xử lý ngoại lệ. Nếu hàm gọi này không thực hiện việc xử lý ngoại lệ, ngoại lệ sẽ được xử lý sớm bởi CLR, điều này dẫn đến chương trình của chúng ta sẽ kết thúc.

Một trình xử lý ngoại lệ là một khối lệnh chương trình được thiết kế xử lý các ngoại lệ mà chương trình phát sinh. Xử lý ngoại lệ được thực thi trong trong câu lệnh catch. Một cách lý tưởng thì nếu một ngoại lệ được bắt và được xử lý, thì chương trình có thể sửa chữa được vấn đề và tiếp tục thực hiện hoạt động. Thậm chí nếu chương trình không tiếp tục, bằng việc bắt giữ ngoại lệ chúng ta có cơ hội để in ra những thông điệp có ý nghĩa và kết thúc chương trình một cách rõ ràng.

Nếu đoạn chương trình của chúng ta thực hiện mà không quan tâm đến bất cứ ngoại lệ nào mà chúng ta có thể gặp (như khi giải phóng tài nguyên mà chương trình được cấp phát), chúng ta có thể đặt đoạn mã này trong khối finally, khi đó nó sẽ chắc chắn sẽ được thực hiện thậm chí ngay cả khi có một ngoại lệ xuất hiện.

### 1.7.2 Cấu trúc câu lệnh try... catch... finally...

Để nắm rõ các bước xử lý ngoại lệ trong C#, bạn hãy xem xét cấu trúc của lệnh *try catch finally* sau đây (và ý nghĩa của từng phần):

```
try
{
    //các lệnh có nguy cơ tạo ra ngoại lệ
}
catch(Exception ex)
{
    //các lệnh xử lý khi xảy ra ngoại lệ
}
finally
{
    //các lệnh sẽ thực hiện dù có ngoại lệ hay không
}
```

### 1.7.3 Phát sinh và bắt giữ ngoại lệ

Trong ngôn ngữ C#, chúng ta chỉ có thể phát sinh (*throw*) những đối tượng các kiểu dữ liệu là *System.Exception*, hay những đối tượng được dẫn xuất từ kiểu dữ liệu này. *Namespace System* của CLR chứa một số các kiểu dữ liệu xử lý ngoại lệ mà chúng ta có thể sử dụng trong chương trình. Những kiểu dữ liệu ngoại lệ này bao gồm *ArgumentNullException*, *InvalidCastException*, và *OverflowException*, cũng như nhiều lớp khác nữa.

### 1.7.4 Câu lệnh *throw*

Để phát tín hiệu một sự không bình thường trong một lớp của ngôn ngữ C#, chúng ta phát sinh một ngoại lệ bằng cách sử dụng từ khóa *throw*. Dòng lệnh sau tạo ra một thể hiện mới của `System.Exception` và sau đó *throw* nó:

***throw new System.Exception();***

Khi phát sinh ngoại lệ thì ngay tức khắc sẽ làm ngừng việc thực thi trong khi *CLR* sẽ tìm kiếm một trình xử lý ngoại lệ. Nếu một trình xử lý ngoại lệ không được tìm thấy trong phương thức hiện thời, thì *CLR* tiếp tục tìm trong phương thức gọi cho đến khi nào tìm thấy. Nếu *CLR* trả về lớp *Main()* mà không tìm thấy bất cứ trình xử lý ngoại lệ nào, thì nó sẽ kết thúc chương trình.

Ví dụ:

```
static void Main(string[] args)
{
    int tuoi;
    try
    {
        Console.WriteLine("nhap tuoi cua ban = ");
        tuoi = int.Parse(Console.ReadLine());
        if (tuoi < 0) throw new Exception("khong hop le vi tuoi < 0");
        Console.WriteLine("tuoi = {0}", tuoi);
        //các lệnh xử lý khi tuoi>0 sẽ được viết ở đây:
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        //rỗng
    }
}
```

### 1.7.5 Dẫn xuất một ngoại lệ

Bước 1: Tạo ứng dụng mới

Bước 2: Viết mã cho lớp `MyException`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExceptionDemo.MyClasses
{
    class MyException:Exception
    {
        public MyException(string message) : base(message)
        {
        }
    }
}
```

Bước 3: Sử dụng lớp `MyException`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ExceptionDemo.MyClasses;

namespace ExceptionDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            double soTien;
            try
            {
                Console.Write("nhap so tien = ");
                bool thanhCong = double.TryParse(Console.ReadLine(), out soTien);
                if (!thanhCong || soTien < 0) throw new MyException("so tien khong hop le");
                Console.WriteLine("so tien da nhap = {0}", soTien);
            }
            catch (MyException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

**Bước 4:** Kiểm thử chương trình với 3 trường hợp (nhập số tiền = abc, số tiền = -123 và số tiền = 456)

## 1.8 Các bước Debug chương trình

\* Biên dịch chương trình:

- Cách 1: Gõ phím *F6*
- Cách 2: Gõ tổ hợp phím *CTRL+SHIFT+B*
- Cách 3: Vào thực đơn *Build* chọn lệnh *Build Solution*

\* Debug chương trình: Gõ phím *F5*

\* Debug chương trình từng bước: Gõ phím *F10*

\* Đặt/Xóa breakpoint: *F9*

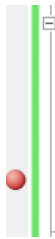
Ví dụ:

**Bước 1:** Tạo dự án mới với các hàm sau:

```
static void Main(string[] args)
{
    int i, tong = 0;
    for (i = 1; i < 11; i++) tong += i;
    Console.WriteLine("tong = {0}", tong);
}
```

**Bước 2:** Đặt *breakpoint*

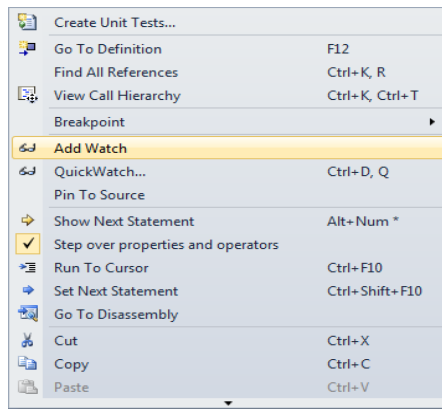




```
static void Main(string[] argv)
{
    int i, tong = 0;
    for (i = 1; i < 11; i++)
        tong += i;
    Console.WriteLine("tong = {0}", tong);
}
```

**Bước 3:** Gõ *F10* để debug

**Bước 4:** Click phải chuột tại dòng lệnh đã đặt *breakpoint* chọn *Add Watch*



**Bước 5:** Gõ tên của đối tượng (biến) cần kiểm tra vào cửa sổ *Watch*

Name	Value	Type
i	0	int
tong	0	int
i<11	true	bool

Chú ý: tại bước này bạn có thể thêm, và xóa các đối tượng cần kiểm tra trong cửa sổ *Watch*

**Bước 6:** Tiếp tục gõ *F10* nhiều lần để quan sát giá trị của các đối tượng cần kiểm tra.

Name	Value	Type
i	11	int
tong	55	int
i<11	false	bool

**Bước 7:** Gõ *SHIFT+F5* để dừng debug.

**Bước 8:** Gõ lại *F9* để xóa *breakpoint* đã đặt

## 1.9 Hàm (Function)

### 1.9.1 Định nghĩa

Hàm là đoạn chương trình thực hiện trọn vẹn một công việc nhất định. Hàm chia cắt việc lớn bằng nhiều việc nhỏ, giúp chương trình sáng sủa, dễ sửa lỗi, dễ quản lý.

## 1.9.2 Khai báo:

```
<Kiểu dữ liệu trả về> <Tên hàm> ([Danh sách tham số])  
{  
    <Danh sách các lệnh>;  
}
```

Trong đó:

- Kiểu dữ liệu trả về: **void, float, int, double, ...**, hay kiểu người dùng định nghĩa.
- Tên hàm: do người dùng đặt thông thường đặt sao cho gợi nhớ về công dụng của hàm (thường đặt theo Pascal Case).
- Tham số: có thể có hoặc không. Tham số nếu có là các biến có giá trị, biến kết quả mà hàm sử dụng.

*Ví dụ 1:* Hàm `output_hello` dưới đây không có tham số truyền vào, trả về kiểu `string`.

```
static string output_hello()  
{  
    return "Hello, welcome to ASP.NET class!";  
}
```

*Ví dụ 2:* Hàm `input` khởi tạo giá trị cho mảng số nguyên, không có giá trị trả về (trả về `null`).

```
static void input(int[] a)  
{  
    //khởi tạo ngẫu nhiên các phần tử số nguyên  
    Random rd = new Random();  
    for (int i = 0; i < a.Length; i++)    a[i] = rd.Next(100);  
}
```

## 1.9.3 Lời gọi hàm

- Mục đích: Yêu cầu thực thi 1 tác vụ (hàm) với dữ liệu cụ thể.
- Cú pháp gọi hàm:  
`Tên_hàm` (các dữ liệu cho từng tham số)
- Lưu ý:
  - Cần truyền đủ số lượng và đúng kiểu tham số đã khai báo khi gọi hàm.
  - Có thể gọi hàm lồng nhau.
  - Đối với hàm có kết quả trả về, lời gọi hàm thường gán kết quả cho biến hay đặt trong biểu thức xử lý.
  - Đối với hàm không có kết quả trả về, lời gọi hàm nằm riêng một dòng lệnh.
- Ví dụ:
  - Gọi hàm không truyền tham số: `string s = output_hello();`
  - Gọi hàm có truyền tham số: `input(a); //a là mảng một chiều`

## 1.9.4 Truyền tham số cho hàm

### 1.9.4.1 Dạng INT

Đây là kiểu truyền tham số mặc định cho hàm. Thân hàm chỉ tham khảo giá trị của tham số mà không thay đổi giá trị của tham số.

Ví dụ: Khai báo và định nghĩa hàm:

```
static int Tong(int n)
{
    int s = 0;
    for (int i = 1; i <= n; i++) s += i;
    return s;
}
```

Gọi hàm:

```
static void Main(string[] args)
{
    //ví dụ truyền tham số
    int n = 5;
    Console.WriteLine("Truoc khi gọi ham n = {0}", n);
    int s = Tong(n);
    Console.WriteLine("Sau khi gọi ham n = {0}", n);
}
```

Kết quả:



#### 1.9.4.2 DẠNG OUT

- Thân hàm cấp phát/khởi tạo giá trị của tham số (chỉ được gán giá trị cho tham số).
- Ra khỏi hàm giá trị tham số thay đổi.
- Khi gọi hàm, thêm chữ **out** vào trước tên tham số.

Ví dụ:

Khai báo và định nghĩa hàm:

```
static int Tong2(out int n)
{
    int s = 0;
    for (int i = 1; i <= 5; i++)    s += i;
    n = 100;
    return s;
}
```

Gọi hàm:

```
static void Main(string[] args)
{
    //ví dụ truyền tham số
    int n = 5;
    Console.WriteLine("Truoc khi gọi ham n = {0}", n);
    int s = Tong2(out n);
    Console.WriteLine("Sau khi gọi ham n = {0}", n);
}
```

Kết quả:



#### 1.9.4.3 DẠNG REF

- Ra khỏi hàm giá trị tham số thay đổi.
- Được phép thao tác đọc/sửa giá trị của tham số.
- Khi gọi hàm, thêm chữ **ref** vào trước tên tham số.

Ví dụ:

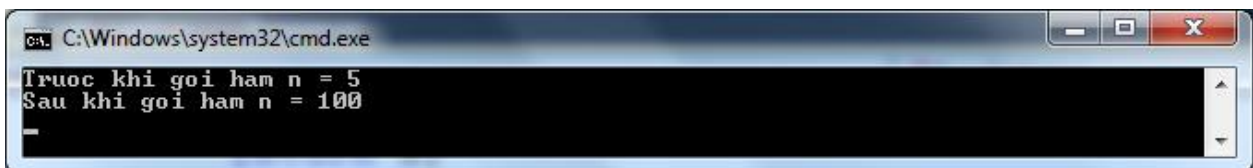
Khai báo và định nghĩa hàm:

```
static int Tong3(ref int n)
{
    int s = 0;
    for (int i = 1; i <= n; i++) s += i;
    n = 100;
    return s;
}
```

Gọi hàm:

```
static void Main(string[] args)
{
    //ví dụ truyền tham số
    int n = 5;
    Console.WriteLine("Truoc khi goi ham n = {0}", n);
    int s = Tong3(ref n);
    Console.WriteLine("Sau khi goi ham n = {0}", n);
}
```

Kết quả:



#### 1.9.4.4 Optional Parameter – Tham số mặc định

- Tham số mặc định dùng cho trường hợp không truyền giá trị của tham số.

Ví dụ: Hàm Optional() bên dưới có tham số truyền vào kiểu string, nếu không truyền giá trị tham số thì sẽ lấy giá trị mặc định (Test).

```
static void Main(string[] args)
{
    Optional();
    Optional("Another value");
}
static void Optional(string Value="Test")
{
    Console.WriteLine(Value);
}
```

Kết quả:

Another value

Test

- Tham số mặc định phải truyền từ phải sang trái, liên tục nhau.

Ví dụ:

```
static void Optional2(string Value1="Test", string Value2) {  
    Console.WriteLine(Value1 + Value2);  
}
```

```
static void Optional3(string Value1="Test", string Value2,  
string Value3="OK") {  
    Console.WriteLine(Value1 + Value2 + Value3);  
}
```

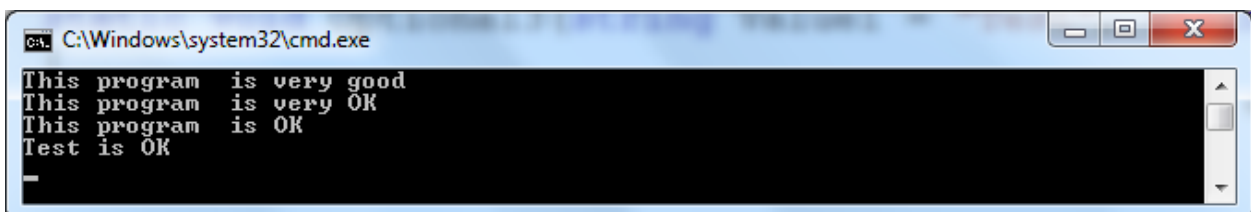
Hàm Optional2, Optional3 bắt buộc tham số Value2 phải là tham số mặc định. Ta sửa tham số Value2 của Optional3 như sau:

```
static void Optional3(string Value1 = "Test", string Value2 = "is", ,  
string Value3="OK") {  
    Console.WriteLine(Value1 + Value2 + Value3);  
}
```

Thực hiện chương trình với dữ liệu sau:

```
static void Main(string[] args)  
{  
    Optional3("This program ", "is very", "good");  
    Optional3("This program ", "is very");  
    Optional3("This program ");  
    Optional3();  
}
```

Kết quả nhận được:



```
cmd C:\Windows\system32\cmd.exe  
This program is very good  
This program is very OK  
This program is OK  
Test is OK  
-
```

#### 1.9.4.5 Named Parameter – Tham số được đặt tên

Sử dụng tham số được đặt tên làm chương trình dễ đọc, dễ trình bày. Bằng cách sử dụng tên tham số chính thức, chúng ta có thể đổi thứ tự các tham số thực tế.

Ví dụ sau đây sử dụng 4 cách gọi hàm khác nhau sử dụng tham số được đặt tên.

```
class Program  
{  
    static void Main()  
    {  
        // Call the Test method several times in different ways.  
    }  
}
```

```
Test(name: "Perl", size: 5); // Cach 1
Test(name: "Dot", size: -1); // Cach 2
Test(6, "Net"); // Cach 3
Test(7, name: "Google"); // Cach 4
}

static void Test(int size, string name)
{
    Console.WriteLine("Size = {0}, Name = {1}", size, name);
}
}
```

### Output

```
Size = 5, Name = Perl
Size = -1, Name = Dot
Size = 6, Name = Net
Size = 7, Name = Google
```

Cách gọi 1,2 hoán đổi vị trí các tham số.

## 1.10 Mảng

### 1.10.1 Giới thiệu mảng trong C#

- Mảng – thành phần quan trọng trong cấu trúc dữ liệu – là tập hợp các phần tử có cùng kiểu dữ liệu được truy xuất thông qua một tên duy nhất.
- Các loại mảng: mảng một chiều (One-Dimensional Array), mảng nhiều chiều (Multidimensional Array) và mảng răng cưa (Jagged Array). Các thuộc tính và phương thức của mảng là một thể hiện của lớp **System.Array**.
- Khi chúng ta tạo một mảng có kiểu dữ liệu giá trị, mỗi thành phần sẽ chứa giá trị mặc định của kiểu dữ liệu.

Ví dụ: Với khai báo `int myIntArray = new int[5]` ; thì:

- Mỗi thành phần của mảng được thiết lập giá trị là 0 (giá trị mặc định của số nguyên).
- Những kiểu tham chiếu trong một mảng không được khởi tạo giá trị mặc định, chúng được khởi tạo giá trị `null`.

### 1.10.2 Mảng 1 chiều

- Chỉ số mảng bắt đầu từ 0, tức là phần tử đầu tiên ở vị trí 0, phần tử cuối cùng ở vị trí `<số_phần_tử> - 1`.
- Mảng có thể được khai báo với kích thước cố định (bị giới hạn bởi số lượng phần tử) hoặc động (có thể thay đổi kích thước mảng tùy tình hình thực tế).
- Mảng là kiểu đối tượng (object), do đó sau khi khai báo mảng cần tạo thể hiện của mảng bằng từ khóa `new`.

```
kiểu dữ liệu>[] <tên mảng> ;
```

Ví dụ:

```
double[] doubleArray = new double[5];
char[] charArray = new char[5];
bool[] boolArray = new bool[2];
```

```
string[] stringArray = new string[10];
```

- Khởi tạo giá trị cho mảng:
  - Có thể khởi tạo ngay khi khai báo và tạo thể hiện

```
int[] staticIntArray = new int[3] {1, 3, 5};  
string[] strArray = new string[] { "Mahesh Chand", "Mike Gold", "Raj  
Beniwal", "Praveen Kumar", "Dinesh Beniwal" };
```

- Hoặc gán trực tiếp từng giá trị cho mảng:

```
int[] staticIntArray = new int[2];  
staticIntArray[0] = 1;  
staticIntArray[1] = 3;
```

- Truy xuất mảng: Sử dụng toán tử [].

```
staticIntArray[0] = 1;
```

### 1.10.2.1 Một số thuộc tính và phương thức thường dùng của lớp System.Array

Thành viên	Mô tả
Sort()	Phương thức sắp xếp giá trị tăng dần trong mảng một chiều
Reverse()	Phương thức sắp xếp giá trị giảm dần trong mảng một chiều
Length	Thuộc tính chiều dài của mảng
SetValue()	Phương thức thiết lập giá trị cho một thành phần xác định trong mảng
Rank	Thuộc tính trả về số chiều của mảng

### 1.10.2.2 Ví dụ:

Các thao tác với mảng một chiều các số nguyên tối đa 20 phần tử.

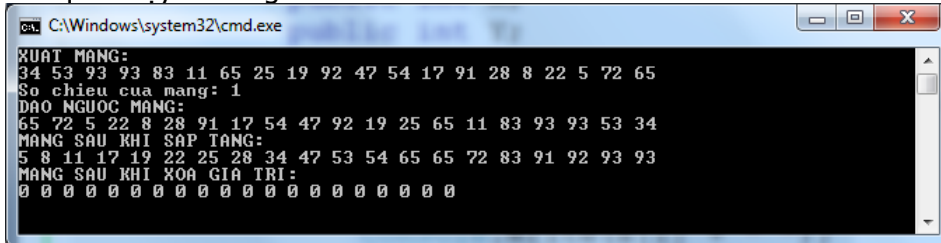
```
static void XuatMang(int [] a)  
{  
    for (int i = 0; i < a.Length; i++)  
        Console.Write(a[i] + " ");  
    Console.WriteLine();  
}  
static void Main(string[] args)  
{  
    //ví dụ mảng 1 chiều  
    //khai báo mảng 1 chiều tối đa 20 phần tử  
    int[] a = new int[20];  
  
    //khởi tạo giá trị ngẫu nhiên cho mảng  
    Random rd = new Random();  
    for (int i = 0; i < a.Length; i++)    a[i] = rd.Next(100);  
  
    Console.WriteLine("XUAT MANG:");  
    XuatMang(a);  
    Console.WriteLine("So chieu cua mang: {0}", a.Rank);  
  
    Console.WriteLine("DAO NGUOC MANG:");  
    Array.Reverse(a);  
    XuatMang(a);  
  
    Console.WriteLine("MANG SAU KHI SAP TANG:");
```

```

Array.Sort(a);
XuatMang(a);

//xóa mảng
Console.WriteLine("MANG SAU KHI XOA GIA TRI:");
Array.Clear(a, 0, a.Length);
XuatMang(a);
}
    
```

Kết quả chạy chương trình:



### 1.10.3 Mảng nhiều chiều

- Mảng nhiều chiều được biết đến như mảng hình chữ nhật với số chiều nhiều hơn 1, thường được gọi là ma trận (matrix). Số lượng phần tử là bằng nhau ở mỗi chiều.
- Khai báo mảng:

<kiểu dữ liệu>[ , ] <tên mảng>

Ví dụ:

```
int[ , ] myRectangularArray ;
```

- Khởi tạo thành phần của mảng

```

int[ , ] myRectangularArray = new int[sodong , socot];
int[ , ] numbers = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
string[ , ] names = new string[2, 2] { { "Ros", "Amy" }, { "Pet", "Albert" } };
    
```

hoặc:

```

int[ , ] numbers = new int[ , ] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
string[ , ] names = new string[ , ] { { "Rosy", "Amy" }, { "Peter", "Albert" } };
    
```

hoặc là:

```

int[ , ] numbers = { { 1, 2 }, { 3, 4 }, { 5, 6 } };
string[ , ] names = { { "Rosy", "Amy" }, { "Peter", "Albert" } };
    
```

hoặc là:

```

int[ , ] numbers = new int[3, 2];
numbers[0, 0] = 1;
numbers[1, 0] = 2;
numbers[2, 0] = 3;
numbers[0, 1] = 4;
numbers[1, 1] = 5;
numbers[2, 1] = 6;
    
```

- Duyệt mảng 2 chiều

```

for (int i = 0; i < sodong; i++)
{
    
```



```
for (int j = 0; j < socot; j++)  
{  
    Xử lý numbers[i,j];  
}
```

#### 1.10.4 Mảng răng cưa

- Là mảng nhiều chiều nhưng số lượng phần tử ở mỗi chiều khác nhau.
- Khai báo mảng:

```
<kiểu dữ liệu>[ ][ ] <tên mảng>;
```

Ví dụ: Khai báo mảng răng cưa 2 chiều có 3 dòng.

```
int[][] intJaggedArray = new int[3][];
```

- Khởi tạo thành phần của mảng.

Ví dụ: Mảng răng cưa 2 chiều có 3 dòng, ứng với mỗi dòng ta khởi tạo số lượng cột khác nhau.

```
intJaggedArray[0] = new int[2];  
intJaggedArray[1] = new int[4];  
intJaggedArray[2] = new int[6];
```

hoặc:

```
intJaggedArray[0] = new int[2]{2, 12};  
intJaggedArray[1] = new int[4]{4, 14, 24, 34};  
intJaggedArray[2] = new int[6] {6, 16, 26, 36, 46, 56 };
```

- Duyệt mảng răng cưa:

Ví dụ: duyệt mảng răng cưa 3 dòng đã khai báo và khởi tạo ở trên.

```
for (int i = 0; i < intJaggedArray.Length; i++)  
{  
    System.Console.WriteLine("Element ({0}): ", i);  
    for (int j = 0; j < intJaggedArray[i].Length; j++)  
    {  
        System.Console.WriteLine("{0}{1}", intJaggedArray[i][j], j ==  
            (intJaggedArray[i].Length - 1) ? "" : " ");  
    }  
    System.Console.WriteLine();  
}
```

Kết quả chạy chương trình:



```
C:\Windows\system32\cmd.exe  
Element <0>: 2 12  
Element <1>: 4 14 24 34  
Element <2>: 6 16 26 36 46 56
```

## 1.11 Tập hợp - Collections

- Là cấu trúc dữ liệu dùng để lưu trữ danh sách các phần tử có kiểu dữ liệu khác nhau. Số lượng phần tử không hạn chế.
- Các lớp này nằm trong namespace `System.Collections` hoặc `System.Collections.Generic` và thường có chung một giao diện.

### 1.11.1 List

- Phải chỉ định rõ kiểu dữ liệu của từng phần tử.
- Khai báo:

```
List<kiểu_dữ_liệu> <tên_biến> = new List<kiểu_dữ_liệu> ();
```

Ví dụ:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main() {
        // Use the List type.
        List<string> list = new List<string>();
        list.Add("cat");
        list.Add("dog");

        foreach (string element in list)
        {
            Console.WriteLine(element);
        }
    }
}
```

**Kết quả:**

```
cat
dog
```

- Truy xuất phần tử thông qua toán tử []. Ví dụ: `list[1]`;
- Một số phương thức và thuộc tính thường dùng:

Thuộc tính/Phương thức	Ý nghĩa
Capacity	Trả về tổng số phần tử tối đa
Count	Trả về số phần tử thật sự
Add(obj)	Thêm một phần tử kiểu obj
AddRange(arr_obj)	Thêm một mảng các phần tử
Clear()	Xóa tất cả phần tử
Contains(T)	Xác định danh sách có chứa phần tử T hay không?
Exist(lamda_expression)	Kiểm tra có phần tử nào thỏa mãn lamda expression hay không? (true hoặc false)
Find(lamda_expression)	Kiểm tra có phần tử nào thỏa mãn lamda expression hay không? (true hoặc false)
Insert(index, T)	Chèn phần tử T vào vị trí index
InsertRange(index, T_collection)	Chèn vào vị trí index danh sách các phần tử
RemoveAll(lamda_expression)	Xóa tất cả phần tử thỏa mãn lamda expression

RemoveAt(index)	Xóa phần tử tại vị trí index
Sort()	Sắp xếp các phần tử

### 1.11.2 ArrayList

- Không chỉ định rõ kiểu dữ liệu của từng phần tử. Do đó, các phần tử có thể có kiểu dữ liệu khác nhau.
- Khai báo:

```
ArrayList <tên_biến> = new ArrayList();
```

Ví dụ:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Use the ArrayList type.
        ArrayList alist = new ArrayList();
        alist.Add("cat");
        alist.Add(1999);
        alist.Add("dog");

        for (int i=0; i < alist.Count; i++)
        {
            Console.WriteLine(alist[i].ToString());
        }
    }
}
```

#### Kết quả:

```
cat
1999
dog
```

- Truy xuất phần tử thông qua toán tử [], chú ý ép kiểu nếu cần.

Ví dụ: `(int) alist[1];`

- Một số phương thức và thuộc tính thường dùng:

Thuộc tính/Phương thức	Ý nghĩa
Capacity	Trả về tổng số phần tử tối đa có thể chứa
Count	Trả về số phần tử thật sự
Add()	Thêm một phần tử kiểu obj vào cuối ArrayList
AddRange(arr_obj)	Thêm một mảng các phần tử vào cuối ArrayList
Clear()	Xóa tất cả phần tử
Contains(T)	Xác định phần tử T có nằm trong ArrayList hay không?
Exist(lamda_expression)	Kiểm tra có phần tử nào thỏa mãn lamda expression hay không? (true hoặc false)

Find(lamda_expression)	Kiểm tra có phần tử nào thỏa mãn lamda expression hay không? (true hoặc false)
Insert(index, T)	Chèn phần tử T vào vị trí index
InsertRange(index, T_collection)	Chèn vào vị trí index danh sách các phần tử
Remove()	Xóa phần tử đầu tiên trong ArrayList
RemoveAll(lamda_expression)	Xóa tất cả phần tử thỏa mãn lamda expression
RemoveAt(index)	Xóa phần tử tại vị trí index
Reverse()	Đảo ngược thứ tự các phần tử
Sort()	Sắp xếp các phần tử

### 1.11.3 Hashtable

- Hashtable là kiểu từ điển, mỗi phần tử bao gồm 1 cặp [key-value]. Hashtable không cần khai báo kiểu dữ liệu cho key, value.
- Hashtable dùng tối ưu cho việc truy xuất nhanh. Các cặp key là không trùng nhau.
- Khai báo:

```
Hashtable <tên_biến> = new Hashtable();
```

Ví dụ:

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main()
    {
        Hashtable ht = new Hashtable();
        ht.Add("pet", "cat");
        ht.Add(1, 1999);
        ht.Add("nick", "mylovepet");

        for (DictionaryEntry de in ht)
            Console.WriteLine("{0} --> {1}", de.Key, de.Value);
    }
}
```

#### Kết quả:

```
cat
dog
```

- Truy xuất phần tử thông qua toán tử [], chú ý ép kiểu nếu cần.  
 Ví dụ: `(int) ht[1];`
- Một số phương thức và thuộc tính thường dùng:

Thuộc tính/Phương thức	Ý nghĩa
Count	Trả về số phần tử có trong Hashtable
Keys	Tập hợp các khóa
Values	Tập hợp các giá trị ứng với khóa
Add(key, value)	Thêm một phần tử với key, value xác định
Clear()	Xóa tất cả phần tử bao gồm cả key, value
Contains(T)	Xác định phần tử T có nằm trong Hashtable hay không?
ContainsKey(k)	Xác định có phần tử nào trong Hashtable có khóa k
ContainsValues(v)	Xác định có phần tử nào trong Hashtable có giá trị là v

Remove(key)

Xóa phần tử có khóa key

## 1.12 Khái niệm lớp (Class) & Đối tượng (Object)

### 1.12.1 Lớp

Lớp là một khái niệm mô tả cho những thực thể có chung tính chất và hành vi, là một khuôn mẫu cho các đối tượng.

Lớp đối tượng là một cấu trúc dữ liệu linh hoạt có thể lưu trữ dữ liệu và thực thi hành động bao gồm hai thành phần sau :

- Thành phần thuộc tính (dữ liệu) bao gồm các thông tin liên quan đến lớp.
- Thành phần phương thức (hành động) bao gồm các hành động liên quan đến lớp đó.

Mỗi lớp đối tượng có thể có nhiều thuộc tính và nhiều phương thức.

Khai báo lớp bằng cách sử dụng từ khóa **class**. Cú pháp đầy đủ như sau:

```
[Thuộc tính] [Bổ sung truy cập] class <Tên lớp> [: Lớp cơ sở]
{
    // Các thuộc tính
    <Thuộc tính>
    // Các phương thức
    <Phương thức>
}
```

Ví dụ: Khai báo lớp Diem biểu diễn thông tin một điểm trong mặt phẳng Oxy.

```
class Diem
{
    // Cac thuc tinh
    private int x; // x viet thuong
    private int y; // y viet thuong

    // Cac phuong thuc
    public override string ToString() // Xuat
    {
        return "(" + x + ", " + y + ")";
    }
}
```

### 1.12.2 Đối tượng

Đối tượng là những đại diện cho lớp, mọi đối tượng đều có chung tính chất và hành vi mà lớp định nghĩa.

Ví dụ: Khai báo đối tượng dinhA – là thể hiện của lớp Diem đã định nghĩa ở trên.

```
Diem dinhA = new Diem();
```

Sau khi khai báo, đối tượng dinhA có đầy đủ các thuộc tính và phương thức của lớp Diem.

### 1.12.3 Thuộc tính (Field)

- Fields là các phần tử dùng để thể hiện các biến trong lớp.
- Fields là những thông tin có thể thay đổi được.

### 1.12.4 Phương thức (Method)

- Phương thức (method) chính là các hàm (function) được tạo trong lớp (class).
- Tên của phương thức thường được đặt theo tên của hành động.

### 1.12.5 Bảng tầm vực thuộc tính truy cập

Thuộc tính	Giới hạn truy cập
public	Không hạn chế. Những thành viên được đánh dấu public có thể được dùng bất kỳ các phương thức của lớp, bao gồm cả những lớp khác.
private	Thành viên trong lớp được đánh dấu private chỉ được dùng các phương thức của lớp này mà thôi.
protected	Thành viên trong lớp được đánh dấu protected chỉ được dùng các phương thức của lớp này; và các phương thức của lớp dẫn xuất từ lớp này.
internal	Thành viên trong lớp được đánh dấu là internal được dùng các phương thức của bất kỳ lớp nào cùng khối hợp ngữ với lớp này.
protected internal	Thành viên trong lớp được đánh dấu là protected internal được dùng các phương thức của lớp này; các phương thức của lớp dẫn xuất từ lớp này; và các phương thức của bất kỳ lớp nào trong cùng khối hợp ngữ với lớp này.

- **private** thì thuộc tính/phương thức đó chỉ được sử dụng trực tiếp bên trong lớp đó.
- **public** thì thuộc tính/phương thức đó có thể được sử dụng trực tiếp bên trong lớp lẫn bên ngoài lớp.

### 1.12.6 Constructor

- Constructor là phương thức đặc biệt của lớp, được gọi thực hiện khi lớp được tạo ra.
- Constructors có tên giống như tên của Class.
- Constructors không có giá trị trả về.
- Ví dụ: Một số hàm constructor cho lớp Diem.

```
// Các phương thức khởi tạo
public Diem()
{
    x = 0;
    y = 0;
}

public Diem(int xx, int yy)
{
    x = xx;
    y = yy;
}

public Diem(Diem p)
{
    X = p.X;
    Y = p.Y;
}
```

Sử dụng hàm constructor:

```
Diem dinhA = new Diem();  
Diem dinhB = new Diem(5, 5);  
Diem dinhC = new Diem(dinhB);
```

### 1.12.7 Object Initialize

Khởi tạo đối tượng kiểu object Initialize gồm có 3 cách sau:

- Cách thông thường: Khởi tạo các đối tượng sau đó gán các thuộc tính.

Ví dụ:

```
Diem dinhA = new Diem();  
dinhA.X = 10;  
dinhA.Y = 10;
```

- Gán các thuộc tính ngay khi khởi tạo đối tượng:

Ví dụ:

```
Diem dinhB = new Diem() { X = 10, Y = 20 };
```

- Khởi tạo đối tượng với kiểu anonymous:

Ví dụ:

```
var dinhC = new { X = 9, Y = 11};
```

### 1.12.8 Properties

Properties là phần tử dùng để cập nhật và truy xuất đến đặc điểm của một đối tượng – field ở mức private. Properties được định nghĩa bằng 2 phần, phần thứ nhất giống như định nghĩa Fields, phần thứ 2 có thêm 2 phần tử `get` và `set`.

Ví dụ: Property cho thuộc tính hoành độ x trong lớp Diem.

```
class Diem  
{  
    // Cac thuc tinh  
    private int x; // x viet thuong  
    private int y; // y viet thuong  
  
    // Cac phuong thuc Properties  
    public int X // X viet hoa  
    {  
        get { return x; } // x viet thuong  
        set { x = value; } // x viet thuong  
    }  
}
```

### 1.12.9 Automatic Properties

Để đơn giản hóa việc định nghĩa các `get/set` giống nhau ở các properties, automatic properties cho phép người dùng khai báo một cách chung chung `get; set;`. Thay vào đó trình biên dịch có thể tự động tạo ra các private field và những thao tác `get/set` mặc định cho chúng.

Ví dụ:

```
public class HangHoa {  
    public string MaHang { get; set; }  
    public string TenHang { get; set; }  
    public int SoLuong { get; set; }  
}
```

### 1.12.10 Từ khóa static

- Các thành viên (biến, phương thức) tĩnh (static) cho phép chúng ta truy cập trực tiếp mà không cần phải tạo thể hiện (đối tượng) của lớp.
- Mọi thao tác truy xuất thông qua tên class.
- Static member:
  - Dữ liệu thuộc mức lớp.
  - Độc lập với các đối tượng.
  - Chỉ có một thể hiện (instance) duy nhất.
  - Dữ liệu được cấp phát khi chương trình bắt đầu chạy.
- Static method:
  - Chỉ sử dụng được biến static.

Ví dụ:

```
class StaticClass  
{  
    //static member  
    static int count;  
  
    //static method  
    public static void Print()  
    {  
        Console.WriteLine("Count = " + count);  
    }  
}
```

Khai báo sử dụng hàm print():

```
StaticClass.Print();
```

### 1.12.11 Phương thức mở rộng

- Extension Methods (phương thức mở rộng) là phương thức được viết thêm vào một class **static** hiện có mà không cần một cấp thừa kế, biên dịch lại, hoặc sửa đổi mã nguồn gốc. Extension Methods được viết dưới dạng hàm tĩnh (static), tức là bạn sẽ gọi hàm này mà không cần phải khởi tạo một đối tượng.
- Khai báo phương thức mở rộng:

```
public static <kiểu trả về hàm> tên_hàm (this <kiểu_đối_tượng  
mở_rộng> tên_đối_tượng)  
{  
    //Nội dung hàm  
}
```

Ví dụ: Cài đặt phương thức đổi sang chữ hoa chuỗi cho trước là phương thức được thêm vào lớp string đã có.



```
static class Program
{
    public static string doisangchuhua(this string s)
    {
        return s.ToUpper();
    }
    static void Main(string[] args)
    {
        string s = "Hello eVery body!";
        Console.WriteLine(s.doisangchuhua());
    }
}
```

### 1.12.12 Kiểu Anonymous Type

Anonymous Type - kiểu dữ liệu trừu tượng - được dùng khi khai báo đối tượng chưa xác định được kiểu. Kiểu dữ liệu của biến sẽ được xác định khi gán giá trị cụ thể cho biến.

```
static void Main(string[] args)
{
    var a1 = new { Item100 = 1234, Item200 = "Hello World", Item300 = true };
    Console.WriteLine(a1.Item100 * 2); // 246
    Console.WriteLine(a1.Item200.ToUpper()); // HELLO WORLD
    Console.WriteLine(a1.Item300 ? "One" : "Two"); // One
}
```

## 1.13 Bài tập

### NHẬP XUẤT CƠ BẢN

- Viết chương trình nhập vào hai số thực dương chỉ chiều dài và chiều rộng của hình chữ nhật. Xuất ra màn hình chu vi và diện tích hình chữ nhật đó.
- Viết chương trình nhập vào độ dài cạnh của hình vuông. Xuất ra màn hình chu vi và diện tích hình vuông đó.
- Viết chương trình nhập vào bán kính của hình tròn. Xuất ra màn hình chu vi và diện tích hình tròn đó.
- Viết chương trình nhập vào họ tên (HoTen), điểm toán (Toan), điểm lý (Ly), điểm hóa (Hoa) của một sinh viên. In ra màn hình họ tên (dạng chữ HOA), điểm trung bình (DTB) lấy hai số thập phân của sinh viên theo công thức  $DTB = (Toan * 2 + Ly + Hoa) / 4$ .
- Viết chương trình nhập vào họ tên, năm sinh một người bất kỳ. Sau đó in ra màn hình các kết quả sau: họ tên, năm sinh, tuổi hiện tại và tuổi ở năm 2020 của họ. (Sử dụng: *DateTime.Now lấy ngày giờ hiện tại*).
- Nhập vào 1 số thực x bất kỳ, xuất ra kết quả của đa thức  $Y = 3x^2 + 4x - 7$ .

IF ... ELSE

7. Giải phương trình bậc 2 ( $ax^2 + bx + c = 0$ ).

- Nhập vào các hệ số a, b và c
- Biện luận và giải phương trình
  - Vô nghiệm ( $\Delta < 0$ )
  - Nghiệm kép ( $\Delta = 0$ )
  - 2 Nghiệm phân biệt ( $\Delta > 0$ )
- Hướng dẫn:
  - `double delta = b*b - 4*a*c;`
  - `X1 = (-b + Math.sqrt(delta))/(2*a);`
  - `X2 = (-b - Math.sqrt(delta))/(2*a);`

8. Viết chương trình nhập vào một số nguyên dương chỉ năm, cho biết năm đó có là năm nhuận hay không?

Hướng dẫn:

- Năm nhuận (là năm có 366 ngày, tháng 2 có 29 ngày) là năm chia hết cho 4, nếu năm chia hết cho 100 thì năm đó phải chia hết cho 400.

Thuật giải:

Nếu `nam % 400 == 0` thì

`//Năm nhuận`

Ngược lại, nếu `(nam % 4 == 0)` và `(nam % 100 != 0)` thì

`//Năm nhuận`

Ngược lại

`//Năm thường`

9. Nhập vào độ dài 3 cạnh của một tam giác. Xuất ra thông báo tam giác vuông (bình phương một cạnh bằng tổng bình phương 2 cạnh còn lại), tam giác cân (hai cạnh bằng nhau), tam giác đều (ba cạnh bằng nhau), tam giác thường hoặc bộ ba số không hợp lệ.

10. Nhập vào tiền thực lãnh của tháng (năm) và số người phụ thuộc, tính thuế thu nhập cá nhân phải nộp theo luật thuế áp dụng từ tháng 7 năm 2013 như sau:

Bậc thuế	Phần thu nhập tính thuế/năm (tr. đồng)	Phần thu nhập tính thuế/tháng (tr. đồng)	Thuế suất (%)
1	Đến 60	Đến 5	5
2	Trên 60 đến 120	Trên 5 đến 10	10
3	Trên 120 đến 216	Trên 10 đến 18	15
4	Trên 216 đến 384	Trên 18 đến 32	20
5	Trên 384 đến 624	Trên 32 đến 52	25
6	Trên 624 đến 960	Trên 52 đến 80	30
7	Trên 960	Trên 80	35

Giảm trừ gia cảnh mỗi người 09 triệu/tháng và mỗi người phụ thuộc 3.6 triệu/tháng.

### SWITCH ... CASE

11. Tính số tiền khách hàng phải trả khi thuê xe. Dữ liệu đầu vào là số ngày thuê xe và loại xe. Mỗi loại xe có một giá thuê riêng: loại A: 1,000,000 đ/ngày; loại B: 700,000 đ/ngày, loại C: 500,000 đ/ngày. Nếu số ngày thuê lớn hơn 10 thì giảm giá thuê xe 10%.

### LẶP (FOR, WHILE,...)

12. Viết chương trình nhập số N sau đó tính các tổng sau:

- a.  $S1 = 1 + 2 + 3 + \dots + N$
- b.  $S2 = 1 + 1/2 + 1/3 + \dots + 1/N$
- c.  $S3 = 11 + 22 + 33 + \dots + NN$
- d.  $S4 = 1 * 2 * 3 * \dots * N$
- e.  $S5 = 1 + 1/2! + 1/3! + \dots + 1/N!$  (trong đó  $N! = 1 * 2 * 3 * \dots * N$ )
- f.  $S6 = 1/(1*2) + 1/(2*3) + 1/(3*4) + \dots + 1/(N*(N+1))$

13. Nhập vào số nguyên dương  $N \leq 150$ . In ra giá trị bình phương các số từ 1 đến  $N$ .

14. Nhập vào một số nguyên, xuất ra số chữ số và tổng các chữ số của nó.

VD: Nhập vào 123456, xuất ra 6 và 21 ( $1 + 2 + 3 + 4 + 5 + 6 = 21$ )

## HÀM

15. Viết chương trình thực hiện chức năng nhập vào 1 số nguyên từ bàn phím ( $n > 20$ ), sau đó tính tổng  $x$  ( $x$  được nhập từ bàn phím) các số chẵn đầu tiên từ 1  $\rightarrow$   $n$ . Nếu người nhập  $n < 20$  thì thông báo nhập lại.

- Viết hàm nhập và kiểm tra số nguyên  $n$ .
- Viết hàm tính tổng các số chẵn.

## HƯỚNG DẪN - GỢI Ý

Bước 1: Thiết kế hàm nhập và kiểm tra  $n > 20$ . *Chú ý kiểu truyền tham số cho hàm.*

```
static void nhap(out int n) {
    int x;
    while(true){
        Console.WriteLine("Nhap vao so nguyen");
        x = int.Parse(Console.ReadLine());
        if (x > 20) break;
        Console.WriteLine("Vui long nhap n > 20.");
    }
    n = x;
}
```

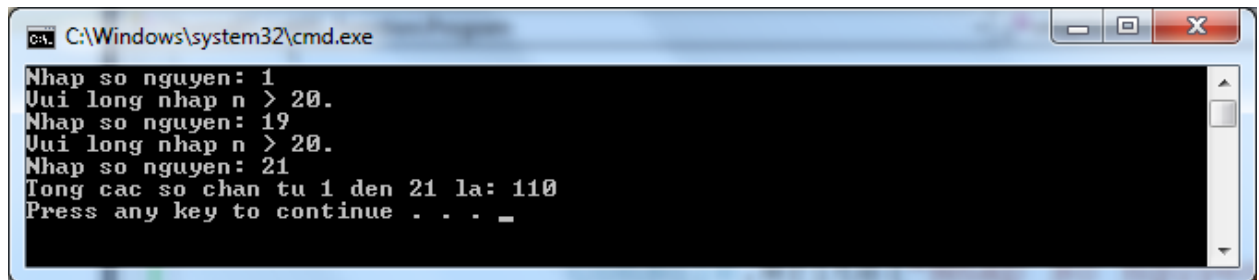
Bước 2: Thiết kế hàm tính tổng các số chẵn từ 1 đến  $n$

```
static int tongsochan(int n)
{
    int s = 0;
    for (int i = 1; i <= n; i++)
        if (i % 2 == 0) s += i;
    return s;
}
```

Bước 3: Thực hiện gọi hàm

```
static void Main(string[] args){
    int n; nhap(out n);
    Console.WriteLine("Tong cac so chan tu 1 den {0} la: {1}", n,
        tongsochan(n));
}
```

Kết quả màn hình chạy với dữ liệu nhập vào lần lượt là 1, 19, 21.



```
C:\Windows\system32\cmd.exe
Nhap so nguyen: 1
Uui long nhap n > 20.
Nhap so nguyen: 19
Uui long nhap n > 20.
Nhap so nguyen: 21
Tong cac so chan tu 1 den 21 la: 110
Press any key to continue . . . _
```

Bước 4: Tinh chỉnh hàm tính tổng

```
static int tongsochan2(int n)
{
    int s = 0;
    for (int i = 2; i <= n; i+=2)
        s += i;
    return s;
}
```

16. Viết chương trình đếm xem có bao nhiêu nguyên tố từ  $x \rightarrow y$  với  $x, y$  là 2 số nguyên được nhập từ bàn phím.
- Viết hàm nhập số nguyên.
  - Viết hàm kiểm tra số nguyên tố
  - Viết hàm đếm số nguyên tố

## MẢNG

17. Khai báo mảng một chiều các số nguyên tối đa 100 phần tử. Viết chương trình:
- Viết hàm nhập vào giá trị cho các phần tử trong mảng.
  - Viết hàm xuất mảng 1 chiều các số nguyên.
  - Viết hàm tính tổng các phần tử trong mảng.
  - Viết hàm tìm số lớn nhất, số nhỏ nhất trong mảng 1 chiều.
  - Viết hàm đếm số lượng số nguyên dương chẵn có trong mảng.
  - Viết hàm xuất giá trị tổng, trung bình cộng các giá trị của các phần tử trong mảng.
  - Viết hàm main thực hiện các yêu cầu trên.
18. Khai báo 1 mảng nguyên 2 chiều 4 dòng, 5 cột. Viết chương trình:
- Nhập giá trị cho các phần tử trong mảng (giá trị = số thứ tự dòng + số thứ tự cột).
  - In giá trị các phần tử trong mảng.
  - In giá trị lớn nhất, giá trị nhỏ nhất của các phần tử trong mảng.
  - In tổng số các giá trị, trung bình cộng các giá trị của các phần tử trong mảng.
  - Viết hàm thực hiện sắp xếp các phần tử trong mảng tăng dần (từ trái sang phải, từ trên xuống dưới).
  - Viết hàm tìm phần tử  $x$  có trong mảng hay không? nếu tìm thấy xuất thông báo.
  - Viết hàm thực hiện tính tổng các phần tử trên đường chéo chính của mảng  $a$  (những phần tử có vị trí dòng = vị trí cột).
  - Viết hàm thực hiện sắp xếp các phần tử trên dòng chẵn tăng dần và dòng lẻ giảm dần.
19. Viết 1 ứng dụng Console thực hiện các chức năng sau với List.
- Nhập các phần tử kiểu chuỗi từ bàn phím, thêm vào List khi nào người dùng nhập vào chuỗi "stop" thì ngừng. Sau đó xuất các phần tử ra màn hình.
  - Tìm phần tử, trả về **true** nếu tìm thấy, ngược lại trả về **false**
  - Xóa một phần tử.
  - Thêm 1 phần tử vào vị trí index bất kỳ với index nhập từ bàn phím.

## HƯỚNG ĐỐI TƯỢNG

### 20. Thao tác lớp cơ bản

Tạo ứng dụng Console, thêm vào 1 class tên HocVien bao gồm các thành phần dữ liệu: MaHV, HoTen, NgaySinh, DiaChi, DienThoai.

- Khai báo và định nghĩa các Constructor tham số và không tham số.
- Khai báo và định nghĩa các properties để truy cập đến giá trị của các thành phần dữ và viết thêm 1 properties chỉ đọc (get) dùng để lấy LayTuoi của HocVien.
- Viết hàm Main() để kiểm tra lớp trên.

### 21. Quản lý Hình học

Tạo ứng dụng Console, thực hiện các yêu cầu sau:

- Thêm vào 1 class HìnhHoc gồm các thành phần biểu diễn diện tích (mDienTich) và chu vi (mChuVi). Viết phương thức để xuất giá trị của mDienTich và mChuVi ra màn hình.
- Thêm vào 1 class HìnhChuNhat kế thừa từ lớp HìnhHoc biểu diễn thông tin hình chữ nhật bao gồm thuộc tính riêng của nó là mChieuDai, mChieuRong.
  - Khai báo thành phần dữ liệu cần thiết để biểu diễn hình chữ nhật.
  - Khai báo và định nghĩa các constructor cần thiết.
  - Khai báo và định nghĩa các properties để truy cập đến giá trị của các thành phần dữ liệu mChieuDai và mChieuRong (get, set).
  - Viết 2 phương thức tính diện tích hình chữ nhật (mChieuDai x mChieuRong) và chu vi (mChieuDai+mChieuRong) x 2, kết quả gán vào thuộc tính mDienTich, mChuVi.
- Thêm 1 lớp tên HìnhTron kế thừa từ lớp HìnhHoc và viết thêm các thành phần sau :
  - Khai báo thêm các thành phần dữ liệu: mBanKinh.
  - Khai báo và định nghĩa các Constructor tham số và không tham số để khởi tạo các giá trị cho các thành phần dữ liệu.
  - Khai báo và định nghĩa các properties để truy cập đến giá trị của các thành phần dữ liệu (get, set).
  - Viết các phương thức tính chu vi và diện tích hình tròn (cách viết giống như lớp hình chữ nhật).
- Viết hàm Main() để kiểm tra các constructor, các properties, các phương thức của các lớp trên.

### 22. Quản lý Nhân viên

Tạo 1 ứng dụng Console, thực hiện các yêu cầu sau:

- Thêm vào 1 class tên Nguoi bao gồm các thành phần dữ liệu: HoTen, NgaySinh, DiaChi.
  - Khai báo và định nghĩa các constructor tham số và không tham số.
  - Khai báo và định nghĩa các properties để truy cập đến giá trị của các thành phần dữ liệu (get, set) và viết thêm 1 properties chỉ đọc (get) dùng để lấy LayTuoi của Nguoi.
  - Viết 1 phương thức tên XemThongTin(): xuất giá trị các thành phần dữ liệu ra màn hình.
- Thêm 1 lớp tên SinhVien kế thừa từ lớp Nguoi và viết thêm các thành phần sau:

- Khai báo thêm các thành dữ liệu: string MaSV , string MaLop, string Email, string DienThoai.
  - Khai báo và định nghĩa các Constructor tham số và không tham số để khởi tạo các giá trị cho các thành phần dữ liệu .
  - Khai báo và định nghĩa các properties để truy cập đến giá trị của các thành phần dữ liệu (get, set) kiểm tra dữ liệu.
- Thêm 1 lớp tên NhanVien kế thừa từ lớp Nguoi và viết thêm các thành phần sau:
- Khai báo thêm các thành dữ liệu: string MaNhanVien , string Email, string DienThoai, DateTime NgayLamViec, string MaCongTy.
  - Khai báo và định nghĩa các Constructor tham số và không tham số để khởi tạo các giá trị cho các thành phần dữ liệu.
  - Khai báo và định nghĩa các properties để truy cập đến giá trị của các thành phần dữ liệu (get, set) và kiểm tra dữ liệu.
- Viết hàm Main() để kiểm tra các constructor, các properties, các phương thức của các lớp trên.
- Mở rộng thêm cho các đối tượng khác như GiamDoc, CaSi. Tất cả các lớp nên override lại phương thức *ToString()*.
- Sử dụng List để khai báo mảng các đối tượng.

```
List<Nguoi> ds = new List<Nguoi>();  
ds.Add(new SinhVien{.....});  
ds.Add(new NhanVien{.....});  
ds.Add(new Nguoi{.....});  
ds.Add(new SinhVien{.....});  
  
//duyet danh sach  
...
```

-----oOo-----

## Chương 2: THIẾT KẾ GIAO DIỆN WEB

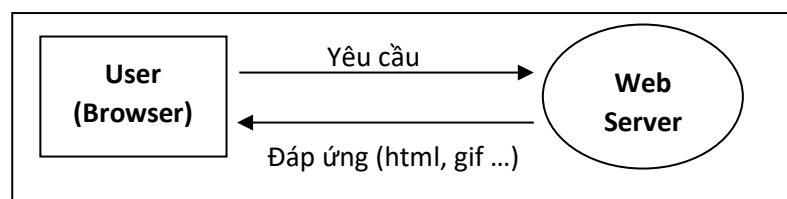
**Sau khi học xong chương này, học viên có khả năng:**

- *Nắm vững các thẻ cơ bản kết hợp CSS để làm layout cho trang web.*
- *Xử lý các hiệu ứng với jQuery.*
- *Vận dụng Bootstrap và jQueryUI trong thiết kế giao diện.*

### 2.1 Ngôn ngữ HTML

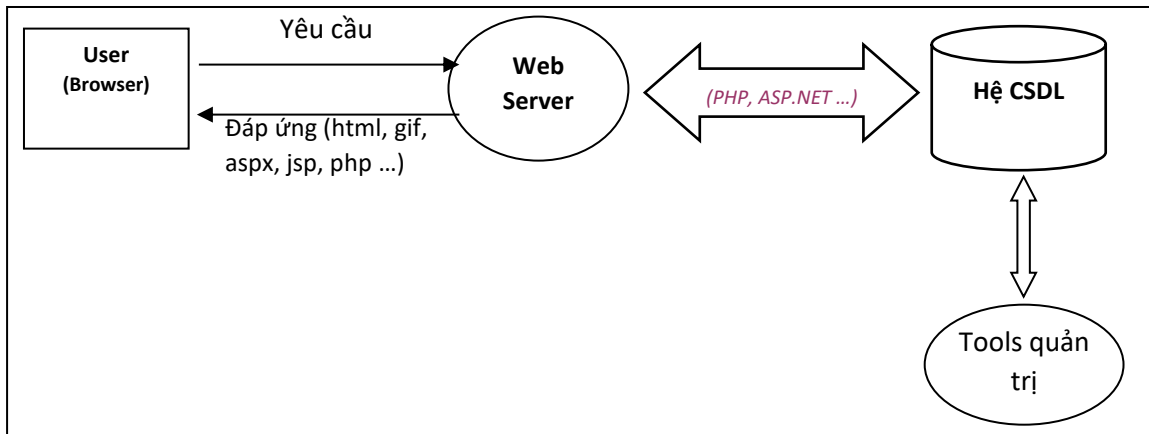
#### 2.1.1 Một số khái niệm

- Trang web:** Trang web (tĩnh) là một file dạng text chứa dữ liệu và các tag HTML. Khi hiển thị trong trình duyệt web, dữ liệu sẽ được hiển thị theo quy định của các tag mà nó nằm bên trong. Dữ liệu trong trang web có thể là văn bản, hình ảnh, âm thanh, video...
- Hyperlink:** là 1 liên kết chỉ đến 1 trang web khác. Một trang web có thể chứa nhiều link.
- Website:** Là tập hợp nhiều trang web thể hiện thông tin của 1 tổ chức, 1 chủ đề nào đó. Mỗi website có 1 trang trang chủ chứa các hyperlink liên kết đến các trang khác trong website. Người xem sẽ vào website bắt đầu từ trang chủ; từ trang này, nhờ các link trong đó mà họ sẽ đến được các trang khác trong toàn website.
- Browser :** Là chương trình dùng để xem các trang web. Các trình duyệt web nổi tiếng là FireFox, Google Chrome, Microsoft Edge.
- WebServer:** Là các máy "phục vụ web", đây là các máy tính trên Internet/Intranet có cài chương trình webserver. Webserver sẽ trả về cho người sử dụng trang web mà họ yêu cầu để họ xem. Webserver liên lạc với browser qua giao thức http(s). Một Webserver có thể chứa nhiều website. Hai chương trình webserver nổi tiếng nhất là: IIS và Apache.
- Http:** là giao thức để browser và web server trao đổi với nhau nhằm đưa trang web cho người dùng xem.
- Web tĩnh:** Là trang web chỉ có tag html và dữ liệu, tất cả đều gõ trực tiếp trong trang chứ không đặt ở nơi khác. File có tên mở rộng là. html hoặc. htm.



Trong mô hình web tĩnh, user yêu cầu 1 trang web html, trang web này đã được thiết kế sẵn và đặt trên webserver, trang web không hề có tương tác đến CSDL. Webserver chỉ việc lấy file html trả về cho user. Vậy là xong.

- Web động:** Là trang web có truy xuất đến cơ sở dữ liệu (Database) hoặc có tương tác với webserver để thực hiện 1 chức năng cao cấp nào đó. Một trang web động có thể trả về những kết quả khác nhau tùy theo yêu cầu của người sử dụng. Thiết kế web động đòi hỏi người thiết kế có nhiều kiến thức: HTML, Javascript, Database, WebServer, ... tốn nhiều công sức và thời gian. Mô hình Web động:



Trong mô hình web động, Webserver sẽ tương tác với các chương trình “hậu trường” phía sau nó (PHP, ASP.NET...) để thực hiện 1 số việc nào đó (thường là kết nối cơ sở dữ liệu), các chương trình này lấy dữ liệu trong hệ quản trị cơ sở dữ liệu và thực hiện định dạng (nếu cần) rồi đưa về cho webserver để webserver sẽ trả về cho user.

Các chương trình “hậu trường” như PHP, ASP.NET... là chương trình trung gian, là cầu nối giữa Webserver và cơ sở dữ liệu. Sở dĩ có chúng là vì tương tác với cơ sở dữ liệu không phải là mục tiêu của webserver, nhiệm vụ chính của WebServer là tương tác với user để trả về trang web (qua giao thức http).

### 2.1.2 Giới thiệu HTML

- HTML (*Hyper Text Markup Language*) là một ngôn ngữ để quy định cách hiển thị thông tin trong trang web. HTML gồm nhiều lệnh, mỗi lệnh gọi là 1 tag. Mỗi tag quy định một cách thức hiển thị dữ liệu trong trang web. Ví dụ như: chữ đậm, chữ nghiêng, màu chữ ... Người xem trang web không thấy các tag mà chỉ thấy các dữ liệu được định dạng bởi các tag. Nói đơn giản : **HTML là 1 ngôn ngữ dùng để tạo ra các trang web.**
- Các tag cùng với dữ liệu trong đó được lưu trong 1 file text, gọi là trang web. File này thường có tên mở rộng là. html hoặc. htm.
- Ví dụ: Nếu bạn gõ như sau khi tạo trang web:

```
Lớp: <b>ASP.NET</b><br>
Họ tên: <u><i>Nguyễn Văn Tèo</i></u>
```

thì kết quả hiện trong Browser sẽ thế này:

```
Lớp: ASP.NET
Họ tên: Nguyễn Văn Tèo
```

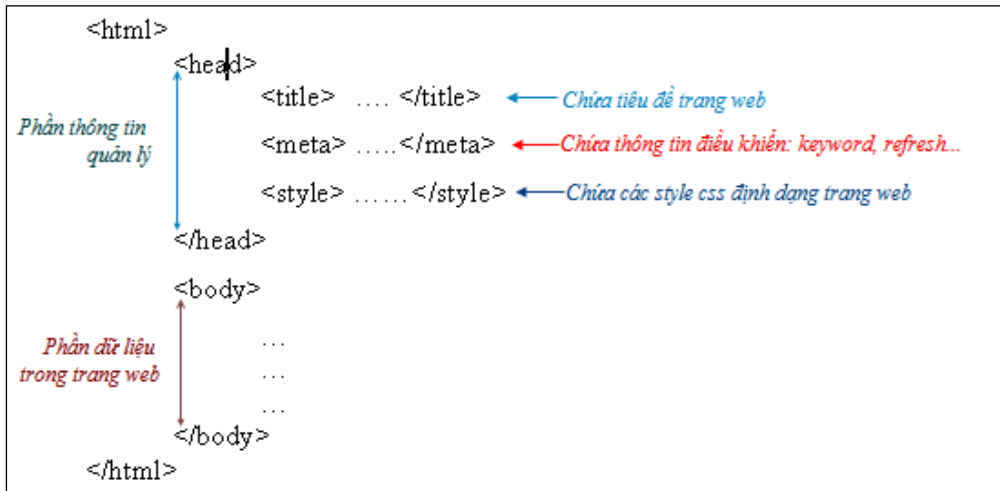
- Tên tag không quan trọng chữ thường chữ hoa, tên tag phải đặt trong 2 dấu < >, thường có mở và đóng. Một số tag chỉ có mở như <hr>, <br>, <img>.

### 2.1.3 Cấu trúc của 1 trang web

- Một trang web thường có mở đầu và kết thúc bởi tag **html**
- Tag **head** chứa những thông tin để quản lý và hoạt động nội tại bên trong trang web, không hiện ra cho user xem.
- Tag **title** là tiêu đề của trang web, bao giờ cũng nằm trong tag head



- Tag **body** chứa dữ liệu hiện ra trong trang web cho user xem.



### 2.1.4 Các tag HTML căn bản

Tên Tag	Cú pháp	Định nghĩa
<a>	<a href=abc.html>Tên hiển thị</a>	Tạo liên kết đến trang abc.html
<b> <strong>	<b>Nội dung đoạn văn</b> <strong>Hello ASP Core</strong>	In đậm
 	Nội dung đoạn văn   bắt đầu một dòng mới	Xuống dòng, không qua đoạn mới
<bgsound>	<bgsound delay="1" loop="-1" src="start.wav">	Nhạc nền cho trang web
<center>	<CENTER>Canh giữa chữ</CENTER>	Canh giữa
<div>	<div>.....</div>	div chứa 1 vùng dữ liệu trong trang
<em>	<em> Nội dung </em>	Định dạng kiểu chữ nghiêng
<h1> to <h6>	<h1>Tiêu đề 1 </h1> <h2>Tiêu đề 2 </h2> <h3>Tiêu đề 3 </h3> <h4>Tiêu đề 4 </h4> <h5>Tiêu đề 5 </h5> <h6>Tiêu đề 6 </h6>	Tạo tiêu đề (cấp 1 đến cấp 6)
<hr>	<hr color="#FF0000">	Tạo một đường gạch ngang
<i> <em>	<i> Nội dung </i> <em>In nghiêng</em>	Chữ in <i>nghiêng</i>
<iframe>	<iframe name="content_frame" width="488" height="244" src="welcome.htm"> </iframe>	Tạo 1 iframe (iframe là 1 vùng trong trang chứa 1 trang web khác)
<img>		Chèn hình vào văn bản.
<marquee>	<marquee direction="left" loop="-1" scrollamount="2" width="100%">Chữ cuộn</marquee>	Là tag dùng để cuộn (hình, văn bản).
<p>	<p>Nội dung đoạn văn bản.</p>	Paragraph
<small>	<small>Nội dung văn bản</small>	Chữ nhỏ

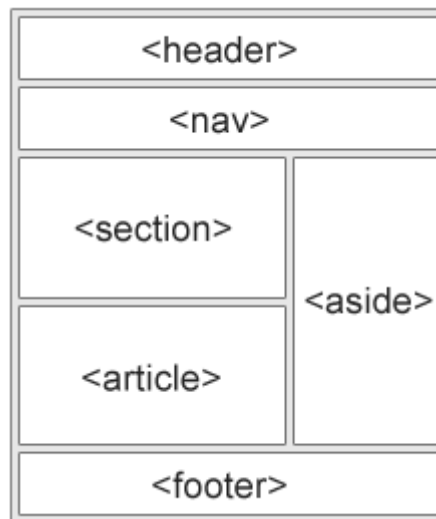
<b>&lt;span&gt;</b>	<b>&lt;span&gt;</b> Nội dung văn bản <b>&lt;/span&gt;</b>	Bao quanh 1 vùng text để định dạng
<b>&lt;strong&gt;</b>	<b>&lt;strong&gt;</b> Nội dung đoạn văn bản <b>&lt;/strong&gt;</b>	Chữ đậm
<b>&lt;sub&gt;</b>	<b>&lt;sub&gt;</b> Nội dung đoạn văn bản <b>&lt;/sub&gt;</b>	Chữ subscript (chữ xuống dưới+nhỏ)
<b>&lt;sup&gt;</b>	<b>&lt;sup&gt;</b> Nội dung đoạn văn bản <b>&lt;/sup&gt;</b>	Chữ superscript (chữ lên cao+nhỏ)
<b>&lt;u&gt;</b>	<b>&lt;u&gt;</b> Nội dung đoạn văn bản <b>&lt;/u&gt;</b>	Gạch dưới

## 2.1.5 Các tag mới trong HTML5

Tag	Mô tả
<b>&lt;article&gt;</b>	Định nghĩa một bài viết, một nội dung riêng biệt.
<b>&lt;aside&gt;</b>	Định nghĩa nội dung bên ngoài nội dung chính (thường là phần sidebar).
<b>&lt;audio&gt;</b>	Định nghĩa âm thanh, như nhạc hay trường audio khác..
<b>&lt;canvas&gt;</b>	Được dùng để hiển thị đồ họa.
<b>&lt;command&gt;</b>	Định nghĩa một nút lệnh, giống như một Radiobutton, hộp kiểm, hoặc một button.
<b>&lt;datalist&gt;</b>	Định nghĩa một danh sách tùy chọn, sử dụng thành phần này cùng với các thành phần input.
<b>&lt;details&gt;</b>	Xác định thêm chi tiết hoặc điều khiển có thể được ẩn hoặc hiển thị theo yêu cầu.
<b>&lt;embed&gt;</b>	Xác định nội dung nhúng như một plugin.
<b>&lt;figcaption&gt;</b>	Xác định một chú thích cho tag figure.
<b>&lt;figure&gt;</b>	Xác định các nội dung liên quan mạch lạc với nhau, như hình ảnh, sơ đồ, code,...
<b>&lt;footer&gt;</b>	Định nghĩa khu vực footer (phần cuối) của trang.
<b>&lt;header&gt;</b>	Định nghĩa khu vực header (phần đầu) của trang.
<b>&lt;hgroup&gt;</b>	Định nghĩa một nhóm các tiêu đề.
<b>&lt;keygen&gt;</b>	Xác định một cặp trường khóa chính sử dụng cho form.
<b>&lt;mark&gt;</b>	Xác định văn bản được đánh dấu, sử dụng khi muốn làm nổi bật văn bản của mình.
<b>&lt;meter&gt;</b>	Định nghĩa một phép đo. Sử dụng chỉ cho phép đo với giá trị tối thiểu và tối đa.
<b>&lt;nav&gt;</b>	Định nghĩa link danh mục (navigation)
<b>&lt;output&gt;</b>	Đại diện cho kết quả của phép tính (giống như được thực hiện bởi script).
<b>&lt;progress&gt;</b>	Mô tả tiến trình làm việc.
<b>&lt;rp&gt;</b>	Hiển thị những nội dung bên trong khi trình duyệt không hỗ trợ ruby.
<b>&lt;rt&gt;</b>	Định nghĩa một lời giải thích hoặc cách phát âm của các ký tự (đối với kiểu chữ Đông Á).

<ruby>	Định nghĩa một chú thích ruby (đối với kiểu chữ Đông Á). Chú thích Ruby được sử dụng trong khu vực Đông Á, hiển thị cách phát âm của các ký tự Đông Á.
<section>	Định nghĩa một khu vực (vùng bao).
<source>	Xác định nguồn cho một media.
<summary>	Xác định một tiêu đề cho các thành phần details, được sử dụng để mô tả chi tiết về tài liệu, hoặc các bộ phận của tài liệu.
<time>	Xác định thời gian, ngày tháng, hoặc năm sinh.
<video>	Xác định một video, chẳng hạn như một đoạn phim hoặc một trường video.
<wbr>	Xác định text quá dài sẽ tự động xuống hàng (không tràn layout)

### 2.1.6 Cấu trúc 1 trang web



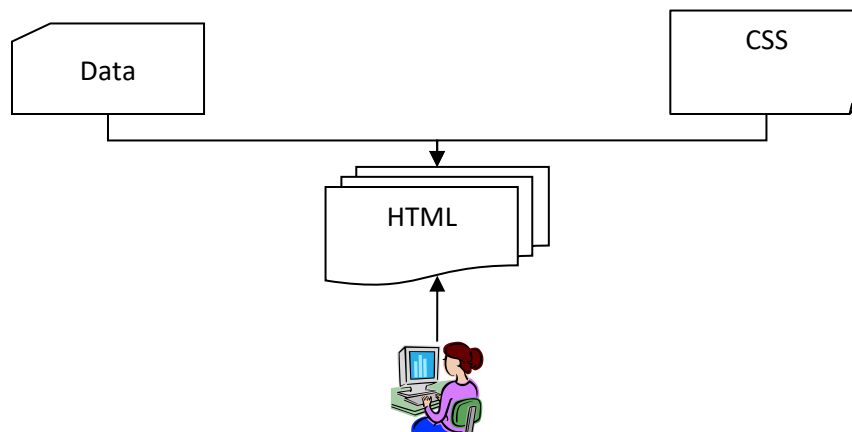
- **Header**: là vùng chứa nội dung giới thiệu hay hỗ trợ tìm kiếm nội dung trang,...và thường đặt đầu trang hay đầu mỗi chuyên đề nào đó (section).
- **Nav**: là vùng chứa các liên kết điều hướng đến các nội dung trong site (thanh điều hướng).
- **Section**: là vùng chứa nội dung liên quan một chuyên đề như kinh doanh, giải trí, giáo dục,..
- **Article**: ý nghĩa vùng này hơi giống **section** nên đôi khi gây nhầm lẫn, nhưng, nếu **section** chứa nội dung có tính chuyên đề và liên quan đến nội dung cả website thì, **article** chứa nội dung có tính độc lập với nội dung tổng thể của website – nghĩa là có thể được dùng lại trên một website khác mà không cần chỉnh sửa. **Article** có thể dùng cho các bài viết trên blog, diễn đàn, các bài trên báo, tạp chí, v.v...
- **Aside**: là vùng chứa một phần của toàn bộ nội dung website như một khối các liên kết điều hướng (bên cạnh thanh điều hướng trong vùng **nav**), thông tin quảng cáo hay sidebar.
- **Footer**: là vùng thường cuối trang, chứa nội dung liên quan thông tin bản quyền, tác giả hay tổ chức sở hữu website, các liên kết liên quan, v.v...

Chú ý rằng, cấu trúc trang web trên chỉ là một trường hợp chung, đơn giản. Tùy theo nội dung, tính năng của website mà cấu trúc của nó sẽ khác nhau.

## 2.2 Bảng định kiểu – CASCADING STYLE SHEET (CSS)

### 2.2.1 Giới thiệu

- CSS là 1 kỹ thuật dùng để định dạng các tag trong trang web. CSS giúp định dạng trang web rất nhanh nhờ nhiều kiểu định dạng tag, class, element... Bạn không thể định dạng 1 trang web cho đẹp khi không có sự am hiểu về CSS. Nếu làm được điều này, chúng ta có được các lợi điểm sau:
  - ✓ Dễ quản lý, bảo trì.
  - ✓ Tái sử dụng. Một qui luật kiểu dáng có thể áp dụng cho nhiều thành phần web khác nhau.
  - ✓ Cải thiện tốc độ.
    - Giảm lượng thông tin truyền tải.
    - Cách hiển thị của trình duyệt



- Style: Là 1 tập hợp các đặc điểm định dạng cho các thành phần trong trang. Để định dạng, ta chuyển sang chế độ code, rồi định nghĩa các style bên trong tag <style>. Tag <style> cần đặt trong tag head.

### 2.2.2 Khởi động nhanh

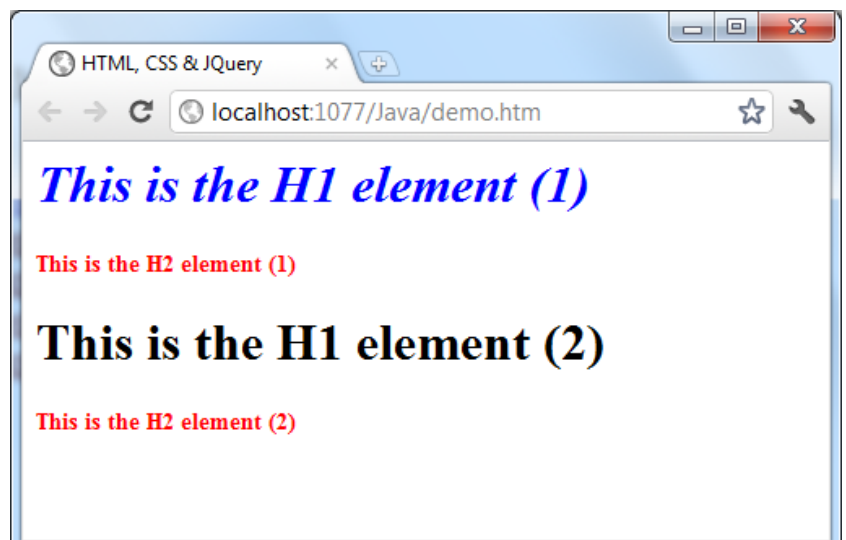
Để không gặp khó khăn của các qui luật của CSS, hãy tìm hiểu một ví dụ đơn giản về nó. Qua đó chúng ta có dịp làm quen với các khái niệm của CSS.

Mã nguồn HTML

```

<HTML>
<HEAD>
<title>HTML, CSS & JQuery</title>
<STYLE TYPE="text/css">
    H2{color:red;font-size: 14px;}
</STYLE>
</HEAD>
<H1 style="color:blue;font-style:italic;">
    This is the H1 element (1)</H1>
<H2>This is the H2 element (1)</H2>
<H1>This is the H1 element (2)</H1>
<H2>This is the H2 element (2)</H2>
</HTML>
  
```

Kết quả thực hiện



Phân tích ví dụ:

- Các thẻ <h2> nhỏ và màu đỏ là do dòng mã CSS ***H2{color:red;font-size: 14px;}***. Còn thẻ <h1> nghiêng và có màu xanh do thuộc tính style chứa CSS ***style="color:blue;font-style:italic;"***.
- Phương pháp định nghĩa CSS cho 1 thẻ phù hợp cho áp dụng hàng loạt ngược lại phương pháp inline (sử dụng thuộc tính style) thì các CSS được tạo ra chỉ để áp dụng cho từng trường hợp đơn lẻ.

### 2.2.3 Tạo style định dạng

#### 2.2.3.1 Tạo style định dạng cho 1 tag trong toàn trang:

Muốn định dạng tag nào thì style sẽ giống như tag muốn định dạng.

Ví dụ: định dạng tag **p** và tag **a**

```
<style>
p {color:#F00}
a { color:#039; text-decoration:none}
</style>
```

#### 2.2.3.2 Tạo style định dạng cho 1 đối tượng cụ thể có tên:

Muốn định dạng cụ thể 1 tag nào đó theo tên do bạn đặt thì tạo style bắt đầu bằng dấu #.

Ví dụ sau định dạng cho 1 tag có tên là box

```
<style>
#box { width:300px; height:150px; text-align:justify}
</style>
```

**Chú ý:** tag phải đặt tên theo id khớp với style đã tạo thì mới có tác dụng

```
<div id="box"> ..... </div>
```

#### 2.2.3.3 Tạo style định dạng cho tag bên trong 1 đối tượng có tên:

Muốn định dạng tag bên trong 1 vùng thì tạo style theo công thức sau: **#TênVùng tag**

Ví dụ sau định dạng cho các tag a trong vùng có tên là box

```
<style>
#box a { color: magenta; text-transform: uppercase}
</style>
```

#### 2.2.3.4 Tạo style dạng class và set class

Muốn tạo class gõ theo công thức sau: **.TênClass** Ví dụ sau tạo class. tieude

```
<style>
.caption {color:#993; padding:5px; margin:0px; text-align:center }
</style>
```

**Set class:** Tạo class xong, muốn tag nào định dạng theo class thì chỉ định thông số class.

Ví dụ:

```
<h4 class="caption">Tin xem nhiều</caption>
```

#### 2.2.3.5 Tạo file css và nhúng vào trang web

##### ❖ Tạo file css

- Nhấp menu File → New → chọn CSS → Create → Lưu với tên file. css
- Khai báo các style

❖ **Nhúng file css vào trang web**

```
<link href="c1.css" rel="stylesheet" type="text/css" />
```

Trong trang web cần áp dụng CSS, gõ code trên để nhúng file css, trong đó c1.css là tên file css muốn nhúng.

### 2.2.4 Các thuộc tính CSS

Khi thiết kế trang web với CSS thì vốn kiến thức CSS là khối lượng các thuộc tính CSS mà bạn có được. Sau đây trình bày danh sách các thuộc tính thường dùng nhất có phân loại.

Các thuộc tính CSS thường sử dụng để định nghĩa cho văn bản trên trang web như font chữ, màu sắc, chế độ hiển thị...

#### 2.2.4.1 Định dạng chữ

Thuộc tính	Mô tả
font-family: Verdana, Geneva, sans-serif;	Chỉ định tên font. Các font được liệt kê các nhau dấu phẩy. Áp dụng cho font được tìm thấy trước
font-size: 12px;	Kích thước font
font-style: italic;	Kiểu font (italic: chữ nghiêng)
line-height: 12px;	Độ cao của mỗi hàng
font-weight: bold;	Độ đậm của font chữ: bold (đậm), 100 (độ đậm 100)
font-variant: small-caps;	Chữ hoa nhỏ, ký tự đầu lớn hơn
text-transform: uppercase;	đổi chữ hoa, chữ thường (capitalize: chữ hoa đầu từ, uppercase: toàn chữ hoa, lowercase: toàn chữ thường)
color: #F00;	Màu sắc có thể dùng mã (Red, Green, Blue) hoặc tên màu
text-decoration: none;	Trang trí chữ: Underline: gạch dưới chữ, Strikethrough: gạch giữa chữ, Overline: gạch đầu chữ, None: không gạch
text-align: center;	canh chữ (left, right, center, justify)
text-shadow	màu bóng của chữ
letter-spacing: 2em;	Khoảng cách giữa các ký tự
text-align: justify;	Canh lề: left, right, center, justify
text-indent: 5px;	Khoảng thụt vào đầu dòng
vertical-align: middle;	Canh lề đứng: top, bottom, middle, base-line
word-spacing: 4em;	Khoảng cách giữ các từ
letter-spacing: 2em;	Khoảng cách giữa các ký tự

#### 2.2.4.2 Background

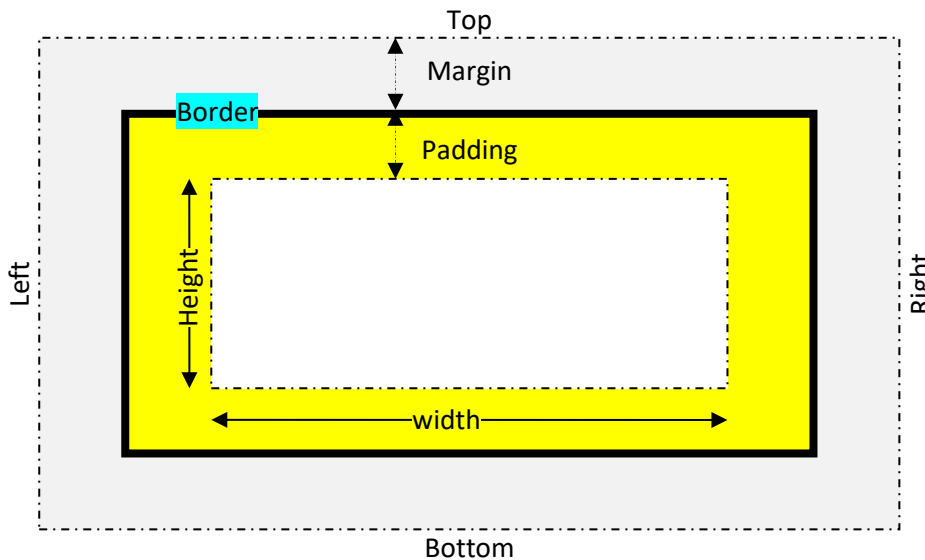
Nền chỉ có 2 loại là màu và ảnh. Nếu là ảnh thì cần điều chỉnh chế độ lặp lại (lát). Trong trường hợp không lặp bạn cần điều chỉnh vị trí đặt ảnh nền.

Thuộc tính	Mô tả
background-color: #F00;	Màu nền
background-attachment: fixed;	Chế độ cuộn ảnh nền ✓ Fixed = cố định ảnh nền khi cuộn nội dung ✓ Scroll = ảnh nền cuộn theo nội dung
background-image: url(anh/abc.jpg);	Ảnh nền
background-repeat: repeat;	Chế độ lặp: ✓ None: không lặp ✓ Repeat: lặp cả 2 chiều

	✓ repeat-x: lặp chiều ngang ✓ repeat-y: lặp chiều đứng
background-position: left center;	Vị trí đặt ảnh nền trường hợp không lặp
Background-size: 100% 100%	Kích thước ảnh nền (width height)

### 2.2.4.3 Box

Mô hình của một hộp gồm kích thước, lề, phần đệm, đường viền, chế độ xếp hộp.



Thuộc tính	Mô tả
height: 222px;	Chiều cao
width: 111px;	Chiều rộng
margin: 6px;	Khoảng cách từ lề của đối tượng với những đối tượng bên ngoài. Sử dụng margin-top, margin-right, margin-bottom, margin-left nếu muốn định nghĩa riêng mỗi cạnh.
padding: 4px;	Phần đệm bên trong hộp (Khoảng cách từ lề của đối tượng với nội dung bên trong). Sử dụng padding-top, padding-right, padding-bottom, padding-left nếu muốn định nghĩa riêng mỗi cạnh.
border: medium dotted #F00;	Đường kẻ theo thứ tự <b>độ dài, kiểu, màu</b> . Sử dụng border-top, border-right, border-bottom, border-left nếu muốn định nghĩa riêng mỗi cạnh.
float: left;	Gâm (chế độ xếp hộp) vào trái: left (gâm trái), right (gâm phải)
clear: right;	Hủy bỏ chế độ ngâm: left(xóa ngâm trái), right(xóa ngâm phải), both(xóa ngâm cả 2 bên)

### 2.2.4.4 Border

- border-style: kiểu đường viền.
- border-width: độ dày.
- border-color: Màu đường viền.
- border-radius: bo tròn góc
- box-shadow: tạo bóng cho đối tượng định dạng

### 2.2.4.5 List

Để điều chỉnh <OL>, <UL> và <LI> bạn cần sử dụng các thuộc tính css sau đây.

Thuộc tính	Mô tả
list-style-position: inside;	Vị trí đặt dấu danh sách
list-style-type: square;	Kiểu dấu danh sách: <i>disc: tròn đen; circle: tròn trắng; square: vuông...</i>
list-style-image: url(xyz/abc.gif);	hình dùng thay thế ký tự bullet

### 2.2.4.6 Layer

Để tạo ra và điều chỉnh các thông số của nó, bạn cần học các thuộc tính css sau đây là đủ.

Thuộc tính	Mô tả
overflow: scroll;	Điều khiển chế độ tràn: scroll, visible, hidden
position: relative;	Chế độ vị trí của layer. Absolute (vị trí tuyệt đối so với layer mẹ), relative (vị trí tương đối tức đặt tại vị trí đặt thẻ)
visibility: visible;	Ẩn hiện layer
left: 0px;	Vị trí layer tính từ bên trái
top: 0px;	Vị trí layer tính từ bên trên
right: 0px;	Vị trí layer tính từ bên phải
bottom: 0px;	Vị trí layer tính từ bên dưới
z-index: 111;	Chiều z hướng từ trong màn hình ra người dùng. Layer nào có z-index cao hơn sẽ nằm trên.

### 2.2.5 Bộ chọn (Selector)

Bộ chọn (selector) là nơi định nghĩa các qui luật kiểu dáng để áp dụng cho các thành phần trên trang web. Có 3 loại bộ chọn cơ bản là Class, ID và HTML

#### 2.2.5.1 Bộ chọn HTML (HTML Selector)

- ✓ Định nghĩa: định nghĩa kiểu dáng bổ sung cho các thẻ HTML

**<tên thẻ>{<khai báo các thuộc tính css>}**

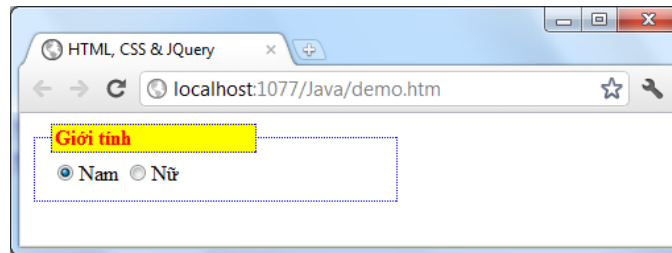
- ✓ Áp dụng: Tự động áp dụng các qui luật css trong phần khai báo cho tất cả các thẻ có tên là <tên thẻ>
- ✓ Ví dụ sau đây định nghĩa lại thẻ <fieldset> và legend với các thuộc tính kích thước (width), đường kẻ (border), màu chữ (color), màu nền (background-color).

```
<HTML>
<HEAD>
<title>HTML, CSS & JQuery</title>
<STYLE TYPE="text/css">
FIELDSET{
width: 250px;
border: 1px dotted #0000FF;
}
LEGEND{
font-weight: bold;
color: #FF0000;
background-color: #FFFF00;
border: 1px dotted #0000FF;
width: 150px;
}
</STYLE>
</HEAD>
<body>
<fieldset>
```



```
<legend>Giới tính</legend>
<input type="radio" name="rdoGioiTinh" checked/>Nam
<input type="radio" name="rdoGioiTinh" />Nữ
</fieldset>
</body>
</HTML>
```

Kết quả hiển thị



### 2.2.5.2 Bộ chọn lớp (Class Selector)

- ✓ Định nghĩa: định nghĩa một lớp được bắt đầu bởi dấu chấm (.) bên trong khai báo nhiều thuộc tính css để áp dụng cho bất kỳ thẻ nào chỉ định bởi thuộc tính class của nó.

**.<tên lớp>{<khai báo các thuộc tính css>}**

- ✓ Áp dụng: tất cả các thẻ sử dụng thuộc tính class có giá trị là <tên lớp>. Chú ý thuộc tính class của mỗi thẻ có thể chỉ đến nhiều class cùng một lúc (cách nhau khoảng trắng).
- ✓ Ví dụ sau định nghĩa 2 bộ chọn lớp sau đó thẻ <H1> áp dụng một còn thẻ <DIV> áp dụng cả hai để tận dụng các đặc điểm tổng hợp.

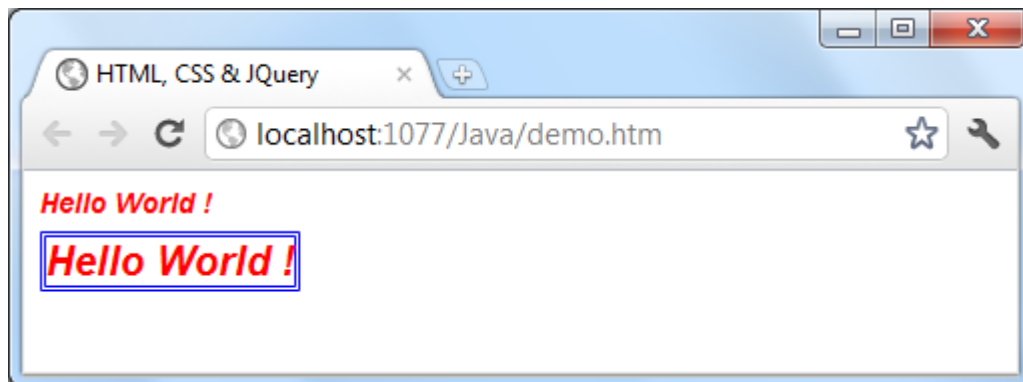
```
<HTML>
<HEAD>
<title>HTML, CSS & JQuery</title>
<STYLE TYPE="text/css">
. MyHeader{
    font-family: Arial, Helvetica, sans-serif;
    font-weight: bold;
    font-style: italic;
    font-size: 14px;
    color: #FF0000;
}
. MyBorder{
    border: 3px double blue;
    font-size: 20px;
    display: inline;
}
</STYLE>
</HEAD>
<body>
```

```
<h1 class="MyHeader">Hello World !</h1>  
<div class="MyHeader MyBorder">Hello World !</div>  
</body>  
</HTML>
```

Trong đó

- ✓ font-family: tên font chữ
- ✓ font-weight: độ đậm
- ✓ font-style: kiểu chữ
- ✓ font-size: kích thước chữ
- ✓ color: màu chữ

Kết quả thực hiện



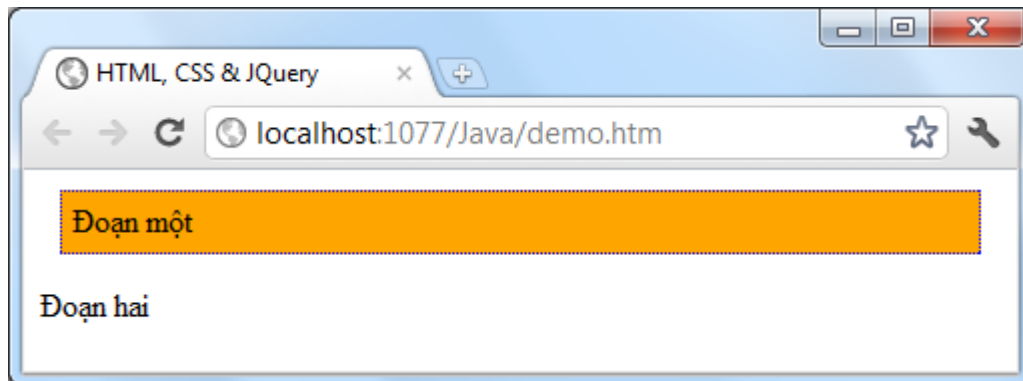
### 2.2.5.3 Bộ chọn định danh (ID Selector)

- ✓ Định nghĩa: giống như bộ chọn lớp nhưng khởi đầu với dấu rằn (#)  
**#<tên định danh>{<khai báo các thuộc tính css>}**
- ✓ Áp dụng: tất cả các thẻ sử dụng thuộc tính id với giá trị là <tên định danh>
- ✓ Ví dụ sau định nghĩa bộ chọn định danh tên là #MyPara sau đó áp dụng cho một thẻ <P> trong trang web. Chú ý thẻ <P> còn lại không hề bị ảnh hưởng gì.

```
<HTML>  
<HEAD>  
<title>HTML, CSS & JQuery</title>  
<STYLE TYPE="text/css">  
#MyPara  
{  
    background-color: orange;  
    background-image: url(images/abc.gif);  
    text-align: justify;  
    margin: 10px;  
    padding: 5px;  
    border: 1px dotted #0000FF;  
}  
</STYLE>
```

```
</HEAD>
<body>
  <p id="MyPara">Đoạn một</p>
  <p>Đoạn hai</p>
</body>
</HTML>
```

Kết quả thực hiện



#### 2.2.5.4 Bộ chọn cho liên kết

Định nghĩa: định nghĩa css cho siêu liên kết. Với liên kết có bốn trạng thái sử dụng là chưa thăm (chưa click), đã thăm, có chuột và tích cực (đang chọn). Vì vậy để định nghĩa CSS áp dụng cho liên kết bạn không chỉ định nghĩa CSS cho thẻ <A> mà còn định nghĩa cả 4 trạng thái của nó. Sau đây là cú pháp chung định nghĩa css cho siêu liên kết

- A: {<khai báo các thuộc tính css>}**
- A:link {<khai báo các thuộc tính css>}**
- A:visited{<khai báo các thuộc tính css>}**
- A:hover{<khai báo các thuộc tính css>}**
- A:active{<khai báo các thuộc tính css>}**

Tên style	Ý nghĩa
<b>a:link</b>	định dạng cho tag a chưa được nhấp trong trang
<b>a:visited</b>	định dạng cho tag a đã được nhấp trong trang
<b>a:hover</b>	định dạng cho tag a trong trang đang được đưa chuột vào
<b>a</b>	định dạng cho tag a trong trang
<b>#menu a:link</b>	định dạng cho tag a chưa nhấp trong vùng có tên là menu
<b>#menu a:visited</b>	định dạng cho tag a đã được nhấp trong vùng có tên là menu
<b>#menu a:hover</b>	định dạng cho tag a đang đưa chuột vào trong vùng menu
<b>#menu a</b>	định dạng cho tag a trong vùng có tên là menu
<b>.menu A:link</b>	định dạng cho tag a (chưa nhấp) trong vùng có class là menu
<b>.menu A:visited</b>	định dạng cho tag a (đã nhấp) trong vùng có class là menu
<b>.menu A:hover</b>	định dạng tag a (đang đưa chuột vào) trong class là menu
<b>.menu A</b>	định dạng cho tất cả liên kết trong vùng có class là menu

<b>.tieudetin</b>	định dạng cho liên kết có class là <b>tieudetin</b>
-------------------	---

- ✓ Áp dụng: Tất cả các liên kết trong trang có định nghĩa CSS cho liên kết
- ✓ Ví dụ

```
<HTML>
<HEAD>
<title>HTML, CSS & JQuery</title>
<STYLE TYPE="text/css">
A{
    font-family: Arial; font-size: 16px; text-decoration: none; }
A:link{ color: Blue; }
A:visited { color: Green; }
A:hover{
    text-decoration: underline; color: Red;
    border: 1px dotted Red; background-color: Yellow;
}
A:active { color: Orange; }
</STYLE>
</HEAD>
<body>
<a href="#1">Link 1</a> | <a href="#2">Link 2</a> | <a href="#3">Link 3</a> |
<a href="#4">Link 4</a> | <a href="#5">Link 5</a> |
</body>
</HTML>
```

Kết quả thực hiện: Link3 đang có chuột, Link2 đã bị click trước đó

[Link 1](#) | [Link 2](#) | [Link 3](#) | [Link 4](#) | [Link 5](#) |

### 2.2.5.5 Nhiều bộ chọn cùng kiểu

- ✓ Định nghĩa: định nghĩa nhiều bộ chọn đồng một số kiểu dáng.

**<bộ chọn 1>, <bộ chọn 2>, ..., <bộ chọn n>{<khai báo các thuộc tính css>}**

- ✓ Áp dụng: áp dụng các khai báo css cho tất cả các thẻ có chỉ định sử dụng css thỏa mãn với các bộ chọn được liệt kê cách nhau dấu phẩy.
- ✓ Ví dụ

```
<html>
<head>
<title>HTML, CSS & JQuery</title>
<style type="text/css">
#A,. B, DIV INPUT, H2
{
    font-weight: bold; font-style: italic; color: #FF0000; font-size: 11pt;
}
</style>
</head>
<body>
<div class="B">Công cha như núi thái sơn</div>
<div id="A">Nghĩa mẹ như nước trong nguồn chảy ra</div>
<div><input value="Một lòng thờ mẹ kính cha" size="55" /></div>
<h2>Cho tròn đạo hiếu mới là đạo con</h2>
<input value="Thẻ input này không bị ảnh hưởng gì" size="55" />
</body>
</html>
```

Kết quả thực hiện

*Công cha như núi thái sơn*

*Nghĩa mẹ như nước trong nguồn chảy ra*

Một lòng thờ mẹ kính cha

*Cho tròn đạo hiếu mới là đạo con*

Thẻ input này không bị ảnh hưởng gì

Lưu ý: "DIV INPUT" có nghĩa là định nghĩa css cho các thẻ <INPUT> đặt trong các thẻ <DIV>. Vì vậy trong bài này thẻ <input> không đặt trong <div> không hề chịu tác dụng của css đã định nghĩa.

### 2.2.5.6 Bộ chọn khoanh vùng

- ✓ Định nghĩa: định nghĩa CSS cho các vùng khác nhau trên trang. Như vậy chúng ta cần xác định vùng cần áp dụng và bộ chọn chứa các CSS để áp dụng.

**<vùng> <bộ chọn>{<khai báo các thuộc tính css>}**

- ✓ Áp dụng CSS của bộ chọn cho các thẻ đặt trong <vùng> và chỉ định áp dụng bộ chọn.
- ✓ Ví dụ:

```
<HTML>
<HEAD>
<title>HTML, CSS & JQuery</title>
<STYLE TYPE="text/css">
    /*--vỏ bọc bên ngoài rộng 900px, canh giữa, nền trắng--*/
    . container{width:900px; margin: 0px auto; background-color: White;}
    /*--đầu trang cao 100px--*/
    . top{height: 100px; background-color: Red;}
    /*--menu trang cao 22px, canh giữa--*/
    . menu{height: 22px; background-color: Yellow; text-align:center}
    /*--giữa trang cao tối thiểu 400px--*/
    . middle{min-height: 400px;}
    /*--giữa-trái cao như middle, rộng 250px, gâm trái--*/
    . middle_left{
        float:left;width: 250px;min-height:inherit;
        background-color: Aqua;
    }
    /*--giữa-phải cao như middle, rộng 650px, gâm phải--*/
    . middle_right{
        float:right;width: 645px;min-height:inherit;
        background-color: White;
    }
    /*--chân trang không gâm, cao 22px--*/
    . bottom{clear:both;height: 22px; background-color: Yellow;}
    /*--fieldset trong. middle_left cao tối thiểu 150--*/
    . middle_left fieldset{min-height: 150px;}
    /*--li trong. middle_left không dùng dấu, kẻ chân--*/
    . middle_left li{list-style-type:none; border-bottom: 1px dotted red;}
    /*--liên kết trong. menu cách nhau 20px--*/
    . menu a{padding: 0px 10px 0px 10px;}
```

```
/*--liên kết trong. middle_left chữ HOA nhỏ--*/
.middle_left a{font-variant:small-caps;}
/*--liên kết có chuột trong. middle_left in đậm--*/
.middle_left a:hover{font-weight: bold;}
/*--liên kết chung cho toàn trang--*/
a{text-decoration: none;}
a:link, a:active, a:visited{color: Blue;}
a:hover{color: Red;}
body{background-color: Gray;}

</STYLE>
</HEAD>
<body>
  <div class="container">
    <div class="top"></div>
    <div class="menu">
      <a href="#1">Home</a> | <a href="#2">About Us</a> |
      <a href="#3">Contact Us</a> | <a href="#4">Feedback</a> |
      <a href="#5">FAQs</a>
    </div>
    <div class="middle">
      <div class="middle_left">
        <fieldset>
          <legend>Member Info</legend>
        </fieldset>
        <fieldset>
          <legend>Products</legend>
          <li><a href="#1">Nokia</a></li>
          <li><a href="#2">Sumsung</a></li>
          <li><a href="#3">Sony Ericsson</a></li>
          <li><a href="#4">Motorola</a></li>
          <li><a href="#5">Apple</a></li>
          <li><a href="#5">Seamen</a></li>
        </fieldset>
        <fieldset>
          <legend>Online Support</legend>
        </fieldset>
      </div>
      <div class="middle_right"></div>
    </div>
    <div class="bottom"></div>
  </div>
</body>
</HTML>
```

Các bạn lưu ý các điểm sau:

✓ Khoanh vùng liên kết cho 2 vùng khác nhau là. menu và. middle\_left:

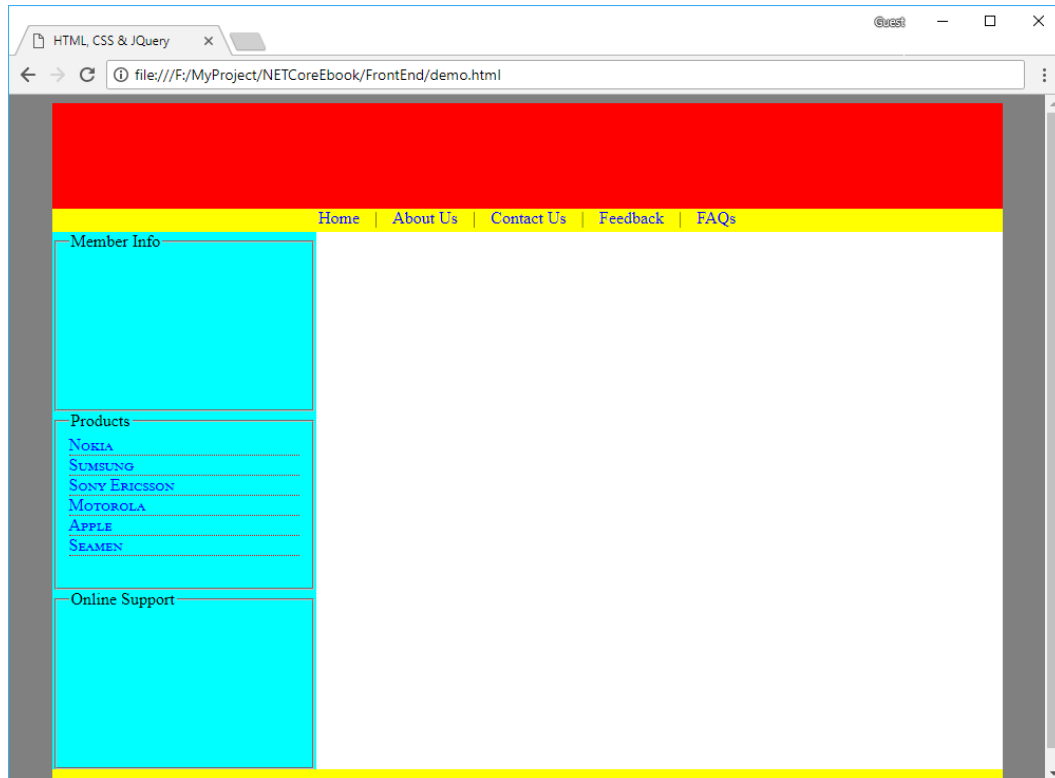
```
/*--liên kết trong. menu cách nhau 20px--*/
.menu a{padding: 0px 10px 0px 10px;}
/*--liên kết trong. middle_left chữ HOA nhỏ--*/
.middle_left a{font-variant:small-caps;}
/*--liên kết có chuột trong. middle_left in đậm--*/
.middle_left a:hover{font-weight: bold;}
```

Chúng ta cũng khoanh vùng cho các <fieldset> và <li> đặt trong middle\_left

```
/*--fieldset trong. middle_left cao tối thiểu 150--*/
```

```
.middle_left fieldset{min-height: 150px;}
/*--li trong. middle_left không dùng dấu, kẻ chân--*/
.middle_left li{list-style-type:none; border-bottom: 1px dotted red;}
```

Kết quả thực hiện:



### 2.2.6 Qui tắc nạp chồng

Nếu nhiều bộ chọn cùng được áp dụng cho cùng một thẻ thì phần khác nhau sẽ được gộp lại và phần giống nhau sẽ bị bộ chọn có độ ưu tiên cao hơn nạp chồng lên các bộ chọn có độ ưu tiên thấp hơn.

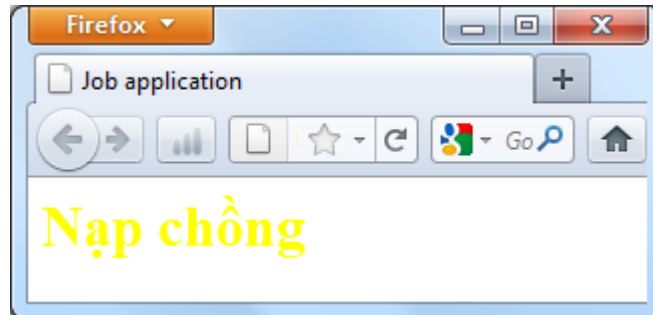
Thể loại và vị trí định nghĩa của các bộ chọn sẽ ảnh hưởng đến độ ưu tiên của chúng khi áp dụng lên một thẻ. Hãy ghi nhớ 2 qui tắc sau:

- ✓ Phân biệt theo thể loại:
  - **Nội tuyến** -> Bộ chọn **ID** -> Bộ chọn **Class** -> Bộ chọn **HTML** -> Mặc định
- ✓ Phân biệt theo vị trí định nghĩa:
  - **Nội tuyến** -> **Nhúng** -> **Liên kết ngoài**

Ví dụ sau thẻ <H1> chịu ảnh hưởng cả 3 bộ chọn khác nhau. Khi chạy sẽ cho dòng chữ "Nạp Chồng" màu vàng

```
<HTML>
<HEAD>
  <TITLE>Job application</TITLE>
  <style type="text/css">
    H1{color: green;}
    #xyz{color: yellow;}
    . abc{color: Red;}
  </style>
```

```
</HEAD>
<BODY>
    <H1 class="abc" id="xyz">Nạp chồng</H1>
</BODY>
</HTML>
```



### 2.2.7 CSS3

Ngoài những thuộc tính đã được hỗ trợ ở CSS1, CSS2; CSS3 còn có các thuộc tính:

Thuộc tính	Mô tả
animation	Xác định một chuyển động của một thành phần.
appearance	Định dạng cho thành phần trông như giao diện chuẩn gần với người dùng.
backface-visibility	Xác định bề mặt sau của thành phần khi thực hiện một chuyển động xoay.
background-clip	Xác định vùng background được cắt bớt theo vùng được giới hạn.
background-origin	Xác định giá trị tương đối của background giới hạn theo vùng giới hạn.
background-size	Xác định lại chiều rộng và chiều cao cho background.
background gradient	Tạo màu sắc cho background theo biên độ giảm dần.
Nhiều background	Sử dụng để khai báo nhiều dạng background khác nhau trong cùng một tag.
border-image	Dùng để định dạng các dạng border bằng hình ảnh.
border-radius	Dùng để định dạng các dạng bo góc của border.
box-align	Xác định vị trí cho thành phần theo chiều dọc hoặc theo chiều thẳng đứng.
box-direction	Xác định hướng cho thành phần.
box-flex	Xác định sự ưu tiên linh hoạt theo các thành phần khác.
box-ordinal-group	Cho biết thứ tự ưu tiên của các thành phần.
box-orient	Xác định thành phần dọc theo phương hướng khối hoặc theo trục.
box-pack	Định vị trí của thành phần theo mép rìa của thành phần.



box-sizing	Xác định lại chiều rộng và chiều cao của thành phần.
box-shadow	Định dạng bóng cho thành phần.
column	Dùng để chia nội dung thành phần thành nhiều cột khác nhau.
@font-face	Định dạng các dạng font chữ khác nhau theo các dạng font riêng.
font-size-adjust	Dùng để định dạng điều chỉnh cho font chữ, độ lớn của chữ được thể hiện bởi phép nhân.
@keyframes	Dùng để điều khiển diễn biến một hoạt động của thành phần, được dùng kèm với thuộc tính <a href="#">animation</a> .
nav	Di chuyển qua lại giữa các thành phần điều hướng (navigate) bằng cách di chuyển các phím mũi tên.
opacity	Hiển thị cấp độ trong suốt cho thành phần.
perspective	Cho ta thấy được chiều sâu của thành phần trong khai báo 3D.
perspective-origin	Định nghĩa trục quay cho thành phần có sử dụng perspective.
resize	Định dạng cho vùng nội dung mà người dùng có thể thay đổi được kích thước.
text-justify	Tăng hoặc giảm khoảng cách giữa các từ và giữa các ký tự sao cho dàn đều thành phần.
text-overflow	Xác định vùng text được cắt bớt.
text-shadow	Xác định bóng đổ cho text.
transform	Xác định một chuyển đổi 2 chiều, 3 chiều, có thể là xoay, tỷ lệ, di chuyển, nghiêng, ...
transform-origin	Xác định trục cho một chuyển đổi 2 chiều, 3 chiều.
transform-style	Các thành phần bên trong sẽ giữ vị trí 3D của nó.
transition	Xác định một quá trình chuyển đổi khi có một hành động.
word-break	Sẽ làm cho những chữ trong một từ không còn là một thể thống nhất, nghĩa là có thể xuống hàng bất cứ vị trí nào trong từ.
word-wrap	Sẽ làm cho những từ dài xuống hàng mà không làm vỡ layout.

Các bộ chọn CSS3 bổ sung:

Bộ chọn	Ví dụ	Mô tả
---------	-------	-------

tag01~tag02	ul~p	Chọn tất cả thành phần 02 khi có thành phần 01 ở trước.
[attribute^=value]	img[src^="bnr_"]	Chọn tất cả thành phần với thuộc tính có <i>giá trị bắt đầu</i> bằng "value".
[attribute\$=value]	img[src\$=".gif"]	Chọn tất cả thành phần với thuộc tính có <i>giá trị kết thúc</i> bằng "value".
[attribute*=value]	img[src*=""]	Chọn tất cả thành phần với thuộc tính có <i>giá trị đặc biệt</i> bằng "value".
:first-of-type	p:first-of-type	Chọn thành phần con đầu tiên hoặc duy nhất trong các thành phần cha.
:last-of-type	p:last-of-type	Chọn thành phần con cuối cùng hoặc duy nhất trong các thành phần cha.
:only-of-type	p:only-of-type	Chọn thành phần con trong các thành phần cha, khi thành phần cha có một thành phần con là chính nó. Trong thành phần cha có thể chứa nhiều thành phần con, tuy nhiên thành phần con được chọn phải là duy nhất, không được có từ 2 trở lên.
:only-child	p:only-child	Chọn thành phần con trong các thành phần cha, khi thành phần cha có mỗi thành phần con là chính nó, không được chứa thành phần con khác. Chỉ chọn thành phần có thành phần cha, những thành phần độc lập sẽ không được chọn.
:nth-child(n)	p:nth-child(3)	Chọn thành phần thứ "n" trong thành phần cha. Chỉ chọn thành phần có thành phần cha, những thành phần độc lập sẽ không được chọn.
:nth-last-child(n)	p:nth-last-child(3)	Chọn thành phần thứ "n" tính từ thành phần cuối trong thành phần cha. Chỉ chọn thành phần có thành phần cha, những thành phần độc lập sẽ không được chọn.
:nth-of-type(n)	p:nth-of-type(3)	Chọn thành phần thứ "n".
:nth-last-of-type(n)	p:nth-last-of-type(3)	Chọn thành phần thứ "n" từ thành phần cuối trong thành phần cha.
:last-child	p:last-child	Chọn thành phần cuối cùng trong thành phần cha. Chỉ chọn thành phần có thành phần cha, những thành phần độc lập sẽ không được chọn.
:root	:root	Chọn thành phần gốc của văn bản.
:empty	p:empty	Chọn thành phần không chứa thành phần khác.

:target	#event:target	Sử dụng trong liên kết anchor name (link neo).
:enabled	input:enabled	Chọn thành phần <i>được kích hoạt</i> (enabled), thường sử dụng cho các thành phần của <a href="#">form</a> .
:disabled	input:disabled	Chọn thành phần <i>bị vô hiệu hóa</i> (disabled), thường sử dụng cho các thành phần của <a href="#">form</a> .
:checked	input:checked	Chọn thành phần <i>được check</i> (checked), thường sử dụng cho các thành phần của <a href="#">form</a> .
:not(bộ chọn)	:not(p)	Chọn tất cả ngoại trừ <i>bộ chọn</i> trong ngoặc.
::selection	::selection	Chọn phần tử được người dùng chọn.

## 2.3 JAVASCRIPT VÀ JQUERY

### 2.3.1 Javascript

- Là ngôn ngữ lập trình dạng script thực thi trong browser.
- Javascript giúp trang web có tính tương tác : đổi màu 1 đối tượng khi đưa chuột vào, đổi nội dung của 1 tag, đưa ra các thông báo cần giao tiếp, phóng to hình.
- Trong trang web, mã lệnh javascript được đặt bên trong tag script.
- Mã lệnh javascript có thể đặt trực tiếp trong trang html hoặc có đặt trong 1 file riêng (\*.js). Khi đó trang html muốn dùng code javascript thì link đến file js.
- Mỗi lệnh javascript kết thúc là dấu ;

#### 2.3.1.1 Đưa javascript vào trang

Là viết mã lệnh Javascript trực tiếp trong file HTML với tag script. Ví dụ:

```
<script> alert("Chào bạn"); </script>
```

#### 2.3.1.2 Viết mã javascript trong file riêng

Là viết mã lệnh trong file. js nằm ngoài trang web. Sau đó nhúng file js vào trang web

a. Tạo file javascript:

- Nhấp menu File → New → Javascript → Create
- Gõ mã lệnh javascript. Ví dụ: gõ

```
<script>
    hoten=prompt("Bạn ơi bạn tên gì?");
    alert("Chào bạn " + hoten);
</script>
```

- Lưu file với tên mở rộng là. js
- b. Liên kết file js đến trang html
  - Mở 1 fle html. Nhấp vị trí muốn chèn (thường trong tag head) rồi kéo thả file js vào

### 2.3.2 jQuery

- jQuery được phát minh bởi John Resig năm 2006, là một trong những thư viện JavaScript được dùng rộng rãi nhất hiện nay giúp viết code JavaScript dễ hơn.
- jQuery có nhiều hàm giúp định dạng, thay đổi nội dung trang web, tạo nhiều hiệu ứng như mờ dần, chạy dọc chạy ngang, tạo request ajax v.v...
- jQuery cho phép tạo ra các Plugin.

#### a. Nhúng jquery vào trang web

- Vào trang chủ [www.jquery.com](http://www.jquery.com) và download phiên bản mới nhất (file js), chép vào folder website của bạn, rồi insert vào trang web bằng tag script.
- Hoặc chèn JQuery từ site chính thức của JQuery:

```
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.js">
</script>
<script type="text/javascript">
    $(document).ready(function(){
        // Mã jquery của bạn
    });
</script>
```

Lệnh đầu tiên trong jQuery mà bạn cần biết là lệnh theo dõi document ready (ready event), lệnh này theo dõi và đợi cho đến khi document sẵn sàng. Code JQuery thường đặt trong sự kiện này để chúng thực thi khi tài liệu sẵn sàng.

```
<script type="text/javascript">
    $(document).ready(function(){
        // Mã jquery của bạn
    });
</script>
```

#### b. Bộ chọn (Selector)

Thao tác cơ bản của JQuery là chọn các phần tử trong tài liệu HTML và thực hiện một việc gì đó. Cú pháp như sau: **\$(query).action()**

\$ là kí hiệu đặc biệt, xác định đây là câu lệnh jquery. **action** là hàm sẽ tác động lên các phần tử được chọn (click, change...). **query** là cách chọn phần tử.

##### Bộ chọn cơ bản:

Selector	Ví dụ	Ý nghĩa
<tên thẻ>	\$("#a")	chọn tất cả các tag <a>
#<định danh>	\$("#left")	chọn phần tử có tên là left
.<tên lớp>	\$(".tensp")	chọn các phần tử có class là tensp
<tên thẻ>.<tên lớp>	\$("#li.app")	Chọn tất cả các thẻ có tên thẻ là <tên thẻ> với thuộc tính class có giá trị là <tên lớp>
*	\$("#*")	Chọn tất cả các element trên document

##### Bộ chọn các phần tử của form:

Selector	Ý nghĩa
:input	Chọn tất cả thẻ input, textarea trên Form
:text	Chọn tất cả text field trên Form

:password	Chọn tất cả password field
:radio	Chọn tất cả radio button
:checkbox	Chọn tất cả checkbox
:submit	Chọn tất cả button submit
:reset	Chọn tất cả button reset
:image	Chọn tất cả image
:button	Chọn tất cả generalized button
:file	Chọn tất cả control upload file
:checked	Chọn tất cả checkbox có check
:unchecked	Chọn tất cả checkbox không check
:blank	Chọn tất cả các ô nhập để trống

### c. Bộ lọc Filter

Filter luôn được dùng để lọc các thẻ trên một selector nào đó.

#### Bộ lọc Filter cơ bản

FILTER	Ý nghĩa
:first	Chọn phần tử đầu tiên trong tập kết quả do Selector trả về
:last	Chọn phần tử cuối cùng trong tập kết quả do Selector trả về
:even	Chọn phần tử chẵn
:odd	Chọn phần tử lẻ
:eq ( index )	Chọn phần tử tại vị trí index
:gt ( index )	Chọn phần tử có vị trí > index
:lt ( index )	Chọn phần tử có vị trí < index
:header	Chọn tất cả header element (H1, H2, .. H6)
:not ( selector )	Chọn phần tử không thỏa selector

#### Bộ lọc Filter thuộc tính

FILTER	Ý nghĩa
[attribute]	Lọc các phần tử có khai báo attribute
[attribute=value]	Lọc các phần tử có attribute với giá trị = value
[attribute!=value]	Lọc các phần tử có attribute với giá trị != value
[attribute^=value]	Lọc các phần tử có attribute với giá trị bắt đầu là value
[attribute\$=value]	Lọc các phần tử có attribute với giá trị kết thúc là value
[attribute*=value]	Lọc các phần tử có attribute chứa giá trị value

#### Bộ lọc Filter nội dung

FILTER	Ý nghĩa
:contains(text)	Lọc các phần tử có chứa chuỗi text
:empty	Lọc các phần tử rỗng
:has(selector)	Lọc các phần tử có chứa ít nhất 1 element thỏa selector
:parent	Lọc các phần tử có ít nhất 1 con hoặc text (Lấy các thẻ không rỗng)
:visible	Lọc các phần tử có trạng thái là visible ( đang hiển thị )
:hidden	Lọc các phần tử có trạng thái hidden ( đang ẩn )

#### Bộ lọc Filter con cháu

FILTER	Ý nghĩa
:nth-child(index)	Lọc các phần tử theo vị trí so với cha của nó

:nth-child(even)	
:nth-child(odd)	
:nth-child(equation)	Lọc phần tử theo vị trí ( vị trí thỏa phương trình tham số ) so với cha của nó
:first-child	Lấy phần tử đầu tiên so với cha của nó
:last-child	Lấy phần tử cuối cùng so với cha của nó
:only-child	Lấy phần tử nếu phần tử này là con duy nhất so với cha của nó

#### d. Sự kiện (Event)

Các sự kiện thường gặp trong lập trình jQuery.

Tên sự kiện	Giải thích
blur()	Xảy ra khi ra khỏi đối tượng
change()	Xảy ra khi giá trị bị thay đổi
click()	Xảy ra khi click vào đối tượng
contextmenu()	Xảy ra khi click chuột phải
dblclick()	Xảy ra click double chuột
delegate()	Bổ sung sự kiện vào đối tượng cả trước và sau khi thêm bằng Javascript
die()	Xóa bỏ sự kiện ra khỏi đối tượng
error()	Xảy ra khi xuất hiện lỗi trên đối tượng
focus()	Xảy ra khi focus vào đối tượng (con trỏ chuột đang xử lý tại đối tượng)
focusin()	Giống focus nhưng bổ sung thêm điều kiện là sự kiện đang ở trạng thái mới vào
focusout()	Giống focus như bổ sung thêm điều kiện là sự kiện đang ở trạng thái dừng
hover()	Xảy ra khi hover chuột vào đối tượng
keydown()	Xảy ra khi bàn phím nhấn xuống
keypress()	Xảy ra khi bàn phím nhấn xuống
keyup()	Xảy ra khi nhả bàn phím
load()	Xảy ra khi đối tượng bắt đầu load

Tên sự kiện	Giải thích
mousedown()	Xảy ra khi nhấn chuột trái xuống
mouseup()	Xảy ra khi nhả chuột trái ra
mouseenter()	Xảy ra khi con trỏ chuột đi vào phạm vi của đối tượng
mouseleave()	Xảy ra khi con trỏ chuột đi ra ngoài phạm vi của đối tượng
mousemove()	Xảy ra khi con trỏ chuột đang di chuyển bên trong đối tượng
mouseover()	Xảy ra một lần duy nhất khi con trỏ chuột bắt đầu đi vào phạm vi đối tượng
mouseout()	Xảy ra một lần duy nhất khi con trỏ chuột đi ra ngoài phạm vi đối tượng
ready()	Xảy ra khi browser đã load xong
resize()	Xảy ra khi resize browser
scroll()	Xảy ra khi kéo thanh cuộn
submit()	Xảy ra khi form được submit

### preventDefault()

Đây là một phương thức của event Object ở trên. Nó không có tham số, dùng để ngăn chặn hành vi mặc định của sự kiện.

```
$(document).ready( function(){
    $(window).contextmenu( function( e ){
        //code ở đây
        e.preventDefault();
    });
});
```

### e. Hàm

#### Xử lý tập hợp

Method	Ý nghĩa
size()	Lấy số phần tử trong tập kết quả của selector
get()	Lấy tập DOM elements trong tập kết quả của selector
get(index)	Lấy DOM element ở vị trí index
find(expression)	Lấy các element con cháu thỏa mãn expression
each()	Gọi thực thi phương thức với từng element trong tập kết quả của selector

#### Thao tác nội dung thẻ

Method	Ý nghĩa
html()	Lấy nội dung html bên trong element đầu tiên thỏa selector var html = \$("#basic").html();
html( newContent )	Thay đổi nội dung html bên trong mọi element thỏa selector ( tương tự innerHTML trong DOM ) \$("#app").html("<b>Hello</b>");
text()	Lấy nội dung text bên trong element đầu tiên var Text = \$("#basic").text();
text(value)	Thay đổi nội dung text bên trong mọi element thỏa selector (tương tự innerText) \$("#app").html("<b>Hello</b>");
append(content)	Chèn content vào sau nội dung có sẵn của các element thỏa selector
appendTo(selector)	Chèn element thỏa selector vào sau nội dung có sẵn của các element thỏa selector tham số
prepend(content)	Chèn content vào trước nội dung có sẵn của các element thỏa selector
prependTo(selector)	Chèn element thỏa selector vào trước nội dung có sẵn của các element thỏa selector tham số
after(content)	Chèn content vào sau các element thỏa selector
before(content)	Chèn content vào trước các element thỏa selector

### Thao tác thuộc tính thẻ

Method	Ý nghĩa
attr(name)	Lấy giá trị một thuộc tính v = \$("#txtId").attr("value")
attr(name, value)	Thiết lập giá trị một thuộc tính \$("#txtId").attr("value", "Hello")
attr(properties)	Thay đổi giá trị nhiều thuộc tính. \$("#txtId").attr({"value": "Hello", "disabled": "true"})
removeAttr(name)	Xóa thuộc tính. \$("#txtId").removeAttr("disabled")
val([value])	Viết tắt của attr("value") để thao tác thuộc tính các phần tử form \$("#txtId").val("Hello"); V=\$("#txtId").val();

### Các thuộc tính và hàm CSS

Method	Ý nghĩa
css (name)	Lấy giá trị một thuộc tính css w = \$("#img").css("width");
css (properties)	Thay đổi giá trị của nhiều thuộc tính css \$("#div:first").css({ "text-align": "center", "background-image": "url(bg.gif)" })
css (property,value)	Thay đổi giá trị một thuộc tính css \$("#tr:odd").css("background-color": "red")
addClass (class)	Thêm class vào các element thỏa selector
hasClass (class)	Kiểm tra class có tồn tại trong các element thỏa selector
removeClass(class)	Xóa class khỏi các element thỏa selector
toggleClass (class)	Thêm class vào các element thỏa selector nếu class chưa khai báo, ngược lại nếu đã tồn tại rồi, class sẽ bị xóa



### Thao tác kích thước

Method	Ý nghĩa
height ()	Lấy chiều cao của element đầu tiên thỏa selector H=\$( "#MyImage" ).height()
width ()	Lấy chiều rộng của element đầu tiên thỏa selector W=\$( "#MyDiv" ).width()
height (val)	Thiết lập chiều cao của mọi element thỏa selector \$( "#MyImage" ).height(300);
width (val)	Thiết lập chiều rộng của mọi element thỏa selector \$( "#MyImage" ).width(400)

### f. Hiệu ứng

Method	Ý nghĩa
show ()	Hiển thị các element thỏa selector ngay tức thì \$( ".abc" ).show();
show(speed, callback)	Hiển thị các element thỏa selector nếu trước đó bị ẩn, <ul style="list-style-type: none"> <li>✓ speed xác định tốc độ hiển thị.</li> <li>✓ callback phương thức sẽ được thực thi khi hiệu ứng thực hiện xong.</li> </ul> \$( "#div1" ).show("normal", function(){ alert("Hello") });
hide ()	Ẩn element nếu trước đó đang hiển thị.
hide(speed, callback)	Ẩn element nếu trước đó đang hiển thị, tham số có ý nghĩa tương tự phương thức show(). \$( "#div1" ).hide("slow"); \$( "#div1" ).hide(4000);
toggle ()	Chuyển qua lại trạng thái ẩn/hiện các element.
toggle(speed, callback)	Chuyển qua lại trạng thái ẩn/hiện các element, tham số có ý nghĩa tương tự phương thức show(). \$( "#div1" ).toggle("fast");
\$(selector).fadeIn(speed, callback);	Hiển thị rõ dần với tốc độ speed ( <b>speed</b> có thể là các chuỗi "slow", "fast", hay số mili giây như 1000, 2000,...)
\$(selector).fadeOut(speed, callback);	Hiển thị mờ dần với tốc độ speed ( <b>speed</b> có thể là các chuỗi "slow", "fast", hay số mili giây như 1000, 2000,...)
\$(selector).fadeToggle(speed, callback);	Hiển thị mờ dần hay rõ dần với tốc độ speed ( <b>speed</b> có thể là các chuỗi "slow", "fast", hay số mili giây như 1000, 2000,...)
\$(selector).fadeTo(speed, opacity, callback);	Hiển thị phần tử <b>selector</b> theo độ mờ (opacity) cho trước với tốc độ <b>speed</b> . <b>speed</b> có thể là các chuỗi "slow", "fast", hay số mili giây như 1000, 2000,...; <b>opacity</b> có giá trị từ 0 đến 1

### 2.3.3 Các thành phần giao diện jQueryUI

#### a. Liên kết thư viện JQueryUI

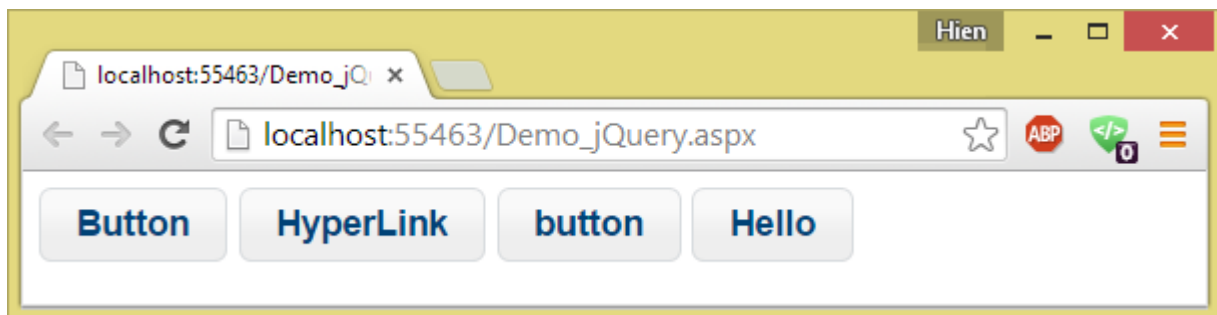
Trước khi viết mã tạo các thành phần giao diện, bạn phải chỉ ra thư viện JQuery bằng 2 liên kết JavaScript và một liên kết css sau:

```
<head runat="server">
  <title></title>
  <script src="jQuery/external/jquery/jquery.js"></script>
  <script src="jQuery/jquery-ui.js"></script>
  <link href="jQuery/jquery-ui.css" rel="stylesheet" />
</head>
```

## b. JQuery button

Để sử dụng kiểu dáng button của JQuery bạn chỉ cần gọi hàm button trên control (nút, liên kết) bạn muốn.

Ví dụ sau đây sẽ tạo kiểu dáng button cho cả <a>, <asp:HyperLink>, <input type='button'> và <asp:Button>



Mã nguồn đoạn nhúng:

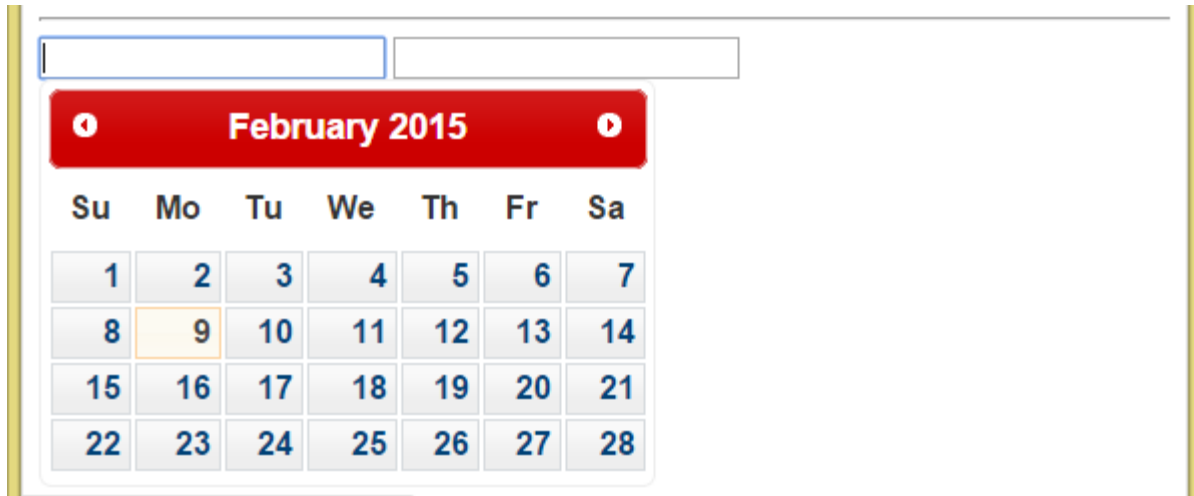
```
<script>
  $(function () {
    $(".my-button").button();
  });
</script>
```

```
<asp:Button CssClass="my-button" ID="Button1" runat="server" Text="Button" />
<asp:HyperLink CssClass="my-button" ID="lnk"
runat="server">HyperLink</asp:HyperLink>
<input class="my-button" id="Button2" type="button" value="button" />
<a class="my-button" href="">Hello</a>
```

## c. JQuery Datepicker

Thành phần giao diện này cho phép nhập ngày tháng bằng cách chọn từ bảng lịch xổ xuống. Bạn cũng có thể định dạng ngày nhận được tùy thích cũng như hạn chế ngày nhập vào nhờ vào các tùy chọn.

Ví dụ sau minh họa nhập ngày đơn giản:



Mã nguồn đoạn nhúng:

```
<script>
    $(function () {
        $(".date").datepicker();
    });
</script>
<asp:TextBox CssClass="date" ID="TextBox" runat="server" />
<input class="date" id="Button2" type="text" />
</div>
```

Nếu bạn muốn định dạng ngày nhận được thì chỉ cần thêm tùy chọn ngày vào hàm datepicker như sau:

```
<script>
    $(function () {
        $(".date").datepicker({dateFormat: 'dd-mm-yy'});
    });
</script>
```

Các ký tự định dạng:

- ✓ y: 2 chữ số năm
- ✓ m: một chữ số tháng
- ✓ d: một chữ số ngày



Nếu bạn muốn hạn chế ngày nhập vào thì thêm tham số `minDate` và `maxDate`

```
<script>
    $(function () {
        $(".date").datepicker({dateFormat: 'yy-mm-dd', minDate: -2, maxDate: "+1M
+5D" });
    });
</script>
```

Tham số `minDate` qui định ngày tối thiểu được phép chọn còn `maxDate` qui định ngày tối đa được chọn.

Trong ví dụ này chúng ta hiểu ngày tối thiểu là trước ngày hiện tại 2 ngày, còn ngày tối đa là sau ngày hiện tại 1 tháng 5 ngày.

Các ký tự định dạng:

- ✓ D: ngày
- ✓ W: tuần
- ✓ M: tháng
- ✓ Y: năm

Nếu thêm dấu `-` thì hiểu là trước ngày hiện tại và dấu `+` thì sau ngày hiện tại.

#### d. JQuery Tabs

Thành phần giao diện này rất quen thuộc với chúng ta. Mỗi tab gồm 2 phần là tiêu đề tab và nội dung tab. Nội dung tab có thể là một phần mã HTML cùng trang hoặc nội dung của một trang web khác được tải theo cơ chế Ajax.

Ví dụ sau đây chúng ta tạo ra giao diện 3 tab. Tab1 và Tab2 chỉ đến một phần HTML cùng trang còn Tab3 sẽ tải nội dung của trang Tab3.aspx.

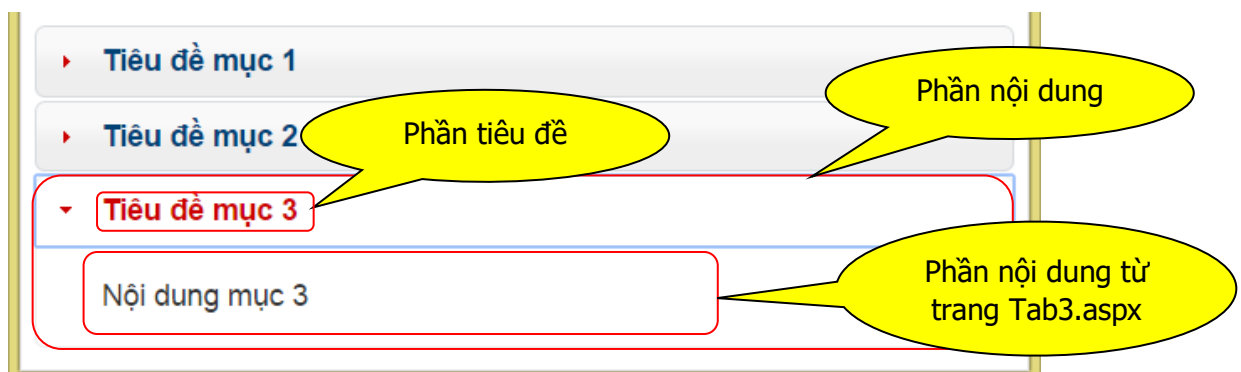


Mã nguồn đoạn nhúng:

```
<div id="mytabs">
  <ul>
    <li><a href="#Tab1">Tab 1</a></li>
    <li><a href="#Tab2">Tab 2</a></li>
    <li><a href="Tab3.aspx">Tab 3</a></li>
  </ul>
  <div id="Tab1">Nội dung Tab 1</div>
  <div id="Tab2">Nội dung Tab 2</div>
</div>
<script>
  $(function () {
    $("#mytabs").tabs();
  });
</script>
```

### e. JQuery Accordion

Thành phần giao diện này cũng gần giống như tabs. Mỗi mục gồm 2 phần là tiêu đề và nội dung. Tại một thời điểm chỉ hiển thị một mục. Tuy nhiên công dụng của thành phần này thường để chứa các menu chức năng để tiết kiệm diện tích trang web.



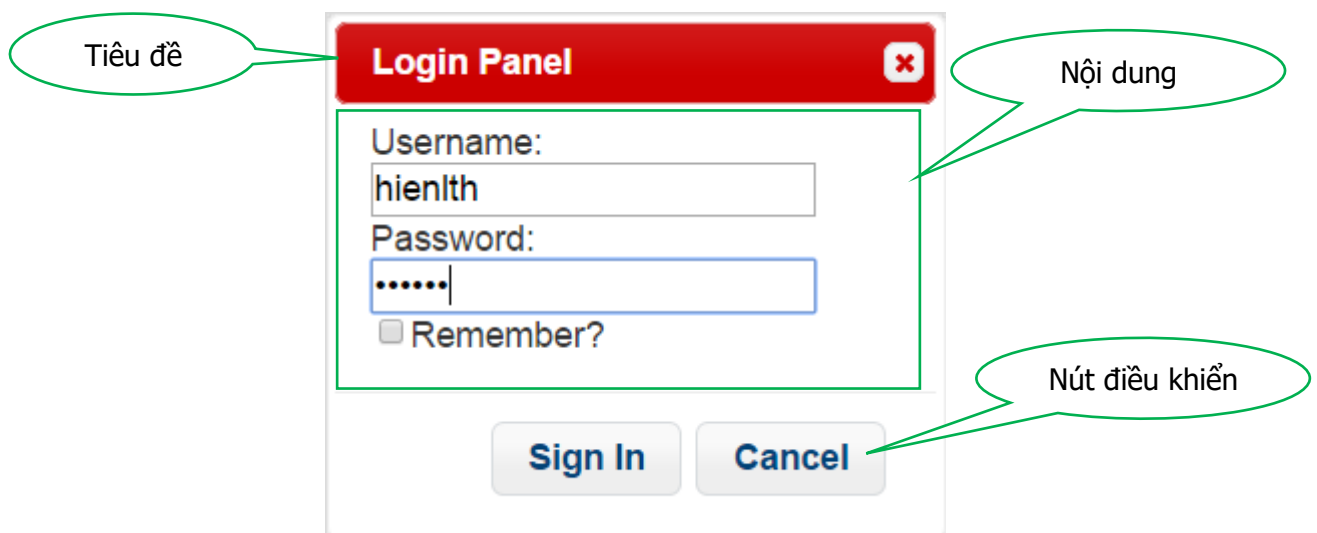
Mã nguồn đoạn nhúng:

```
<script>
$(function () {
    $(".my-acco").accordion();
});
</script>

<div class="my-acco">
<h3>Tiêu đề mục 1</h3>
<div>Nội dung mục 1</div>
<h3>Tiêu đề mục 2</h3>
<div>Nội dung mục 2</div>
<h3>Tiêu đề mục 3</h3>
<div>Nội dung mục 3</div>
</div>
```

## f. JQuery Dialog

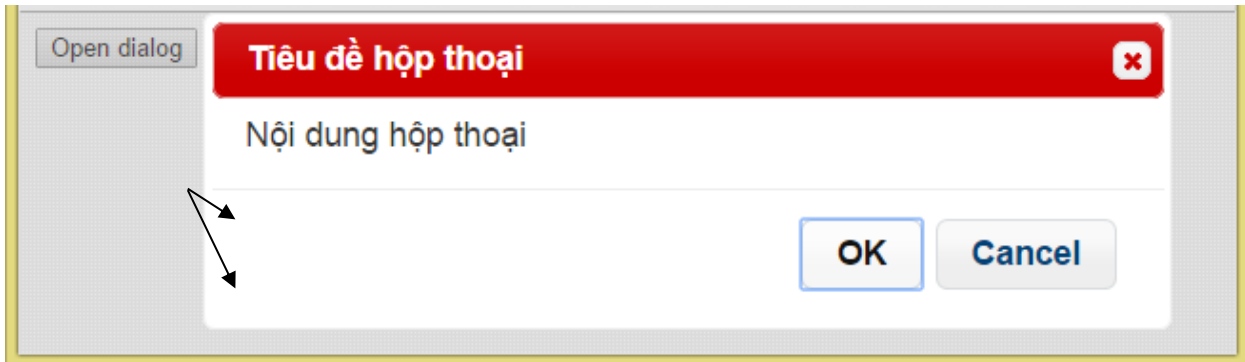
Thành phần này dùng để tạo hộp thoại tương tác với người dùng. Mỗi hộp thoại có tiêu đề, nội dung và thậm chí có cả hệ thống nút xử lý tương tác.



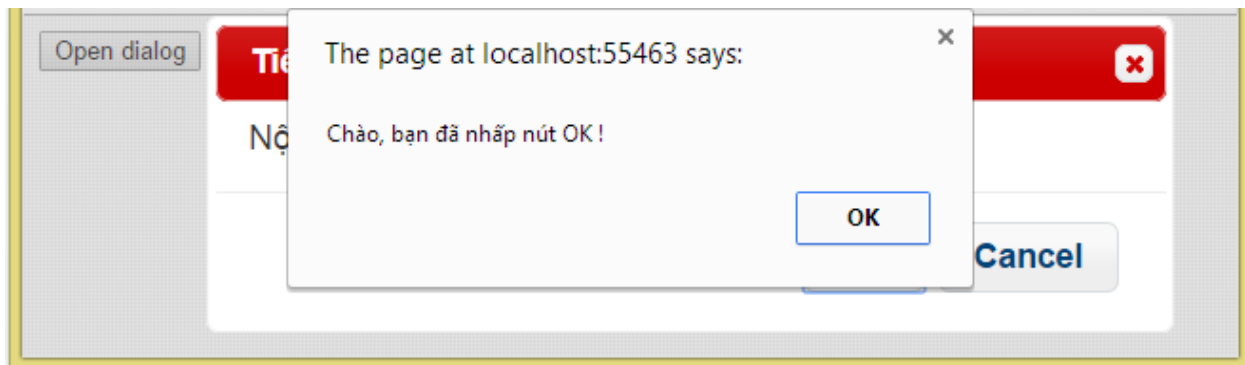
Có 3 công việc chính khi viết mã JQuery làm việc với hộp thoại là:

- ✓ Định nghĩa hộp thoại
- ✓ Mở hộp thoại
- ✓ Xử lý tương tác các nút điều khiển

Ví dụ sau đây cho thấy điều đó:



Sau khi nhấp nút Cancel hoặc dấu chéo góc trên-phải thì hộp thoại được đóng lại. Còn nếu nhấp vào nút OK sẽ nhận được thông báo



Mã nguồn đoạn nhúng:

```
<script>
    $(function () {
        // công việc 1: định nghĩa hộp thoại
        $(".my-dialog").dialog({
            width: 500,
            autoOpen: false,
            modal: true,
            show: 'blind',
            hide: 'explode',
            buttons: {
                "OK": function () {
                    // công việc 3: xử lý tương tác nút sự kiện
                    alert("Chào, bạn đã nhấp nút OK !");
                },
                "Cancel": function () {
                    // công việc 3: xử lý tương tác nút sự kiện
                    $(".my-dialog").dialog("close");
                }
            }
        });
    });

    // công việc 2: mở hộp thoại
    $(".my-button").click(function () {
        $(".my-dialog").dialog("open");
    });
</script>
```

```
});
</script>

<div class="my-dialog" title="Tiêu đề hộp thoại">
    Nội dung hộp thoại
</div>

<input class="my-button" id="Button1" type="button" value="Open dialog" />
```

Qua ví dụ trên ta thấy công việc 1: định nghĩa hộp thoại có cú pháp sau

```
$(".my-dialog").dialog({thiết lập các tùy chọn});
```

Trong trường hợp này chúng ta sử dụng các tùy chọn với ý nghĩa sau

Tùy chọn	Mô tả
width	Định nghĩa chiều rộng của hộp thoại
autoOpen	Giá trị false sẽ không mở hộp thoại khi định nghĩa. Ngược lại giá trị true sẽ mở hộp thoại lúc định nghĩa.
modal	Không cho tương tác lên trang web khi hộp thoại hiện ra nếu là true. Ngược lại sẽ cho tương tác với cửa sổ mẹ.
show	Chỉ ra hiệu ứng lúc mở hộp thoại
hide	Chỉ ra hiệu ứng lúc đóng hộp thoại
buttons	Định nghĩa hệ thống nút điều khiển của hộp thoại

## 2.4 Bootstrap

### 2.4.1 Giới thiệu

Bootstrap là một CSS framework phổ biến nhất hiện nay do Twitter phát triển. Đây là công cụ giúp design trang web bằng css nhanh chóng nhờ các class có sẵn và các thành phần có trên trang web như form, navbar, tooltip, dropdown-menu, modal, button... Một trong tính năng quan trọng của Bootstrap là hỗ trợ Responsive – tự động co giãn theo kích thước màn hình trình duyệt.

Để sử dụng, download phiên bản mới nhất về máy và bỏ vào website từ trang <https://getbootstrap.com/>. Phiên bản hiện tại tính đến tháng 4/2018 là Bootstrap 4.1. Thường khi tạo ứng dụng .NET Core MVC thì project đã cung cấp sẵn bootstrap.

### 2.4.2 Hệ thống lưới – Grid System

Hệ thống lưới trong Bootstrap được chia thành các hàng (row) và các cột (column). Cụ thể, layout Bootstrap sẽ được chia thành 12 cột. Hệ thống lưới hoạt động hiệu quả khi đặt trong class **.container** (chiều rộng cố định) hoặc **.container-fluid** (chiều rộng full màn hình).

```
<div class="row">
    <div class="col-sm-2">Left sidebar</div>
    <div class="col-sm-8">Content</div>
    <div class="col-sm-2">Right sidebar </div>
</div>
<div class="row">
    <div class="col">.col</div>
    <div class="col">.col</div>
```



```
<div class="col">.col</div>
</div>
```

Ở ví dụ trên, layout của chúng ta gồm 2 hàng. Hàng thứ nhất có 3 cột với tỉ lệ 2-8-2, hàng thứ 2 cũng 3 cột với tỉ lệ đều nhau (4-4-4). Cú pháp đặt class cột: **.col-a-b** hoặc **.col**

Class	Thiết bị	Mô tả
<b>.col-</b>	extra small devices	Kích thước màn hình nhỏ hơn 576px
<b>.col-sm-</b>	small devices	Kích thước màn hình lớn hơn hay bằng 576px
<b>.col-md-</b>	medium devices	Kích thước màn hình lớn hơn hay bằng 768px
<b>.col-lg-</b>	large devices	Kích thước màn hình lớn hơn hay bằng 992px
<b>.col-xl-</b>	xlarge devices	Kích thước màn hình lớn hơn hay bằng 1200px

### 2.4.3 Định dạng cơ bản

#### a. Typography:

Cách trình bày như bản in gồm Heading, căn lề và text, màu chữ, màn nền, ...

#### b. Text Color

Màu chữ gồm các bộ **.text-muted**, **.text-primary**, **.text-success**, **.text-info**, **.text-warning**, **.text-danger**, **.text-secondary**, **.text-white**, **.text-dark**, **.text-body** và **.text-light**.

```
<p class="text-muted">This text is muted.</p>
<p class="text-primary">This text is important.</p>
<p class="text-success">This text indicates success.</p>
<p class="text-info">This text represents some information.</p>
<p class="text-warning">This text represents a warning.</p>
<p class="text-danger">This text represents danger.</p>
<p class="text-secondary">Secondary text.</p>
<p class="text-dark">Dark grey text.</p>
<p class="text-body">Body color.</p>
```

This text is muted.

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Secondary text.

Dark grey text.

Body text.

### c. Background Color

Màu nền gồm các bộ class: `.bg-primary`, `.bg-success`, `.bg-info`, `.bg-warning`, `.bg-danger`, `.bg-secondary`, `.bg-dark` và `.bg-light`

This text is important.

This text indicates success.

This text represents some information.

This text represents a warning.

This text represents danger.

Secondary background color.

Dark grey background color.

Light grey background color.

### d. Button

Button cơ bản trong Bootstrap sử dụng class `.btn`, ngoài ra bổ sung các class khác: `.btn-primary`, `.btn-secondary`, `.btn-success`, `.btn-info`, `.btn-warning`, `.btn-danger`, `.btn-dark`, `.btn-light`, `.btn-link`.

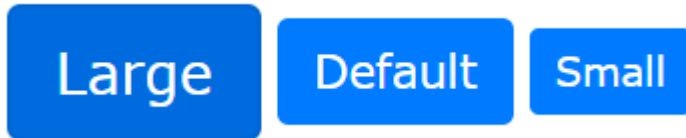
Basic Primary Secondary Success Info Warning Danger Dark Light Link

Bootstrap 4 còn cung cấp 8 class cho button dạng outline/border gồm: `.btn-outline-primary`, `.btn-outline-secondary`, `.btn-outline-success`, `.btn-outline-info`,

`.btn-outline-warning`, `.btn-outline-danger`, `.btn-outline-dark`, `.btn-outline-light`



Về kích thước của button, sử dụng thêm các class: `.btn-lg`, `.btn-sm`



```
<button type="button" class="btn btn-primary btn-lg">Large</button>
<button type="button" class="btn btn-primary">Default</button>
<button type="button" class="btn btn-primary btn-sm">Small</button>
```

### e. Table

Sử dụng class cơ bản nhất là `.table`, ngoài ra còn sử dụng các class khác như `.table-striped` (kiểu hàng sọc, các hàng chẵn lẻ xem kẽ), `.table-bordered` đóng khung, `.table-hover` thêm hiệu ứng hover lên từng dòng, `.table-dark` nền tối.

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

### f. Hình ảnh

Sử dụng class cơ bản nhất là `.rounded`, `.rounded-circle`, `.img-thumbnail` để có hình ảnh như ý muốn.

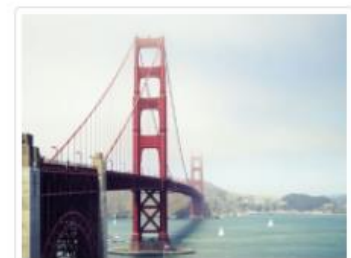
Rounded Corners:



Circle:



Thumbnail:



### g. Alert

Sử dụng class cơ bản `.alert` kết hợp với các class khác như `.alert-success`, `.alert-info`, `.alert-warning`, `.alert-danger`, `.alert-primary`, `.alert-`

`secondary`, `.alert-light`, `.alert-dark` để có màu chữ + background phù hợp ngữ cảnh thông báo.

<b>Success!</b> This alert box indicates a successful or positive action.	×
<b>Info!</b> This alert box indicates a neutral informative change or action.	×
<b>Warning!</b> This alert box indicates a warning that might need attention.	×
<b>Danger!</b> This alert box indicates a dangerous or potentially negative action.	×
<b>Primary!</b> This alert box indicates an important action.	×
<b>Secondary!</b> This alert box indicates a less important action.	×
<b>Dark!</b> Dark grey alert box.	×
<b>Light!</b> Light grey alert box.	×

```
<div class="alert alert-success alert-dismissible">
  <button type="button" class="close" data-dismiss="alert">&times;</button>
  <strong>Success!</strong> Indicates a successful or positive action.
</div>
```

#### 2.4.4 Form

Thêm thuộc tính class `.form-control` để có width 100%.

Email:

Password:

Remember me

```
<form>
  <div class="form-group">
    <label for="email">Email address:</label>
    <input type="email" class="form-control" id="email">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control" id="pwd">
  </div>
</form>
```

```
</div>
<div class="form-group form-check">
  <label class="form-check-label">
    <input class="form-check-input" type="checkbox"> Remember me
  </label>
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Thêm thuộc tính class `.form-inline` vào thẻ form để các phần tử trên một dòng.

Email:  Password:   Remember me

```
<form class="form-inline" action="">
  <label for="email">Email address:</label>
  <input type="email" class="form-control" id="email">
  <label for="pwd">Password:</label>
  <input type="password" class="form-control" id="pwd">
  <div class="form-check">
    <label class="form-check-label">
      <input class="form-check-input" type="checkbox"> Remember me
    </label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Bootstrap đã định dạng sẵn css cho các thẻ input : input, textarea, checkbox, radio, select

Muốn checkbox lên cùng một dòng, thêm thuộc tính `.form-check-inline` vào khối div bao quanh nó.

Option 1  Option 2  Option 3

```
<div class="form-check form-check-inline">
  <label class="form-check-label">
    <input type="checkbox" class="form-check-input" value="">Option 1
  </label>
</div>
<div class="form-check form-check-inline">
  <label class="form-check-label">
    <input type="checkbox" class="form-check-input" value="">Option 2
  </label>
</div>
<div class="form-check form-check-inline disabled">
  <label class="form-check-label">
    <input type="checkbox" class="form-check-input" value="">Option 3
  </label>
</div>
```

Tương tự thêm thuộc tính class `.radio-inline` cho để các radio nằm trên 1 dòng.

Option 1 Option 2 Option 3

```
<label class="radio-inline"><input type="radio" name="optradio">Option  
1</label>  
<label class="radio-inline"><input type="radio" name="optradio">Option  
2</label>  
<label class="radio-inline"><input type="radio" name="optradio">Option  
3</label>
```

Các ví dụ khác về JS Tab, JS Dropdown, JS Carousel, JS Modal xem thêm tại  
<https://www.w3schools.com/bootstrap4/>

## Chương 3: **TỔNG QUAN VỀ ASP.NET CORE 2.2**

Sau khi học xong bài này, học viên có khả năng :

- Trình bày được kiến trúc ứng dụng WEB
- Nắm được đặc điểm của ứng dụng web ASP.NET Core 2.2
- Tạo ứng dụng với ASP.NET Core 2.2

### 3.1 Giới thiệu Ứng dụng Web

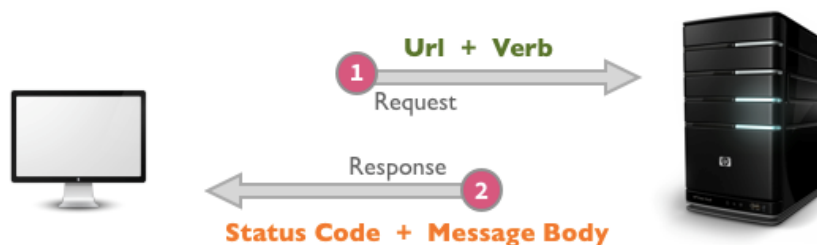
#### 3.1.1 Giới thiệu

Hàng ngày chúng ta mở máy tính, vào các trang web nổi tiếng như Google, Yahoo để tìm kiếm, đọc và gửi email. Chúng ta cũng thường vào các trang báo để đọc tin tức; các trang đào tạo trực tuyến để học hành; các trang bán hàng để tìm và mua hàng; tham gia vào diễn đàn để tranh luận. Tất cả rất tuyệt vời và thật sự có ý nghĩa với cuộc sống mỗi một con người trong thời đại internet của chúng ta.

Sự phát triển mạnh mẽ về công nghệ thông tin, đặc biệt là internet, nhiều lĩnh vực ngày nay như : thương mại, y tế, giáo dục..., nhu cầu trao đổi thông tin thực sự là cần thiết, giúp cho công việc được triển khai nhanh chóng, chính xác, dễ dàng và tiết kiệm chi phí, thông tin được cập nhật kịp thời. Do đó vấn đề đặt ra là chúng ta cần phải có một ứng dụng cho phép trao đổi thông tin mọi lúc, mọi nơi, dễ sử dụng,... thông qua mạng. Ứng dụng Web đáp ứng được các yêu cầu đặt ra và sau đây là các lý do tại sao chúng ta phải sử dụng Web :

- ✓ Dễ dàng trao đổi và chia sẻ thông tin thông tin qua mạng.
- ✓ Sử dụng giao diện đồ họa giúp cho người dùng dễ sử dụng.
- ✓ Hỗ trợ về multimedia như : hình ảnh, âm thanh, phim ảnh,...
- ✓ Hỗ trợ nhiều chương trình(web-browser) để truy cập Web.
- ✓ Hỗ trợ truy cập web trên các thiết bị di động: Tablet, SmartPhone,...
- ✓ Hỗ trợ nhiều ngôn ngữ để phát triển Web: ASP.NET, JSP, PHP, Python, ...

#### 3.1.2 Nguyên lý hoạt động



Hình 3-1: Nguyên tắc hoạt động

Để có được kết quả hiển thị của trang web yêu cầu, các bước thực hiện truyền thông xảy ra ở phía hậu cảnh bao gồm:

- ✓ Chuyển đổi "url" thành "ip"
- ✓ Gửi request đến Web Server

- ✓ Web Server thực hiện các xử lý cần thiết theo request
- ✓ Kết quả được response đến Browser
- ✓ Web Browser trình bày dữ liệu trên kết quả trả về và các thẻ markup

### 3.1.3 Các khái niệm

#### 3.1.3.1 Web client (Browser)

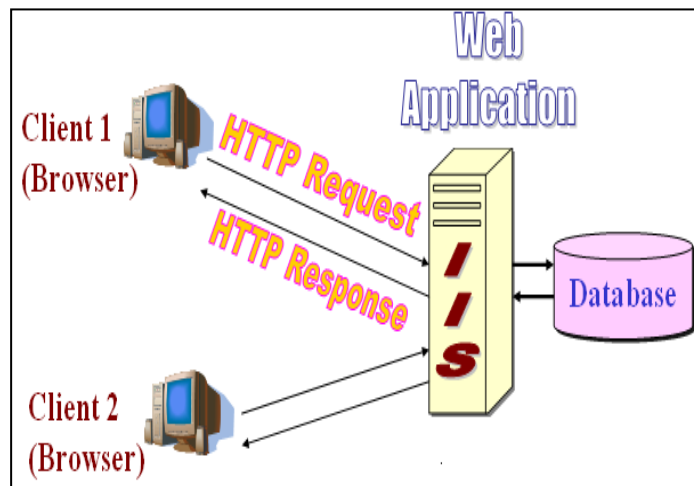
Máy khách (Client) thường là Web Browser sẽ sử dụng chương trình để truy cập đến các trang web gọi là trình duyệt web hay browser. Hiện rất nay có nhiều trình duyệt web như : Chrome, Edge, Opera, Mozilla FireFox, ...

#### 3.1.3.2 Web server

Các máy chủ (Server) chứa các ứng dụng Web, sẵn sàng truy xuất các trang web hay các tài liệu và gửi về cho client khi nhận được yêu cầu từ phía Client. Hiện nay có rất nhiều Web server và chạy trên nhiều hệ thống như : Apache, Nginx, Microsoft IIS, ...

#### 3.1.3.3 Giao thức HTTP

Quá trình giao tiếp giữa client và server được thực hiện thông qua giao thức chuẩn HTTP(HyperText Transfer Protocol). Hình minh họa sau mô tả việc truy cập ứng dụng Web.



Hình 3-2: Giao thức HTTP

- ✓ Web được phát triển trên mô hình client-server.
- ✓ Giao thức HTTP: Quá trình giao tiếp giữa client và server được thực hiện thông qua giao thức chuẩn HTTP (HyperText Transfer Protocol).
- ✓ Mô hình gồm hai thành phần chính là: máy khách(client) và máy phục vụ(server). Máy phục vụ(server) sẽ chứa các ứng dụng Web và các ứng dụng Web này sẽ được quản lý tập trung bởi trình quản lý gọi là Web Server (IIS,...). Các máy khách(client) truy cập đến ứng dụng web sử dụng trình duyệt web(browser).
- ✓ Client sử dụng giao thức HTTP Request để gửi yêu cầu (trang web) lên Server, Server xử lý và sử dụng giao thức HTTP Response để gửi kết quả về cho Client.

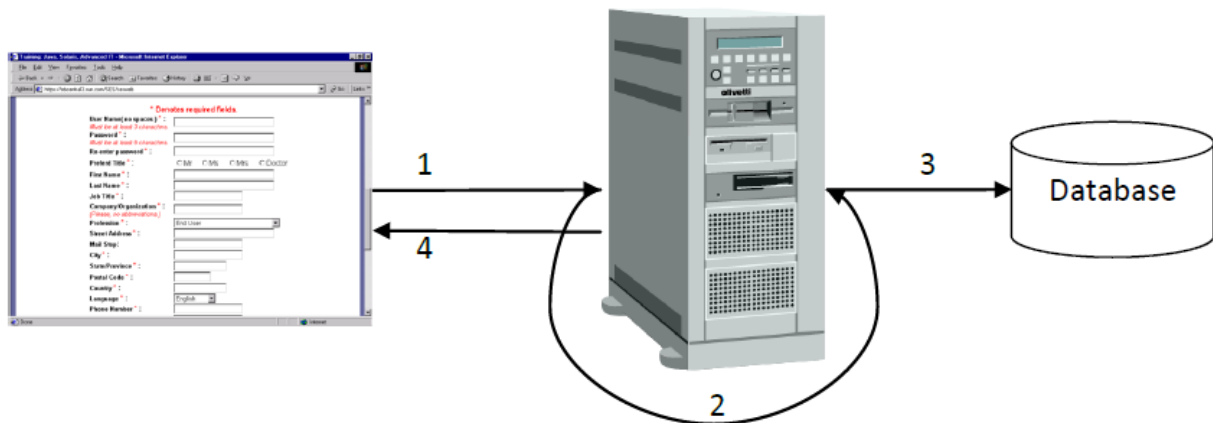
#### 3.1.3.4 Client Scripting và Server Scripting

Các ngôn ngữ dùng để viết mã cho trang web. Một trang web được xử lý ở Server và trả kết quả về cho Client. Do đó các ngôn ngữ viết mã cho trang web được chia thành hai dạng:



- ✓ **Client Scripting:** được xử lý tại trình duyệt (Browser) trên máy Client. Các ngôn ngữ dùng để viết là :VBScript, JavaScript, ...
  - JavaScript là ngôn ngữ phổ biến sử dụng nhiều nhất hiện nay. JavaScript được dùng để kiểm tra việc nhập liệu, kiểm tra trình duyệt,...
  - DHTML: là sự kết hợp của HTML, Style Sheet (CSS) và JavaScript nhằm làm cho trang web dễ tương tác, điều khiển và giảm bớt việc xử lý phía Server.
  - VBScript là ngôn ngữ script của Microsoft. Chức năng của VBScript cũng giống như JavaScript.
- ✓ **Server Scripting:** được xử lý tại Web server trên máy Server. Các ngôn ngữ dùng để viết là : ASP, ASP.NET, PHP, JSP, Python, ... Trong giáo trình này chúng ta sẽ khảo sát ngôn ngữ ASP.NET C#.

### 3.1.4 Kiến trúc công nghệ ứng dụng web



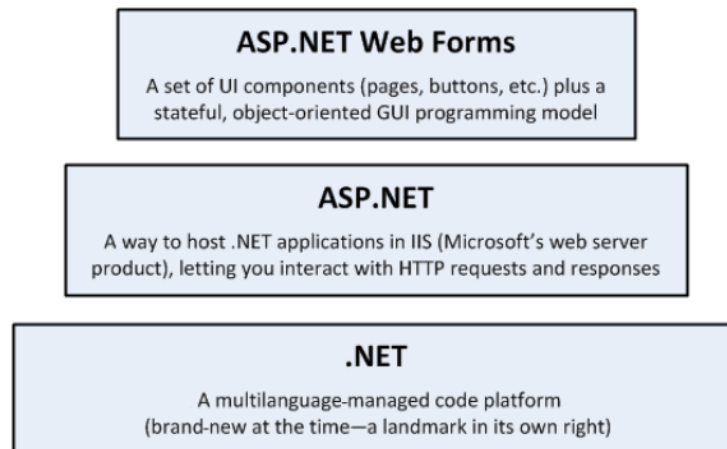
**Hình 3-3: Kiến trúc công nghệ ứng dụng Web**

Trình duyệt giúp người sử dụng giao tiếp với ứng dụng web cài đặt phía server. Phần mềm trung gian (ứng dụng web) này sẽ nhận và xử lý các yêu cầu của người sử dụng. Nếu ứng dụng cần truy vấn hay lưu trữ thông tin, nó sẽ kết nối với CSDL để được trợ giúp bởi các hệ quản trị CSDL.

1. Thông tin trên form chuyển đến Web Server thông qua request
2. Server thực hiện các xử lý được cài đặt (server script – Server Side)
3. Kết nối và thao tác CSDL
4. Kết quả xử lý được trả về qua response (HTML)

## 3.2 Tổng quan về ASP.NET Core MVC

### 3.2.1 Giới thiệu về ASP.NET



**Hình 3-4. The ASP.NET Web Forms technology stack**

Đầu năm 2002, Microsoft giới thiệu một kỹ thuật lập trình Web khá mới mẻ với tên gọi ban đầu là ASP+, tên chính thức sau này là ASP.Net. Với ASP.Net, không những không cần đòi hỏi bạn phải biết các tag HTML, thiết kế web, mà nó còn hỗ trợ mạnh lập trình hướng đối tượng trong quá trình xây dựng và phát triển ứng dụng Web. ASP.Net là kỹ thuật lập trình và phát triển ứng dụng web ở phía Server (Server-side) dựa trên nền tảng của Microsoft .Net Framework.

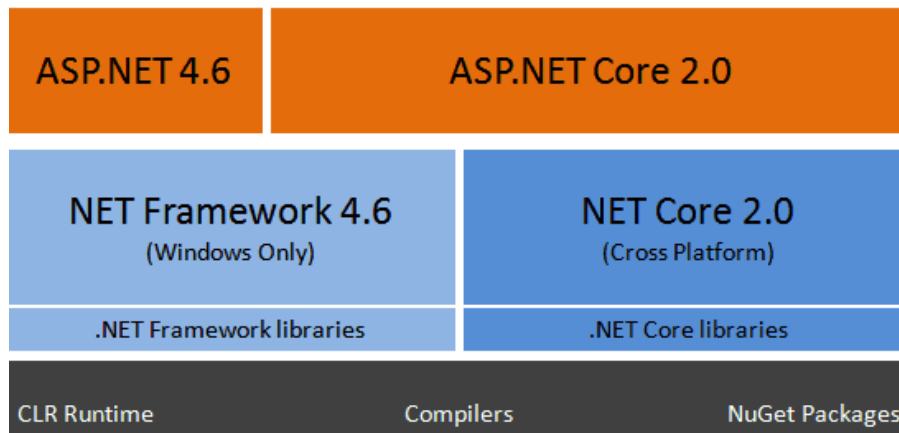
Hầu hết, những người mới đến với lập trình web đều bắt đầu tìm hiểu những kỹ thuật ở phía Client (Client-side) như: HTML, Java Script, CSS (Cascading Style Sheets). Khi Web browser yêu cầu một trang web (trang web sử dụng kỹ thuật client-side), Web server tìm trang web mà Client yêu cầu, sau đó gửi về cho Client. Client nhận kết quả trả về từ Server và hiển thị lên màn hình.

ASP.Net sử dụng kỹ thuật lập trình ở phía server thì hoàn toàn khác, mã lệnh ở phía server sẽ được biên dịch và thi hành tại Web Server. Sau khi được Server đọc, biên dịch và thi hành, kết quả tự động được chuyển sang HTML/JavaScript/CSS và trả về cho Client. Tất cả các xử lý lệnh ASP.Net đều được thực hiện tại Server và do đó, gọi là kỹ thuật lập trình ở phía server.

### 3.2.2 ASP.NET Core là gì?

ASP.NET Core là một Open-source mới và là Cross-Platform framework giúp xây dựng các ứng dụng web hiện đại dựa trên đám mây như là web apps, IoT apps, mobile backends. Các ứng dụng ASP.NET Core có thể chạy trên .NET Core hoặc trên .NET framewrok. ASP.NET Core cung cấp một kiến trúc để tối ưu hóa việc xây dựng các ứng dụng đám mây (Microsoft Azure) hoặc các ứng dụng độc lập. Nó bao gồm các Module thành phần với chi phí tối thiểu. Do đó bạn có thể giữ lại tính linh hoạt khi xây dựng ứng dụng của mình.

### ASP.NET Core in .NET Ecosystem



**Hình 3-5: Hệ sinh thái ASP.NET**

ASP.NET Core được module hóa, không còn dựa vào System.Web.dll mà dựa vào nhu cầu người dùng cần cài đặt các gói trên Nuget và các thành phần khác. Điều này giúp bạn optimize apps, chỉ lấy các gói Nuget mà bạn cần. Lợi ích của ứng dụng nhỏ bao gồm bảo mật chặt chẽ hơn, giảm dịch vụ không cần thiết, hiệu suất được cải thiện và giảm chi phí tương ứng với các module mà bạn sử dụng.

Với ASP.NET Core bạn sẽ có những cải tiến nền tảng sau:

- Hợp nhất việc xây dựng web UI và web APIs.
- Tích hợp các framework hiện đại cho client-side, quy trình phát triển hiện đại.
- Một hệ thống cấu hình dựa trên môi trường đám mây.
- Build-in dependency injection (DI).
- Gọn, nhẹ hơn, module hóa HTTP request.
- Khả năng lưu trữ trên IIS, Nginx, Apache, Docker hoặc máy chủ riêng của bạn.
- Build trên .NET Core, hỗ trợ từng bước một theo phiên bản.
- Đầy đủ các gói trên Nuget.
- Các công cụ mới hiện đại, đơn giản cho phát triển web.
- Xây dựng và chạy cross-platform trên Windows, Mac và Linux.
- Là mã nguồn mở nên có cộng đồng quan tâm lớn.

#### **Xây dựng Web UI và WebAPI sử dụng ASP.NET Core MVC.**

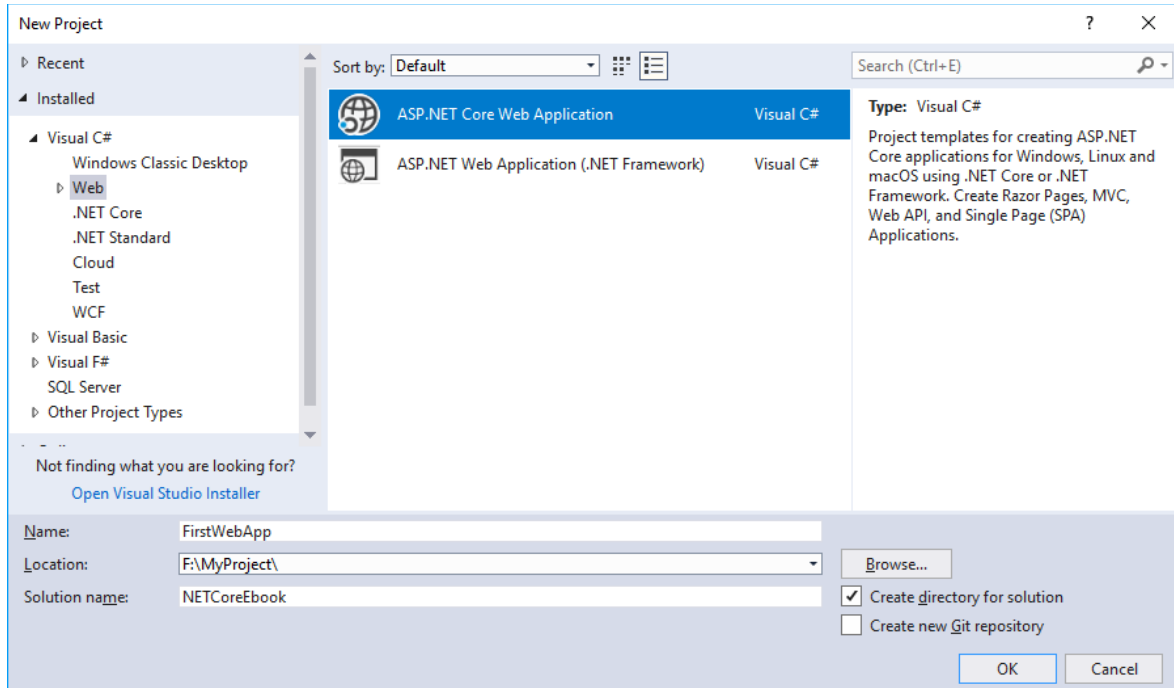
- Mô hình MVC (pattern) giúp dễ dàng kiểm thử cho web API và web Apps.
- Razor Page giúp xây dựng web UI dễ dàng và hiệu quả hơn.
- Trình trợ giúp thẻ cho phép mã phía máy chủ tham gia tạo và hiển thị các phần tử HTML trong các tệp Razor
- Tag Helpers cho phép viết code trên server-side sẽ tạo và hiển thị các phần tử HTML tương ứng ở Razor file.
- Hỗ trợ tích cực cho nhiều dạng dữ liệu (JSON, XML) cho phép web API tiếp cận nhiều khách hàng hơn bao gồm browsers và thiết bị di động.
- Mô hình Binding tự động ánh xạ dữ liệu từ các HTTP request đến các tham số trong phương thức hành động (action method).
- Mô hình kiểm tra (validation) tự động thực hiện xác thực cả client-side và server-side.

#### **Phát triển phía máy khách**

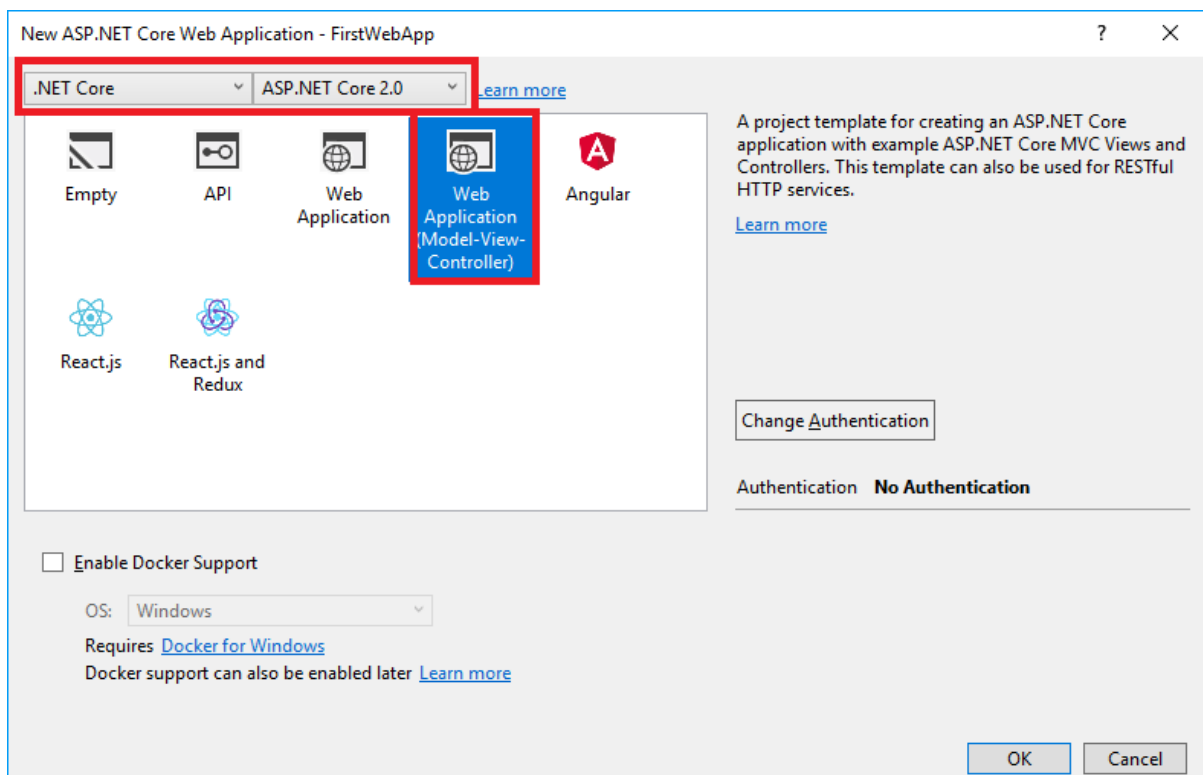
ASP.NET Core được thiết kế để tích hợp liền mạch với nhiều khuôn khổ phía khách hàng, bao gồm AngularJS, ReactJS, KnockoutJS và Bootstrap.

### 3.2.3 Tạo ứng dụng ASP.NET Core MVC

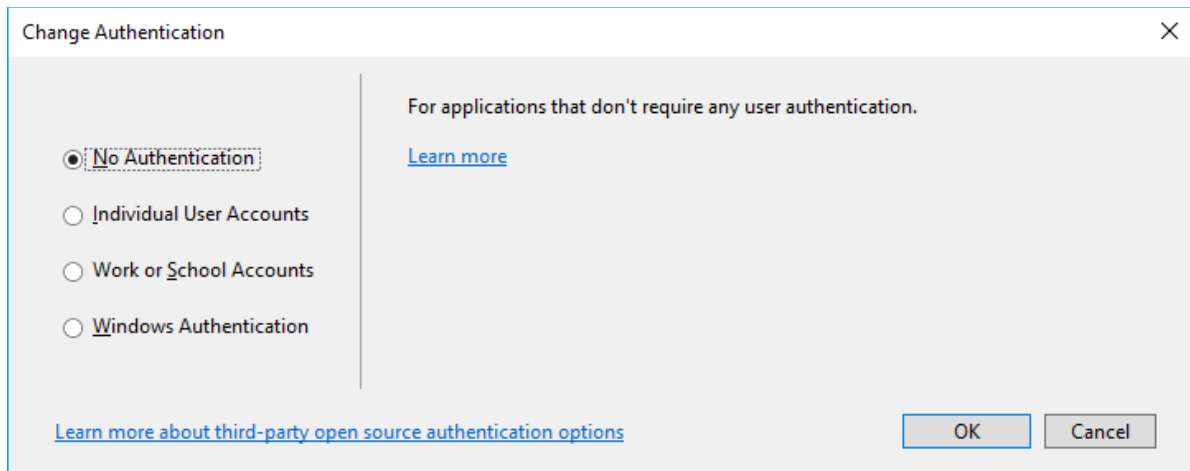
Vào menu **File** chọn **New Project**. Trong cửa sổ tạo mới chọn template **Web** và chọn **ASP.NET Core Web Application**. Điền thông tin project và bấm OK.



Chọn **.NET Core** và **ASP.NET Core 2.0**, template là **Web Application (Model-View-Controller)**.

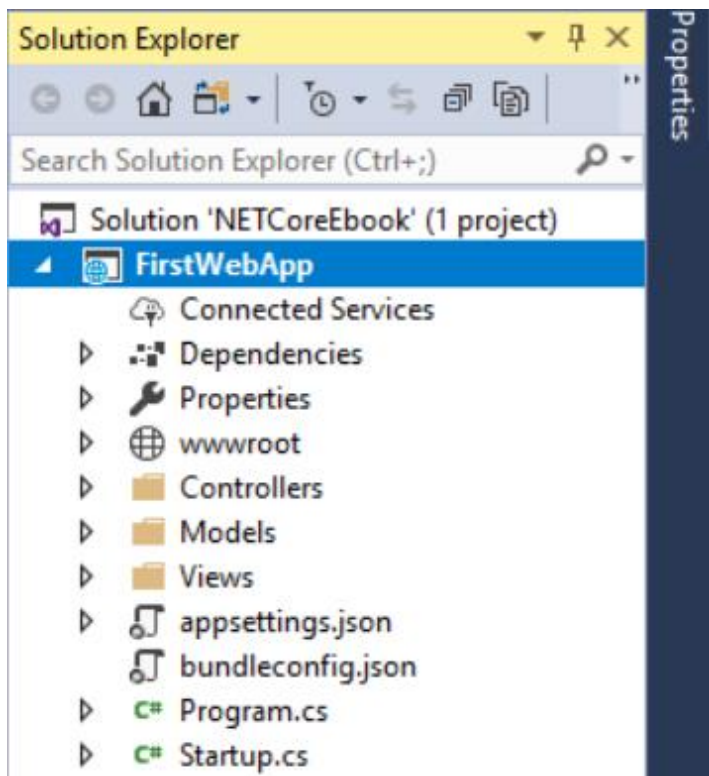


### Các loại Authentication:

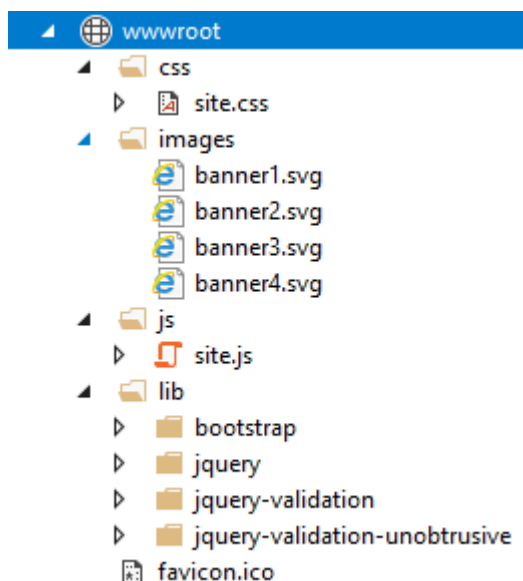


- **No Authentication** : Không bao gồm phần security, project phải tự thiết lập security riêng.
- **Individual User Accounts** : Project được tạo ra gồm có cả phần security với các chức năng như đăng nhập, đăng ký, đăng xuất, đổi mật khẩu, cho phép đăng nhập từ các ứng dụng bên ngoài.
- **Work or School Accounts** : dự án sử dụng tài khoản từ cloud computing. Ứng dụng này phù hợp với các doanh nghiệp lớn xuyên quốc gia.
- **Windows Authentication** : đây là dự án intranet bao gồm cả phần security nhưng tài khoản quản lý ở mạng nội bộ.

### Cấu trúc file và thư mục của project ASP.NET Core MVC:



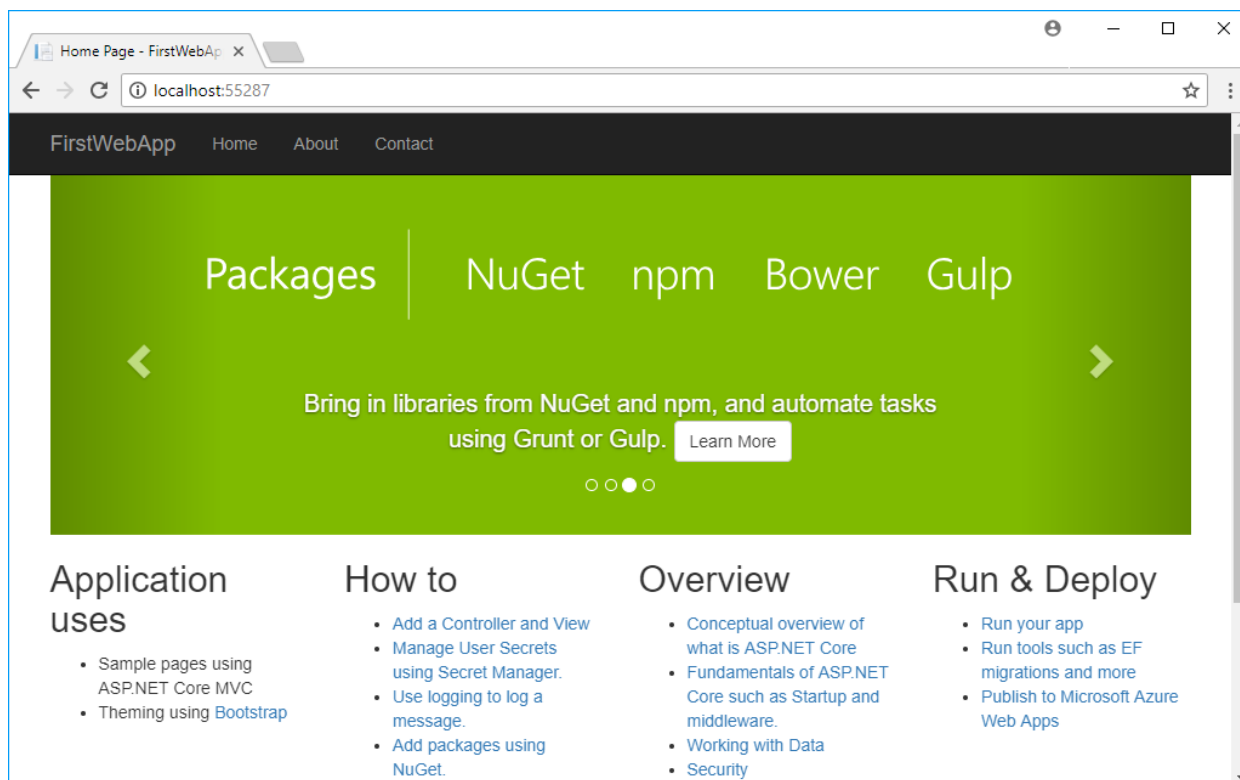
Thư mục **wwwroot** chứa những file tĩnh như html, javascript, CSS, hình, ...



Bạn có thể truy xuất trực tiếp file site.css trong thư mục css như sau:  
`http://localhost:<port>/css/site.css`

Thư mục **Dependencies** chứa các thư viện cài đặt từ Nuget.

Chạy ứng dụng bằng F5.



### 3.2.4 Application Startup

Trong project luôn có 2 file đặc biệt là **Program.cs**.

Class Program khởi tạo webserver trong hàm main() còn class StartUp cấu hình yêu cầu pipeline từ ứng dụng.

```
using System;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

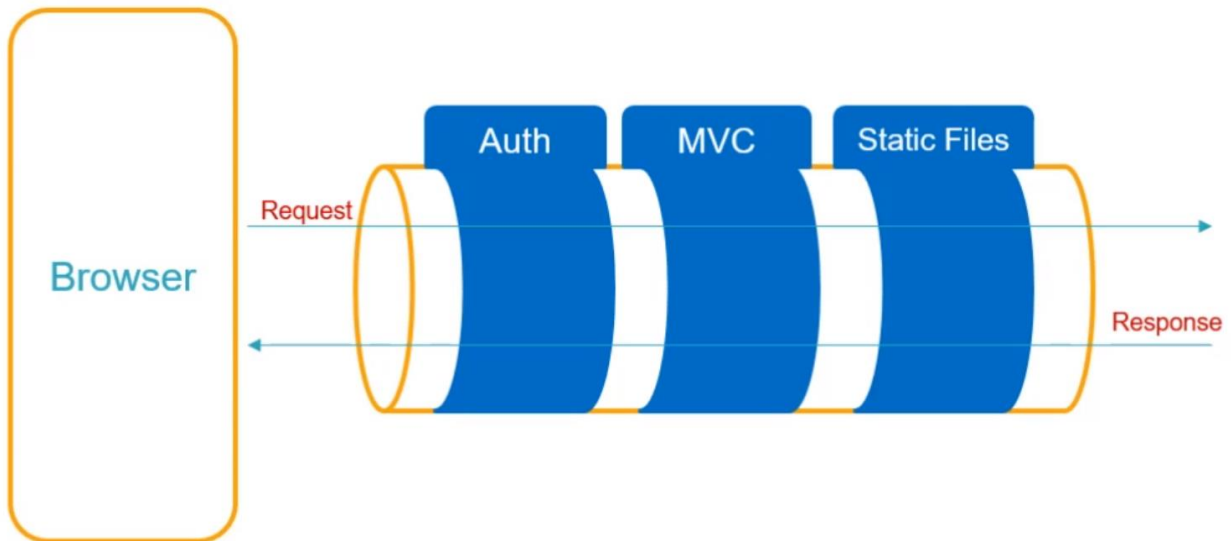
namespace HelloWorld
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run(); ← 3. Run it
        }

        public static IWebHost BuildWebHost(string[] args) => ← 1. Create a Host
        {
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build(); ← 2. Build it
        }
    }
}
```

Phương thức tĩnh `BuildWebhost` dùng để cấu hình, xây dựng và sẽ trả về tham chiếu Host.

Hàm `CreateDefaultBuilder` của lớp `WebHost` dùng để thực thi các task sau:

1. Cấu hình `Kestrel` web server.
2. Đặt content root ở `Directory.GetCurrentDirectory`.
3. Load cấu hình mặc định từ:
  - a) `Appsettings.json`
  - b) `Appsettings.{Environment}.json`.
  - c) User secrets when the app runs in the Development environment.
  - d) Environment variables
  - e) Command-line arguments.
4. Enable logging
5. Tích hợp `Kestrel` với `IIS`



**Hình 3-6: ASP.NET Pipeline**

### Kestrel là gì?

- Là một web-server đa nền tảng được xây dựng cho ASP.NET Core dựa trên libuv - một thư viện xử lý I/O async đa nền tảng.
- Là web-server mặc định khi dùng ASP.NET Core template
- Bên cạnh đa nền tảng, nó có một lợi thế nữa là tốc độ rất nhanh
- Không có đầy đủ tính năng của một web server
- Nên dùng nó sau một web server đầy đủ như IIS/Nginx/Apache

Startup là class đơn giản, không kế thừa từ bất kỳ lớp nào, thực thi 2 nhiệm vụ chính:

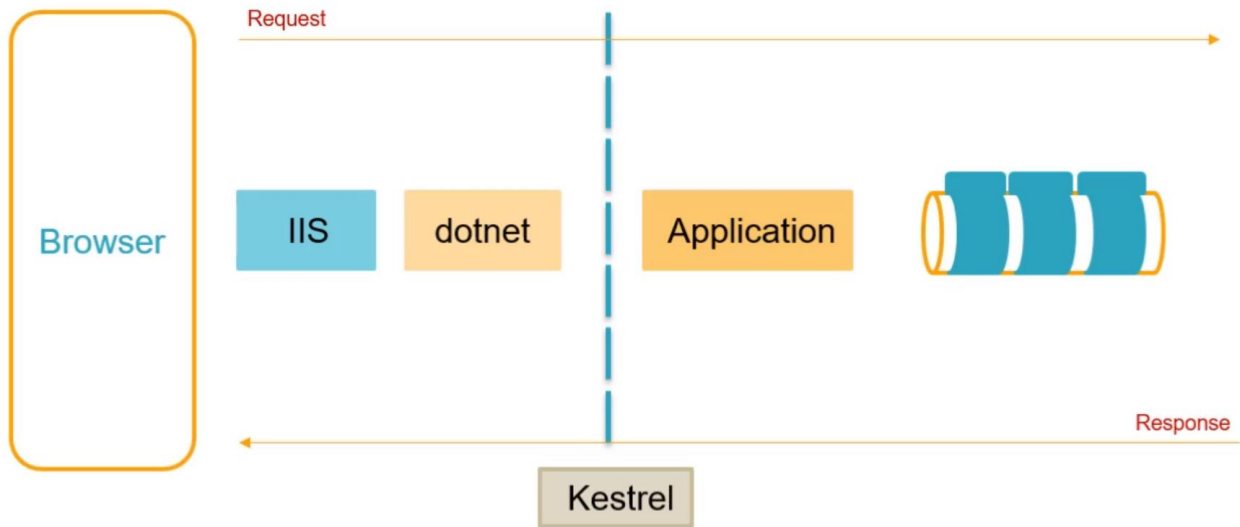
- Cấu hình đường ống

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
}
```

- Cấu hình các dịch vụ phụ thuộc (dependency injection).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
}
```





**Hình 3-7: ASP.NET Core Request Processing**

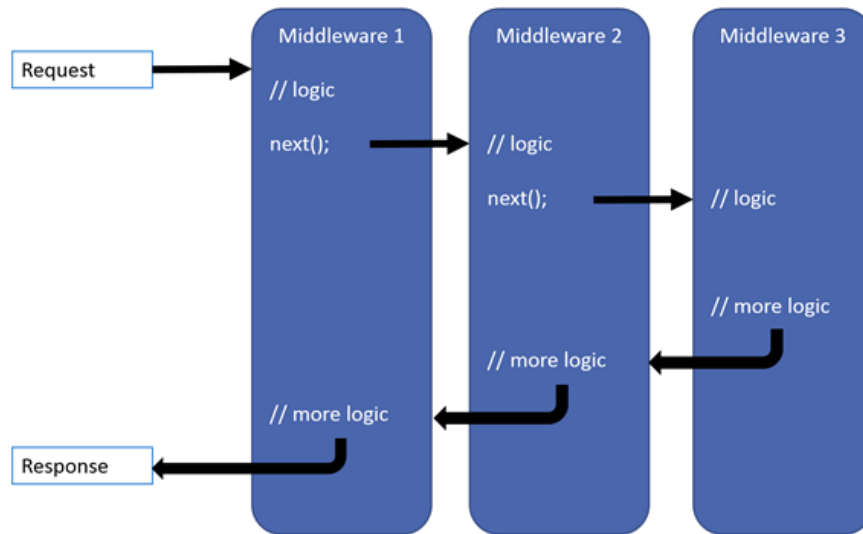
### 3.2.5 Dependency Injection

- Đây là một kỹ thuật nhằm loại bỏ sự liên kết lẫn nhau giữa các object và các object dùng trong nó, loại bỏ các phụ thuộc
- ASP.NET Core được thiết kế để hỗ trợ dependency injection
- ASP.NET Core dùng DI để dẫn các service được xây dựng sẵn hoặc service tự tạo vào trong các phương thức, được config trong class Startup.
- Các đăng ký và quản lý lifetime của DI trong ASP.NET Core:
  - Transient
  - Scoped
  - Singleton
- Lợi ích của DI đó là quản lý resource, test dễ dàng và khả năng mở rộng cao.

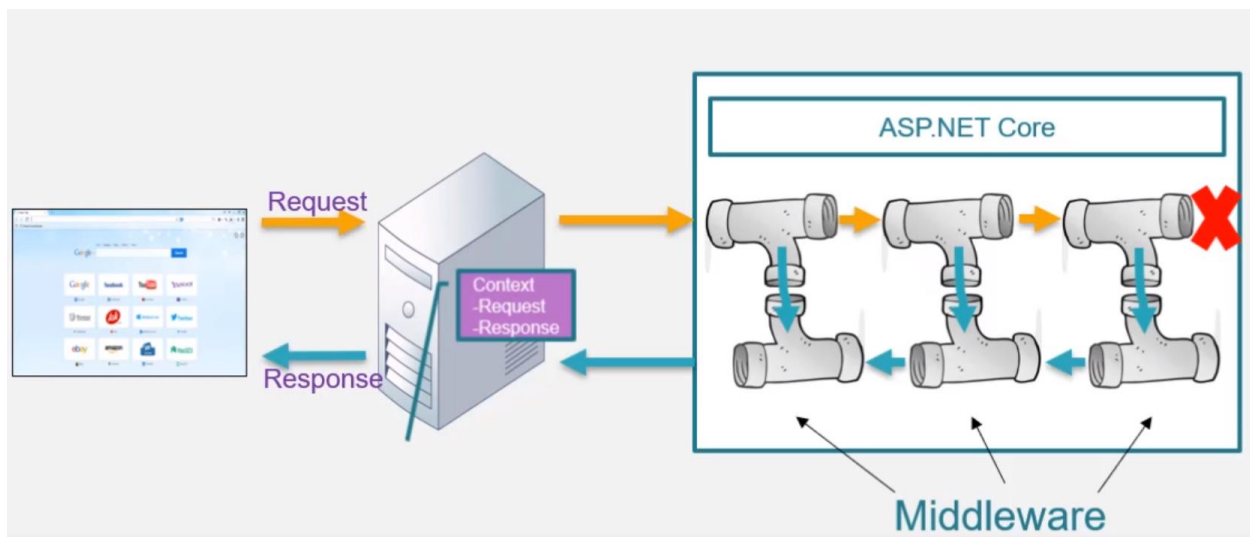
Service là các dịch vụ được tạo ra để xử lý một việc cụ thể (database, log, cache...), được quản lý bởi DI trong ASP.NET Core và được đăng ký bên trong Startup của ứng dụng. ASP.NET Core được xây dựng theo cơ chế service, tất cả đều là service và quản lý bởi DI.

### 3.2.6 Middleware

Middleware là các software component được gắn vào trong pipeline của ứng dụng để xử lý request và response. Mỗi component trong pipeline là một request delegate. Mỗi delegate có thể khởi chạy component tiếp theo trong một chuỗi các component được gắn vào pipeline. Có thể ngắt hoặc chạy tiếp đến các component trong một chuỗi các component trong pipeline được sắp xếp tuần tự.

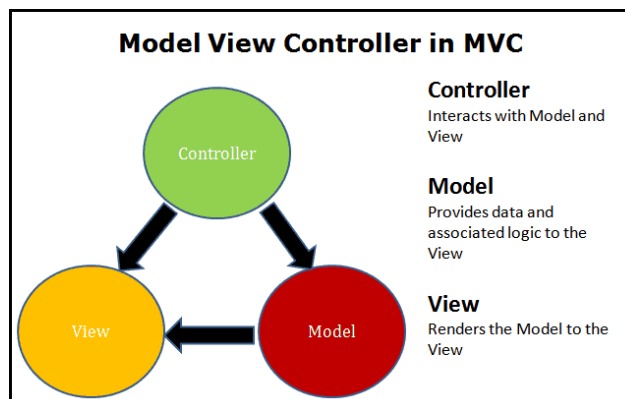


Hình 3-8: Middle ware



Hình 3-9: Middleware in ASP.NET Core

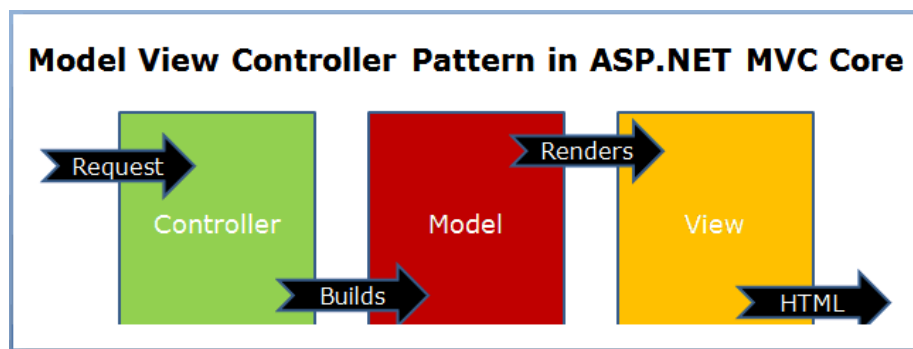
### 3.2.7 Mô hình Model – View – Controller



Hình 3-10: Mô hình M-V-C

- Model-View-Controller (MVC) là mô hình kiến trúc xây dựng ứng dụng tách ra làm ba phần chính riêng biệt **Model**, **View** và **Controller**. MVC giúp bạn xây dựng ứng dụng dễ dàng bảo trì, cập nhật hơn các ứng dụng truyền thống.
  - ✓ **Model**: là các class đại diện cho dữ liệu và logic để thực thi nghiệp vụ của dữ liệu đó. Thông thường, mỗi model sẽ đại diện cho một table trong database.
  - ✓ **View**: là thành phần hiển thị giao diện người dùng (UI) của ứng dụng. Nhìn chung, view sẽ hiển thị dữ liệu từ model.
  - ✓ **Controller**: Có nhiệm vụ xử lý các request từ trình duyệt (GET, POST, PUT ...) và sau đó trả về các data tương ứng cho view.
- MVC giúp bạn tách biệt các phần của ứng dụng (input logic, business logic, UI logic) và cung cấp kết nối giữa các lớp này. Sự tách biệt này giúp bạn dễ dàng quản lý những ứng dụng phức tạp vì nó cho phép bạn làm việc trên một phần và không ảnh hưởng đến những phần khác.

Trong mô hình MVC, cái yêu cầu gửi đến (incoming request) được xử lý bởi các bộ điều khiển (controller). Mỗi phương thức công khai (public method) trong controller được gọi là một phương thức hành động (action method), nghĩa là bạn có thể gọi nó từ Web thông qua địa chỉ URL để thực hiện một action. Các controller được đặt trong thư mục Controllers của project.



Hình 3-11 : MVC Core Pattern

Khi tạo mới Project, mặc định Visual Studio luôn tạo một controller Home đặt tên là **HomeController.cs**.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using FirstWebApp.Models;

namespace FirstWebApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

```
    }

    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";

        return View();
    }

    public IActionResult Contact()
    {
        ViewData["Message"] = "Your contact page.";

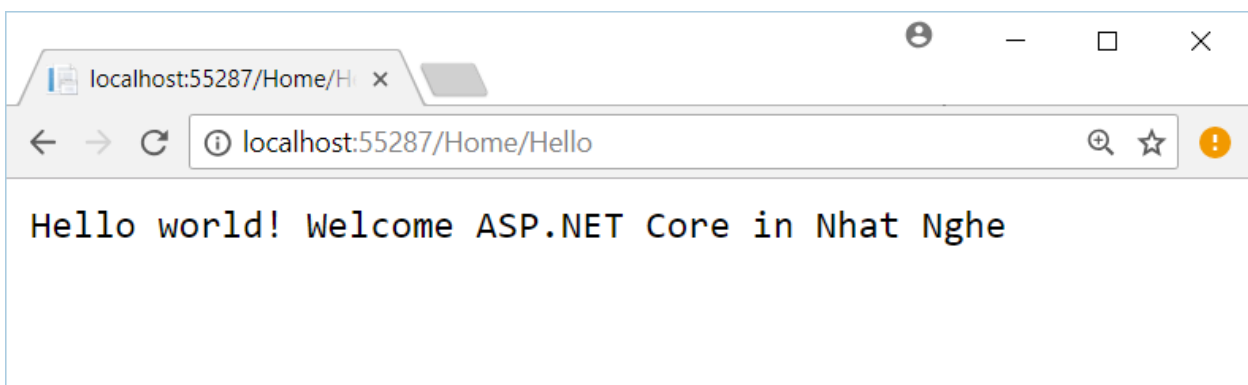
        return View();
    }

    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id
?? HttpContext.TraceIdentifier });
    }
}
}
```

Thử viết thêm một action Hello trong controller trên:

```
public string Hello()
{
    return "Hello world! Welcome ASP.NET Core in Nhat Nghe";
}
```

Chạy ứng dụng thử kết nối địa chỉ <http://localhost:<port>/Home/Hello>



Định tuyến – Routing

Để truy xuất đến action Index trong Home controller có các cách sau:

- /
- /Home
- /Home/Index

Bởi vì action **Index** và controller **Home** là mặc định được khai báo trong **Startup.cs**:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

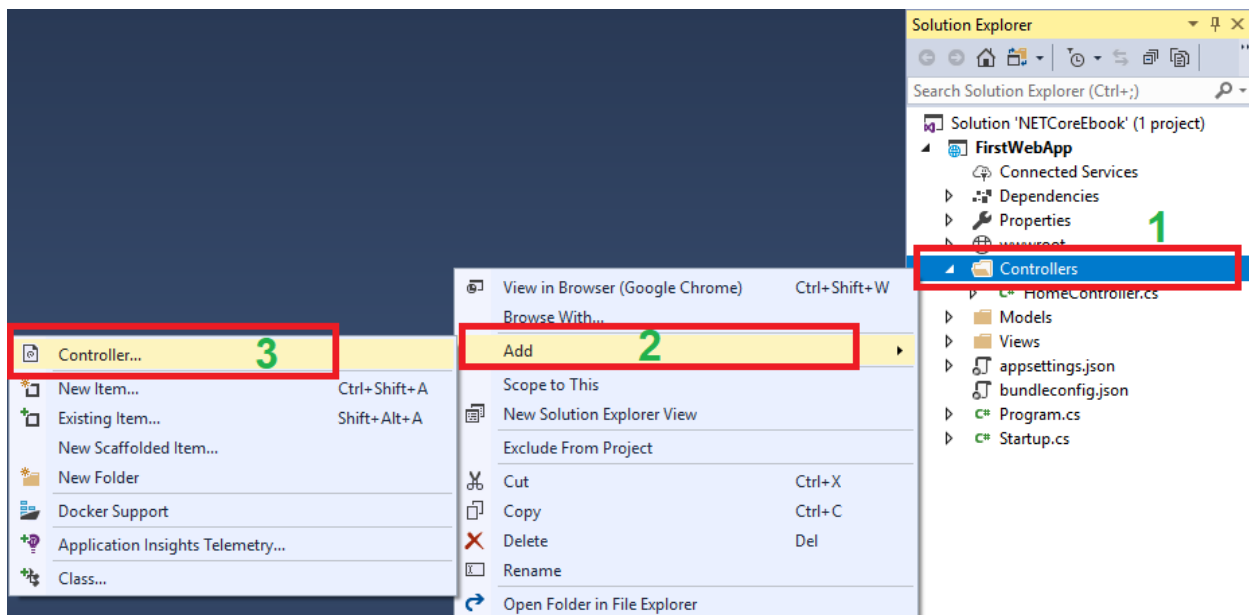
Nguyên tắc gọi 1 action trong MVC như sau:

**host[:port]/ControllerName/ActionName**

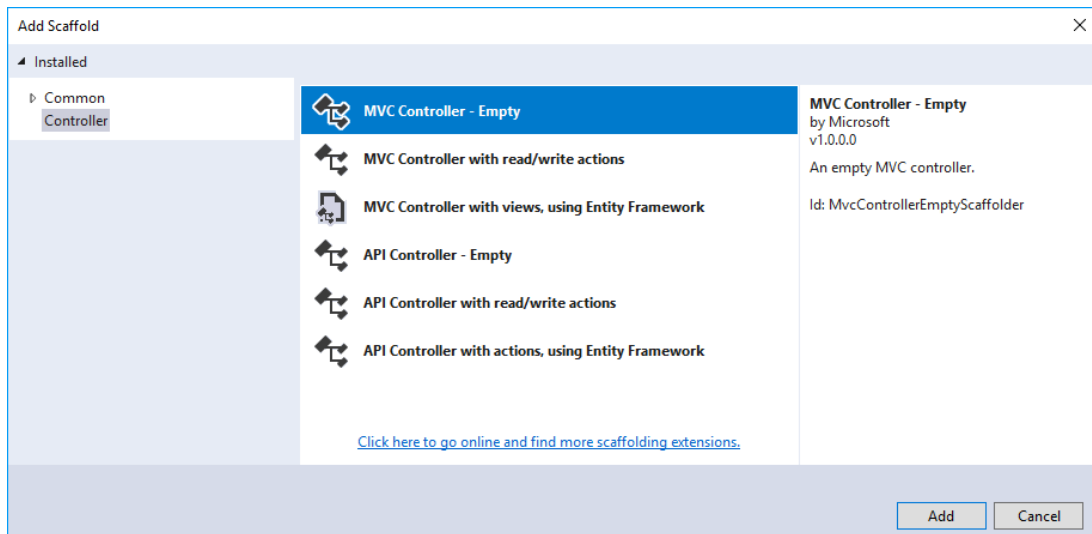
### 3.2.8 Thêm mới Controller

Controller là thành phần chứa các Action là các phương thức điều khiển và đáp ứng yêu cầu từ người dùng. Phần hướng dẫn dưới đây giúp bạn thêm mới một Controller và một Action đơn giản, sau đó thử chạy để biết hoạt động của chúng.

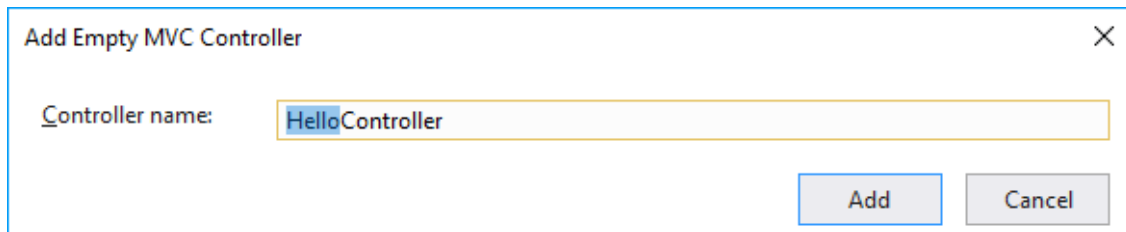
Chuột phải vào thư mục **Controller**, chọn **Add**, chọn **Controller**.



Hình 3-12: Thêm mới Controller

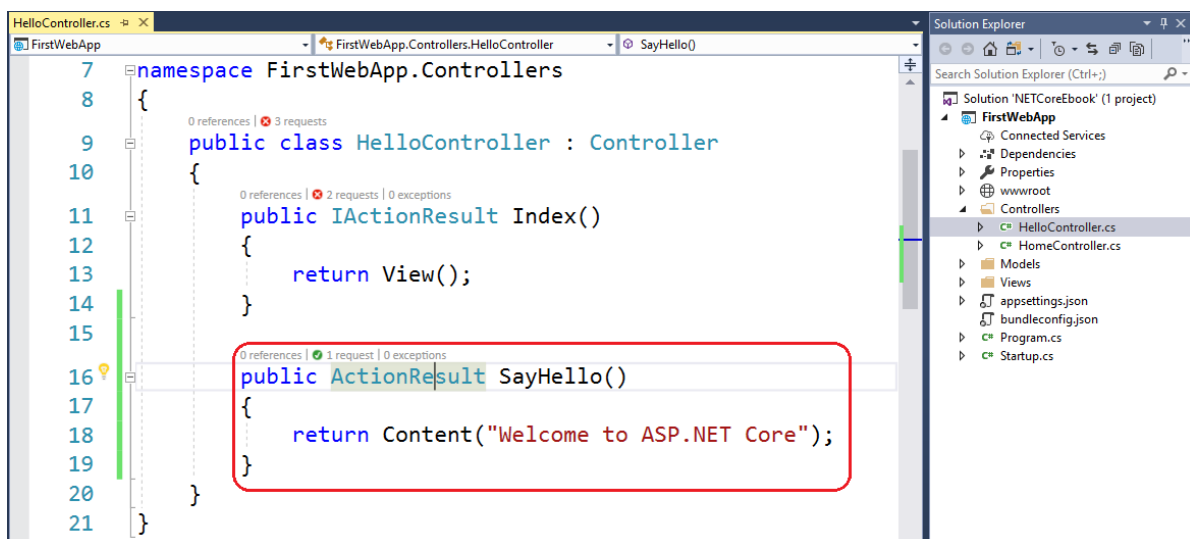


**Hình 3-13: Chọn loại Controller trống**

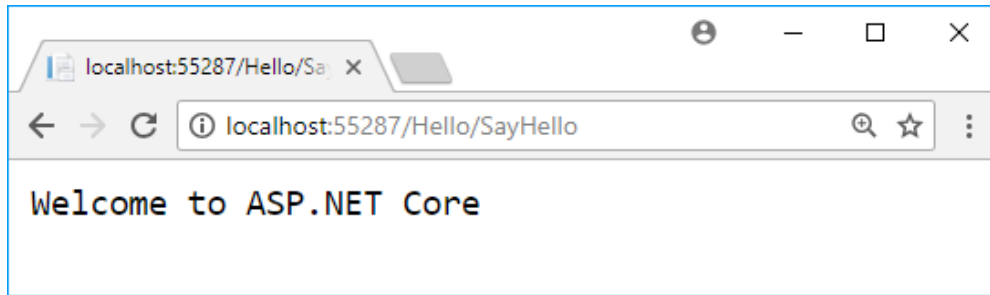


**Hình 3-14: Đặt tên Controller**

*Chú ý:* Tên controller phải có phần cuối ngữ là Controller, trường hợp này **HelloController**. Khi dựng controller, ta dùng tên Hello, còn tiếp cuối ngữ là quy ước để hệ thống xử lý phía hậu trường.



**Hình 3-15: Thêm mới một Action**

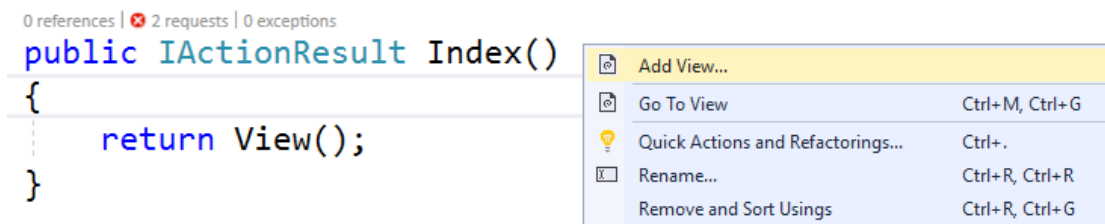


Hình 3-16: Chạy action SayHello của controller Hello

### 3.2.9 Thêm mới View

Ở trên action SayHello return Content("Welcome to ASP.NET Core") để hồi đáp yêu cầu. Kết quả sẽ gửi về dòng chữ "Welcome to ASP.NET Core" như hình 3.9. Nếu muốn trả về một trang web với nội dung phức tạp và hình thức đẹp thì phải xây dựng một View riêng để đáp ứng yêu cầu. Lúc đó action phải return View() thay vì Content().

Để tạo view cho Action, chuột phải trên action chọn **Add View**.



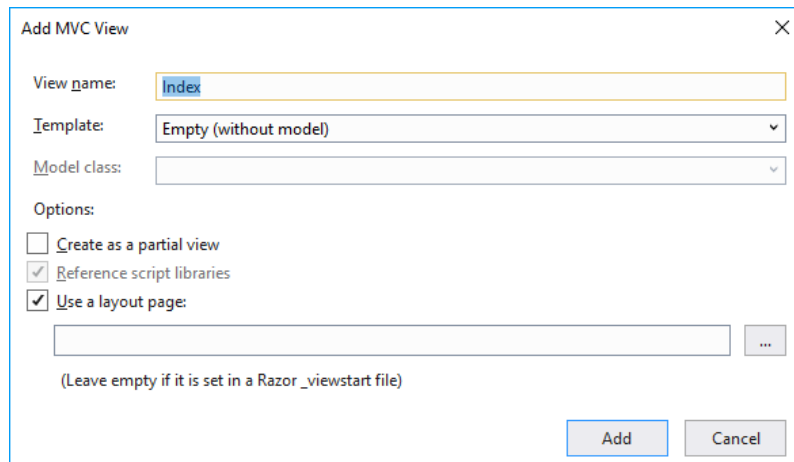
Hình 3-17: Thêm View cho action

Lệnh return của Action có thể lựa chọn bất kỳ View tên gì để hiển thị nếu đưa tên View làm tham số cho phương thức View().

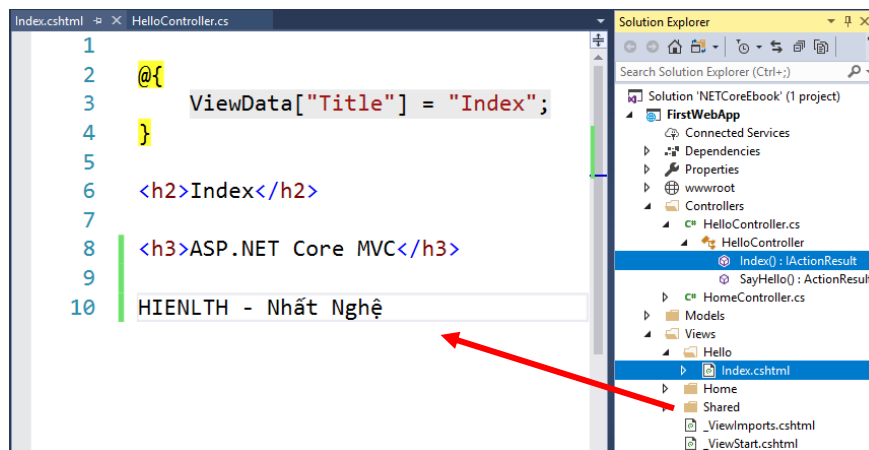
Ví dụ:

- return View("Chao") sẽ gọi view Chao.cshtml
- return View("SayHello") sẽ gọi view SayHello.cshtml

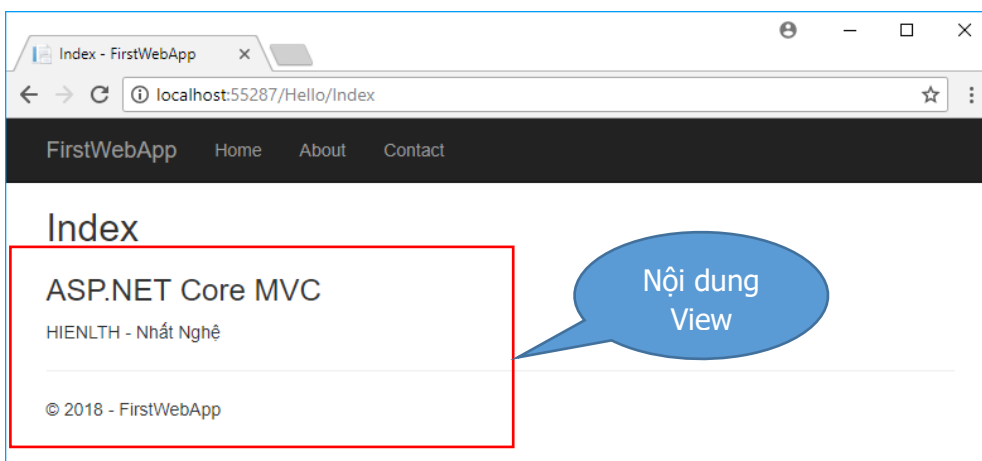
Trong trường hợp không chỉ ra tên view cần gọi thì MVC hiểu tên view chính là tên action. Vì vậy trường hợp hình 3-7 được hiểu là return View("index").



**Hình 3-18: Đặt tên View**



Nội dung View Index đơn giản, chưa có dữ liệu từ CSDL, chưa có tính thẩm mỹ cao, sẽ được khám phá ở các bài tiếp theo.



**Hình 3-19: Kết quả hiển thị**



## Chương 4: CONTROLLER

### 4.1 Cấu trúc Controller

Controller là một class kế thừa từ Controller trong MVC Core. Trong Controller có thể có nhiều Action phục vụ yêu cầu người dùng.

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    public IActionResult About()
    {
        ViewData["Message"] = "Your application description page.";

        return View();
    }

    public IActionResult Contact()
    {
        ViewData["Message"] = "Your contact page.";

        return View();
    }

    public string Hello()
    {
        return "Hello world! Welcome ASP.NET Core in Nhat Nghe";
    }
}
```

HomeController trên định nghĩa 4 action là Index(), About(), Contact() và Hello(). Các action này sẽ được phục vụ các yêu cầu từ người dùng thông qua các url như sau:

- ✓ http://localhost -> Index()
- ✓ http://localhost/Home -> Index()
- ✓ http://localhost/Home/Index -> Index()
- ✓ http://localhost/Home/About -> About()
- ✓ http://localhost/Home/Contact -> Contact()
- ✓ http://localhost/Home/Hello -> Hello()

### 4.2 Action Method

Bất kỳ method public nào được gọi từ Controller thì phải gọi phương thức Action, cụ thể là thông qua đường dẫn URL trên trình duyệt.

Action Method thường gọi service layer để phản hồi yêu cầu người dùng. Service layer thường truy vấn hoặc thay đổi cơ sở dữ liệu bằng cách sử dụng Data Access layer và sau đó chuyển kết quả trả về cho Model và chuyển ngược lại cho phương thức Action.

Một số lưu ý khi tạo Action method:

- ✓ Luôn là public method, không thể là static method hay extension method.
- ✓ Các phương thức kế thừa trong Controller không được xem là action method.

- ✓ Tham số của action method không là ref, out.
- ✓ Không chứa thuộc tính [NonAction].
- ✓ Không thể overload các phương thức action.

### 4.3 Tiếp nhận tham số

Khi yêu cầu một action dữ liệu từ người dùng sẽ được chuyển cho các action thông qua tham số. Tham số tồn tại dưới 2 dạng chuỗi truy vấn (sau dấu ? của url) hoặc các trường trên form

- ✓ Ví dụ: Gọi action và truyền tham số với chuỗi truy vấn

Với action ChiTiet() như sau:

```
public ActionResult ChiTiet(int id, string TenLoai, string MoTa)
{
    Loai loai = new Loai
    {
        MaLoai = id, TenLoai = TenLoai, MoTa = MoTa
    };

    return View(loai);
}
```

Thì liên kết tới:

```
<a href="/Loai/ChiTiet?Id=2&TenLoai=Bia&MoTa=Bla-bla">Chi tiết</a>
<a href="/Loai/ChiTiet/2?TenLoai=Bia&MoTa=Bla-bla">Chi tiết</a>
```

Theo định tuyến 2 liên kết trên có tác dụng như nhau là đều gọi action ChiTiet() và truyền 3 tham số Id, TenLoai, MoTa.

- ✓ Ví dụ: Gọi action và truyền tham số với các trường form

```
<form asp-action="Create" asp-controller="Loai">
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <div class="form-group">
    <label class="control-label">Tên loại</label>
    <input asp-for="TenLoai" class="form-control" />
    <span asp-validation-for="TenLoai" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label class="control-label">Mô tả</label>
    <input asp-for="MoTa" class="form-control" />
    <span asp-validation-for="MoTa" class="text-danger"></span>
  </div>
  <div class="form-group">
    <input type="submit" value="Tạo mới" class="btn btn-default" />
  </div>
</form>
```

Khi nhấn vào nút "Tạo mới" thì yêu cầu được gửi đến action Create() của controller Loai(), đồng thời chuyển các tham số TenLoai, MoTa cho action() này.

### 4.4 ActionResult

Kết quả trả về của action là ActionResult, một kiểu dữ liệu chung chung từ kiểu đơn giản string, int, ... cho đến kiểu phức tạp như JSON, html, file (dùng để download).

**IActionResult** là một Interface, định nghĩa các hợp đồng kết quả của ActionResult.

**ActionResult** là một lớp cơ sở trừu tượng cài đặt cho **IActionResult**. Các **ViewResult**, **PartialViewResult**, **JsonResult**, ... đều kế thừa lớp ActionResult này.

Các loại:

Name		Behavior
<b>ContentResult</b>	Content()	trả về nội dung văn bản không bao gồm layout, phù hợp cho việc test và làm việc với ajax.
<b>FileContentResult</b>	File()	trả về nội dung file văn bản (không bao gồm layout). Ví dụ: <pre>return File("~/downloads/pdf-sample.pdf", "application/pdf");</pre> hoặc: <pre>public IActionResult FileContentResult() {     //Get the byte array for the document     var pdfBytes =         System.IO.File.ReadAllBytes("wwwroot/downloads/pdf-sample.pdf");      //FileContentResult needs a byte array and returns a     file with the specified content type.     return new FileContentResult(pdfBytes,         "application/pdf"); }</pre>
<b>FilePathResult</b>		
<b>FileStreamResult</b>		
<b>EmptyResult</b>	Empty()	Không trả về gì cả
<b>JavaScriptResult</b>	JavaScript()	trả về mã javascript phù hợp với tương tác ajax.
<b>JsonResult</b>	Json()	trả về dữ liệu dạng JSON. Ví dụ: <pre>public IActionResult JsonResult() {     return Json(new { message = "This is a JSON result.", date = DateTime.Now }); }</pre>
<b>RedirectToResult</b>	RedirectTo()	Chuyển đến địa chỉ URL chỉ định. Ví dụ: <pre>return Redirect("https://www.exceptionnotfound.net"); return LocalRedirect("/redirects/target") return RedirectToAction("target");</pre>
<b>HttpUnauthorizedResult</b>		Trả về mã trạng thái HTTP 403
<b>RedirectToRouteResult</b>		Redirect to different action/ different controller action. Ví dụ: <pre>return RedirectToRoute("default", new { action = "target", controller = "redirects" });</pre>
<b>ViewResult</b>	View()	lựa chọn View để hiển thị được bao bọc bởi layout
<b>PartialViewResult</b>	PartialView()	lựa chọn View để hiển thị KHÔNG được bao bọc bởi layout, phù hợp với module hóa giao diện hoặc làm việc với ajax

Ví dụ:

```
public class SampleController : Controller
{
    public string SayHello()
```

```
{
    return "Hello, ASP.NET Core!";
}

public double Add(double a, double b)
{
    return a + b;
}

public IActionResult CylinderVolume(double r, double h)
{
    double v = Math.PI * Math.Pow(r, 2) * h;
    return new JsonResult(v);
}
}
```

Khi khai báo kiểu IActionResult có thể trả về kiểu dữ liệu tùy ý.

Gọi các action như sau:

- <http://localhost:5000/sample/sayhello>
- <http://localhost:5000/sample/add?a=10&b=20>
- <http://localhost:5000/sample/cylindervolume?r=10&h=20>

Ngoài ra có thể trả về các mã trạng thái (Status Code Result) như:

- **OkResult:** trả về mã 200 OK. Ví dụ: `return Ok();`  

```
public IActionResult OkResult()
{
    return Ok();
}
```
- **CreatedResult:** trả về mã 201. Ví dụ:  

```
public IActionResult CreatedResult()
{
    return Created("http://example.org/myitem", new { name = "testitem"
});
}
```
- **NoContentResult:** trả về mã 204 – NoContent Status. Ví dụ:  

```
public IActionResult NoContentResult()
{
    return NoContent();
}
```
- **BadRequestResult:** trả về mã 400 – Bad Request. Ví dụ:  

```
public IActionResult BadRequestResult()
{
    return BadRequest();
}
```
- **UnauthorizedResult:** trả về mã 401.  

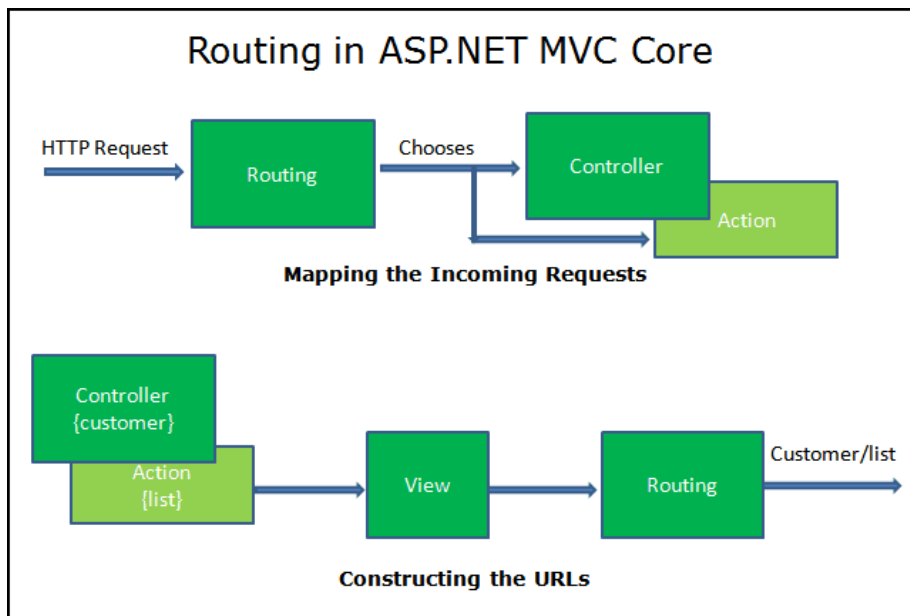
```
public IActionResult UnauthorizedResult()
{
    return Unauthorized();
}
```
- **NotFoundResult:** trả về mã 404.

```
public IActionResult NotFoundResult()  
{  
    return NotFound();  
}
```

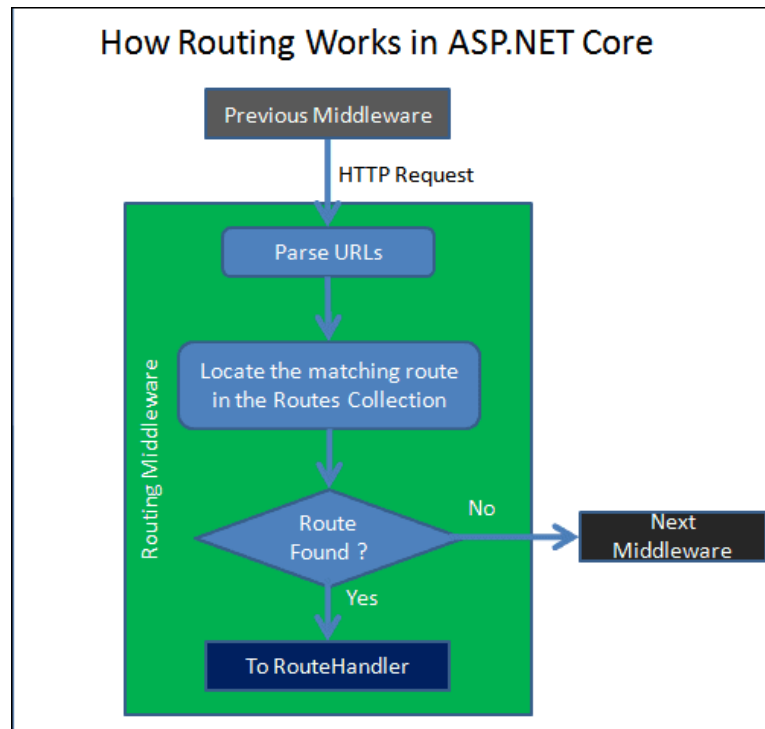
## 4.5 Routing

### 4.5.1 Routing

Routing là quá trình mà ASP.NET Core kiểm tra các URL đến và ánh xạ chúng vào các controller, action tương ứng. Ngoài ra routing còn để sinh các URL đi. Quá trình route được xử lý bởi Routing Middle có sẵn trong namespace Microsoft.AspNetCore.Routing.



Hình 3-20: Nhiệm vụ của Routing



**Hình 3-21: Sơ đồ hoạt động của Routing**

Khi có một yêu cầu, Routing Middleware thực hiện như sau:

- ✓ Phân tích cú pháp URL.
- ✓ Tìm kiếm định tuyến phù hợp trong RouteCollection.
- ✓ Nếu tìm thấy sẽ chuyển đến RouteHandle, nếu không tìm thấy sẽ bỏ và thực thi Middleware tiếp theo.

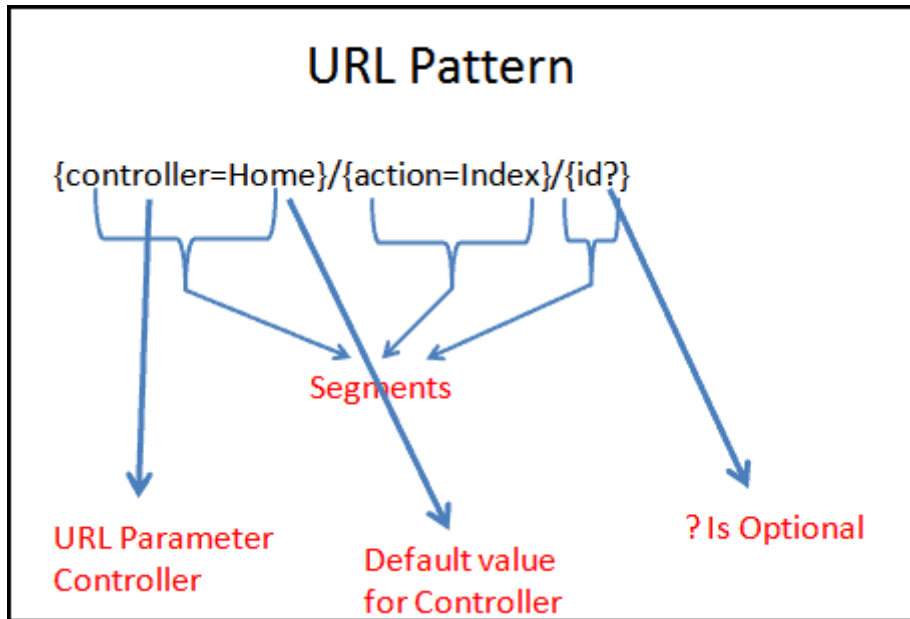
Mỗi **Route** bao gồm tên (Name), URL Pattern (Template), Defaults và Constraints. **URL Pattern** nhằm so khớp yêu cầu gửi đến từ URL, ví dụ: `{controller=Home}/{action=Index}/{id?}`

```

    MapRoute Api

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            "default",
            "{controller=Home}/{action=Index}/{id?}");
    });
  
```

Name of the Route  
URL Pattern



Trong phần URL Pattern có dấu ? để xác định tùy chọn hay không.

Việc định nghĩa các route nằm trong hàm Configure của lớp StartUp (StartUp.cs).

Ví dụ:

```
routes.MapRoute("default", "{controller=Home}/{action=Index}");
```

sẽ map với:

URL	MATCH ?	PARSED AS
/	Yes	Controller=Home, Action=Index
/Home	Yes	Controller=Home, Action=Index
/Home/Index	Yes	Controller=Home, Action=Index

Còn định nghĩa route như sau:

```
routes.MapRoute("default", "{admin}/{controller=Home}/{action=Index}");
```

sẽ map với:

URL	MATCH ?	PARSED AS
/	No No defaults for admin. Hence first segment is mandatory	

URL	MATCH ?	PARSED AS
/Home	Yes  The First segment Home matches to the Admin	Admin=Home Controller=Home Action=Index
/Abc	Yes	Admin=Abc Controller=Home Action=Index
/Home/Index	No  Admin=Home Controller=Index  There is No IndexController, Hence it fails	
/Xyz/Home	Yes	Admin=Xyz Controller=Home Action=Index
/Admin/Home	Yes	Admin=Admin Controller=Home Action=Index

Ví dụ 3:

```
routes.MapRoute("default", "admin/{controller=Home}/{action=Index}");
```

URL	MATCH ?	PARSED AS
/	No because First segment is mandatory	
/Home	No The first segment must contain the word Admin	
/Abc	No The first segment must contain the word Admin	
/Admin	Yes	Controller=Home Action=Index
/Admin/Home	Yes	Controller=Home Action=Index

#### 4.5.2 Attribute Routing

Bạn có thể thêm các thuộc tính chỉ định Route trực tiếp trước các Action.

```
[Route("Home")]
public string Index()
{
    return "Hello from Index method of Home Controller";
}
```

Khi gọi URL /Home thì action Index của controller Home được thực thi.



URL	MATCH ?	PARSED AS
/	No	
/Home	Yes	Controller=Home Action=Index
/Home/Index	No	

Thêm action vào route property:

```
[Route("Home/Index")]
public string Index()
{
    return "Hello from Index method of Home Controller";
}
```

URL	MATCH ?	PARSED AS
/	No	
/Home	No	
/Home/Index	Yes	Controller=Home Action=Index

Ví dụ 3:

```
[Route("Say/Hello")]
public string Index()
{
    return "Hello from Index method of Home Controller";
}
```

URL	MATCH ?	PARSED AS
/	No	
/Home	No	
/Home/Index	No	
/Say/Hello	Yes	Controller=Home Action=Index

Multiple Routes: có thể áp dụng nhiều route cho cùng một action.

```
[Route("")]
[Route("Home")]
```

```
[Route("Home/Index")]
public string Index()
{
    return "Hello from Index method of Home Controller";
}
```

URL	MATCH ?	PARSED AS
/	Yes	Controller=Home Action=Index
/Home	Yes	Controller=Home Action=Index
/Home/Index	Yes	Controller=Home Action=Index
/Say/Hello	No	

```
[Route("Home/Index/{id:int}")]
public string Index(int id)
{
    return "I got " + id.ToString();
}
```

## 4.6 Action Selector

Action Selector là thuộc tính được áp dụng cho action method của controller, giúp cho bộ định tuyến chọn đúng phương thức action để xử lý cho URL đã cho. Action Selector bao gồm : Action Name, Non Action và Action Verb.

Ví dụ 1: Cần truy cập URL <http://host/Home/Modify> có 2 cách dưới đây tương đương nhau:

```
[ActionName("Modify")]
public string Edit()
{
    return "Hello from Edit Method";
}
```

Là tương đương với:

```
[Route("Home/Modify")]
public string Edit()
{
    return "Hello from Edit Method";
}
```

Ví dụ 2: Chỉ định phương thức NonAction:

```
[NonAction]
public string Edit()
{
    return "Hello from Edit Method";
}
```

Ví dụ 3: Sử dụng các động từ HTTP Verbs như GET, POST, PUT, DELETE, HEAD, OPTIONS, PATCH

```
[HttpGet]
public ActionResult Edit(string id)
{
    //Return the Edit Form
    return View();
}

[HttpPost]
public ActionResult Edit(Loai Model)
{
    //Update the database here
    return View();
}
```

Ví dụ 4: Cho phép sử dụng nhiều Verb cùng lúc

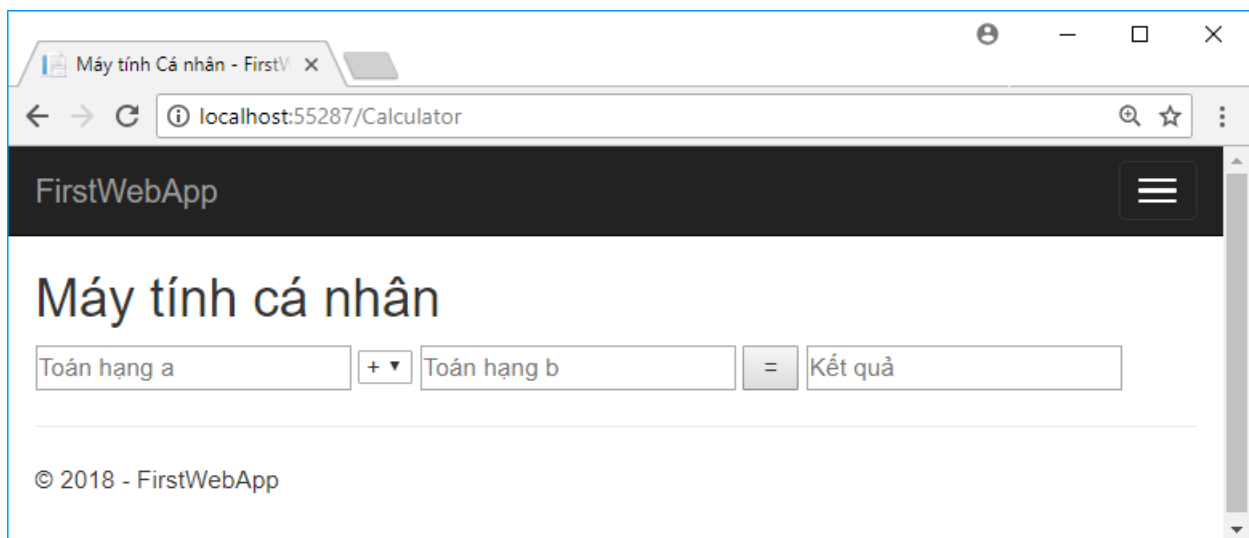
```
[AcceptVerbs(HttpVerbs.Get | HttpVerbs.Post)]
public ActionResult AboutUs()
{
    return View();
}
```

## 4.7 Bài tập Ứng dụng

### 4.7.1 Máy tính cá nhân

#### ❖ MÔ TẢ

Xây dựng trang ứng dụng cho phép thực hiện các phép tính đơn giản như cộng, trừ, nhân và chia có giao diện như hình sau. Khi người dùng nhập các toán hạng và chọn toán tử thực hiện sau đó nhấp nút [=] thì chương trình sẽ thực hiện phép tính và hiển thị kết quả lên ô nhập [Kết quả].



Để hoàn thiện bài này, bạn cần thực hiện các bước sau :

Bước 1: Tạo controller CalculatorController

Bước 2: Xây dựng giao diện

Bước 3: Thêm action Calculate để thực hiện phép tính

Bước 4: Hiển thị kết quả

### Bước 1: Tạo controller **CalculatorController**

```
public class CalculatorController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

### Bước 2: **Xây dựng giao diện**

Phải chuột lên action Index() để thêm giao diện cho action này và viết mã Razor như sau:

```
Index.cshtml
1  @{
2      ViewData["Title"] = "Máy tính Cá nhân";
3      Layout = "~/Views/Shared/_Layout.cshtml";
4  }
5
6  <h2>Máy tính cá nhân</h2>
7
8  <form asp-action="Calculate" asp-controller="Calculator" method="post">
9      <input name="a" placeholder="Toán hạng a" />
10     <select name="op">
11         <option>+</option>
12         <option>-</option>
13         <option>x</option>
14         <option>:</option>
15     </select>
16     <input name="b" placeholder="Toán hạng b" />
17     <input type="submit" value=" = " />
18     <input placeholder="Kết quả" value="@ViewBag.KetQua" />
19 </form>
```

Giao diện gồm form có action gọi đến action Calculate() của controller CalculatorController và truyền cho action này 3 tham số a (toán hạng a), b (toán hạng b) và op (toán tử op). Đồng thời form này cũng hiển thị giá trị của thuộc tính ViewBag.KetQua lên ô nhập kết quả.

### Bước 3: Thêm action **Calculate** để thực hiện phép tính

Để xử lý form, bạn cần bổ sung action Calculate() vào controller CalculatorController để tiếp nhận tham số thực hiện việc tính toán sau đó truyền kết quả về form này để hiển thị kết quả.

```
public ActionResult Calculate(double a = 0, double b = 0, char op = '+')
{
    switch (op)
    {
        case '+':
            ViewBag.KetQua = a + b;
    }
}
```

```
        break;  
    case '-':  
        ViewBag.KetQua = a - b;  
        break;  
    case 'x':  
        ViewBag.KetQua = a * b;  
        break;  
    case ':':  
        ViewBag.KetQua = a / b;  
        break;  
    }  
    return View("Index");  
}
```

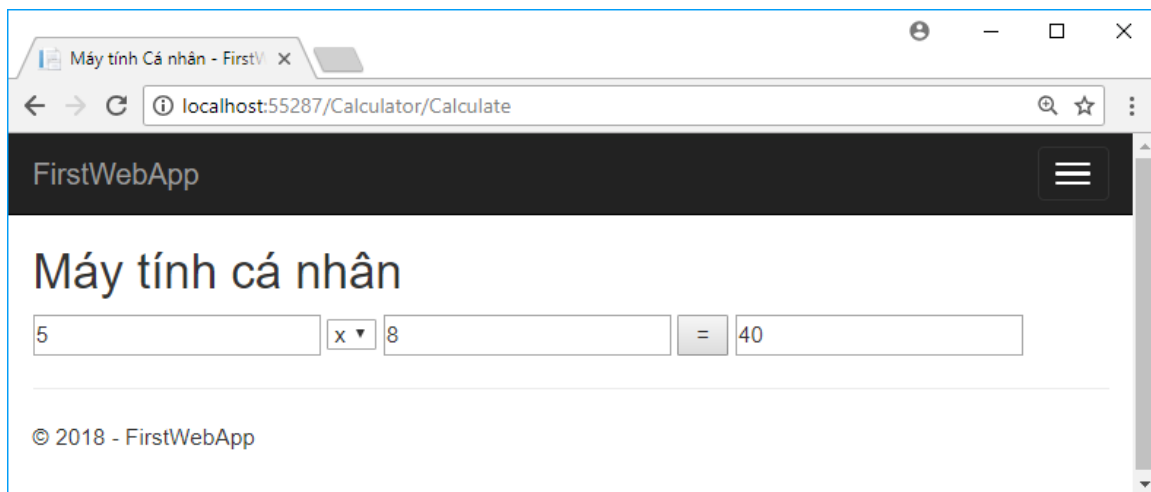
Ở đây chúng ta sử dụng phương pháp nhận tham số bằng đối số action. Như vậy các trường form a, b và op được chuyển vào các đối số của action và tự động chuyển đổi sang kiểu phù hợp. Việc còn lại là xét xem toán tử được chọn là gì để thực hiện phép toán.

Kết quả thực hiện được lưu vào thuộc tính động KetQua của đối tượng ViewBag để được truyền cho view sử dụng sau này.

Với dòng lệnh `return View("Index")` ở cuối action thì View Index.cshtml được chọn để hiển thị. Trong form của view này có dòng mã HTML là `<input placeholder="Kết quả" value="@ViewBag.KetQua" />` do đó kết quả sẽ được hiển thị vào đúng ô kết quả.

#### Bước 4: **Hiển thị kết quả**

Giao diện sau được thực hiện sau khi nhập 5 và a và 8 vào b và chọn toán tử là x sau đó nhấp nút [=] 40 sẽ hiển thị lên ô kết quả.



Chúng ta thấy rằng giá trị của các toán hạng cũng như toán tử bị bất sau khi bấm nút [=]. Muốn giữ được các giá trị này lại chúng ta phải sử dụng các Helper do MVC cung cấp (sẽ được học ở các bài sau).

#### 4.7.2 Đọc ghi file

##### ❖ MỤC TIÊU

Kết thúc bài thực hành, bạn có khả năng:

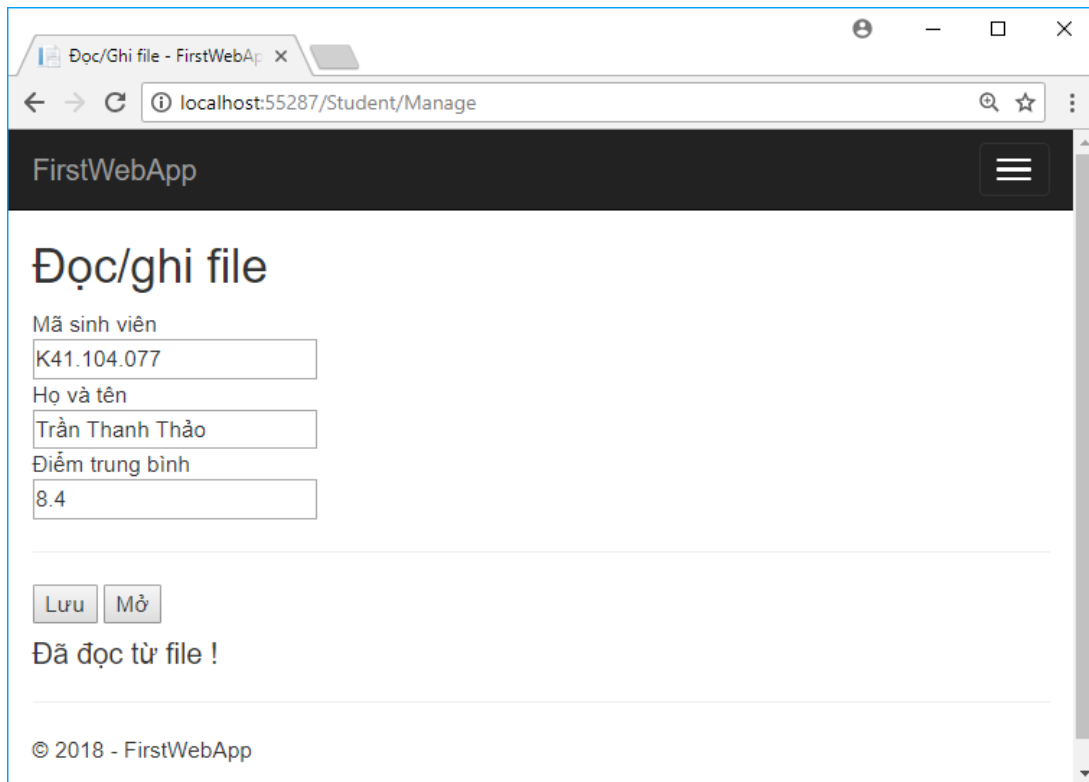
- Kết hợp đối số và model để nhận tham số
- Đọc/ghi mảng từ/vào file văn bản

#### ❖ MÔ TẢ

Cụ thể trong bài này bạn phải xây dựng trang web có hình minh họa sau. Sau khi nhập dữ liệu vào form và nhấn nút **[Lưu]** ứng dụng sẽ lưu thông tin vào file có tên **Student.txt** đặt tại thư mục gốc của website (wwwroot).

Sau khi đã lưu 1 lần, bạn có thể đọc dữ liệu từ file và hiển thị lên form bằng cách nhấn nút **[Mở]**.

Trạng thái	Mã sinh viên	Họ và tên	Điểm trung bình
Trước khi lưu			
Sau khi lưu	K41.104.077	Trần Thanh Thảo	8.4



## ❖ THỰC HIỆN

Để thực hiện ứng dụng trên, bạn cần thực hiện theo các bước sau:

Bước 1: Tạo controller **StudentController.cs**

```
public class StudentController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Bước 2: **Tạo giao diện form nhập**

Phải chuột vào action `Index()` và tạo view có mã razor như sau. Mã gồm 1 form có action gọi đến action `Manage()` của controller `StudentController` và chuyển các trường `Id`, `Name`, `Marks` và nút command được nhập đến action này.

Form cũng hiển thị các thuộc tính `Id`, `Name`, `Marks` và `Message` của `ViewBag` được chuyển từ controller lên các trường form và thông báo cuối form.

```
@{
    ViewData["Title"] = "Độc/Ghi file";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Độc/ghi file</h2>

<form asp-action="Manage" asp-controller="Student" method="post">
```

```
<div>Mã sinh viên</div>
<input name="Id" value="@ViewBag.Id" />

<div>Họ và tên</div>
<input name="Name" value="@ViewBag.Name" />

<div>Điểm trung bình</div>
<input name="Marks" value="@ViewBag.Marks" />
<hr />
<input type="submit" value="Lưu" name="command" />
<input type="submit" value="Mở" name="command" />
</form>

<h4>@ViewBag.Message</h4>
```

### Bước 3: Tạo lớp model **StudentInfo.cs**

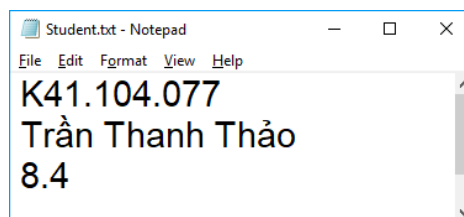
Lớp model này được sử dụng để nhận dữ liệu từ form. Các tham số sẽ chuyển vào các thuộc tính cùng tên của model.

```
public class StudentInfo
{
    public string Id { get; set; }
    public string Name { get; set; }
    public double Marks { get; set; }
}
```

### Bước 4: Bổ sung action **Manage()** vào controller để điều khiển hành động [Lưu] và [Mở]

Action Manage() sử dụng model để nhận thông tin nhân viên và đối số command để nhận nút submit bị nhấn. Action sẽ phân biệt 2 trường hợp dựa vào giá trị của nút nhấn.

Nếu nhấn nút [Lưu] thì thực hiện lưu model vào file. Phương thức System.IO.File.WriteAllLines(path, lines) sẽ lưu mảng vào file. Mỗi phần tử mảng lưu trên một hàng.



Nếu nhấn nút [Mở] thì đọc dữ liệu từ file và truyền cho view thông qua các thuộc tính Id, Name và Marks của ViewBag. Phương thức System.IO.File.ReadAllLines(path) giúp đọc mảng chuỗi từ file. Cứ mỗi hàng sẽ đọc thành 1 phần tử của mảng.

Bổ sung action Manage() để mở và đọc file:

```
public ActionResult Manage(StudentInfo model, String command)
{
    var path = Path.Combine(Directory.GetCurrentDirectory(),
        "wwwroot", "Student.txt");

    if (command == "Lưu")
    {
        String[] lines = {model.Id, model.Name, model.Marks.ToString()};
```



```
        System.IO.File.WriteAllLines(path, lines);
        ViewBag.Message = "Đã ghi vào file !";
    }
    else if (command == "Mở")
    {
        String[] lines = System.IO.File.ReadAllLines(path);
        ViewBag.Id = lines[0];
        ViewBag.Name = lines[1];
        ViewBag.Marks = Convert.ToDouble(lines[2]);

        ViewBag.Message = "Đã đọc từ file !";
    }
    return View("Index");
}
```

#### Bước 5: Chạy ứng dụng

- ❖ Chạy <http://localhost:55287/Student/Index>
- ❖ Nhập thông tin và nhấn nút \*Lưu+ sau đó kiểm tra thông tin của file được tạo ra ở thư mục gốc của website
- ❖ Nhấn nút [Mở+ để hiển thị lại thông tin đã nhập

### 4.7.3 Upload file

Kết thúc bài thực hành, bạn có khả năng:

- Tạo form upload file
- Tiếp nhận file upload và lưu vào thư mục với tên file gốc
- Hiển thị thông tin file upload

#### Bước 1: Cấu hình Startup.cs

Để dễ dàng thao tác trên thư mục wwwroot, bạn thêm phần cấu hình sau vào hàm ConfigureServices():

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IFileProvider>(
        new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(),
                "wwwroot"))));

    services.AddMvc();
}
```

#### Bước 2: Tạo FileUploadController

```
public class FileUploadController : Controller
{
    private readonly IFileProvider fileProvider;

    public FileUploadController(IFileProvider fileProvider)
    {
        this.fileProvider = fileProvider;
    }
}
```

```
public IActionResult Index()
{
    return View();
}
```

### Bước 3: Tạo form upload file

Ở View Index() này sẽ thiết kế 3 form riêng biệt dùng để upload một file, upload nhiều file và upload file dựa vào Model. Đặc biệt form upload luôn luôn phải thiết lập giá trị của thuộc tính method là POST và enctype là MULTIPART/FORM-DATA.

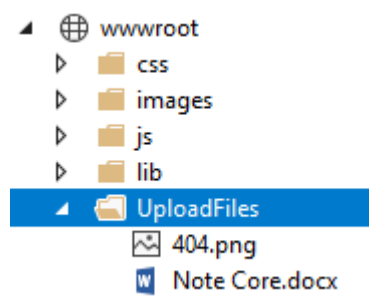
```
@{
    ViewData["Title"] = "Upload file";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Upload file</h2>

<form asp-controller="FileUpload" asp-action="UploadFile" method="post"
    enctype="multipart/form-data" class="form-inline">
    <h4>Upload single-file</h4>
    <input type="file" name="file" />
    <button type="submit">Upload File</button>
</form>
<hr />
<form asp-controller="FileUpload" asp-action="UploadFiles" method="post"
    enctype="multipart/form-data">
    <h4>Upload multi-files</h4>
    <input type="file" name="files" multiple />
    <button type="submit">Upload Files</button>
</form>
<hr />
<form asp-controller="FileUpload" asp-action="UploadFileViaModel" method="post"
    enctype="multipart/form-data" class="form-inline">
    <input type="file" name="FileToUpload" />
    <button type="submit">Upload File (model)</button>
</form>
```

Bước 4: Bổ sung các action `UploadFile()`, `UploadFiles()`, `UploadFileViaModel()` vào controller.

Tất cả các tập tin upload lên đều lưu vào thư mục **UploadFiles** trong **wwwroot**.



```
[HttpPost]
public async Task<IActionResult> UploadFile(IFormFile file)
```

```
{
    if (file == null || file.Length == 0)
        return Content("file not selected");

    var path = Path.Combine(
        Directory.GetCurrentDirectory(), "wwwroot",
"UploadFiles",
        file.GetFilename());

    using (var stream = new FileStream(path, FileMode.Create))
    {
        await file.CopyToAsync(stream);
    }

    return RedirectToAction("ListFiles");
}
```

```
[HttpPost]
public async Task<IActionResult> UploadFiles(List<IFormFile> files)
{
    if (files == null || files.Count == 0)
        return Content("files not selected");

    foreach (var file in files)
    {
        var path = Path.Combine(
"UploadFiles",
            Directory.GetCurrentDirectory(), "wwwroot",
            file.GetFilename());

        using (var stream = new FileStream(path, FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }
    }

    return RedirectToAction("ListFiles");
}
```

```
[HttpPost]
public async Task<IActionResult> UploadFileViaModel(FileInputModel
model)
{
    if (model == null ||
        model.FileToUpload == null || model.FileToUpload.Length == 0)
        return Content("file not selected");

    var path = Path.Combine(
"UploadFiles",
        Directory.GetCurrentDirectory(), "wwwroot",
```

```
        model.FileToUpload.GetFilename());

        using (var stream = new FileStream(path, FileMode.Create))
        {
            await model.FileToUpload.CopyToAsync(stream);
        }

        return RedirectToAction("ListFiles");
    }
}
```

Tất cả các action trên sau khi xử lý xong đều gửi tới action ListFiles() để hiển thị danh sách file trong thư mục chỉ định thông qua lệnh `return RedirectToAction("ListFiles");`

Chú ý:

- ✓ Action UploadFile() để xử lý upload một file thì tham số truyền vào là biến đơn: `IFormFile file`, `file` chính là tên control `input[type=file]`.
- ✓ Action UploadFiles() để xử lý upload nhiều file thì tham số truyền vào là danh sách: `List<IFormFile> files`, `files` chính là tên control `input[type=file]`.
- ✓ Action UploadFileViaModel() xử lý upload file qua model cần truyền vào biến model: `FileInputModel model`, trong đó lớp FileInputModel định nghĩa như sau:

```
public class FileInputModel
{
    public IFormFile FileToUpload { get; set; }
}
```

và `FileToUpload` chính là tên của control `input[type=file]`

**Bước 5:** Bổ sung action ListFile()

```
public IActionResult ListFiles()
{
    var model = new FilesViewModel();
    foreach (var item in
this.fileProvider.GetDirectoryContents("UploadFiles"))
    {
        model.Files.Add(
            new FileDetails { Name = item.Name, Path = item.PhysicalPath
});
    }
    return View(model);
}
```

Model chuyển qua chứa thông tin tên file và đường dẫn. Do đó cần định nghĩa thêm các lớp để lưu các thông tin này.

```
public class FileDetails
{
    public string Name { get; set; }
    public string Path { get; set; }
}

public class FilesViewModel
{
    public List<FileDetails> Files { get; set; } = new List<FileDetails>();
}
```

}

Nội dung view ListFiles.cshtml dùng để hiển thị thông tin file có trong thư mục như sau:

```
ListFiles.cshtml* x
1  @model FirstWebApp.Models.FilesViewModel
2
3  @{
4      ViewData["Title"] = "ListFiles";
5      Layout = "~/Views/Shared/_Layout.cshtml";
6  }
7
8  <h2>List of Files</h2>
9
10 <ul>
11     @foreach (var item in Model.Files)
12     {
13         <li>
14             <a asp-action="Download" asp-route-filename="@item.Name">
15                 @item.Name
16             </a>
17         </li>
18     }
19 </ul>
```

**Bước 6:** Thực nghiệm

## Chương 5: **TỔ CHỨC WEBSITE**

**Sau khi học xong bài này, học viên có khả năng :**

- Xây dựng layout cho ứng dụng Web MVC.
- Đóng và sử dụng các gói tài nguyên đã đóng.
- Mô đun hóa giao diện để quản lý và sử dụng lại.
- Phân quyền ứng dụng và định tuyến theo namespace.

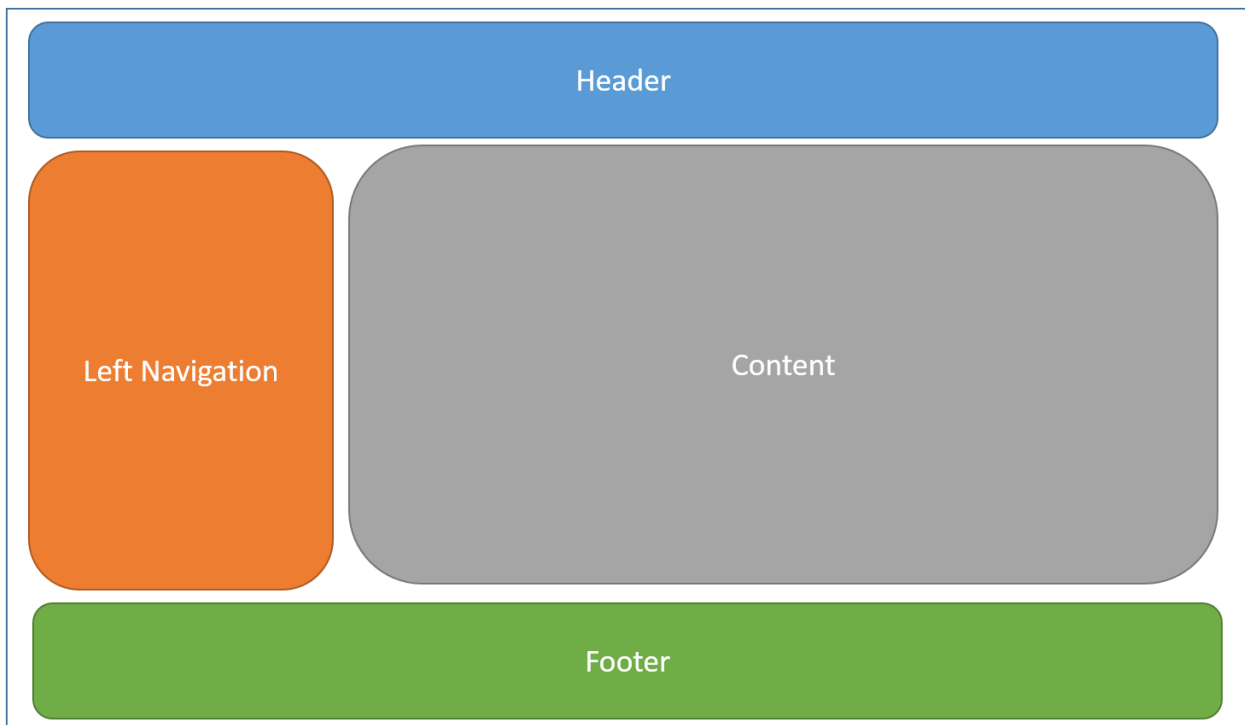
### 5.1 Các thành phần layout

#### 5.1.1 Giới thiệu

Trong mỗi website thường giao diện của một vài nhóm trang có cấu trúc tương tự nhau, chỉ khác nhau ở phần dữ liệu thay đổi. Vì vậy, nếu tạo các trang riêng rẽ sẽ tốn nhiều thời gian. Nếu có sai sót cần thay đổi sẽ phải thay đổi ở tất cả các trang con, vì vậy việc quản lý các giao diện này gặp nhiều khó khăn.

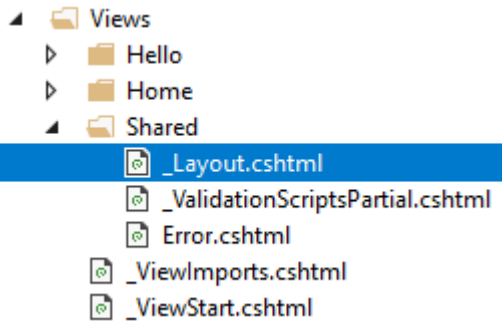
Từ đó, khái niệm Layout sẽ giải quyết vấn đề trên. Layout là giao diện của trang web, thường mỗi trang có những phần giống nhau như header, footer, menu.

View là thành phần hiển thị, tương tác trực tiếp với người dùng. Trong phần này sẽ tổ chức bố cục, các thành phần layout và thiết lập các lệnh dùng chung trước khi render xuống ứng dụng.



**Hình 5-1 : Các thành phần layout**

Theo quy ước, tập tin giao diện dùng chung thường nằm trong thư mục Views/Shared của ứng dụng. Khi tạo project, ta có sẵn file `_Layout.cshtml`.



File này sẽ định nghĩa template level top. Có thể một số view không cần layout template và cũng có view xác định từ layout template riêng rẽ cho từng trường hợp cụ thể.

Ví dụ file `_Layout.cshtml` mẫu:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - FirstWebApp</title>

  <environment include="Development">
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="~/css/site.css" />
  </environment>
  <environment exclude="Development">
    <link rel="stylesheet"
href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
      asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
      asp-fallback-test-class="sr-only" asp-fallback-test-
property="position" asp-fallback-test-value="absolute" />
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true"
  />
  </environment>
</head>
<body>
  <nav 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="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a asp-area="" asp-controller="Home" asp-action="Index"
class="navbar-brand">FirstWebApp</a>
      </div>
      <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li><a asp-area="" asp-controller="Home" asp-
action="Index">Home</a></li>
          <li><a asp-area="" asp-controller="Home" asp-
action="About">About</a></li>
```

```

        <li><a asp-area="" asp-controller="Home" asp-
action="Contact">Contact</a></li>
    </ul>
</div>
</div>
</nav>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; 2018 - FirstWebApp</p>
    </footer>
</div>

<environment include="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
</environment>
<environment exclude="Development">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.2.0.min.js"
asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
asp-fallback-test="window.jQuery"
crossorigin="anonymous"
integrity="sha384-
K+ctZQ+LL8q6tP7I94W+qzQsFRV2a+AfHIi9k8z8l9ggpc8X+Ytst4yBo/hH+8Fk">
    </script>
    <script
src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
asp-fallback-test="window.jQuery && window.jQuery.fn &&
window.jQuery.fn.modal"
crossorigin="anonymous"
integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA712mCWNIpG9mGCD8wGNICPD7Txa">
    </script>
    <script src="~/js/site.min.js" asp-append-version="true"></script>
</environment>

@RenderSection("Scripts", required: false)
</body>
</html>

```

### 5.1.2 Đánh dấu vùng động

Bạn có thể thiết kế trang layout như mọi trang bình thường khác. Sự khác biệt ở trang layout là việc đánh dấu các vị trí hiển thị dữ liệu của các trang thành viên. Trong MVC, bạn dùng các chỉ thị và thuộc tính động sau đây để đánh dấu:

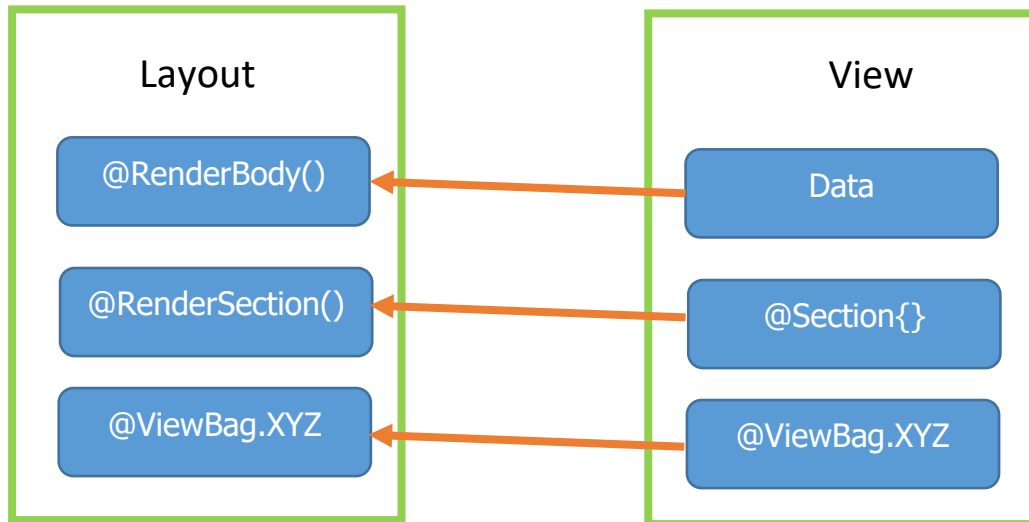
Chỉ thị/Thuộc tính động	Mô tả
@RenderBody()	Vị trí đặt nội dung trang
@RenderSection()	Vị trí đặt nội dung vùng
@ViewBag.<Property>	Vị trí đặt giá trị thuộc tính động trong ViewBag

Có thể cung cấp dữ liệu cho các vùng đánh dấu trên layout được thể hiện như sau:

- Giá trị thuộc tính động của @ViewBag.XYZ sẽ được đặt vào đúng vị trí @ViewBag.XYZ trong layout.
- @section xyz{data} của view sẽ được đặt vào vị trí @RenderSection(xyz) của layout.



- Tất cả các phần dữ liệu còn lại của view sẽ được đặt vào @RenderBody() của layout.
- Các thành phần Layout: @RenderBody() và @RenderSection()



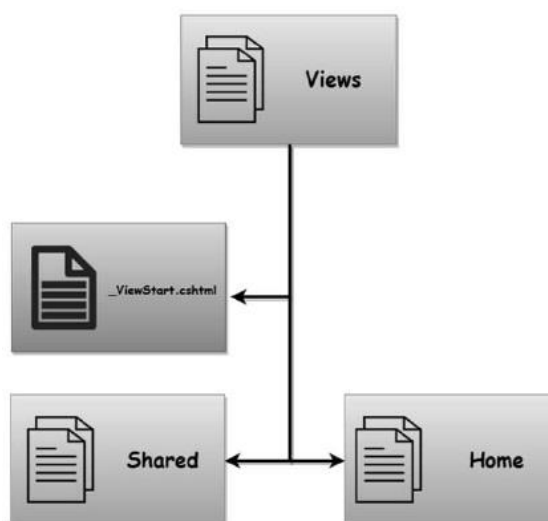
Layout view là một Razor view với phần mở rộng là \*.cshtml.

**Chỉ định bố cục:** Sử dụng thuộc tính Layout để chỉ định template cần dùng. Ví dụ:

```
@{
    Layout = "_Layout";
}
```

### 5.1.3 Tập tin \_ViewStart.cshtml, \_ViewImports.cshtml

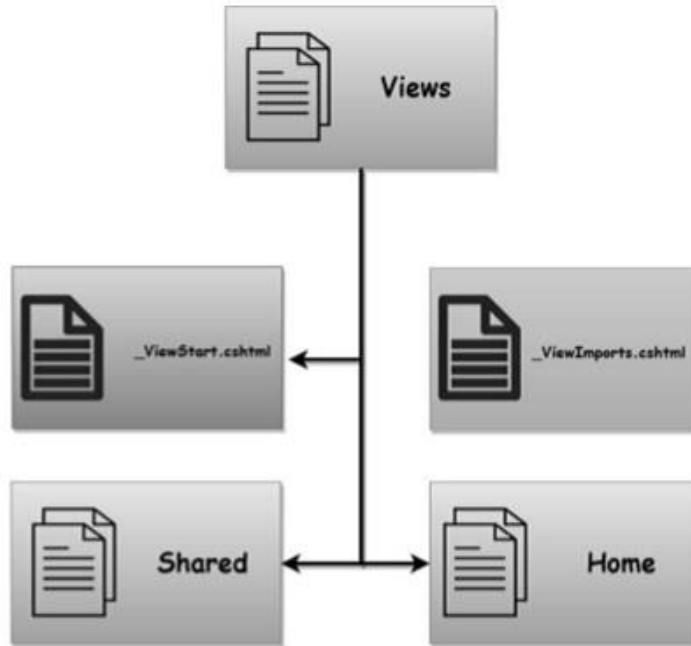
Bất kỳ một LayoutView nào cũng luôn tìm đến tập tin \_ViewStart.cshtml để thực thi mã bên trong file này trước khi thực thi mã của chính layout view đó. Vì vậy để chuẩn bị những gì chung cho tất cả các view bạn nên đặt mã ở tập tin \_ViewStart.cshtml.



Ví dụ tập tin \_ViewStart.cshtml chèn đoạn lệnh khai báo sử dụng layout chung là \_Layout.cshtml như sau:

```
_ViewStart.cshtml  + X
1  @{
2      Layout = "_Layout";
3  }
```

Tương tự các View có chèn khai báo đoạn mã chung thì bỏ vào \_ViewImports.cshtml.



```
_ViewImports.cshtml*  + X
1  @using CodeFirstDB
2  @using CodeFirstDB.Models
3  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
.
```

## 5.2 Bundles – đóng gói tài nguyên css và javascript

Trong layout thông thường bạn phải chuẩn bị đủ các tài nguyên css và script cần thiết cho các trang thành viên. Cũng như mọi trang web khác, bạn chỉ việc nhúng các tài nguyên này bằng các thẻ <link> và <script> là được.

```
<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
<link rel="stylesheet" href="~/css/site.css" />

<script src="~/lib/jquery/dist/jquery.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
```

Tuy nhiên với MVC bạn có cách làm khác khoa học hơn và dễ quản lý hơn rất nhiều đó là đóng gói các tài nguyên liên quan, sau đó nhúng gói tài nguyên đó vào.

**Bundling và minification là gì?**

Đóng gói và rút gọn là hai tối ưu hóa hiệu suất riêng biệt mà bạn có thể áp dụng trong ứng dụng web. Được sử dụng cùng nhau, gói và rút gọn sẽ cải thiện hiệu suất bằng cách giảm số lượng yêu cầu máy chủ và giảm kích thước của nội dung tĩnh được yêu cầu.

Để đóng gói javascript bạn sử dụng ScriptBundle, còn với CSS thì dùng StyleBundle.

## Bundling – Đóng gói

Bundling kết hợp nhiều tập tin vào một tập tin duy nhất. Tính năng nhóm giảm số lượng yêu cầu máy chủ cần thiết để hiển thị nội dung web, chẳng hạn như trang web. Bạn có thể tạo bất kỳ số gói riêng lẻ nào cho CSS, JavaScript, v.v. Ít tập hơn có nghĩa là ít yêu cầu HTTP hơn từ trình duyệt đến máy chủ hoặc từ dịch vụ cung cấp ứng dụng của bạn. Điều này giúp cải thiện hiệu suất tải trang đầu tiên.

## Minification – Tối thiểu hóa

Việc rút gọn sẽ xóa các ký tự không cần thiết khỏi mã mà không cần thay đổi chức năng. Kết quả là giảm kích thước đáng kể trong các nội dung được yêu cầu (chẳng hạn như tệp CSS, hình ảnh và JavaScript). Tác dụng phụ thường gặp của việc rút gọn bao gồm rút ngắn tên biến thành một ký tự và xóa nhận xét cũng như khoảng trắng không cần thiết.

Xem xét đoạn mã javascript dưới đây:

```
AddAltToImg = function (imageTagAndImageID, imageContext) {  
    ///    ///    // </summary>  
    ///    ///    ///<</signature>  
    var imageElement = $(imageTagAndImageID, imageContext);  
    imageElement.attr('alt', imageElement.attr('id').replace(/ID/, ''));  
}
```

Sẽ được rút gọn thành:

```
AddAltToImg=function(n,t){var  
i=$(n,t);i.attr("alt",i.attr("id").replace(/ID/,""))};
```

## Cấu hình đóng gói và rút gọn

Các mẫu dự án MVC và Razor Pages cung cấp tập tin cấu hình bundleconfig.json xác định các tùy chọn cho mỗi gói. Theo mặc định, một cấu hình gói đơn được xác định cho các tệp JavaScript tùy chỉnh (wwwroot / js / site.js) và tệp định kiểu (wwwroot / css / site.css):

```
// Configure bundling and minification for the project.  
// More info at https://go.microsoft.com/fwlink/?LinkId=808241  
[  
  {  
    "outputFileName": "wwwroot/css/site.min.css",  
    // An array of relative input file paths. Globbing patterns supported  
    "inputFiles": [  
      "wwwroot/css/site.css"  
    ]  
  },  
  {
```

```
"outputFileName": "wwwroot/js/site.min.js",  
"inputFiles": [  
  "wwwroot/js/site.js"  
],  
// Optionally specify minification options  
"minify": {  
  "enabled": true,  
  "renameLocals": true  
},  
// Optionally generate .map file  
"sourceMap": false  
}  
]
```

## 5.3 Module hóa giao diện

Khi giao diện quá phức tạp hoặc cần sử dụng lại một số thành phần giao diện thì cần module hóa các thành phần giao diện.

Trong MVC, bạn có thể tách các module giao diện trên các file riêng biệt, sau đó lắp trở lại thông qua 2 HTML helper sau: `@Html.Partial()` và `@Html.PartialAsync()`.

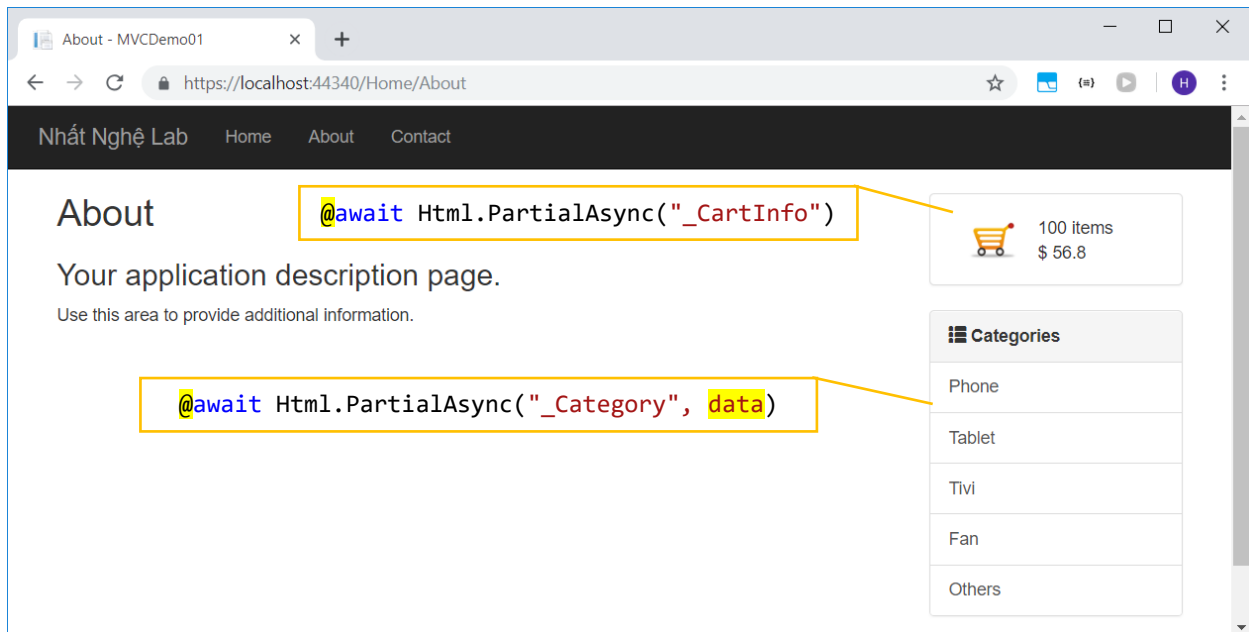
### 5.3.1 Sử dụng Partial View

Giả sử đã định nghĩa PartialView có tên `AuthorPartial.cshtml`. Để chèn vào View có các cách sau:

Các cách sử dụng một PartialView:

```
// Nếu view cùng thư mục hoặc nằm trong thư mục Shared  
@await Html.PartialAsync("AuthorPartial ")  
//hoặc:  
@{  
    await Html.RenderPartialAsync("AuthorPartial");  
}  
  
// Nếu view cùng thư mục  
@await Html.PartialAsync("AuthorPartial.cshtml")  
  
// Xác định view thông qua thư mục root, sử dụng "/" hoặc "~/"  
@await Html.PartialAsync("~/Views/Folder/AuthorPartial.cshtml")  
@await Html.PartialAsync("/Views/Folder/AuthorPartial.cshtml")  
  
// Xác định view thông qua địa chỉ tương đối  
@await Html.PartialAsync("../Account/AuthorPartial.cshtml")
```

Ví dụ:



### Các bước thực hiện

- ✓ Tạo view \_CartInfo.cshtml
- ✓ Tạo view \_Category.cshtml
- ✓ Thiết kế layout và nhúng các thành phần giao diện vào layout

## THỰC HIỆN

### Bước 1: Tạo view \_CartInfo.cshtml trong Views/Shared

Giao diện này chỉ gồm 1 ảnh và 2 thông tin tổng hợp của giỏ hàng là số mặt hàng và tổng tiền.

```
<style>
    .nn-cart ul {
        padding: 0px;
        margin: 0px;
        list-style: none;
        padding-top: 3px;
    }

    .nn-cart img {
        width: 100%;
    }
</style>

<div class="panel panel-default nn-cart">
    <div class="panel-body">
        <a class="col-md-5" asp-controller="ShoppingCart" asp-action="Index">
            
        </a>
        <ul class="col-md-7">
            <li>100 items</li>
            <li>$ 56.8</li>
        </ul>
    </div>
</div>
```

### Bước 2: Tạo view \_Category.cshtml trong Views/Shared

```
@model String[]
<div class="panel panel-default">
  <div class="panel-heading">
    <span class="glyphicon glyphicon-th-list"></span>
    <strong>Categories</strong>
  </div>
  <div class="list-group">
    @foreach (var cate in Model)
    {
      <a href="#" class="list-group-item">@cate</a>
    }
  </div>
</div>
```

### Bước 3: Thiết kế layout

Bạn hãy thiết kế lại phần thẻ div.body-content để chia giao diện thành 2 phần (9 cột và 3 cột). Phần đầu chứa nội dung trang web @RenderBody() phần còn lại nhúng partial view và action đã tạo ra ở trên vào.

Ngoài ra trong layout này bạn còn tìm thấy một partial view khác cũng được nhúng vào là \_LoginPartialView. Chúng ta sẽ tìm hiểu view này ở phần security của bài học sau này.

```
<div class="container body-content">
  <article class="col-md-9">@RenderBody()</article>
  <article class="col-md-3">
    @await Html.PartialAsync("_CartInfo")
    @await Html.PartialAsync("_Category", new string[]{"Phone", "Tablet", "Tivi",
"Fan", "Others"})
  </article>
</div>
```

## 5.3.2 Truyền dữ liệu cho PartialView

Sử dụng ViewData:

```
@await Html.PartialAsync("PartialName", customViewData)
```

Sử dụng Model:

```
@await Html.PartialAsync("PartialName", viewModel)
```

## 5.4 Phân vùng ứng dụng (Areas)

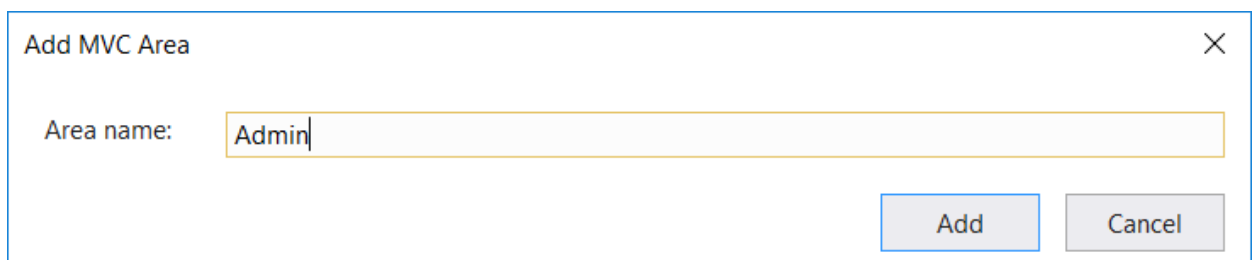
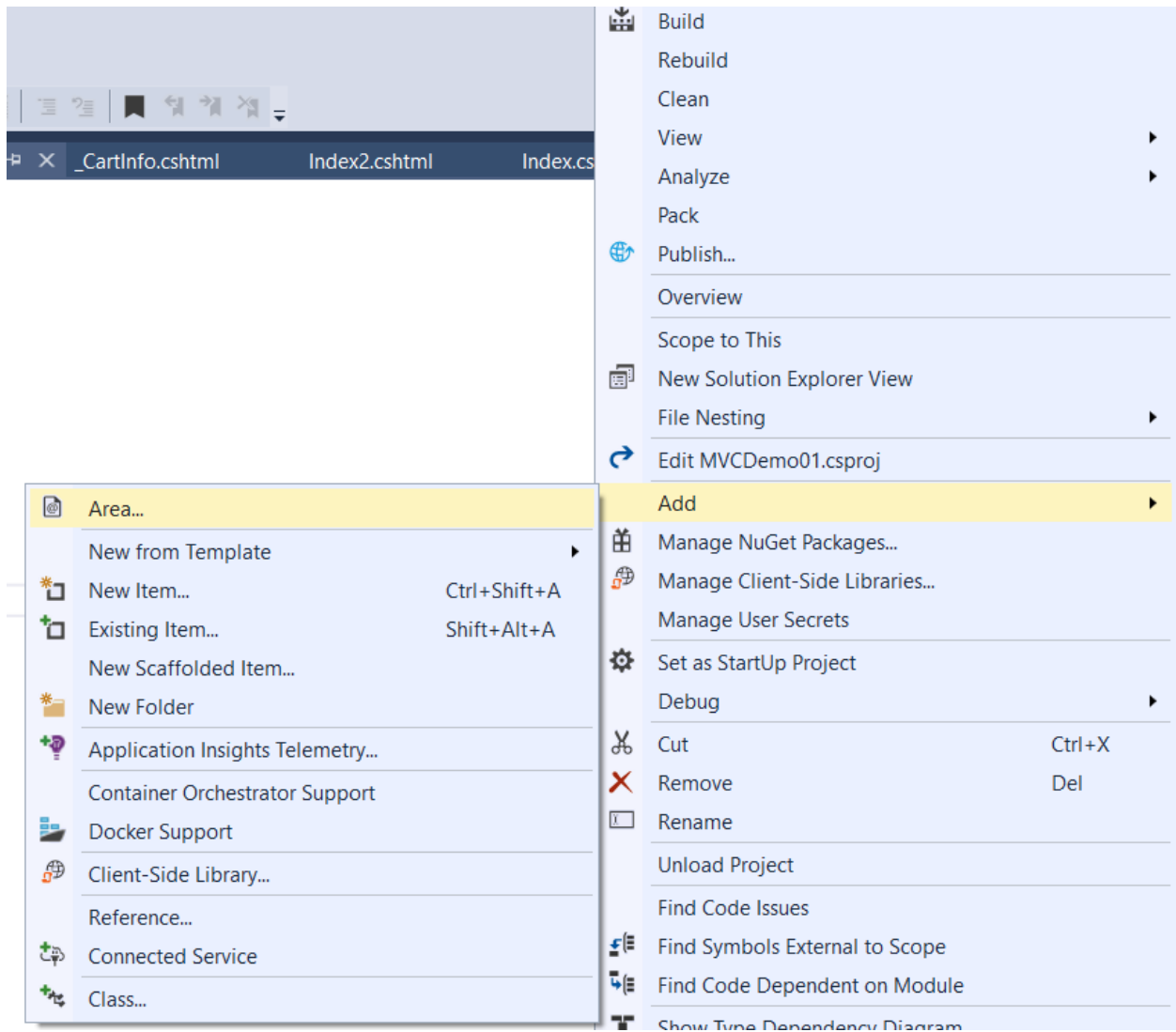
Khi ứng dụng web của bạn có nhiều module chức năng tách biệt thì nên phân chia thành nhiều vùng khác nhau. Làm như vậy sẽ tiện cho việc phát triển cũng như bảo mật riêng cho từng phân vùng.

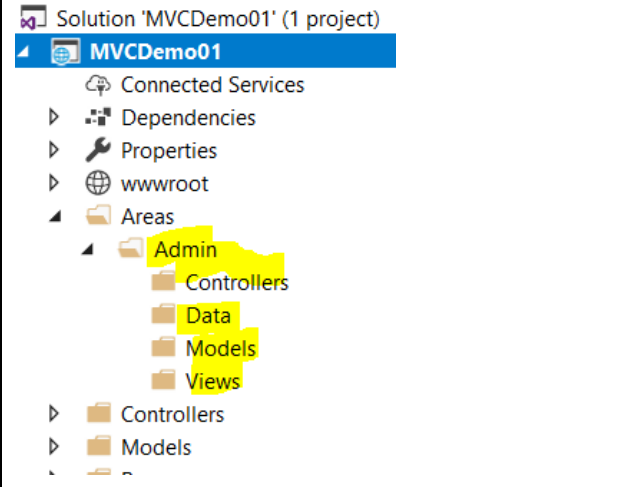
Trong một website thường có 2 vùng: một dành cho khách web và phần còn lại dành cho người quản trị. Công việc của khách web là tìm kiếm, tra cứu và đặt mua trong khi người quản trị là quản lý CSDL, báo cáo thống kê và phân quyền sử dụng.

### 5.4.1 Tạo phân vùng

Trong bài này, bạn được biết kỹ thuật phân vùng trong MVC và cấu hình namespace để tránh xung đột về tên của các action giữa các vùng.

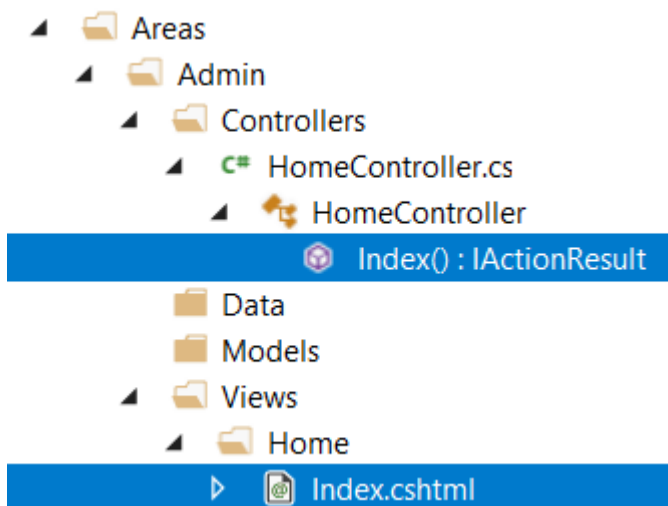
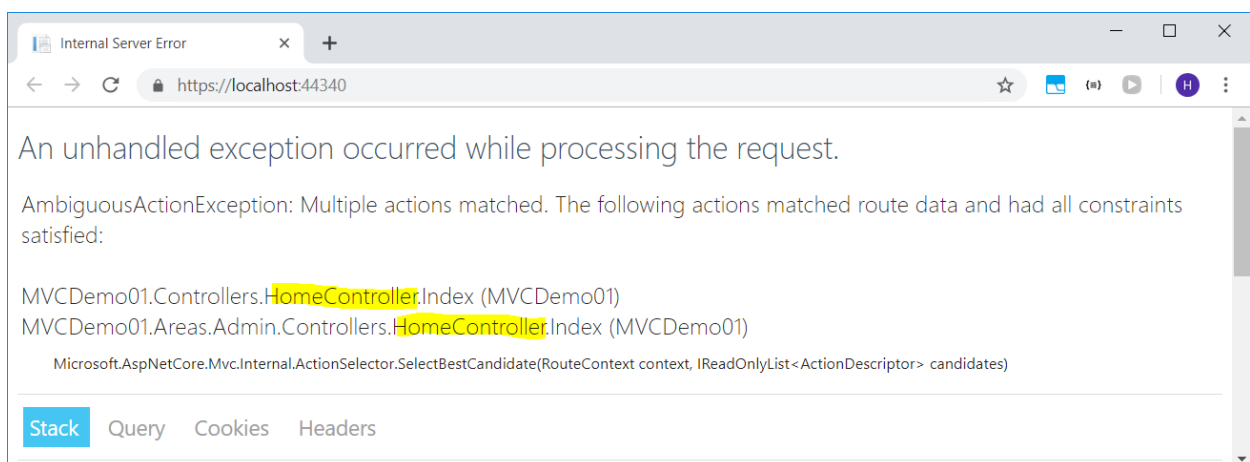
Chuột phải trên project, chọn **Add**, chọn **Area...**



	<p>Sau khi nhấp nút Add, bạn sẽ thấy một phân vùng có tên Admin được sinh ra.</p> <p>Trong mỗi phân vùng có cấu trúc như dự án chính, tức cũng bao gồm các thư mục để các thành phần MVC.</p>
---	---

### 5.4.2 Định tuyến

Nếu bạn tạo thêm một controller có tên HomeController bên trong phân vùng Admin và thêm action Index() tương ứng thì khi chạy `http://{host}:{port}/` sẽ nhận được trang thông báo lỗi.

Lý do là xung đột về tên. MVC tìm thấy có 2 HomeController bên trong hệ thống nên không biết chọn action của Home nào để chạy.

Để sửa lỗi này, bạn mở file Startup.cs và bổ sung thêm thuộc tính Area cho controller.



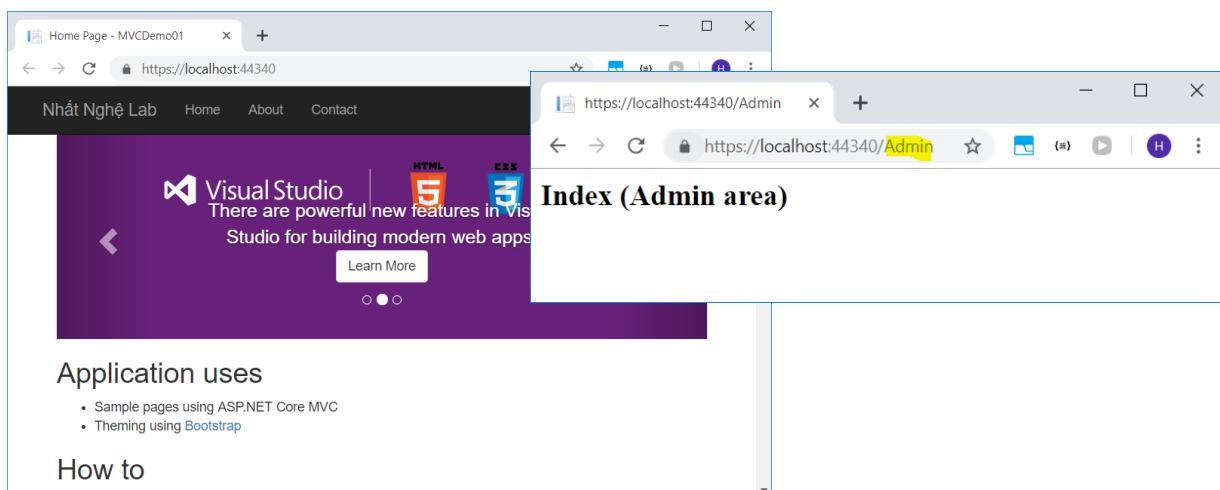
```
namespace MVCDemo01.Areas.Admin.Controllers
{
    [Area("Admin")]
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

và thêm định tuyến trong file Startup.cs:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");

    routes.MapRoute(
        name: "areas",
        template: "{area}/{controller=Home}/{action=Index}/{id?}");
});
```

Lúc đó các đường dẫn sẽ map với các controller khác nhau:



## Chương 6: CHIA SẼ DỮ LIỆU

**Sau khi học xong bài này, học viên có khả năng :**

- Trình bày và sử dụng được các đối tượng *Server, Session* và *Application*.
- Trình bày và sử dụng được các phương pháp truyền tham số trong *ASP.NET* : *GET, POST* và *Cross-Page*.
- Mô tả được cách sử dụng *Cookie/Session* để lưu thông tin.

### 6.1 Dẫn nhập

Khi MVC nhận một yêu cầu HTTP, nó sẽ xác định controller và action tương ứng để thực hiện (dựa vào phần định nghĩa Route).

Ví dụ: <http://localhost:1234/Product/Edit/2>

Theo cú pháp định tuyến sẽ gọi tới controller *Product*, action *Edit* và tham số *id* là 2.

```
public IActionResult Edit(int? id){
}
```

Chú ý: Đường dẫn URL không phân biệt chữ hoa hay chữ thường.

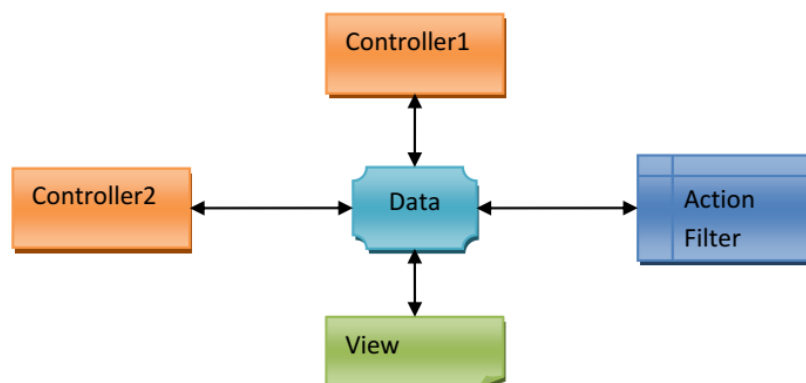
Ba cách cơ bản để ánh xạ HTTP request đến action:

- Form values: Sử dụng các giá trị phần tử trong form gửi lên bằng phương thức *POST*, kể cả các yêu cầu gửi lên bằng *jQuery POST*.
- Route values: Tập hợp các giá trị tham số được định tuyến
- Query strings: Thông qua chuỗi truy vấn trên URL

Lưu ý: Form values, route data và query strings đều được lưu trữ dạng cặp name-value.

#### Chia sẻ dữ liệu là gì?

Trong website có rất nhiều thành phần khác nhau. Các thành phần này được truy cập hoặc được gọi vào một thời điểm nào đó. Vấn đề đặt ra là làm sao để tạo một đối tượng dữ liệu ở thành phần này sau đó được sử dụng ở một thành phần khác tại cùng hoặc khác thời điểm tạo.



Hình 6-1 : Chia sẻ dữ liệu

Trong MVC dữ liệu được chia sẻ giữa các thành phần theo một số cách sau:

- ✓ Truyền dữ liệu từ controller cho view thông qua ViewBag và Model
- ✓ Chia sẻ dữ liệu theo phiên làm việc thông qua session
- ✓ Chia sẻ dữ liệu trên toàn ứng dụng thông qua Application
- ✓ Chia sẻ dữ liệu trên cùng một máy khác thông qua cookie

## 6.2 Truyền từ Controller qua View

### 6.2.1 Sử dụng ViewBag và ViewData

**ViewData** và **ViewBag** trong ASP.NET Core là những tập hợp dữ liệu dạng *weak types* (hay còn gọi là *loose types*), tức là chúng ta không cần định nghĩa trước kiểu dữ liệu một cách rõ ràng. Một số trường hợp sử dụng **ViewBag** và **ViewData**:

Truyền dữ liệu	Ví dụ
Controller và View	Truyền dữ liệu vào một Dropdown List (Select box).
View và Layout View	Thiết lập lại nội dung của element <b>&lt;title&gt;</b> trong Layout View từ một View con.
PartialView và View	Một Widget hiển thị dữ liệu phụ thuộc vào trang mà người dùng truy cập.

Vì là *weak types* nên **ViewBag** và **ViewData** chỉ được xử lý lúc *runtime* và sẽ không có kiểm tra kiểu dữ liệu lúc *compile-time* như *viewmodel* nên sẽ dễ bị lỗi hơn. Do đó chúng ta chỉ nên sử dụng chúng khi cần truyền một lượng dữ liệu nhỏ và dễ kiểm soát.

*Chú ý là ViewBag không sử dụng được trong Razor Pages.*

#### 6.2.1.1 ViewData

Là một `ViewDataDictionary` object (cài đặt cho interface `IDictionary<string, object>`) được truy xuất qua một *string key* (cho phép có khoảng trắng). Với kiểu dữ liệu dạng **string** thì chúng ta có thể lưu trữ và sử dụng trực tiếp, còn với kiểu dữ liệu dạng **object** thì khi sử dụng chúng ta sẽ phải ép kiểu (cast) sang kiểu dữ liệu xác định.

Ví dụ khi truyền một object từ Controller sang View:

Tạo model view Customer:

```
public class Customer
{
    public int CustomerID { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }

    public Customer()
    {
        CustomerID = 1;
        Name = "Nhất Nghệ";
        Address = "105 Bà Huyện Thanh Quan, Quận 3";
    }
}
```

Tạo action SomeAction() trong controller Home:

```
public IActionResult SomeAction()
{
    // Lưu dữ liệu kiểu string
    ViewData["Greeting"] = "Hello";

    // Lưu dữ liệu kiểu object
    ViewData["KhachHang"] = new KhachHang();

    return View();
}
```

Sử dụng ViewData bên View:

```
@{
    // Customer không phải là 1 string nên sẽ cần ép kiểu
    var customer = ViewData["KhachHang"] as Customer;
}
```

```
<p>Id :@customer.CustomerID</p>
<p>Name :@customer.Name </p>
<p>Name :@customer.Address </p>
```

### 6.2.1.2 ViewBag

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

Ví dụ 1:

```
public IActionResult AnotherAction()
{
    // Lưu dữ liệu kiểu string
    ViewBag.Greeting = "Hello";

    // Lưu dữ liệu kiểu object
    ViewBag.KhachHang = new Customer();

    return View();
}
```

Sử dụng ViewBag bên View:

```
@ViewBag.Greeting World!

<p>Id :@ViewBag.KhachHang.CustomerID</p>
<p>Name :@ViewBag.KhachHang.Name </p>
<p>Name :@ViewBag.KhachHang.Address </p>
```

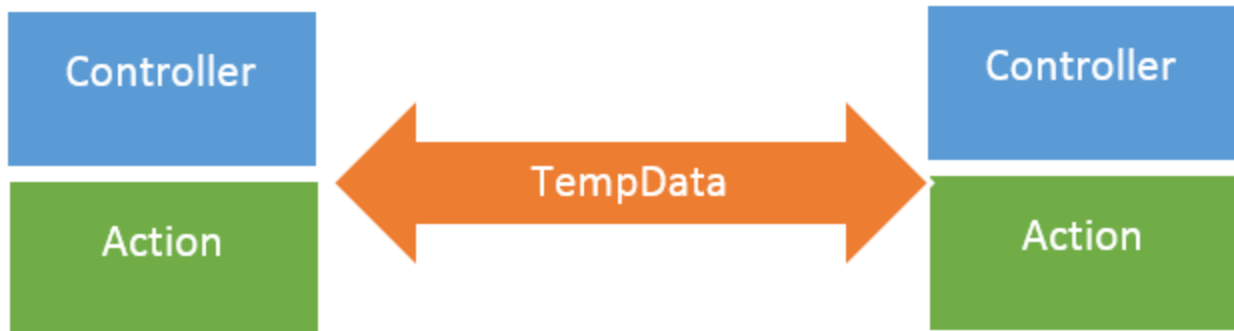
Ví dụ 2: Truyền đối tượng

```
ViewBag.film = new Film{Name = "Transformer", Year = 2017}
```

Thì bên view lấy ra như sau: @ViewBag.film.Name.

### 6.2.1.3 Sử dụng TempData

Tương tự ViewData và ViewBag, TempData cũng dùng để truyền dữ liệu ra view. Tuy nhiên sẽ hơi khác một chút, đó là TempData sẽ tồn tại cho đến khi nó được đọc. Tức là ViewBag và ViewData chỉ hiển thị được dữ liệu ngay tại trang người dùng truy cập, còn TempData có thể lưu lại và hiển thị ở một trang sau đó và nó chỉ biến mất khi người dùng đã "đọc" nó.



TempData thường được ứng dụng để hiển thị các thông báo thành công, thất bại ở trang kế tiếp.

#### Ví dụ với ứng dụng Quản lý Book có 2 trang Danh sách Book và Tạo mới Book.

Khi người dùng *Tạo mới Book* xong sẽ được chuyển qua trang *Danh sách Book* kèm một thông báo tạo book thành công. Lúc này chúng ta sẽ không thể dùng ViewBag hay ViewData để lưu thông báo được vì thông báo nằm ở trang khác (Action khác). Chúng ta sẽ dùng TempData để lưu thông báo ở trang *Tạo mới Book* và hiển thị ra ở trang *Quản lý Book*:

```
namespace CodeFirstDB.Controllers
{
    public class BookController : Controller
    {
        // Trang Danh sách Book
        public IActionResult Index()
        {
            // Logic hiển thị Book ...

            // Render View
            return View();
        }

        // Trang Tạo mới Book
        public IActionResult Create()
        {
            // Logic tạo mới Book ...

            // Lưu message vào TempData
            TempData["StatusMessage"] = "Create book successfully";

            // Điều hướng sang trang Danh sách Book
            return RedirectToAction(nameof(Index));
        }
    }
}
```

Ngoài View của trang *Danh sách Book* chỉ việc hiển thị message:

```
@TempData["StatusMessage"]
```

Một cách khác đó là tạo một thuộc tính cho Controller và gán cho nó Attribute **TempData** sau đó sử dụng thuộc tính đó để lưu message:

```
public class BookController : Controller
{
    [TempData]
    public string StatusMessage { get; set; }
}
```

## TempData Provider

Có 2 loại *TempData Provider* là **cookie-based** (lưu dữ liệu trong Cookie) và **session-base** (lưu dữ liệu trong Session). Việc chọn loại Provider nào là tùy mục đích sử dụng:

- Trong ứng dụng có cần dùng đến Session state hay không, nếu không thì nên dùng *cookie-based provider*.
- Nếu dữ liệu lưu trong TempData lớn thì nên dùng *session-based provider* vì Cookie có thể bị hạn chế dung lượng bởi trình duyệt và sẽ làm tăng thêm chi phí request (cost).

Mặc định ASP.NET Core 2 sử dụng *cookie-base provider*, để chuyển qua dùng *session-base provider* thì ta sẽ phải cấu hình trong file **Startup.cs**:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().AddSessionStateTempDataProvider();
    services.AddMvc();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
{
    app.UseStaticFiles();
    app.UseSession();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

### 6.2.2 Sử dụng model

Trong controller bạn có thể truyền dữ liệu cho view thông qua return View(model) hay return PartialView(model). Trong đó đối tượng model là một object với kiểu bất kz. Sau đó trong view bạn có thể khai thác thông tin của đối tượng này thông qua đối tượng Model.

Nếu trong controller bạn có đoạn mã:

- ✓ var model = new StudentInfo{Id="SV001", Name="Tuấn"};
- ✓ return View(model);

Thì trong view bạn có thể truy xuất thông tin sinh viên thông qua đối tượng Model như sau:

- ✓ Id: @Model.Id

✓ Name: @Model.Name

Ví dụ: sử dụng model để truyền thông tin sinh viên từ controller sang view. Đối tượng được sử dụng để chuyển thông tin cho Model của View là StudentInfo.

Model: **StudentInfo.cs**

Lớp này gồm 3 thuộc tính Id, Name và Marks. Mã nguồn như sau:

```
public class StudentInfo
{
    public string Id { get; set; }
    public string Name { get; set; }
    public double Marks { get; set; }
}
```

Controller: **StudentController.cs**

Một đối tượng student được tạo ra và chuyển cho view thông qua lệnh return View(model).

```
public class StudentController : Controller
{
    ...
    public ActionResult Detail()
    {
        // Tạo đối tượng
        var model = new StudentInfo
        {
            Id = "SV001",
            Name = "Nguyễn Ngọc Hân",
            Marks = 9.5
        };
        // Truyền đối tượng model cho view return View(model);
        return View(model);
    }
}
```

View: **Detail.cshtml**

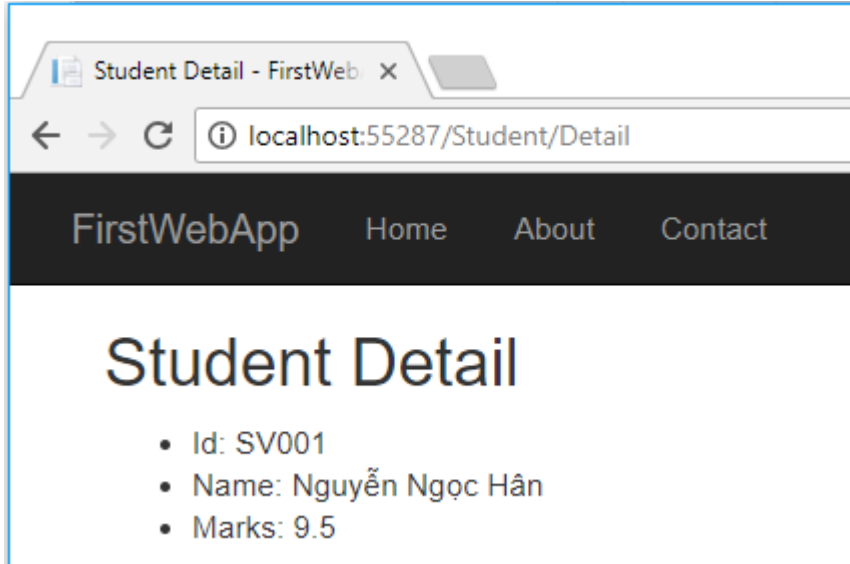
Sử dụng @Model để truy xuất thông tin chia sẻ của controller. @Model chính là đối tượng được truyền từ controller thông qua lệnh return View(model).

```
@{
    ViewBag.Title = "Student Detail";
}

<h2>Student Detail</h2>

<ul>
    <li>Id: @Model.Id</li>
    <li>Name: @Model.Name</li>
    <li>Marks: @Model.Marks</li>
</ul>
```

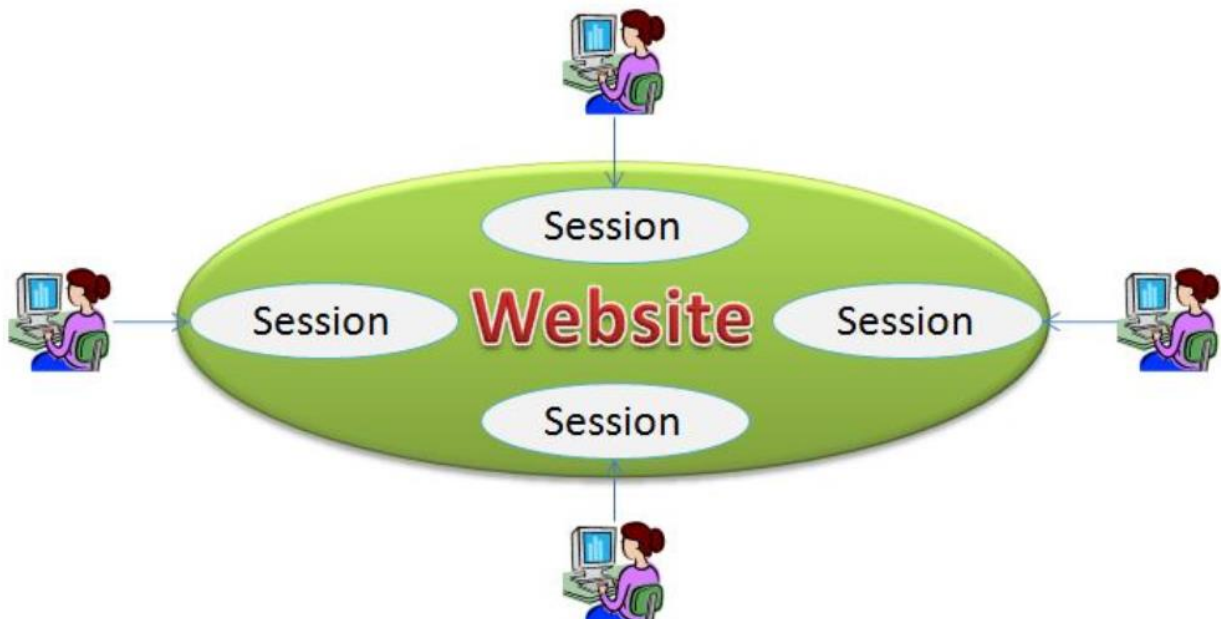
Kết quả:



### 6.3 Session

Chia sẻ dữ liệu theo phiên là kỹ thuật chia sẻ dữ liệu cơ bản của ứng dụng công nghệ web. Theo đó dữ liệu được lưu trữ trong đối tượng Session sẽ được dùng chung cho các thành phần (controller, view, action filter...) làm việc trong cùng một phiên làm việc.

Về mặt bản chất thì Session được server cấp phát riêng cho từng user để lưu dữ liệu của riêng mình. Vùng nhớ đặc biệt này chỉ tồn tại trong phiên làm việc. Kết thúc phiên thì vùng nhớ này được giải phóng. Vì tính chất này nên với các website có số khách truy cập đồng thời đông và sử dụng session để duy trì dữ liệu lớn thì rất có thể dẫn đến thiếu bộ nhớ và website sẽ hoạt động không hiệu quả.

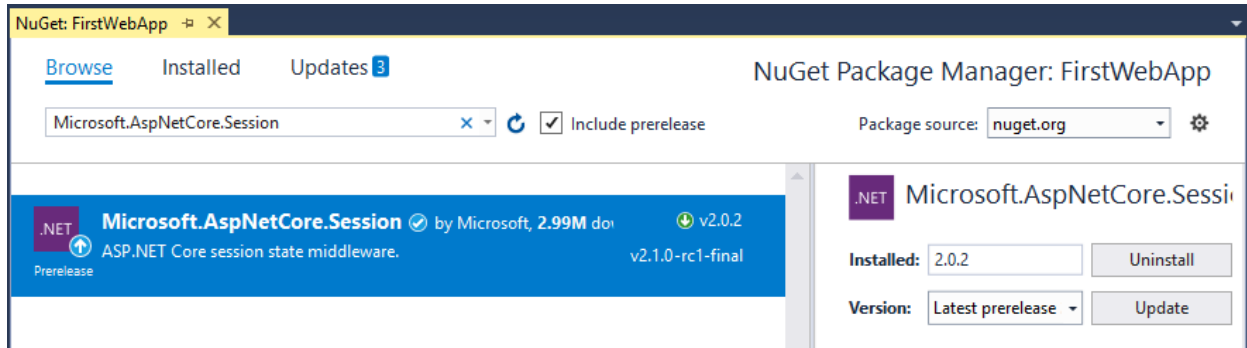


Hình 6-2 : Session chia sẻ dữ liệu riêng của từng phiên làm việc



### 6.3.1 Cài đặt & Cấu hình

Mở cửa sổ "Manage NuGet Packages..." để tiến hành cài thư viện "Microsoft.AspNetCore.Session".



Cấu hình thời gian hết hạn Session trong class **Startup** bằng cách thêm vào hàm **ConfigureServices ()** đoạn code sau:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDistributedMemoryCache();
    services.AddSession(options => {
        //You can set Time
        options.IdleTimeout = TimeSpan.FromMinutes(1);
    });
    services.AddMvc();
}
```

và bổ sung lệnh **app.UseSession();** vào hàm **Configure()** trong lớp **Startup**:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())...
    else...

    app.UseStaticFiles();
    app.UseSession();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

### 6.3.2 Sử dụng Session

Khai báo biến Session ở controller:

```
// Requires using Microsoft.AspNetCore.Http;
HttpContext.Session.SetString(SessionKeyName, "HIENLTH");
HttpContext.Session.SetInt32(SessionKeyYearsMember, 3);
```

Ở Razor view lấy giá trị Session:

```
@using Microsoft.AspNetCore.Http
Session Value = @Context.Session.GetString("_Name")
```

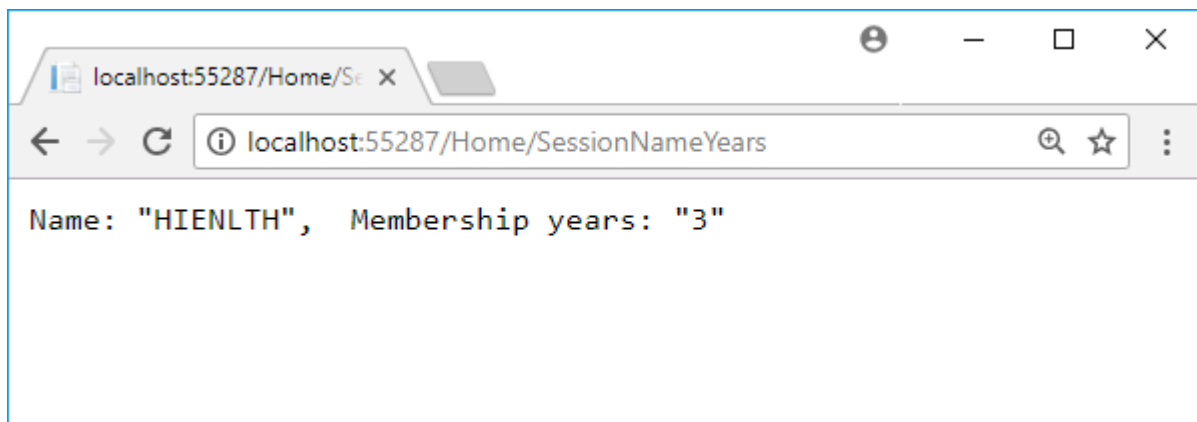
Ví dụ trên Controller Home:

```
public class HomeController : Controller
{
    const string SessionKeyName = "_Name";
    const string SessionKeyYearsMember = "_YearsMember";
    const string SessionKeyDate = "_Date";
    public IActionResult Index()
    {
        // Requires using Microsoft.AspNetCore.Http;
        HttpContext.Session.SetString(SessionKeyName, "HIENLTH");
        HttpContext.Session.SetInt32(SessionKeyYearsMember, 3);
        return RedirectToAction("SessionNameYears");
    }

    public IActionResult SessionNameYears()
    {
        var name = HttpContext.Session.GetString(SessionKeyName);
        var yearsMember = HttpContext.Session.GetInt32(SessionKeyYearsMember);

        return Content($"Name: \"{name}\", Membership years: \"{yearsMember}\"");
    }
}
```

Kết quả chạy:



Để đơn giản việc code, có thể khai báo biến `_session` kiểu `HttpContext.Session`:

```
public class SomeOtherClass
{
    private readonly IHttpContextAccessor _httpContextAccessor;
    private ISession _session =>
        _httpContextAccessor.HttpContext.Session;

    public SomeOtherClass(IHttpContextAccessor httpContextAccessor)
    {
        _httpContextAccessor = httpContextAccessor;
    }

    public void TestSet()
    {
        _session.SetString("Test", "Ben Rules!");
    }
}
```

```
public void TestGet()
{
    var message = session.GetString("Test");
}
}
```

### Lưu trữ đối tượng phức:

Sử dụng `JSON SerializeObject()` để lưu và lấy lại giá trị bằng `JSON DeSerializeObject()`.

```
public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonConvert.SerializeObject(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);

        return value == null ? default(T) :
            JsonConvert.DeserializeObject<T>(value);
    }
}
```

### Sử dụng:

```
var myComplexObject = new MyClass();
HttpContext.Session.Set<MyClass>("Test", myComplexObject);
```

hoặc:

```
var myComplexObject = HttpContext.Session.Get<MyClass>("Test");
```

### Ví dụ:

```
public IActionResult SetDate()
{
    // Requires you add the Set extension method mentioned in the article.
    HttpContext.Session.Set<DateTime>(SessionKeyDate, DateTime.Now);
    return RedirectToAction("GetDate");
}

public IActionResult GetDate()
{
    // Requires you add the Get extension method mentioned in the article.
    var date = HttpContext.Session.Get<DateTime>(SessionKeyDate);
    var sessionTime = date.TimeOfDay.ToString();
    var currentTime = DateTime.Now.TimeOfDay.ToString();

    return Content($"Current time: {currentTime} - "
```

```
+ $"session time: {sessionTime}");  
}
```

### 6.3.3 Ví dụ áp dụng

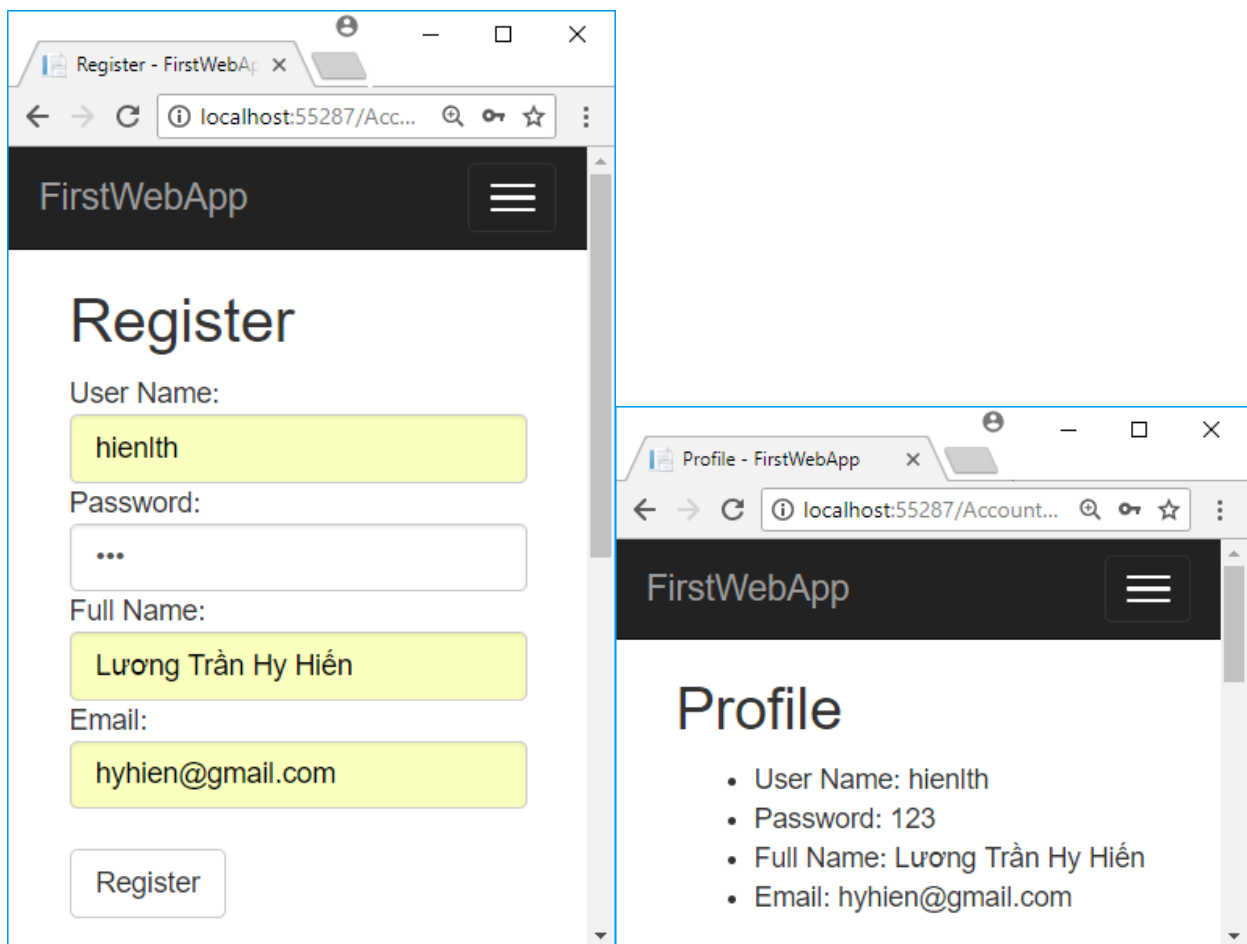
Với tính chất của session như vậy cho phép duy trì thông tin riêng tư của từng phiên khác nhau. Cụ thể là:

- Duy trì giỏ hàng
- Duy trì thông tin người dùng
- Duy trì thông tin về giao diện tùy biến cho từng phiên...

Ví dụ sau đây giúp bạn hiểu hơn về sử dụng session để chia sẻ thông tin giữa các thành phần trong cùng phiên làm việc.

#### MÔ TẢ:

- Vào trang đăng ký nhập thông tin sinh viên và nhấn nút **[Lưu]** thì thông tin của sinh viên được lưu lại trong Session.
- Vào trang tài khoản để xem lại thông tin đã đăng ký trước đó được lấy từ Session. Nếu trong session chưa có thông tin (nghĩa là chưa đăng ký) thì khi vào trang này sẽ tự động chuyển về trang đăng ký.
- Nhấp vào liên kết **Log Off** để xóa user khỏi Session và trở về trang đăng ký.



#### THỰC HIỆN

Để hoàn thành ví dụ trên, bạn cần phải thực hiện các bước

Bước 1: Tạo lớp model UserInfo dùng để tiếp nhận thông tin form và duy trì trong session

Bước 2: Tạo controller AccountController gồm 3 action:

- GET: /Account/Register: hiển thị giao diện đăng ký.
- POST: /Account/Register: tiếp nhận thông tin user và lưu vào session
- GET: /Account/Profile: hiển thị thông tin tài khoản đã đăng ký. Nếu không tồn tại trong session (chưa đăng k{}) thì chuyển về GET: /Account/Register.
- GET: /Account/LogOff: xóa session user và trở về trang đăng ký.

Bước 3: Tạo view cho action Register và Profile

### **Bước 1: Tạo lớp model UserInfo dùng để tiếp nhận thông tin form và duy trì trong session**

```
public class UserInfo
{
    public string UserName { get; set; }
    public string Password { get; set; }
    public string FullName { get; set; }
    public string Email { get; set; }
}
```

### **Bước 2: Tạo controller AccountController gồm 3 action**

```
public class AccountController : Controller
{
    private readonly IHttpContextAccessor _httpContextAccessor;
    private ISession _session => _httpContextAccessor.HttpContext.Session;
    public AccountController(IHttpContextAccessor httpContextAccessor)
    {
        _httpContextAccessor = httpContextAccessor;
    }

    // GET: /Account/Register
    public ActionResult Register()
    {
        return View();
    }

    // POST: /Account/Register
    [HttpPost]
    public ActionResult Register(UserInfo model)
    {
        // Lưu thông tin user vào session
        _session.Set<UserInfo>("user", model);
        //chuyển trang xem thông tin Session
        return RedirectToAction("Profile", "Account");
    }

    // GET: /Account/Profile
    public ActionResult Profile()
    {
        // Chuyển về trang đăng ký nếu user chưa có trong session
        if (_session.Get<UserInfo>("user") == null)
        {
            return RedirectToAction("Register", "Account");
        }
        return View(_session.Get<UserInfo>("user"));
    }
}
```

```
    }  
  
    // GET: /Account/LogOff  
    public ActionResult LogOff()  
    {  
        _session.Remove("user");  
        return RedirectToAction("Register", "Account");  
    }  
}
```

**Bước 3:** Tạo view cho action Register() và Profile()

### Register.cshtml

```
@{  
    ViewData["Title"] = "Register";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>Register</h2>  
  
<form asp-action="Register" asp-controller="Account" method="post">  
    <div>User Name:</div>  
    <input name="UserName" class="form-control" />  
  
    <div>Password:</div>  
    <input name="Password" type="password" class="form-control" />  
  
    <div>Full Name:</div>  
    <input name="FullName" class="form-control" />  
  
    <div>Email:</div>  
    <input name="Email" class="form-control" />  
  
    <br />  
    <input type="submit" value="Register" class="btn btn-default" />  
</form>
```

### Profile.cshtml

```
@{  
    ViewData["Title"] = "Profile";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>Profile</h2>  
<!--Hiển thị thông tin user-->  
<ul>  
    <li>User Name: @Model.UserName</li>  
    <li>Password: @Model.Password</li>  
    <li>Full Name: @Model.FullName</li>  
    <li>Email: @Model.Email</li>  
</ul>  
<!--Liên kết đăng xuất-->  
<a asp-controller="Account" asp-action="LoggOff">Log Off</a>
```

## Chương 7: Razor & Helper

### 7.1 Razor

#### 7.1.1 Giới thiệu

Razor là 1 ngôn ngữ ngắn gọn, rõ ràng và hữu ích, mà nó cho phép bạn tạo ra các giao diện cho ứng dụng ASP.NET MVC trong khi vẫn giữ được sự phân chia rõ ràng, khả năng có thể kiểm tra, và sự phát triển dựa trên mô hình lập trình. Các lập trình viên ASP.NET MVC đang tìm kiếm cho mình 1 ngôn ngữ có cú pháp rõ ràng, ngắn gọn, và bây giờ nó đã được xây dựng sẵn với ngôn ngữ quen thuộc là C#.

Trong Razor bạn cần hiểu rõ các khái niệm và qui ước sau:

- ✓ Khối mã razor được đặt trong @ {...}
- ✓ Biểu thức nội tuyến (các biến và chức năng) bắt đầu với @
- ✓ Mã lệnh kết thúc bằng dấu chấm phẩy
- ✓ Biến được khai báo với từ khóa var
- ✓ Chuỗi được đóng mở bằng dấu nháy kép
- ✓ Mã C# phân biệt hoa thường
- ✓ File C# có phần mở rộng .cshtml

Sau đây là ví dụ viết mã với Razor:

```
<!-- Khối lệnh đơn -->
@{ var message = "Hello World"; }

<!-- Biểu thức nội tuyến -->
<p>Giá trị của message là: @message</p>

<!-- Khối nhiều dòng mã lệnh -->
@{
    var greeting = "Welcome to our site!";
    var weekDay = DateTime.Now.DayOfWeek;
    var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>Lời chào là: @greetingMessage</p>
```

#### 7.1.2 Làm thế nào nó làm việc?

Razor là một cú pháp lập trình đơn giản cho việc nhúng mã chạy phía máy chủ trong các trang web. Cú pháp Razor được dựa trên cú pháp ASP nhưng được thiết kế đặc biệt để tạo các ứng dụng web thuận tiện hơn.

Cú pháp Razor cung cấp cho bạn tất cả sức mạnh của ASP, nhưng dễ học hơn đối với người mới vào nghề và làm hiệu quả hơn đối với các chuyên gia.

Các trang web như Razor có thể được mô tả như các trang HTML với hai loại nội dung: nội dung HTML và mã Razor.

Khi máy chủ lần đọc trang web nó chạy mã Razor trước khi gửi trang HTML cho trình duyệt. Các mã được thực thi trên máy chủ có thể thực hiện nhiệm vụ mà không thể được thực hiện trong trình duyệt, ví dụ như truy cập vào một cơ sở dữ liệu máy chủ. Mã máy chủ có thể tạo ra nội

dung HTML động trước khi nó được gửi đến trình duyệt. Phía trình duyệt nếu bạn xem mã nguồn thì chỉ thấy HTML giống như trang web tĩnh mà không thấy mã Razor vì đã được thực thi trên server.

Các trang web với cú pháp Razor viết bằng C# có phần mở rộng .cshtml, viết bằng VB sẽ có phần mở rộng là .vbhtml.

### 7.1.3 Làm việc với các đối tượng

Viết mã phía server thường liên quan đến các đối tượng. DateTime là đối tượng điển hình được xây dựng trong C# đối tượng (bạn cũng có thể xây dựng đối tượng riêng của mình). Bạn có thể gọi các phương thức hoặc sử dụng các thuộc tính của đối tượng đó.

Sau đây là ví dụ truy xuất các thuộc tính của đối tượng DateTime:

```
<table border="1">
  <tr>
    <th width="100px">Name</th>
    <td width="100px">Value</td>
  </tr>
  <tr>
    <td>Day</td>
    <td>@DateTime.Now.Day</td>
  </tr>
  <tr>
    <td>Hour</td>
    <td>@DateTime.Now.Hour</td>
  </tr>
  <tr>
    <td>Minute</td>
    <td>@DateTime.Now.Minute</td>
  </tr>
  <tr>
    <td>Second</td>
    <td>@DateTime.Now.Second</td>
  </tr>
</table>
```

### 7.1.4 Câu lệnh điều khiển

Bạn có thể viết câu lệnh rẽ nhánh if, if...else hay if...else if...else; switch...case; lặp (while, for, do, foreach)... trong khối mã như trong C#

Ví dụ 1

```
@{
  var txt = "";
  if (DateTime.Now.Hour > 12)
  {
    txt = "Good Evening";
  }
  else
  {
    txt = "Good Morning";
  }
}
```



```
<html>
<body>
  <p>The message is @txt</p>
</body>
```

Ví dụ 2:

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

Ví dụ 3:

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

</html>

### 7.1.5 Bảng tham khảo lệnh Razor

Khái niệm	Mã Razor
Khởi mã	<pre>@{     int x = 123;     string y = "because."; }</pre>
Biểu thức (đã mã hóa HTML)	<code>&lt;span&gt;@model.Message&lt;/span&gt;</code>
Biểu thức (chưa mã hóa HTML)	<pre>&lt;span&gt;     @Html.Raw(model.Message) &lt;/span&gt;</pre>
Kết hợp text và HTML	<pre>@foreach (var item in items) {     &lt;span&gt;@item.Prop&lt;/span&gt; } </pre>
Trộn code và text	<pre>@if (foo) {     &lt;text&gt;Plain Text&lt;/text&gt; } </pre>
	<pre>@if (foo) {     @:Plain Text is @bar } </pre>
Khởi using	<pre>@using (Html.BeginForm()) {     &lt;input type="text" value="input here"&gt; } </pre>
Địa chỉ email	Hi philha@example.com
Biểu thức (tường minh)	<code>&lt;span&gt;ISBN@(isbnNumber)&lt;/span&gt;</code>
Mã hóa ký hiệu @	<pre>&lt;span&gt;     In Razor, you use the     @@foo to display the value     of foo &lt;/span&gt;</pre>
Chú thích phía server	<pre>@*     This is a server side     multiline comment *@</pre>
Trộn biểu thức và text	Hello @title. @name

## 7.2 Tag Helper

Một trong những tính năng mới của ASP.NET Core là Tag Helper. Tag Helper cho phép người lập trình viết mã trên server-side sau đó render thẻ HTML chuẩn trong Razor file, nhằm mang lại trải nghiệm người dùng.

Một số điểm nổi bật của Tag Helper:

- ✓ Làm sao cho giống thẻ HTML nhất.
- ✓ Hỗ trợ IntelliSense phong phú.
- ✓ Viết mã mạnh mẽ, tin cậy và dễ bảo trì.

Khai báo sử dụng Tag Helper trong file `_ViewsImports.cshtml`

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

## 7.2.1 Anchor Tag Helper

Sử dụng:

```
@Html.ActionLink("Register", "Register", "Account")
```

```
<a href="@Url.Action("Register", "Account")">Register</a>
```

Đều cho kết quả như nhau:

```
<a href="/Account/Register">Register</a>
```

Sử dụng anchor tag helper ta có thể viết dạng thuộc tính bắt đầu bởi **asp-...** ngay trong thẻ HTML:

```
<a asp-controller="Account" asp-action="Register">Register</a>
```

Danh sách các thuộc tính:

Thuộc tính	Mô tả
asp-action	Tên action của Controller
asp-area	Tên của MVC Area
asp-controller	Tên của MVC Controller
asp-host	Tên domain
asp-protocol	Giao thức http, https, ftp, ...
asp-all-route-data	Truyền dictionary route data  <pre>@{     var d = new Dictionary&lt;string, string&gt;     {         { "key1", "value1" },         { "key2", "value2" }     }; }</pre> <pre>&lt;a asp-all-route-data="d"&gt;Click&lt;/a&gt;</pre> Kết quả: <pre>&lt;a href="/Page?key1=value1&amp;key2=value2"&gt;Click&lt;/a&gt;</pre>
asp-fragment	VD:  <pre>&lt;a asp-fragment="notes"&gt;Click&lt;/a&gt;</pre> tương đương với: <pre>&lt;a href="/Page#notes"&gt;Click&lt;/a&gt;</pre>
asp-page	<pre>&lt;a asp-page="page"&gt;Click&lt;/a&gt;</pre> tương đương với: <pre>&lt;a href="/Page"&gt;Click&lt;/a&gt;</pre>
asp-route	Tên của route

Ngoài ra ta còn thêm thông số định tuyến bằng cách sử dụng thuộc tính **asp-route-....** Ví dụ:

```
<a asp-controller="Product" asp-action="Display"
    asp-route-id="@ViewBag.ProductId">
    View Details
</a>
```

Ngoài ra, để có thể xuất hiện HTML như sau:

```
<a href="https://aspecificdomain.com/Account/Register#fragment">Register</a>
```

Trước đây, nếu sử dụng HTML helper ta sẽ viết:

```
@Html.ActionLink("Register", "Register", "Account", "https",
    "aspecificdomain.com", "fragment", null, null)
```

Nhưng sử dụng tag helper có thể route như sau:

```
<a asp-controller="Account" asp-action="Register"
    asp-protocol="https" asp-host="asepecificdomain.com"
    asp-fragment="fragment">Register</a>
```

## 7.2.2 Các Model Helper

Giả sử đã có Model:

```
public class Movie
{
    public int ID { get; set; }
    public string Title { get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Genre { get; set; }
    public decimal Price { get; set; }
}
```

Thì phần viết mã ở view cho phần nhập liệu Model:

```
<label asp-for="Movie.Title"></label>
```

Sẽ cho mã HTML tương ứng:

```
<label for="Movie_Title">Title</label>
```

Ví dụ dành cho ô nhập password:

```
<div class="form-group">
    <th:label asp-for="Password" class="col-md-2"></th:label>
    <div class="col-md-10">
        <th:input asp-for="Password" class="form-control" />
        <span asp-validation-for="Password" class="text-danger"></span>
    </div>
</div>
```

## 7.2.3 Form Tag Helper

Khởi tạo form với khai báo action và route:

```
<form asp-controller="Demo" asp-action="Register" method="post">
    <!-- Input and Submit elements -->
</form>
```

Bổ sung thuộc tính chỉ định return Url: `asp-route-returnurl="@ViewData["ReturnUrl"]"`

## Input Tag Helper

Cú pháp: `<input asp-for="<Expression Name>" />`

- ✓ Input tag Helper sẽ sinh ra Id, Name tương ứng với giá trị trong asp-for.
- ✓ Tự động sinh kiểu type phụ thuộc vào data annotation của thuộc tính trong model, không ghi đè nếu có chỉ định trước.

Bảng mapping thuộc tính giữa kiểu dữ liệu C# với loại thẻ input:

.NET type	Input Type
Bool	type="checkbox"
String	type="text"
DateTime	type="datetime"
Byte	type="number"
Int	type="number"
Single, Double	type="number"

Bảng mapping giữa annotation với thẻ input:

Attribute	Input Type
[EmailAddress]	type="email"
[Url]	type="url"
[HiddenInput]	type="hidden"
[Phone]	type="tel"
[DataType(DataType.Password)]	type="password"
[DataType(DataType.Date)]	type="date"
[DataType(DataType.Time)]	type="time"

## 7.2.4 Tag Helper tùy biến

Chúng ta có thể tùy chọn Tag Helper cho mục đích riêng của mình bằng cách xây dựng lớp con kế thừa lớp `TagHelper`, sau đó chỉ định các thuộc tính tag và quan trọng nhất là override phương thức `Process()`:

Xây dựng các Model cần dùng:

```
public class EmployeesViewModel
{
    public List<Employee> Employees { get; set; }
}

public class Employee
{
    public string Name { get; set; }
    public string JobTitle { get; set; }
    public string Profile { get; set; }
    public List<Friend> Friends { get; set; }
}

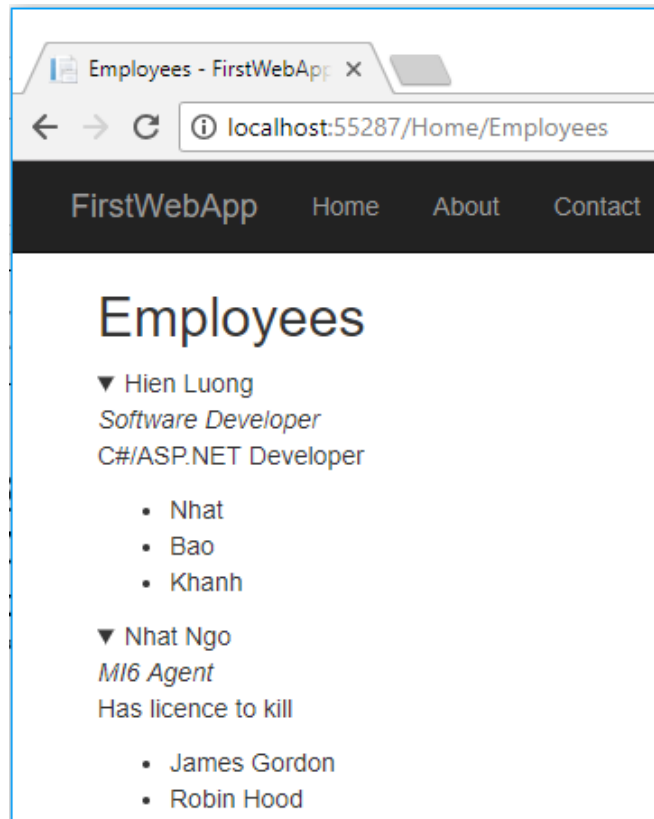
public class Friend
{
    public string Name { get; set; }
}
```

Xây dựng action `Employees()` để hiển thị `EmployeeViewModel`:

```
public IActionResult Employees()
{
    var model = new EmployeesViewModel
    {
        Employees = new List<Employee>
        {
            new Employee {
                Name = "Hien Luong",
                JobTitle = "Software Developer",
                Profile = "C#/ASP.NET Developer",
                Friends = new List<Friend>
                {
                    new Friend { Name = "Nhat" },
                    new Friend { Name = "Bao" },
                    new Friend { Name = "Khanh" },
                }
            },
            new Employee {
                Name = "Nhat Ngo",
                JobTitle = "MI6 Agent",
                Profile = "Has licence to kill",
                Friends = new List<Friend>
                {
                    new Friend { Name = "James Gordon" },
                    new Friend { Name = "Robin Hood" },
                }
            }
        }
    }
}
```

```
    },  
    }  
};  
return View(model);  
}
```

Xây dựng view Employees.cshtml để hiển thị như hình dưới:



```
@model FirstWebApp.Models.EmployeesViewModel  
@{  
    ViewData["Title"] = "Employees";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>Employees</h2>  
@foreach (var employee in Model.Employees)  
{  
    <details>  
        <summary>@employee.Name</summary>  
        <em>@employee.JobTitle</em>  
        <p>@employee.Profile</p>  
        <ul>  
            @foreach (var friend in employee.Friends)  
            {  
                <li>@friend.Name</li>  
            }  
        </ul>  
    </details>  
}
```

}

Định nghĩa thẻ employee cho phép custom các thuộc tính bằng cách xây dựng lớp EmployeeTagHelper kế thừa từ TagHelper:

```
[HtmlTargetElement("employee")]
public class EmployeeTagHelper : TagHelper
{
    [HtmlAttributeName("summary")]
    public string Summary { get; set; }

    [HtmlAttributeName("job-title")]
    public string JobTitle { get; set; }

    [HtmlAttributeName("profile")]
    public string Profile { get; set; }

    public override void Process(
        TagHelperContext context,
        TagHelperOutput output)
    {
        output.TagName = "details";
        output.TagMode = TagMode.StartTagAndEndTag;

        var sb = new StringBuilder();
        sb.AppendFormat("<summary>{0}</summary>", this.Summary);
        sb.AppendFormat("<em>{0}</em>", this.JobTitle);
        sb.AppendFormat("<p>{0}</p>", this.Profile);
        sb.AppendFormat("<ul>");

        output.PreContent.SetHtmlContent(sb.ToString());
        output.PostContent.SetHtmlContent("</ul>");
    }
}
```

Bổ sung view Employees.html:

```
@foreach (var employee in Model.Employees)
{
    <employee summary="@employee.Name"
              job-title="@employee.JobTitle"
              profile="@employee.Profile">
        @foreach (var friend in employee.Friends)
        {
            <friend name="@friend.Name" />
        }
    </employee>
}
```

Kết quả chạy output xuống:



```
<employee summary="Hien Luong"
    job-title="Software Developer"
    profile="C#/ASP.NET Developer">
    <friend name="Nhat" />
    <friend name="Bao" />
    <friend name="Khanh" />
</employee>
<employee summary="Nhat Ngo"
    job-title="MI6 Agent"
    profile="Has licence to kill">
    <friend name="James Gordon" />
    <friend name="Robin Hood" />
</employee>
```

Như vậy chúng có thể định nghĩa các thuộc tính cũng như thẻ tùy chọn cho mỗi đối tượng riêng biệt tùy theo nhu cầu sử dụng.

### 7.3 HTML Helper

Hiện tại trong MVC Core sử dụng Tag Helper thay cho HTML Helper. Tuy nhiên bạn vẫn có thể sử dụng HTML Helper trong Razor View bình thường.

Giống như các control trong ASP.NET, HTML helper được sử dụng để tùy chỉnh HTML đầu ra tuy nhiên HTML Helper nhẹ hơn. Không giống Web Form control, một HTML Helper không có sự kiện và view state.

Có nhiều HTML Helper cho kết quả trả về là một chuỗi. Với MVC, bạn có thể tạo ra các helper của mình hoặc sử dụng các HTML helper có sẵn.

#### 7.3.1 HTML Links

Html.ActionLink() là helper được sử dụng để tạo liên kết trong MVC. Với MVC, Html.ActionLink() không liên kết đến một view mà là một action.

Cú pháp Razor:

```
@Html.ActionLink("linkText", "actionName", "controllerName",
htmlAttributes: new { attribute = "value" }, routeValues: new { parameter =
"value" });
```

Html.ActionLink() helper có một số thuộc tính như sau:

Thuộc tính	Mô tả
.linkText	Văn bản hiển thị (nhãn)
.actionName	Tên Action
.controllerName	Tên controller
.routeValues	Gía trị gửi đến action (tham số yêu cầu)
.htmlAttributes	Các thuộc tính của thẻ <a>

Với cú pháp trên của `Html.ActionLink()` sẽ sinh thẻ liên kết như sau

```
<a href="/controllerName/actionName?parameter=value" attribute="value">linkText</a>
```

Sau đây là một số tình huống thường dùng:

```
@Html.ActionLink("Text", "Action")
@Html.ActionLink("Text", "Action", "Controller")
@Html.ActionLink("Text", "Action", "Controller",
htmlAttributes: new { title = "Hello" }, routeValues: new { name = "Tuan", id = "SV01" })
```

Mã được sinh ra:

```
<a href="/Cookie/Action">Text</a>
<a href="/Controller/Action">Text</a>
<a href="/Controller/Action/SV01?name=Tuan" title="Hello">Text</a>
```

Để tạo liên kết với ảnh, bạn cần nhớ đến helper `@Url.Action()`. Khi đó liên kết sẽ là:

### 7.3.2 Các phần tử HTML Form

Các HTML Helper dưới đây có thể được dùng để tạo ra form HTML như sau:

HTML Helper	Thẻ HTML được sinh ra
<code>@Html.BeginForm()</code>	<code>&lt;form&gt;</code>
<code>@Html.EndForm()</code>	<code>&lt;/form&gt;</code>
<code>@Html.TextArea()</code>	<code>&lt;textarea&gt;</code>
<code>@Html.TextBox()</code>	<code>&lt;input type="text"&gt;</code>
<code>@Html.CheckBox()</code>	<code>&lt;input type="checkbox"&gt;</code>
<code>@Html.RadioButton()</code>	<code>&lt;input type="radio"&gt;</code>
<code>@Html.ListBox()</code>	<code>&lt;select multiple&gt;</code>
<code>@Html.DropDownList()</code>	<code>&lt;select&gt;</code>
<code>@Html.Hidden()</code>	<code>&lt;input type="hidden"&gt;</code>
<code>@Html.Password()</code>	<code>&lt;input type="password"&gt;</code>

### 7.3.3 DropDownList và ListBox

Để sinh các phần tử này, bạn phải sử dụng `@Html.DropDownList()` và `@Html.ListBox()`. Sau đây là ví dụ giúp bạn hiểu điều này.

```
@using System.Collections.Generic
@{
    ViewData["Title"] = "MyListHTMLHelper";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@{
    List<UserInfo> userInfos = new List<UserInfo>
    {
        new UserInfo
        {
            FullName = "Hien Luong", Username = "hienlth"
        },
        new UserInfo
    {
```

```

        FullName = "Nhat Ngo", UserName = "nhatngo"
    }
};
ViewBag.UserInfos = new SelectList(userInfos, "UserName", "FullName");
}
@using (Html.BeginForm())
{
    @Html.Label("UserInfos", "Users : ");
    @Html.DropDownList("UserInfos", "Please select one user");
}
@Html.ListBox("UserInfos");

```

Mã HTML sinh ra cho các Helper như sau:

Users :

Hien Luong ▲

Nhat Ngo ▼

```

<form action="/Home/MyListHTMLHelper" method="post">
  <label for="UserInfos">Users : </label>
  <select id="UserInfos" name="UserInfos">
    <option value="">Please select one user</option>
    <option value="hienlth">Hien Luong</option>
    <option value="nhatngo">Nhat Ngo</option>
  </select>
</form>

```

```

<select id="UserInfos" multiple="multiple" name="UserInfos">
  <option value="hienlth">Hien Luong</option>
  <option value="nhatngo">Nhat Ngo</option>
</select>

```

### 7.3.4 Custom HTML Helper

Bạn có quyền tạo ra các HTML Helper riêng của mình bằng cách thêm vào interface `IHtmlHelper`.

Ví dụ sai đây tôi định nghĩa method `ColorfulHeading()`. Method này bổ sung cho interface `IHtmlHelper` nên cần phải định nghĩa dạng extension method.

Giả sử tạo thư mục **Extensions** ở thư mục gốc và tạo lớp thêm method Extension:

```

public static class MyHtmlHelperExtensions
{
    public static IHtmlContent ColorfulHeading(this IHtmlHelper
htmlHelper, int level, string color, string content)

```

```
{
    level = level < 1 ? 1 : level;
    level = level > 6 ? 6 : level;
    var tagName = $"h{level}";
    var tagBuilder = new TagBuilder(tagName);
    tagBuilder.Attributes.Add("style", $"color:{color ?? "green"}");
    tagBuilder.InnerHtml.Append(content ?? string.Empty);
    return tagBuilder;
}
```

Ở view, muốn sử dụng Helper phải @using class định nghĩa.

Code minh họa:

```
@using MyLabMVC.Extensions
```

```
@Html.ColorfulHeading(1, "green", "Welcome to Nhất Nghệ")
```

```
@Html.ColorfulHeading(2, "orange", "ASP.NET Core 2.0")
```

Kết quả thực nghiệm:

# Welcome to Nhất Nghệ

## ASP.NET Core 2.0

Code HTML sinh ra:

```
<h1 style="color:green">Welcome to Nhất Nghệ</h1>
```

```
<h2 style="color:orange">ASP.NET Core 2.0</h2>
```

## Chương 8: Kiểm lỗi dữ liệu vào

### 8.1 Giới thiệu

Kiểm soát tính hợp lệ của dữ liệu form nhập trước khi chuyển dữ liệu đến server để xử lý là nhiệm vụ cực kỳ quan trọng. Rất nhiều tiêu chí được đặt ra để kiểm duyệt tùy thuộc vào yêu cầu cụ thể của form nhập liệu. Dù sao vẫn có thể liệt kê một số tiêu chí kiểm duyệt chung chung như sau:

- Không cho để trống ô nhập...
- Dữ liệu nhập vào phải theo một khuôn dạng nhất định nào đó: email, creditcard, url...
- Dữ liệu phải nhập vào phải đúng kiểu: số nguyên, số thực, ngày giờ...
- Dữ liệu nhập vào phải có giá trị tối thiểu, tối đa, trong phạm vi...
- Dữ liệu nhập phải đúng theo một kết quả tính toán riêng của bạn...

Trong MVC việc kiểm lỗi chỉ được viết một nơi nhưng xảy ra cả 2 phía là client và server. Vì nếu với cách nào đó, người sử dụng vượt qua phía client thì vẫn còn một chốt chặn ở phía server. Ở phía client MVC sử dụng JQuery validate plugin còn phía server là sự kết hợp giữa Model và Controller.

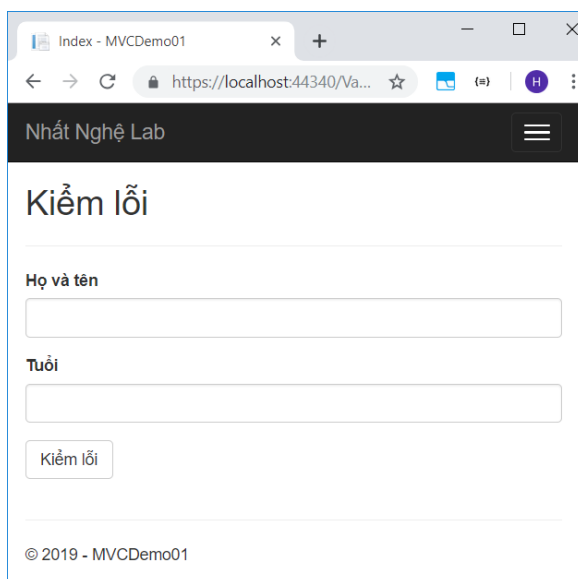
Để đơn giản trong việc học kiểm lỗi, bạn sẽ được làm quen thông qua một ví dụ đơn giản. Từ đó chúng ta sẽ tìm hiểu sâu hơn về điều này.

### 8.2 Mô hình lập trình kiểm lỗi

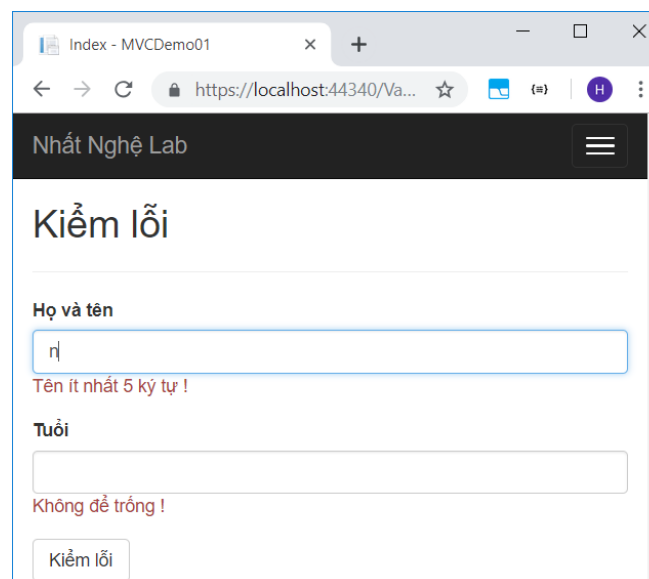
Để thực hiện kiểm lỗi trong MVC bạn cần thực hiện 3 công việc chính

- Bước 1: Khai báo luật kiểm lỗi cho các thuộc tính của Model
- Bước 2: Thực hiện kiểm lỗi trong Controller
- Bước 3: Hiển thị lỗi trong view

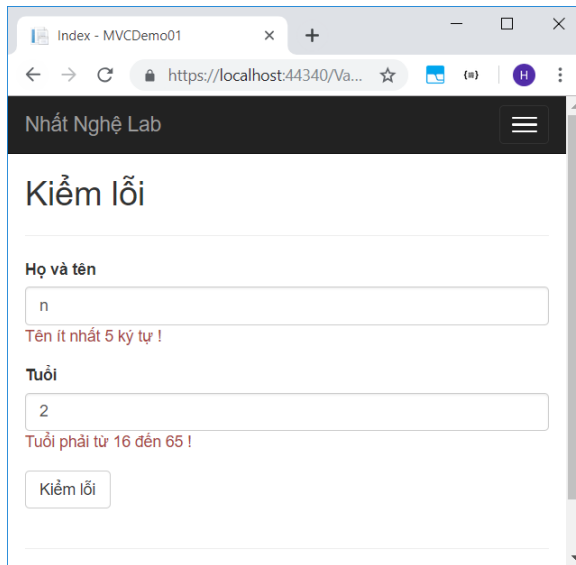
Để hiểu rõ 3 bước này bạn cần thực hiện theo ví dụ được mô tả như ở bên dưới.



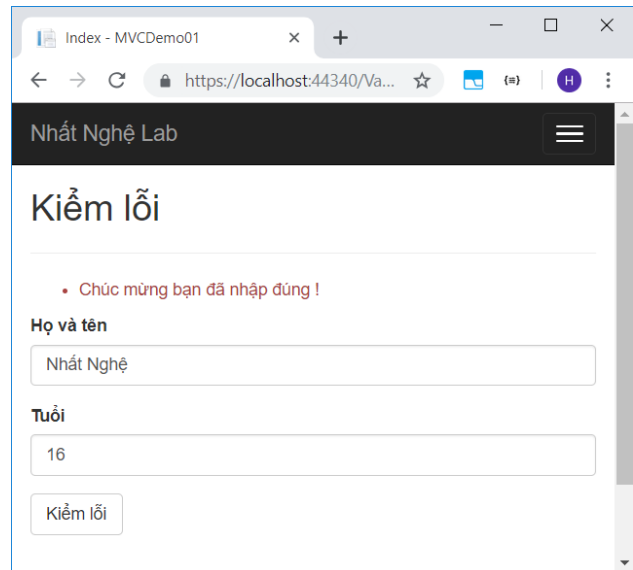
Hình 8-1: Chạy lần đầu tiên



Hình 8-2: Nhập tên, không nhập tuổi



Hình 8-3: Nhập sai tên và tuổi



Hình 8-4: Nhập đúng

### Bước 1: Khai báo luật kiểm lỗi cho các thuộc tính của Model

Đính kèm các annotation kiểm lỗi ngay trên các thuộc tính cần kiểm tra trong lớp model. Trong bài này cần sử dụng các annotation sau:

Annotation	Thuộc tính	Mô tả
[MinLength]	FullName	Giới hạn số lượng ký tự tối thiểu là 5. Nếu không nhập vẫn hợp lệ vì không sử dụng Required
[Required]	Age	Không để trống
[Range]	Age	Giới hạn tuổi từ 16 đến 65

Sau đây là mã nguồn của lớp model có đính kèm các annotaton kiểm lỗi.

```
public class EmployeeInfo
{
    [MinLength(5, ErrorMessage = "Tên ít nhất 5 ký tự !")]
    public String FullName { get; set; }

    [Required(ErrorMessage = "Không để trống !")]
    [Range(16, 65, ErrorMessage = "Tuổi phải từ 16 đến 65 !")]
    public int Age { get; set; }
}
```

### Bước 2: Thực hiện kiểm lỗi trong Controller

Controller sau đây gồm 2 action

- Index() để hiển thị form
- Validate() để nhận thông tin form bằng model và kiểm lỗi thông tin trong model. Action này sẽ kiểm lỗi model của đối số xem có hợp lệ hay chưa thông qua thuộc tính **ModelState.IsValid**. Nếu giá trị của thuộc tính này là true thì dữ liệu trong model đã hợp lệ. Action này bổ sung dòng thông báo khi dữ liệu trong model hợp lệ bằng cách sử dụng phương thức **ModelState.AddModelError(Property, Message)**. Trong đó:
  - Property: thuộc tính phạm lỗi. Nếu không chỉ rõ trên thuộc tính thì đây là lỗi chung.
  - Message: thông báo lỗi

```
public class ValidatorController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    public ActionResult Validate(EmployeeInfo model)
    {
        if (ModelState.IsValid)
        {
            ModelState.AddModelError("", "Chúc mừng bạn đã nhập đúng !");
        }
        return View("Index");
    }
}
```

### Bước 3: Hiển thị lỗi trong view

Lỗi được hiển thị trên view có thể tập trung hoặc riêng cho từng thuộc tính của model.

- **asp-validation-for="FieldName"**: Hiển thị lỗi riêng cho từng thuộc tính trong model.
- **asp-validation-summary="ModelOnly"**: Hiển thị lỗi tập trung. Nếu giá trị thuộc tính là **ModelOnly** thì chỉ hiển thị lỗi do model gây ra; **None** là không hiển thị và **All** là tất cả các lỗi (kể cả lỗi do người dùng thêm vào).

```
@model FirstWebApp.Models.EmployeeInfo
@{
    ViewData["Title"] = "Kiểm lỗi";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

```
<h2>Kiểm lỗi</h2>
<div class="row">
    <div class="col-md-4">
        <form asp-controller="Validator" asp-action="Validate" method="post">
            <div asp-validation-summary="ModelOnly" class="text-
danger"></div>
            <div class="form-group">
                <label asp-for="FullName" class="control-label"></label>
                <input asp-for="FullName" class="form-control" />
                <span asp-validation-for="FullName" class="text-
danger"></span>
```

```

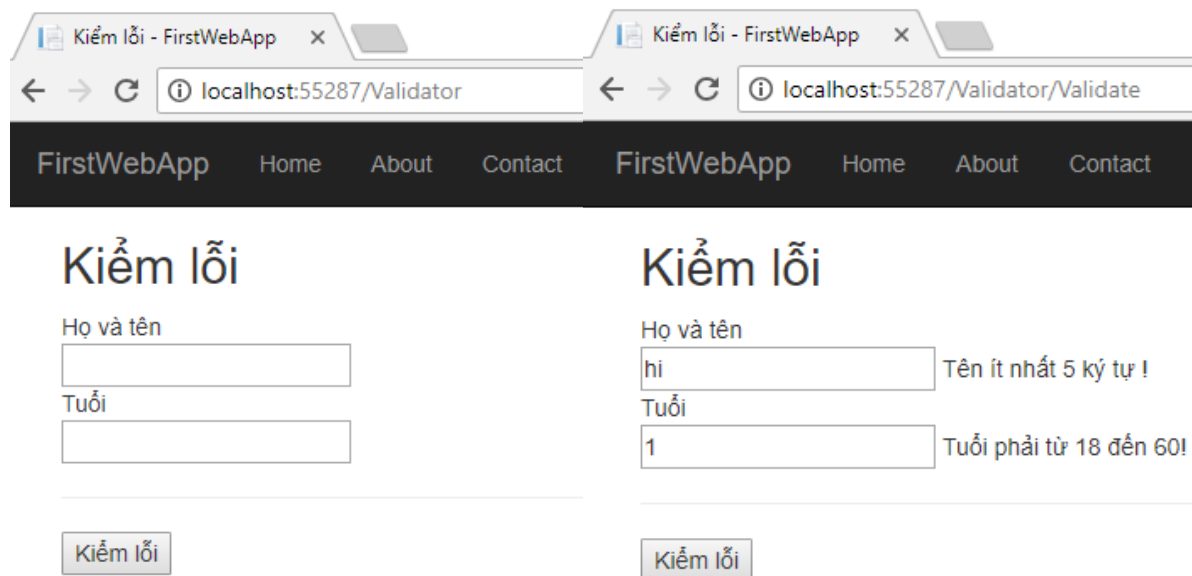
</div>
<div class="form-group">
    <label asp-for="Age" class="control-label"></label>
    <input asp-for="Age" class="form-control" />
    <span asp-validation-for="Age" class="text-danger"></span>
</div>
<div class="form-group">
    <input type="submit" value="Kiểm lỗi" class="btn btn-default"
/>
</div>
</form>
</div>
</div>

```

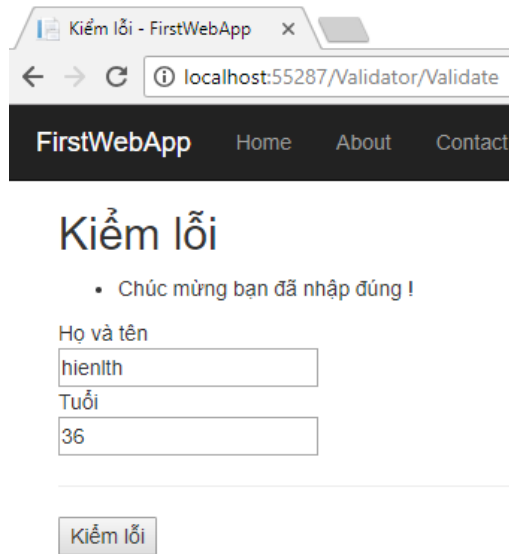
Với cách viết ở trên là bạn đã có thể chạy ứng dụng và lỗi đã được kiểm tốt nhưng chỉ xảy ra phía server, nghĩa là khi nhấp nút [Kiểm lỗi] thì dữ liệu phải được chuyển đến server để kiểm tra và thông báo lỗi được gửi về để hiển thị. Với phương pháp này sẽ dẫn đến phản ứng chậm đến người dùng. Nếu mạng không tốt người dùng phải đợi.

Để có thể kiểm lỗi ngay trước khi chuyển dữ liệu lên server thì bạn phải nhờ đến phương pháp kiểm lỗi phía client với jquery.

Thực nghiệm:







### 8.3 Annotation kiểm lỗi

Ngoài các annotation đã sử dụng, MVC còn định nghĩa sẵn rất nhiều annotation kiểm lỗi khác. Sau đây là danh sách các annotation kiểm lỗi trong MVC.

Annotation	Mô tả	Ví dụ
[Required]	Bắt buộc	[Required] public String Name{get;set}
[Range(Min, Max)]	Giới hạn số trong khoảng	[Range(16, 65)] public String Age{get;set}
[StringLength(Max)]	Giới hạn độ dài chuỗi	[StringLength (20, MinimumLength=5)] public String Password{get;set}
[EmailAddress]	Định dạng email	[EmailAddress] public String Email{get;set}
[CreditCard]	Định dạng số thẻ tín dụng	[CreditCard] public String CardNumber{get;set}
[Url]	Định dạng url	[Url] public String Website{get;set}
[Compare(Property)]	So sánh giá trị	[Compare("Password")] public String ConfirmPassword{get;set}
[RegularExpression(Regex)]	So khớp chuỗi	[RegularExpression("\d{9}")] public String IdCard{get;set}
[MinLength(Min)]	Giới hạn tối thiểu chuỗi, mảng	[MinLength(1)] public String[] Hobbies{get;set}
[MaxLength (Max)]	Giới hạn tối đa chuỗi, mảng	[MaxLength (255)] public String Description{get;set}

Ngoài các annotation kiểm lỗi ở trên, MVC cũng cung cấp annotation \*DataType()+ dùng để sinh mã các phần tử giao diện và kiểm lỗi tự động vào chuẩn HTML5 cho mọi thiết bị và trình duyệt hỗ trợ.

DataType	Trường nhập/Chấp nhận	Ví dụ
Password	Mật khẩu	[DataType(DataType.Password)] public String Password{get;set}
CreditCard	Credicard	[DataType(DataType.Credicard)] public String CardNumber{get;set}
Currency	Tiền tệ theo ngôn ngữ	[DataType(DataType.Currency)] public String UnitPrice{get;set}
Date	Ngày theo ngôn ngữ	[DataType(DataType.Date)] public String Birthday{get;set}
DateTime	Ngày giờ theo ngôn ngữ	[DataType(DataType.DateTime)] public String RegisterDate{get;set}
Duration	Slider	[DataType(DataType.Duration)] public String Duration{get;set}
EmailAddress	Email	[DataType(DataType.Email)] public String Email{get;set}
Html	Mã HTML	[DataType(DataType.Html)] public String Description{get;set}
ImageUrl	Địa chỉ ảnh	[DataType(DataType.ImageUrl)] public String Photo{get;set}
MultilineText	Textarea	DataType(DataType.MultilineText)] public String Description{get;set}
PhoneNumber	Số điện thoại	DataType(DataType.PhoneNumber)] public String Phone{get;set}
PostalCode	Mã số bưu điện	[DataType(DataType.PostalCode)] public String PostalCode{get;set}
Text	Văn bản	[DataType(DataType.Text)] public String Name{get;set}
Time	Thời gian	[DataType(DataType.Time)] public String TimePoint{get;set}
Upload	File upload	[DataType(DataType.Upload)] public String Photo{get;set}

Bảng ánh xạ các thuộc tính Data anotation thông dụng ánh xạ thành thẻ html tương ứng:

Attribute	Input Type
[EmailAddress]	type="email"
[Url]	type="url"
[HiddenInput]	type="hidden"
[Phone]	type="tel"
[DataType(DataType.Password)]	type="password"
[DataType(DataType.Date)]	type="date"

Attribute	Input Type
[DataType(DataType.Time)]	type="time"

Mỗi thuộc tính có thể được thực hiện kiểm nhiều lỗi. Và chúng ta có 2 cách viết annotation kiểm lỗi. Ví dụ sau cho chúng ta thấy 2 cách kiểm lỗi thuộc tính Age là Range và Required.

- Cách 1: Đặt các annotation ngay trên thuộc tính cần kiểm lỗi

[Required]

[Range(16, 65)]

```
public String Age{get;set}
```

- Cách 2: Đặt các kiểm lỗi trong cùng 1 dấu ngoặc vuông và cách nhau dấu phẩy

[Required, Range(16, 65)]

```
public String Age{get;set}
```

Trên bảng chỉ trình bày các đối số bắt buộc. Thực ra mỗi annotation có nhiều đối số khác nhau. Trong đó có đối số ErrorMessage cho phép bạn thay đổi thông báo lỗi mặt định.

[Required(ErrorMessage="Vui lòng nhập tuổi !"), Range(16, 65)]

```
public String Age{get;set}
```

Sau khi kiểm lỗi nhớ kiểm tra đã hợp lệ trên Server hay chưa ở thuộc tính **ModelState.IsValid**.

```
public IActionResult Update(Product pro)
{
    if (ModelState.IsValid == true)
    {
        // business logic ...
    }
    else
    {
        // let user re-input the data
    }
}
```

Ngoài chức năng generate view template dựa trên model, bạn cũng có thể tự định nghĩa các thẻ nhập liệu sử dụng input tag helper theo cú pháp sau:

```
<input asp-for="ModelFieldName" />
```

Tương ứng với các tiêu đề:

```
<label asp-for="ModelFieldName"></label>
```

Riêng thẻ textarea cần ghi rõ: 

```
<textarea asp-for="Description"></textarea>
```

Validation tag helper tương ứng: `<span asp-validation-for="ModelFieldName"></span>`

## 8.4 Kiểm lỗi tùy biến

### 8.4.1 Kiểm lỗi phía Server

Với phương pháp kiểm lỗi đã giới thiệu ở trên, bạn thấy rằng chúng ta cần có model có đính kèm khai các annotation kiểm lỗi cho các thuộc tính. Hai câu hỏi lớn đặt ra là:

- Kiểm lỗi form mà không sử dụng model để tiếp nhận dữ liệu của form thì sao?
- Có thể viết thêm các annotation khác hay không?

#### Kiểm lỗi form không sử dụng model

Chúng ta phải tự viết mã bằng tay mà không có được sự trợ giúp của annotation. Trên server sử dụng tự do lập trình kiểm tra tính hợp lệ của dữ liệu của từng tham số và sử dụng ModelState.AddModelError() để tích lũy lỗi.

Trong view bạn sử dụng thuộc tính `asp-validation-for` để hiển thị lỗi riêng cho các thuộc tính đã add trên ModelState.AddModelError(name, message) và `asp-validation-summary` để hiển thị lỗi tập trung.

Ví dụ: Action Validate()

```
public ActionResult Validate(String FullName, int Age)
{
    if (String.IsNullOrEmpty(FullName))
    {
        ModelState.AddModelError("FullName", "Không để trống họ và tên");
    }
    else if (FullName.Length < 5)
    {
        ModelState.AddModelError("FullName", "Ít nhất 5 ký tự !");
    }

    if (Age < 16 || Age > 65)
    {
        ModelState.AddModelError("Age", "Tuổi phải từ 16 đến 65 !");
    }

    if (ModelState.Count == 0) // không có lỗi nào
    {
        ModelState.AddModelError("", "Chúc mừng bạn đã nhập đúng !");
    }
    return View("Index");
}
```

View Index.cshtml:

```
@{
    ViewData["Title"] = "Kiểm lỗi";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

```
<h2>Kiểm lỗi</h2>
```

```
@Html.ValidationSummary(true)
<form asp-action="Validate" asp-controller="Validator" method="post">
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <div>Họ và tên</div>
  @Html.TextBox("FullName")
  @Html.ValidationMessage("FullName")

  <div>Tuổi</div>
  @Html.TextBox("Age")
  @Html.ValidationMessage("Age")

  <hr />
  <input type="submit" value="Kiểm lỗi" />
</form>
```

## Annotation tùy biến

Bên cạnh các annotation dựng sẵn, MVC cũng cho phép bạn định nghĩa annotation riêng cho mình để sử dụng vào các mục đích riêng.

Ví dụ nếu bạn muốn kiểm lỗi số chẵn thì không có annotation nào có thể giúp bạn thực hiện điều này. Nếu bạn viết mã bằng tay thì lần sau gặp bạn phải viết lại. Bạn rất muốn có annotation riêng dùng để kiểm lỗi này đúng không?

Giả sử chúng ta muốn có annotation [EvenNumber] để kiểm lỗi thì công việc phải làm của bạn là viết annotation này theo mẫu sau.

```
public sealed class EvenNumberAttribute : ValidationAttribute
{
    public EvenNumberAttribute() : base("Vui lòng nhập số chẵn !") { }

    public override bool IsValid(object value)
    {
        if (value == null)
        {
            return true;
        }
        return Convert.ToInt64(value) % 2 == 0;
    }
}
```

Bạn chỉ cần viết mã xử lý lỗi ở bên trong phương thức IsValid(). Nếu kết quả là true có nghĩa là dữ liệu của thuộc tính muốn kiểm tra là đúng.

Ngoài ra bạn cũng cần định nghĩa thông báo lỗi mặc định cho annotation kiểm lỗi này. Trong bài này là "Vui lòng nhập số chẵn !"

Sau khi viết xong lớp này, bạn có thể sử dụng annotation [EvenNumber] y hệt như các annotation dựng sẵn. Ví dụ:

```
[EvenNumber]  
public String Age { get; set}
```

### 8.4.2 Kiểm lỗi phía client

Kiểm duyệt dữ liệu form nhập trước khi chuyển dữ liệu đến server để xử lý là nhiệm vụ cực kỳ quan trọng. Rất nhiều tiêu chí được đặt ra để kiểm duyệt tùy thuộc vào yêu cầu cụ thể của form nhập liệu. Dù sao vẫn có thể liệt kê một số tiêu chí kiểm duyệt chung chung như sau:

- ✓ Không cho để trống ô nhập...
- ✓ Dữ liệu nhập vào phải theo một khuôn dạng nhất định nào đó: email, creditcard, url...
- ✓ Dữ liệu phải nhập vào phải đúng kiểu: số nguyên, số thực, ngày giờ...
- ✓ Dữ liệu nhập vào phải có giá trị tối thiểu, tối đa, trong phạm vi...
- ✓ Dữ liệu nhập phải đúng theo một kết quả tính toán riêng của bạn...

Bây giờ chúng ta hãy khám phá khả năng kiểm duyệt dữ liệu đầu vào của JQuery.

Để sử dụng jQuery Validation, bạn vào trang <http://jqueryvalidation.org/> để tải lấy bản mới nhất.

Mã view cshtml:

```
@model FirstWebApp.Models.EmployeeInfo  
  
@{  
    ViewData["Title"] = "jQueryValidate";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>jQuery Validate</h2>  
  
<hr />  
<div class="row">  
    <div class="col-md-4">  
        <form asp-action="jQueryCheckValidate" id="form1">  
            <div>Name</div>  
            @Html.TextBox("FullName")  
            <div>Age</div>  
            @Html.TextBox("Age")  
            <hr />  
            <input type="submit" value="Submit" />  
            <div id="errors" />  
        </form>  
    </div>  
</div>  
  
@section Scripts {  
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");  
}  
  
<script type="text/javascript">  
    $(document).ready(function () {  
        $("#form1").validate({  
            rules: {  
                txtName: { required: true, minlength: 3 },  
                txtAge: { required: true, digits: true, range: [25, 65] }  
            },  
            messages: {
```

```

        txtAge: { digits: "Nhập số !" },
        txtName: { required: "Không để trống !", minlength: "Ít nhất
3 ký tự !" }
    },
    errorLabelContainer: "#myError",
    wrapper: "li",
    submitHandler: function (form) {
        if (confirm("Dữ liệu form đã hợp lệ. Bạn có muốn submit không
?")) {
            form.submit();
        }
    }
});
});
</script>
<style type="text/css">
    label.error {
        color: Red;
    }

    input.error {
        background-color: Red;
        color: yellow;
    }
</style>

```

Phân tích ví dụ:

Thư viện cần thiết cho việc bẫy lỗi:

Cấu trúc cơ bản của phương thức validate() dùng để cài đặt các tùy chọn bẫy lỗi.

```

<script type="text/javascript">
    $(document).ready(function () {
        $("#form1").validate(
        {
            rules: {<khai báo luật bẫy lỗi cho các trường>},
            messages: {<định nghĩa các thông báo lỗi>},
            errorLabelContainer: ,<khai báo thẻ chứa lỗi>',
            wrapper: ,<khai báo thẻ bọc lỗi>',
            submitHandler: <hàm xử lý submit>
        }
    });
});
</script>

```

Trong bài này:

- ✓ Khai báo luật bẫy lỗi cho các trường

```

rules:
{
    txtName: { required: true, minlength: 3 },
    txtAge: { required: true, digits: true, range: [20,60] }
}

```

- txtName: không được để trống, phải có ít nhất 3 ký tự

- txtAge: không được để trống, phải là số nguyên và thuộc khoản (20, 60)
- ✓ Định nghĩa các thông báo lỗi

Sau đây là danh sách các luật kiểm lỗi trong JQuery

Luật	Mô tả	Ví dụ
required	Bắt buộc nhập	required:true
required	Bắt buộc nhập nếu tập kết quả của selector rỗng	required:"#chkHobby:blank"
required	Bắt buộc nhập nếu kết quả trả về có giá trị false.	required: function(){return true;}
email	Định dạng email	email:true
url	Định dạng url	url:true
date	Định dạng ngày javascript	date:true
number	Số thực	number:true
digits	Số nguyên	digits:true
creditcard	Định dạng creditcard	creditcard:true
minlength	Số ký tự tối thiểu	minlength:10
maxlength	Số ký tự tối đa	maxlength:100
rangelength	Số ký tự từ min đến max	rangelength:[10, 100]
min	Giá trị tối thiểu	min:10
max	Giá trị tối thiểu	max:100
range	Giá trị từ min đến max	range:[10,100]
accept	Kiểu mở rộng file	accept:"doc xsl pdf"
equalTo	So sánh giá trị của phần tử và giá trị của selector	equalTo:"#txtPassword"
remote	Hợp lệ khi kết quả kiểm tra từ xa là false.	remote: "check.aspx"

Chú ý: bạn có 2 cách để khai báo luật bẫy lỗi

- Khai báo trong tùy chọn **rules** như trong ví dụ trên trên
- Khai báo ngay trong thẻ bạn muốn bẫy lỗi

Ví dụ để kiểm lỗi cho ô nhập txtAge của ví dụ trên, bạn có thể khai báo ngay trên thẻ <input> như sau:

```
<input class="required digits" min="25" max="65" id="txtAge" />
```

### Luật kiểm lỗi do người dùng định nghĩa

Trên đây chỉ là danh sách các luật phổ thông hàng ngày. Bạn có thể có những qui luật riêng của mình mà chỉ có bạn mới có thể hiểu và định nghĩa được. Vì vậy JQuery cung cấp cho bạn một cách định nghĩa các luật mới của riêng mình. Hãy xem và phân tích ví dụ sau để hiểu rõ cách để định nghĩa một luật mới.

```
<html>
<head>
<script src="jquery.min.js"></script>
<script src="jquery.validate.js"></script>
<script type="text/javascript">

/*--Định nghĩa hàm kiểm tra số di động việt nam--*/
function fnValidateMobile(value, element) {
    var regex = /^0[0-9]{9,10}$/g;
    return this.optional(element) || regex.test(value);
}

/*--Định nghĩa hàm kiểm tra số xe gắn máy sài gòn--*/
function fnValidateSaigonMoto(value, element) {
```



```
var regex = /^5\d-[A-Z]\d-\d{4}$/g;
return this.optional(element) || regex.test(value);
}

/*--Định nghĩa hàm kiểm tra IP mạng máy tính--*/
function fnValidateNetuworkIP(value, element) {
var regex = /^(\d{3}\.\d{3}\.\d{3}\.\d{3})$/g;
if (this.optional(element) || regex.test(value)) {
var nums = value.split(".");
for (var i = 0; i < nums.length; i++) {
if (parseInt(nums[i]) > 255) {
return false;
}
}
}
else {
return false;
}
return true;
}

/*--Định nghĩa hàm kiểm tra mục chọn của combo box--*/
function fnValidateSelectOne(value, element) {
return (element.value != "none");
}

/*--Định nghĩa luật kiểm tra kết hợp với hàm và một thông
báo lỗi nếu kết quả trả về của hàm có giá trị false--*/
$.validator.addMethod("selectone", fnValidateSelectOne, "Please select an item.");
$.validator.addMethod("vinaphone", fnValidateMobile, "Please enter a valid VinaPhone number.");
$.validator.addMethod("saigonmoto", fnValidateSaigonMoto, "Please enter a valid Saigon moto
number.");
$.validator.addMethod("networkip", fnValidateNetuworkIP, "Please enter valid a network IP.");
</script>
<script type="text/javascript">
$(document).ready(function () {
$("#form1").validate(
{
rules:
{
sport: { selectone: true },
mobile: { vinaphone: true }
},
messages:
{
sport: { selectone: "Vui lòng chọn môn thể thao" },
mobile: { vinaphone: "Không phải số di động ở Việt nam" }
}
});
});
</script>
<style type="text/css">
label.error
{
color: Red;

```

```
}
</style>
</head>
<body>
<h1>Luật kiểm tra tùy biến</h1>
<form id="form1">
Số xe máy Sài gòn:
<input type="text" id="moto" name="moto" class="required saigonmoto">

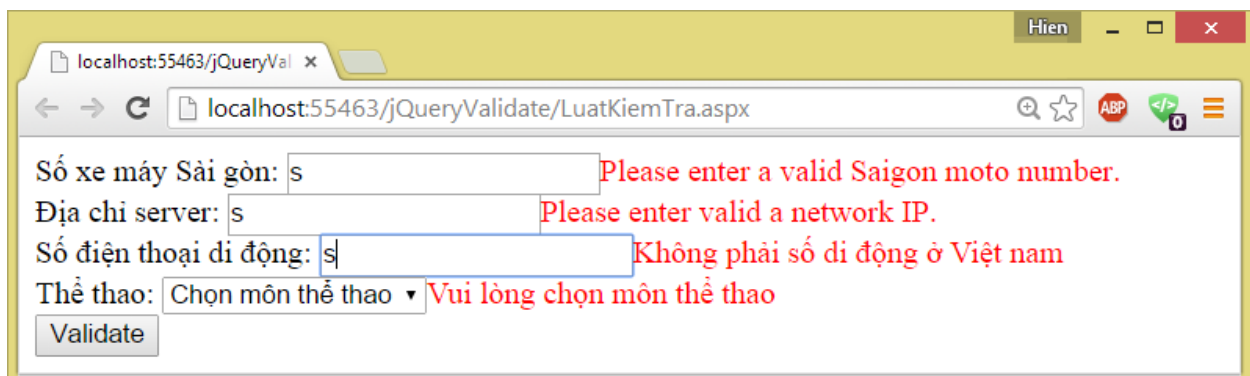
Địa chỉ server:
<input type="text" id="ip" name="ip" class="networkip">

Số điện thoại di động:
<input type="text" id="mobile" name="mobile">

Thể thao:
<select name="sport" id="sport">
<option value="none">Chọn môn thể thao</option>
<option value="baseball">Bóng chày</option>
<option value="basketball">Bóng rổ</option>
<option value="volleyball">Bóng chuyền</option>
<option value="football">Bóng đá</option>
</select>

<input class="submit" type="submit" value="Validate">
</form>
</body>
</html>
```

Trong bài trên chúng ta định nghĩa 4 luật kiểm tra mới là **vinaphone**, **saigonmoto**, **networkip** và **selectone**. Và sau đó áp dụng để kiểm tra dữ liệu cho các thành phần giao diện trên form. Để hiểu được cơ chế định nghĩa và sử dụng chúng ta cần thực hiện các bước sau.



**Bước 1: Định nghĩa.** cần 2 bước là viết hàm kiểm tra và khai báo luật kiểm với JQuery

✓ **Viết hàm kiểm tra:**

```
/*--Định nghĩa hàm kiểm tra số di động việt nam--*/
function fnValidateMobile(value, element) {
    var regex = /^0[0-9]{9,10}$/g;
    return this.optional(element) || regex.test(value);
}
```

Cú pháp của hàm này phải nhận 2 tham số vào là value (giá trị nhập vào) và element (phần tử gây lỗi). Hàm này phải trả về kết quả true (đã hợp lệ) hoặc false (không hợp lệ). Bạn có

thể phân tích giá trị để thực hiện kiểm tra nhờ vào tham số value và thay đổi css hay giá trị của phần tử này thông qua tham số element.

### ✓ Khai báo luật kiểm với JQuery

Sau khi đã định nghĩa hàm kiểm tra, bước tiếp theo là định nghĩa luật kiểm tương ứng với hàm trên và tất nhiên cung cấp thông báo lỗi.

```
/*--Định nghĩa luật kiểm tra kết hợp với hàm, thông báo lỗi nếu kết quả trả về của hàm là false--*/
$.validator.addMethod("vinaphone", fnValidateMobile,
    "Please enter a valid VinaPhone number.");
```

Sử dụng phương thức \$.validator.addMethod(rule, method, message) để khai báo luật kiểm. Tham số rule ("vinaphone") là tên luật mới, tham số method ("fnValidateMobile") là tên phương thức kết hợp với luật mới và message ("Please enter a valid VinaPhone number.") là thông báo lỗi.

### Bước 2: Sử dụng

Bạn sử dụng các luật mới như các luật đã định nghĩa sẵn trong JQuery. Cụ thể là bạn có thể chỉ định trong tùy chọn rules (rules: { mobile: { vinaphone: true } }) của phương thức validate hoặc chỉ ra trên thẻ cần kiểm tra "<input type="text" id="moto" name="moto" class="required saigonmoto">".

### 8.4.3 Regular Expression

Regular expression (biểu thức thường quy) là một chuỗi kí tự đặc biệt được dùng như mẫu (pattern) để so khớp (matching) với các chuỗi.

Loại	Mô tả
Các kí số	\d (hoặc [0-9])
Kí tự không phải số	[^\d] hay [^0-9] hay \D
Kí tự	\w
Kí tự đặc biệt	\W
{m,n}	So khớp từ m đến n lần
{m}	So khớp chính xác m lần
{m,}	So khớp m lần hay hơn
• Hay {0,}	So khớp từ 0 đến n lần
+ hay {1,}	So khớp từ 1 đến n lần
? hay {0,1}	So khớp từ 0 hay 1 lần
\	Dùng để biến kí tự đó là kí tự bình thường

### Cú pháp sử dụng trong javascript:

/pattern/modifier;

**pattern:** là chuỗi mẫu dùng so khớp;

**modifier:** dùng để so khớp theo tiêu chuẩn, gồm:

Modifier	Mô tả
<b>i</b>	So khớp không phân biệt hoa thường.
<b>g</b>	So khớp tất cả thay vì dừng tại mẫu so khớp đầu tiên khi tìm thấy.
<b>m</b>	Thực hiện so khớp nhiều dòng.

**Dùng đối tượng RegExp**

Phương thức	Mô tả
<b>exec()</b>	Kiểm tra một so khớp trong một chuỗi. Trả về so khớp đầu tiên.
<b>test()</b>	Kiểm tra một so khớp trong một chuỗi. Trả về true hay false.
<b>toString()</b>	Trả về giá trị chuỗi của regular expression.
<b>compile()</b>	Biên dịch một regular expression. Không chấp nhận trong phiên bản 1.5.

## Chương 9: Database & EntityFramework

### 9.1 SQL và cơ sở dữ liệu quan hệ

#### 9.1.1 Khái niệm SQL

- SQL (Structured Query Language - ngôn ngữ hỏi có cấu trúc) là công cụ sử dụng để tổ chức, quản lý và truy xuất dữ liệu được lưu trữ trong các cơ sở dữ liệu.
- SQL là một hệ thống ngôn ngữ bao gồm tập các câu lệnh sử dụng để tương tác với cơ sở dữ liệu quan hệ.
- SQL được sử dụng để điều khiển tất cả các chức năng mà một hệ quản trị cơ sở dữ liệu cung cấp cho người dùng bao gồm:
  - **Định nghĩa dữ liệu:** SQL cung cấp khả năng định nghĩa các cơ sở dữ liệu, các cấu trúc lưu trữ và tổ chức dữ liệu cũng như mối quan hệ giữa các thành phần dữ liệu.
  - **Truy xuất và thao tác dữ liệu:** Với SQL, người dùng có thể dễ dàng thực hiện các thao tác truy xuất, bổ sung, cập nhật và loại bỏ dữ liệu trong các cơ sở dữ liệu.
  - **Điều khiển truy cập:** SQL có thể được sử dụng để cấp phát và kiểm soát các thao tác của người sử dụng trên dữ liệu, đảm bảo sự an toàn cho cơ sở dữ liệu.
  - **Đảm bảo toàn vẹn dữ liệu:** SQL định nghĩa các ràng buộc toàn vẹn trong cơ sở dữ liệu nhờ đó đảm bảo tính hợp lệ và chính xác của dữ liệu trước các thao tác cập nhật cũng như các lỗi của hệ thống.

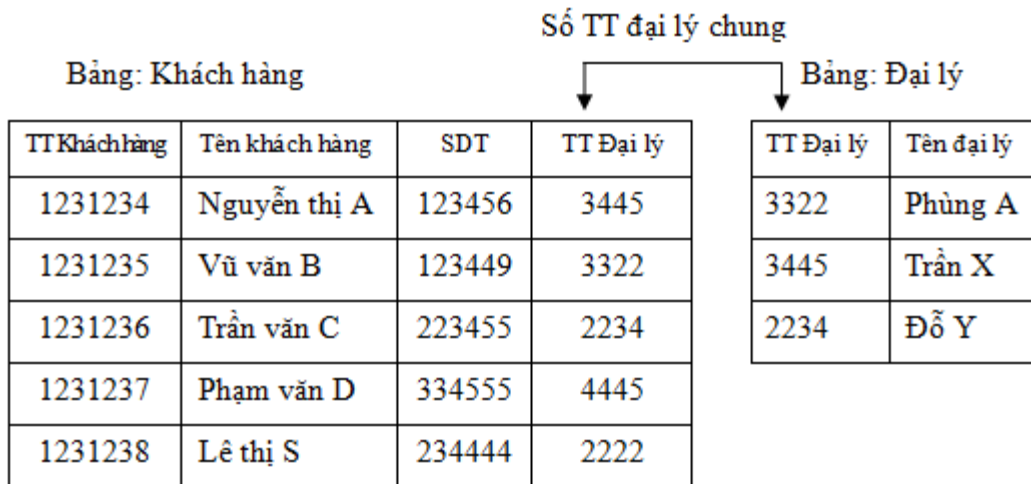
#### 9.1.2 Vai trò của SQL

- SQL không phải là một hệ quản trị cơ sở dữ liệu, do nó không thể tồn tại độc lập.
- SQL là một phần của hệ quản trị cơ sở dữ liệu, nó xuất hiện trong các hệ quản trị cơ sở dữ liệu với vai trò ngôn ngữ và là công cụ giao tiếp giữa người sử dụng và hệ quản trị cơ sở dữ liệu.
- SQL có những vai trò như sau:
  - **SQL là ngôn ngữ hỏi có tính tương tác:** Người sử dụng có thể dễ dàng thông qua các trình tiện ích để gửi các yêu cầu dưới dạng các câu lệnh SQL đến cơ sở dữ liệu và nhận kết quả trả về từ cơ sở dữ liệu.
  - **SQL là ngôn ngữ lập trình cơ sở dữ liệu:** Các lập trình viên có thể nhúng các câu lệnh SQL vào trong các ngôn ngữ lập trình để xây dựng nên các chương trình ứng dụng giao tiếp với cơ sở dữ liệu.
  - **SQL là ngôn ngữ quản trị cơ sở dữ liệu:** Thông qua SQL, người quản trị cơ sở dữ liệu có thể quản lý được cơ sở dữ liệu, định nghĩa các cấu trúc lưu trữ dữ liệu, điều khiển truy cập cơ sở dữ liệu, ..
  - **SQL là ngôn ngữ cho các hệ thống khách/chủ (client/server):** Trong các hệ thống cơ sở dữ liệu khách/chủ, SQL được sử dụng như là công cụ để giao tiếp giữa các trình ứng dụng phía máy khách với máy chủ cơ sở dữ liệu.
  - **SQL là ngôn ngữ truy cập dữ liệu trên Internet:** Cho đến nay, hầu hết các máy chủ Web cũng như các máy chủ trên Internet sử dụng SQL với vai trò là ngôn ngữ để tương tác với dữ liệu trong các cơ sở dữ liệu.
  - **SQL là ngôn ngữ cơ sở dữ liệu phân tán:** Đối với các hệ quản trị cơ sở dữ liệu phân tán, mỗi một hệ thống sử dụng SQL để giao tiếp với các hệ thống khác trên mạng, gửi và nhận các yêu cầu truy xuất dữ liệu với nhau.
  - **SQL là ngôn ngữ sử dụng cho các cổng giao tiếp cơ sở dữ liệu:** Trong một hệ thống mạng máy tính với nhiều hệ quản trị cơ sở dữ liệu khác nhau, SQL thường được sử dụng như là một chuẩn ngôn ngữ để giao tiếp giữa các hệ quản trị cơ sở

dữ liệu (HQTCSDL).

### 9.1.3 Mô hình dữ liệu quan hệ

- CSDL quan hệ là một CSDL trong đó tất cả dữ liệu được tổ chức trong các bảng (table) có mối quan hệ với nhau. Mỗi bảng (table) bao gồm các dòng (record/bản ghi/bộ) và các cột field/trường/thuộc tính.
- Tóm lại, một CSDL bao gồm nhiều bảng (table) có mối quan hệ với nhau (relationship).
- Ví dụ:



### 9.1.4 Bảng (Table)

Bảng (table) bao gồm các yếu tố sau:

- Tên của bảng: được xác định duy nhất.
- Cấu trúc của bảng: tập hợp các cột (field/trường/thuộc tính).
- Dữ liệu của bảng: tập hợp các dòng (record/bản ghi/bộ) hiện có trong bảng.

Ví dụ: Table **DONVI**

MADONVI	TENDONVI	DIENTHOAI
01	Phòng Kế toán	8214514
02	Phòng Tổ chức	8314144
03	Phòng Điều hành	8232356
04	Phòng Đối ngoại	8412345
05	Phòng Tài vụ	8214515

### 9.1.5 Khóa chính của bảng (Primary Key)

- Mỗi bảng phải có một cột (hoặc một tập các cột) mà giá trị dữ liệu của nó xác định duy nhất một dòng trong tập hợp các dòng trong bảng.
- Một cột (hoặc một tập các cột) có tính chất này gọi là khóa chính của bảng (Primary Key).
- Ví dụ: Table **DONVI** ở trên có khóa chính là MADONVI.

### 9.1.6 Mối quan hệ (Relationship) và khóa ngoại (Foreign Key)

- Mối quan hệ (Relationship) được thể hiện thông qua ràng buộc giá trị dữ liệu xuất hiện ở bảng này phải có xuất hiện trước ở một bảng khác.
- Một cột (hoặc tập hợp các cột) (field/trường/thuộc tính) trong một bảng mà giá trị của nó được xác định từ khóa chính (Primary Key) của một bảng khác được gọi là khóa ngoại (Foreign Key).

MADONVI	TENDONVI	DIENTHOAI
01	Phòng Kế toán	8214514
02	Phòng Tổ chức	8314144
03	Phòng Điều hành	8232356
04	Phòng Đối ngoại	8412345
05	Phòng Tài vụ	8214515

MANV	HOTEN	NGAYSINH	DIACHI	DIENTHOAI	MADONVI
NV01001	Nguyễn Ngọc Hoa	15/05/1985	123 Trương Định	38352030	01
NV02001	Lê Thanh Tùng	03/05/1996	32 Trần Phú	38236463	02
NV02002	Hoàng Đình Tùng	08/08/1988	66 Hoàng Diệu	39353535	02
NV03001	Nguyễn Ngọc	19/09/1989	77 Nguyễn Huệ	39292174	03
NV03002	Lý Thanh Tùng	12/02/1992	7 Thành Thái	26636363	03
NV04001	Lê Sao Mai	06/05/1965	123 Lê Lợi	0909123654	04

## 9.2 Sơ lược về câu lệnh SQL

### 9.2.1 Các câu lệnh

Câu lệnh	Chức năng
<i>Thao tác dữ liệu</i>	
SELECT	Truy vấn dữ liệu
INSERT	Thêm mới dữ liệu
UPDATE	Sửa/Cập nhật dữ liệu
DELETE	Xóa dữ liệu
TRUNCATE	Xóa toàn bộ dữ liệu trong bảng
<i>Định nghĩa dữ liệu</i>	
CREATE TABLE	Tạo bảng
DROP TABLE	Xóa bảng
ALTER TABLE	Sửa bảng
CREATE FUNCTION	Tạo hàm (do người sử dụng định nghĩa)
ALTER FUNCTION	Sửa đổi hàm
DROP FUNCTION	Xóa hàm
CREATE TRIGGER	Tạo trigger
ALTER TRIGGER	Sửa trigger
DROP TRIGGER	Xóa trigger

### 9.2.2 Quy tắc sử dụng tên trong SQL

- Trong câu lệnh SQL, nếu ta cần chỉ đến một bảng *do một người dùng khác sở hữu* (hiển nhiên là phải được phép) thì *tên của bảng* phải được viết sau tên của người sở hữu và *phân cách* với tên người sở hữu bởi dấu chấm theo công thức: *tên\_người\_sở\_hữu.tên\_bảng*

- Trong câu lệnh SQL, nếu có sử dụng từ *hai cột* trở lên có *cùng tên* trong các bảng khác nhau thì bắt buộc phải chỉ định thêm *tên bảng trước tên cột*; *tên bảng* và *tên cột* được *phân cách* nhau bởi *dấu chấm* theo công thức: *tên\_bảng.tên\_cột*

### 9.2.3 Kiểu dữ liệu

Tên kiểu	Mô tả
CHAR (n)	Kiểu chuỗi với độ dài cố định
NCHAR (n)	Kiểu chuỗi với độ dài cố định hỗ trợ UNICODE
VARCHAR (n)	Kiểu chuỗi với độ dài chính xác
NVARCHAR (n)	Kiểu chuỗi với độ dài chính xác hỗ trợ UNICODE
INTEGER	Số nguyên có giá trị từ $-2^{31}$ đến $2^{31} - 1$
INT	Giống kiểu INTEGER
TINYINT	Số nguyên có giá trị từ 0 đến 255
SMALLINT	Số nguyên có giá trị từ $-2^{15}$ đến $2^{15} - 1$
BIGINT	Số nguyên có giá trị từ $-2^{63}$ đến $2^{63} - 1$
NUMERIC (p, s)	Kiểu số với độ chính xác cố định
DECIMAL (p, s)	Giống kiểu NUMERIC
FLOAT	Số thực có giá trị từ $-1.79E+308$ đến $1.79E+308$
REAL	Số thực có giá trị từ $-3.4E+38$ đến $3.4E+38$
MONEY	Kiểu tiền tệ
BIT	Kiểu bit (có giá trị 0 hoặc 1)
DATETIME	Kiểu ngày giờ (chính xác đến phần trăm của giây)
SMALLDATETIME	Kiểu ngày giờ (chính xác đến phút)
BINARY	Dữ liệu nhị phân với độ dài cố định (tối đa 8000 bytes)
VARBINARY	Dữ liệu nhị phân với độ dài chính xác (tối đa 8000 bytes)
IMAGE	Dữ liệu nhị phân với độ dài chính xác ( $\leq 2,147,483,647$ bytes)
TEXT	Dữ liệu kiểu chuỗi với độ dài lớn (tối đa 2,147,483,647 ký tự)
NTEXT	Dữ liệu kiểu chuỗi với độ dài lớn và hỗ trợ UNICODE (tối đa 1,073,741,823 ký tự)

### 9.2.4 Toán tử

Toán tử	Ý nghĩa
a) Logic	
AND / OR	Và / Hoặc
b) So sánh	
=	Bằng
>	Lớn hơn
>=	Lớn hơn hay bằng
<	Nhỏ hơn
<=	Nhỏ hơn hay bằng
<> hoặc !=	Khác
c) Danh sách	
IN	Nằm trong danh sách
NOT IN	Không nằm trong danh sách
d) Giới hạn dữ liệu	
BETWEEN	BETWEEN a AND b: nghĩa là $a \leq \text{giá trị} \leq b$
NOT BETWEEN	NOT BETWEEN a AND b: nghĩa là (giá trị < a) và (giá trị > b)
LIKE	Mô tả định dạng dữ liệu sử dụng ký tự đại diện: <ul style="list-style-type: none"> <li>• %: ký tự bất kỳ (không hoặc nhiều)</li> <li>• _: một ký tự bất kỳ</li> <li>• []: một ký tự bất kỳ nằm trong danh sách chỉ định</li> </ul>



- [^]: một ký tự bất kỳ không nằm trong danh sách chỉ định

## 9.3 View, Stored Procedure, Trigger, Function

SQL Server cho phép bạn tạo ra 4 đối tượng bằng cách lập trình: Stored Procedure, View, Trigger, Function.

### 9.3.1 Bảng ảo – View

- View có thể được xem như một Table ảo (nó không được lưu trữ lên đĩa về mặt vật lý như Table thông thường) mà dữ liệu của nó được lấy ra từ một câu truy vấn, có chứa cột và dữ liệu từ một hay nhiều Table khác nhau, hay từ những View khác nhau.
- Đặc điểm của View là ta có thể join dữ liệu từ nhiều Table và trả về một tập kết quả đơn. Ngoài ra ta có thể thao tác dữ liệu trước khi trả về cho user bằng cách dùng các lệnh SQL như where, case...
- Lợi ích của View.
  - Có khả năng tăng tính bảo mật. View giúp ta che giấu cấu trúc của câu truy vấn bên trong. Sau khi tạo xong View ta có thể Insert, Delete, Update như 1 Table bình thường.
  - Giảm độ phức tạp. Ví dụ như User chỉ muốn xem thông tin của một vài cột với một điều kiện nào đó, View cung cấp đúng dữ liệu họ muốn mà không có dữ liệu thừa.
- Cú pháp tạo VIEW:

```
CREATE VIEW <tên_view>  
AS  
    <câu_lệnh_sql>
```

- Sử dụng VIEW: giống như table.

### 9.3.2 Stored Procedure

- SP (Stored Procedure) là tên được đặt cho một nhóm các mệnh đề SQL được tạo ra và lưu trong server database.
- SP cho phép truyền tham số, và có thể được gọi bởi nhiều Client qua mạng với các tham số khác nhau. Khi SP bị thay đổi, tất cả Client sẽ tự động nhận được bản mới, vì SP được lưu ở Server, chứ không phải Client.
- SQL Server cung cấp một số các thủ tục được lưu trữ sẵn trong hệ thống giúp thực hiện một số công việc thường xuyên. Nó được gọi là thủ tục hệ thống – System stored procedures. Còn những thủ tục do người sử dụng tự viết gọi là User stored procedures.
- Lợi ích của SP:
  - Tăng tốc độ thực hiện, tốc độ truy cập dữ liệu nhanh hơn
  - Chương trình được module hóa
  - Nhất quán
  - Nâng cao khả năng bảo mật dữ liệu
- Cú pháp định nghĩa User-defined Stored Procedure:

```
CREATE PROC[EDURE] <procedure_name>  
    <@ParaName1> <DataType1> = <DefaultValue1>,  
    <@ParaName2> <DataType2> = <DefaultValue2>,  
    ...  
    <@ParaNameN> <DataTypeN> = <DefaultValueN>  
AS  
BEGIN
```

```
--SQL statements
```

```
END
```

- Thực thi Stored Procedure:

```
EXEC[UTE] <tên_stored_procedure> [danh_sách_tham_số]
```

### 9.3.3 Trigger

- Trigger gắn liền với một bảng và được tự động thực hiện khi có sự thay đổi dữ liệu (Insert, Delete, hay Update tác động trên một bảng cụ thể).
- Tuy nhiên khác với Stored Procedure, Trigger hoàn toàn không có tham số.
- Trigger có thể gọi thực thi Stored Procedure và được lưu trữ, quản lý trên Server Database.
- Dùng Trigger trong trường hợp ta muốn kiểm tra các ràng buộc toàn vẹn trong Database.
- Cú pháp chung để tạo một Trigger như sau:

```
CREATE TRIGGER Ten_Trigger  
ON Ten_Bang  
FOR {[INSERT] | [UPDATE] | [DELETE]}  
AS  
BEGIN  
    --Cac_Cau_Lenh_Cua_Trigger  
END
```

### 9.3.4 Function

- Function là hàm trong ngôn ngữ lập trình để thực hiện một phép tính hay một xử lý nào đó.
- Mục đích: Dùng để tính giá trị từ 01 hay nhiều câu lệnh SQL
- Cú pháp:

```
CREATE FUNCTION <tên_hàm>  
(  
    -- danh_sách_tham_số  
    <@ParaName1> <DataType>  
)  
RETURNS <ReturnDataType>  
AS  
BEGIN  
    -- Câu lệnh SQL  
    RETURN <kết_quả_trả_về>  
END
```

- Ví dụ:

```
CREATE FUNCTION fnDoanhSo  
(  
    @MaHH INT  
)  
RETURNS FLOAT  
AS  
BEGIN  
    DECLARE @DoanhSo FLOAT  
  
    SELECT @DoanhSo = SUM(SoLuong * DonGia) FROM ChiTietHoaDon  
    WHERE MaHH = @MaHH
```

```
RETURN @DoanhSo
END
```

Và gọi hàm (chú ý thêm dbo trước tên hàm):

```
SELECT MaHH, dbo.fnDoanhSo(MaHH) FROM HangHoa.
```

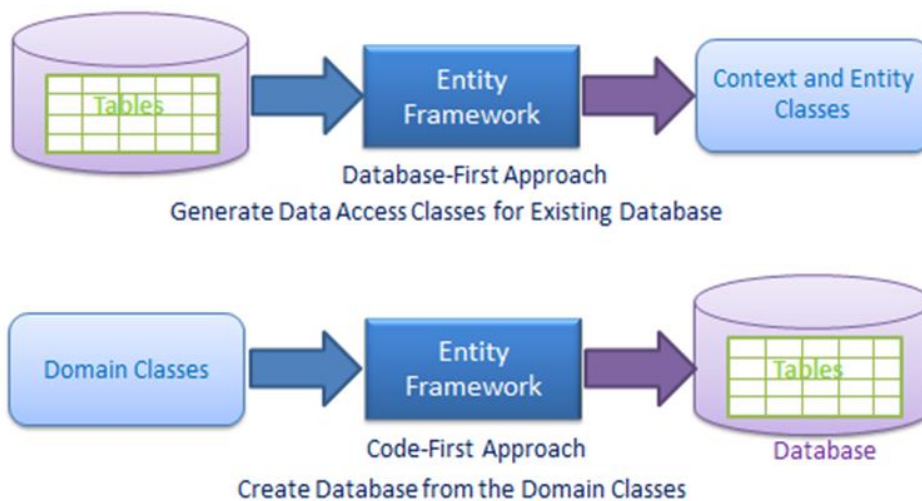
## 9.4 Giới thiệu Entity Framework Core

Entity Framework Core là một phiên bản mã nguồn mở, nhỏ, nhẹ, có thể mở rộng và đa nền tảng nằm trong bộ Entity Framework. EF Core là bộ ánh xạ đối tượng – quan hệ (Object Relation Mapping) cho phép các lập trình .NET làm việc với CSDL quan hệ thông qua các đối tượng (object), giúp các lập trình viên không cần viết mã cho những gì liên quan tới dữ liệu.

EF Core là đã có các version sau:

EF Core Version	Release Date
EF Core 2.0	August 2017
EF Core 1.1	November 2016
EF Core 1.0	June 2016

Trong EF Core có 2 hướng tiếp cận khi làm việc với CSDL:

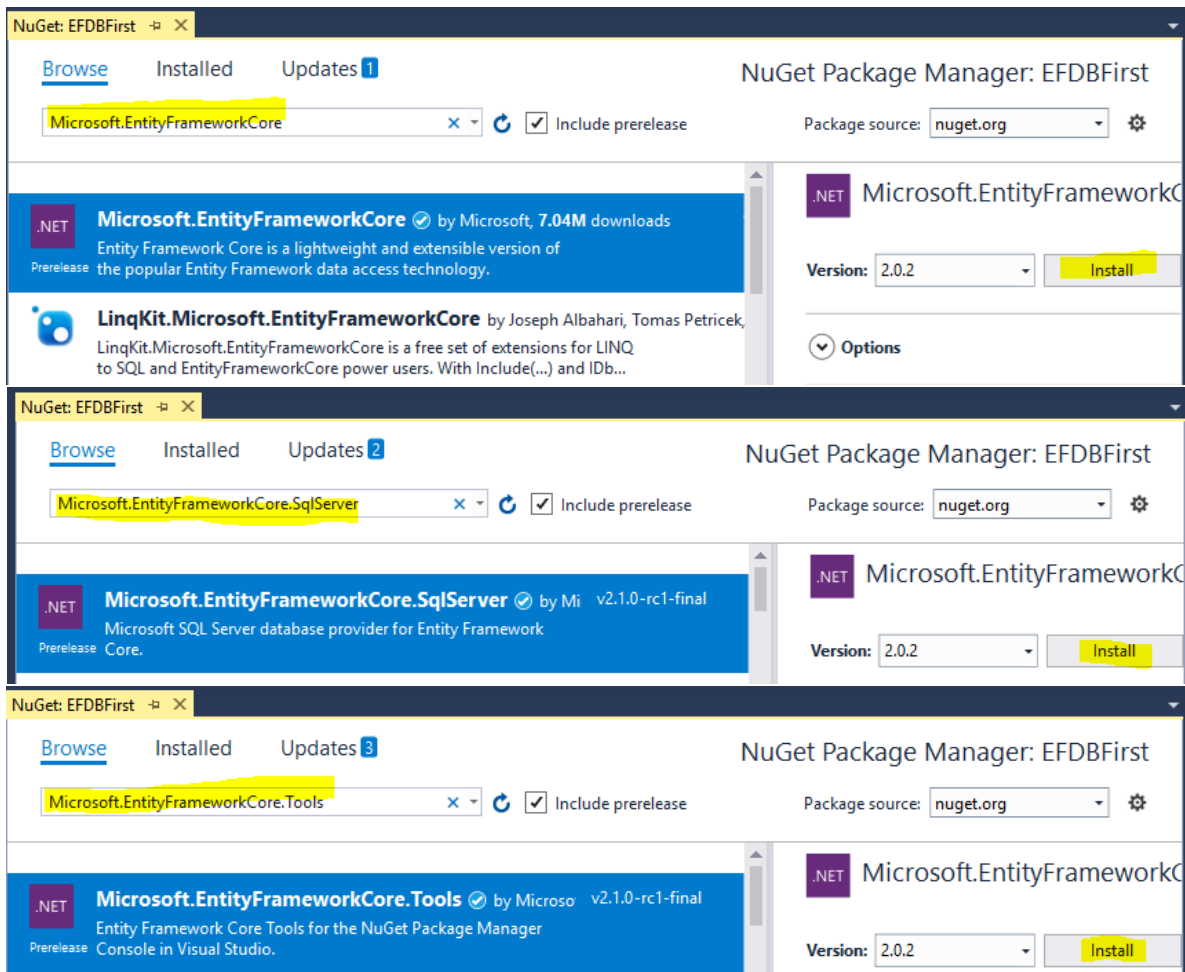


- ✓ Code First: Tạo model các đối tượng dữ liệu, sau đó migration vào Database.
- ✓ Database First: sử dụng khi database có sẵn, sau đó phát sinh các model dữ liệu tương ứng.

Để làm việc với Entity Framework Core, cần sử dụng Nuget package để cài database provider tương ứng. Trong giáo trình này sử dụng database là SQL Server nên cần cài 2 Nuget:

- ✓ Microsoft.EntityFrameworkCore.SqlServer
- ✓ Microsoft.EntityFrameworkCore.Tools

Có thể sử dụng giao diện để cài đặt bằng cách chuột phải trên project chọn **Manage Nuget packages**, sau đó chọn gói tương ứng:



## 9.5 Làm việc với CSDL theo mô hình Database First

Theo mô hình này, bạn cần chuẩn bị sẵn Database. Từ đây sẽ sinh ra các Model.



Mở Package Manage Console gõ lệnh:

```
PM> Scaffold-DbContext "Server=.; Database=EFCoreDBFirst_QLBH;Integrated Security=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Kết quả sau khi generate:

```
└─ Models
  ├── EFCoreDBFirst_QLBHContext.cs
  ├── ErrorViewModel.cs
  ├── HangHoa.cs
  └── Loai.cs
```

Sau đây là mã nguồn các lớp thực thể và lớp ngữ cảnh sinh ra:

Thực thể Loai: thực thể này có mối quan hệ một nhiều với thực thể HangHoa được biểu diễn bởi thuộc tính `HangHoas` có kiểu `Icollection<HangHoa>`.

```
public partial class Loai
{
    public Loai()
    {
        HangHoa = new HashSet<HangHoa>();
    }

    public int MaLoai { get; set; }
    public string Hinh { get; set; }
    public string MoTa { get; set; }
    public string TenLoai { get; set; }

    public ICollection<HangHoa> HangHoas { get; set; }
}
```

Thực thể HangHoa: Có quan hệ nhiều – một với thực thể Loai được biểu diễn bằng thuộc tính `Loai`.

```
public partial class HangHoa
{
    public int MaHh { get; set; }
    public double DonGia { get; set; }
    public string Hinh { get; set; }
    public int MaLoai { get; set; }
    public int SoLuong { get; set; }
    public string TenHh { get; set; }

    public Loai Loai { get; set; }
}
```

Lớp ngữ cảnh DbContext: là đầu mối làm việc với CSDL. Bạn có thể khai báo chuỗi kết nối trực tiếp trong hàm `OnConfiguring()` hoặc có thể kết hợp lưu trong `appsettings.json` và gọi từ hàm `Configuration` của lớp `Startup`.

```
public partial class EFCoreDBFirst_QLBHContext : DbContext
{
    public virtual DbSet<HangHoa> HangHoa { get; set; }
    public virtual DbSet<Loai> Loai { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
```

```
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=.;
Database=EFCoreDBFirst-QLBH;Integrated Security=True;");
    }
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<HangHoa>(entity =>
    {
        entity.HasKey(e => e.MaHh);

        entity.Property(e => e.MaHh).HasColumnName("MaHH");

        entity.Property(e => e.TenHh)
            .IsRequired()
            .HasColumnName("TenHH")
            .HasMaxLength(50);

        entity.HasOne(d => d.MaLoaiNavigation)
            .WithMany(p => p.HangHoa)
            .HasForeignKey(d => d.MaLoai);
    });

    modelBuilder.Entity<Loai>(entity =>
    {
        entity.HasKey(e => e.MaLoai);

        entity.Property(e => e.TenLoai)
            .IsRequired()
            .HasMaxLength(50);
    });
}
}
```

## 9.6 Mô hình Code First của EF Core

Mô hình Code First của EF Core cho phép bạn định nghĩa các Entity trước, sau đó xây dựng lớp ngữ cảnh DbContext và ánh xạ tạo CSDL.

### 9.6.1 Entity

Định nghĩa các lớp thực thể (Entity Model).

- ✓ Loai: ánh xạ với bảng Loai.
- ✓ HangHoa: ánh xạ với bảng HangHoa.

Mã nguồn class Loai:

```
namespace EFCodeFirst.Models
{
    public class Loai
    {
        [Key]
```

```
public int MaLoai { get; set; }
    [Required]
    [MaxLength(50)]
    public string TenLoai { get; set; }
    public string MoTa { get; set; }
    public string Hinh { get; set; }
    public virtual List<HangHoa> HangHoas { get; set; }
}
}
```

Mã nguồn class HangHoa:

```
namespace EFCodeFirst.Models
{
    public class HangHoa
    {
        [Key]
        public int MaHH { get; set; }
        [Required]
        [MaxLength(50)]
        public string TenHH { get; set; }
        public double DonGia { get; set; }
        public int SoLuong { get; set; }
        public string Hinh { get; set; }

        public int MaLoai { get; set; }
        [ForeignKey("MaLoai")]
        public virtual Loai Loai { get; set; }
    }
}
```

Chúng ta dùng từ khoá **virtual** cho **Navigation property** (**Lop** và **HangHoas**) để tận dụng tính năng **Lazy Loading** của **EF**. Tính năng **Lazy Loading** cho phép các thuộc tính điều hướng được cập nhật tự động từ cơ sở dữ liệu khi chúng ta truy cập chúng.

Trong các model trên có sử dụng các từ khóa **Required**, **MaxLength**, **Key**, **ForeignKey** để biểu diễn các ràng buộc (data annotation).

Các quy ước quan trọng của EF khi xây dựng thực thể:

- ✓ Tên thực thể số ít sẽ ánh xạ với bảng cùng tên số nhiều. Nếu bạn muốn ánh xạ đến bảng có tên bất kỳ thì sử dụng [Table("<tên-bảng>")].
- ✓ Tên thuộc tính cùng tên với cột. Nếu bạn muốn ánh xạ đến cột có tên bất kỳ thì sử dụng [Column("<tên-cột>")].
- ✓ Tên thuộc tính khóa thường đặt là Id hoặc EntityId. Nếu bạn muốn đặt tên khác làm khóa thì thêm [Key] phía trước.
- ✓ Thêm các ràng buộc để tạo thông tin cột cho cụ thể.

Bảng ánh xạ kiểu dữ liệu của thực thể với kiểu dữ liệu của SQL Server:

C# Data Type	Mapping to SQL Server Data Type
int	int
string	nvarchar(Max)
decimal	decimal(18,2)
float	real
byte[]	varbinary(Max)
datetime	datetime
bool	bit
byte	tinyint
short	smallint
long	bigint
double	float
char, object	No mapping
sbyte	No mapping (throws exception)

### 9.6.2 Tạo lớp DbContext: Xây dựng lớp ngữ cảnh CSDL.

```
namespace EFCodeFirst.Models
{
    public class MyDbContext : DbContext
    {
        public DbSet<Loai> Loai { get; set; }
        public DbSet<HangHoa> HangHoa { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=.;Database=EFCoreCodeFirst-
QLBH;Integrated Security=True;");
        }
    }
}
```

### 9.6.3 Thực hiện Migration CSDL.

PM > **Add-Migration** EFCodeFirst.MyDBContext

Sau khi chạy, project sẽ tự thêm thư mục Migration:

```

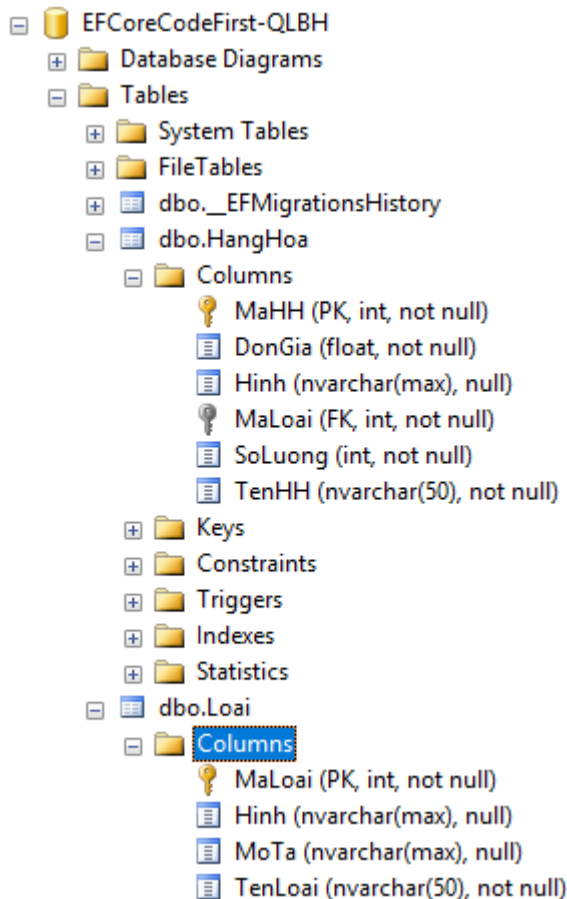
└─ Migrations
    └─ C# 20180512155147_EFCodeFirst.MyDBContext.cs
        └─ C# 20180512155147_EFCodeFirst.MyDBContext.Designer.cs
            └─ EFCodeFirstMyDBContext
                └─ C# MyDbContextModelSnapshot.cs
```



Sau đó chạy lệnh:

PM> **Update-Database**

Kết quả chạy:



## 9.7 Lập trình Entity Framework

Sau khi model được tạo, bạn có thể lập trình làm việc với CSDL thông qua một số hàm ít ỏi nhưng vô cùng đầy đủ và hiệu quả đã được EF xây dựng sẵn.

### 9.7.1 Tóm tắt

Bước 1: Tạo đối tượng DbContext

- MyDbContext db = `new MyDbContext();`

Bước 2: Thao tác và truy vấn thực thể

- Thêm mới thực thể
  - `db.Add(loai);`
- Cập nhật thông tin thực thể
  - `db.Update(loai);`
- Xóa thực thể
  - `db.Loais.Remove(loai);`

- Truy vấn một thực thể theo mã
  - `var loai = db.Loais.SingleOrDefault(m => m.MaLoai == id);`
- Truy vấn tất cả thực thể
  - `var list = db.Loais.ToList();`

**Bước 3:** Lưu sự thay đổi

- `db.SaveChanges();`

## 9.7.2 Màn hình hiển thị Loại

MaLoai	TenLoai	MoTa	
2	Máy tính bảng	Máy tính bảng	Edit   Details   Delete
3	Điện thoại	Điện thoại	Edit   Details   Delete
1002	Túi xách	Túi xách	Edit   Details   Delete
1003	Vali	Vali	Edit   Details   Delete
1004	Phụ kiện	Phụ kiện	Edit   Details   Delete

Bạn cần tạo `LoaiController` và viết mã cho action `Index()` để truy vấn lấy tất cả các loại hiển thị các loại.

```
public class LoaiController : Controller
{
    MyDbContext db = new MyDbContext();
    public ActionResult Index()
    {
        return View(db.Loais.ToList());
        //return View(await db.Loais.ToListAsync());
    }
}
```

Phần view hiển thị bảng như trên:

```
@model IEnumerable<CodeFirstDB.Models.Loai>
```

```
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Loai</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.MaLoai)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.TenLoai)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.MoTa)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.MaLoai)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.TenLoai)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.MoTa)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-id="@item.MaLoai">Edit</a> |
                    <a asp-action="Details" asp-route-id="@item.MaLoai">Details</a> |
                    <a asp-action="Delete" asp-route-id="@item.MaLoai">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

@model để khai báo kiểu của đối tượng model với mục đích tận dụng tính thông minh của công cụ (chấm-xổ). Vòng lặp foreach sẽ duyệt tất cả các loại có trong Model, mỗi loại hiển thị một hàng trên bảng.

Có 3 liên kết mỗi hàng:

- ✓ Liên kết Edit: sẽ gọi action Edit để hiển thị chi tiết của loại cần chỉnh sửa và cập nhật.
- ✓ Liên kết Details: sẽ gọi action Details để hiển thị thông tin chi tiết của loại.
- ✓ Liên kết Delete: sẽ gọi action Delete để xóa loại của hàng tương ứng trên bảng.

### 9.7.3 Màn hình hiển thị chi tiết Loại (Detail)

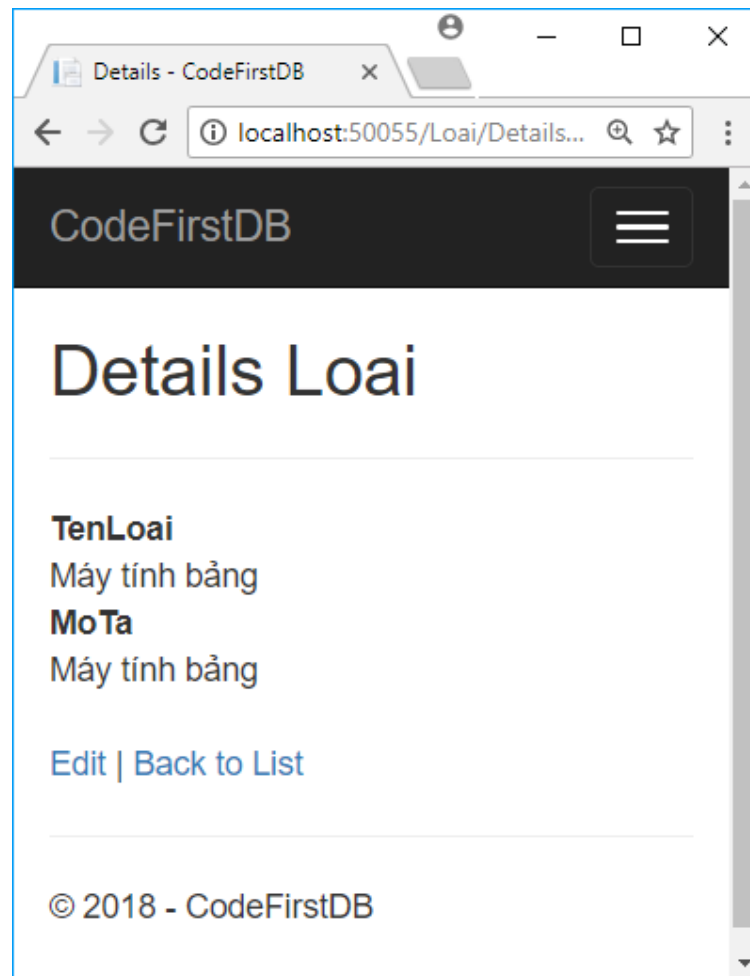
Nội dung Action Details():

```
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var loai = db.Loais
        .SingleOrDefault(m => m.MaLoai == id);
    if (loai == null)
    {
        return NotFound();
    }

    return View(loai);
}
```

Màn hình hiển thị:



Mã của View Details.cshtml tương ứng:

```
@model CodeFirstDB.Models.Loai
```

```
@{
    ViewData["Title"] = "Details";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Details Loai</h2>

<div>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.TenLoai)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.TenLoai)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.MoTa)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.MoTa)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.MaLoai">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>
```

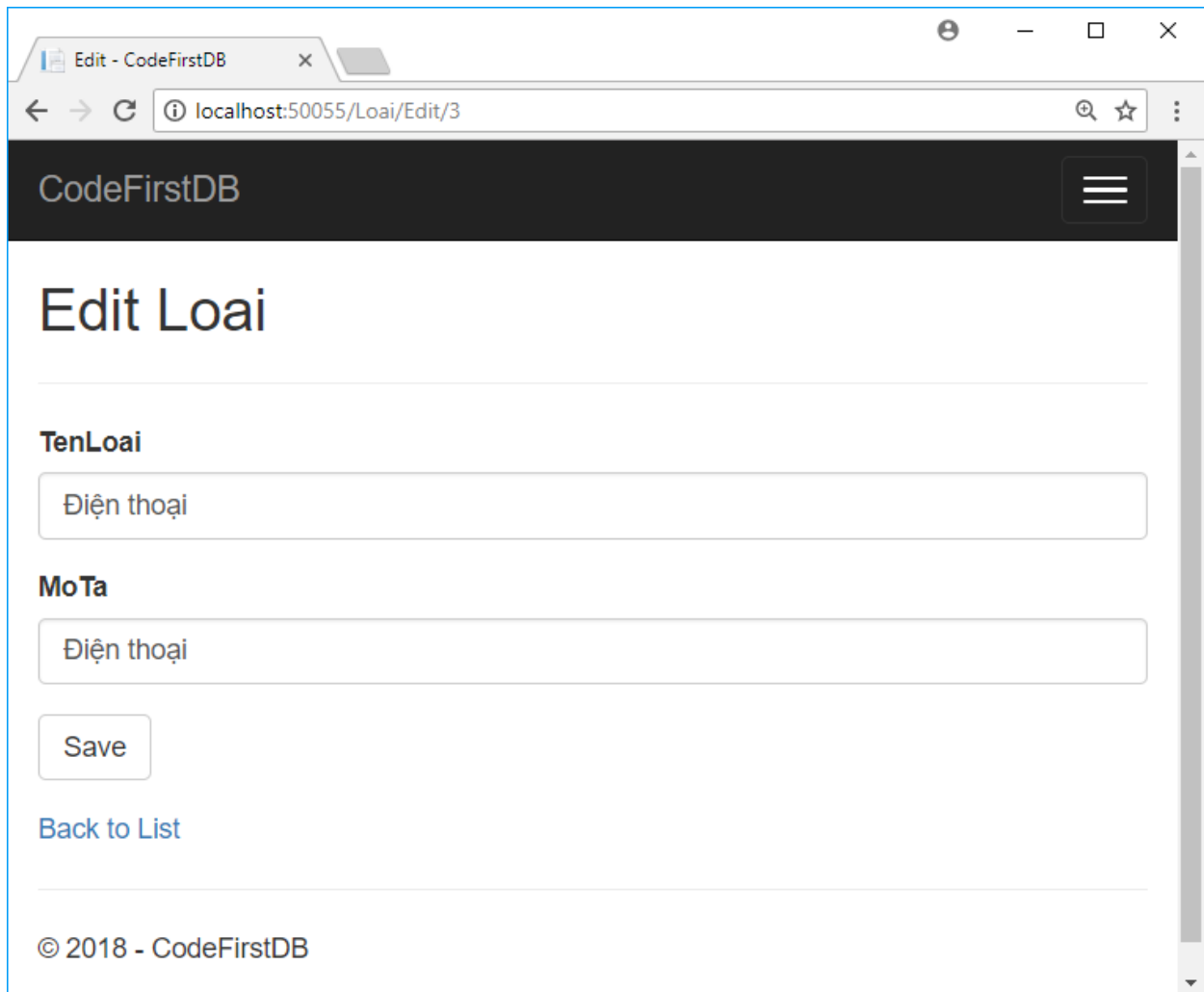
## 9.7.4 Màn hình chỉnh sửa Loại

Nội dung action Edit():

```
// GET: Loai/Edit/5
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var loai = db.Loais.SingleOrDefault(m => m.MaLoai == id);
    if (loai == null)
    {
        return NotFound();
    }
    return View(loai);
}
```

View của action Edit hiển thị thông tin loại được chọn lên form và đợi thao tác cập nhật như sau:



Mã của view Edit được thiết kế như sau:

```
@model CodeFirstDB.Models.Loai

@{
    ViewData["Title"] = "Edit";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Edit Loai</h2>

<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="MaLoai" />
            <div class="form-group">
                <label asp-for="TenLoai" class="control-label"></label>
                <input asp-for="TenLoai" class="form-control" />
                <span asp-validation-for="TenLoai" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="MoTa" class="control-label"></label>
                <input asp-for="MoTa" class="form-control" />
                <span asp-validation-for="MoTa" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" />
            </div>
        </form>
    </div>
    <div class="col-md-4">
        <a href="#">Back to List</a>
    </div>
</div>

<hr />
<div class="row">
    <div class="col-md-4">
        <small>© 2018 - CodeFirstDB</small>
    </div>
</div>
```

```
        <input type="submit" value="Save" class="btn btn-default" />
    </div>
</form>
</div>
</div>
<div>
    <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Action xử lý nút Save vẫn là action hiện hành (Edit) nhưng xử lý với phương thức là POST và đối số là model để nhận dữ liệu toàn form. Bổ sung thêm vào LoaiController action sau để xử lý cập nhật:

```
// POST: Loai/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(int id, [Bind("MaLoai,TenLoai,MoTa")] Loai loai)
{
    if (id != loai.MaLoai)
    {
        return NotFound();
    }

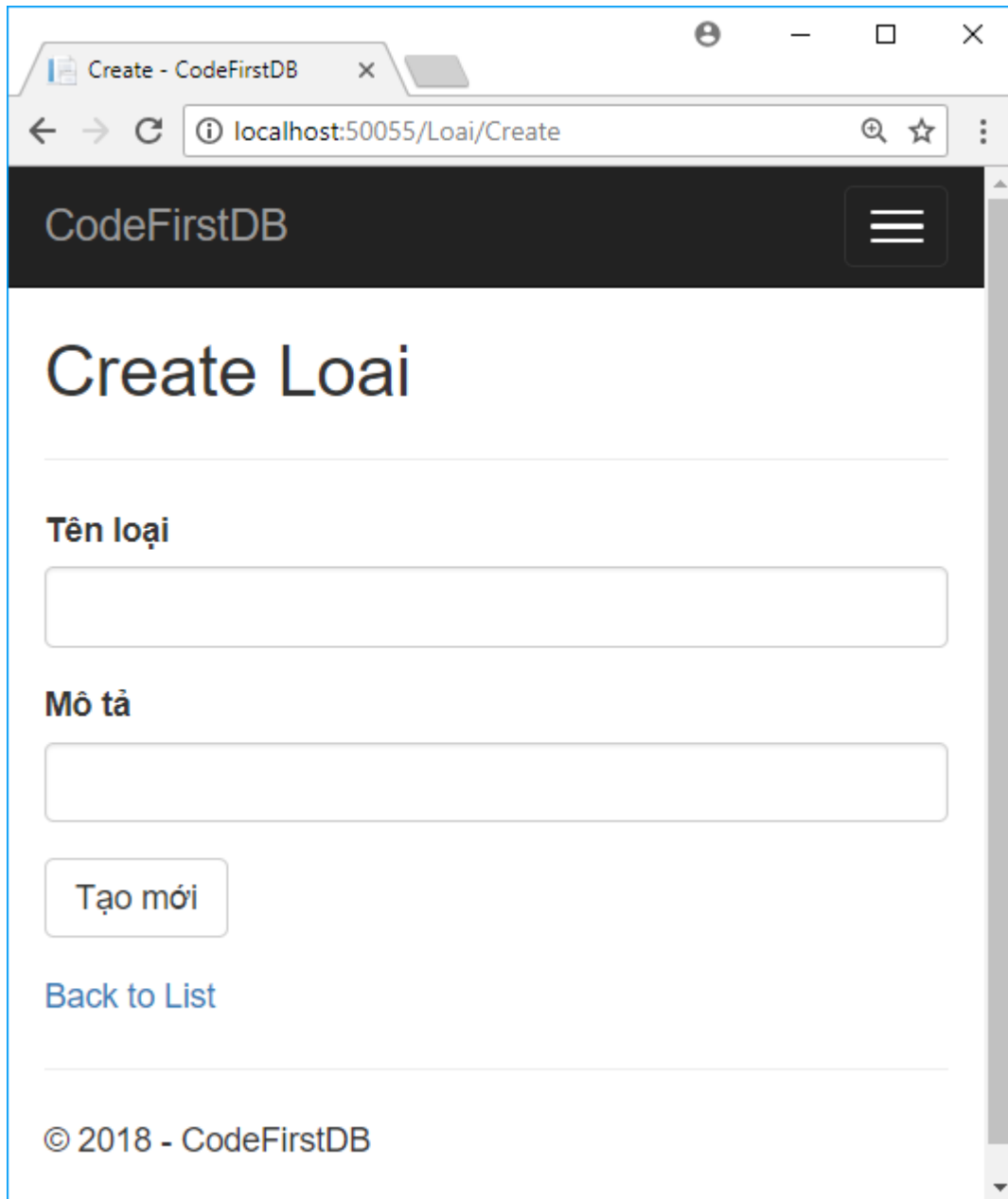
    if (ModelState.IsValid)
    {
        try
        {
            db.Update(loai);
            db.SaveChanges();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!LoaiExists(loai.MaLoai))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(loai);
}
```

### 9.7.5 Màn hình thêm mới (Create)

Để hoàn hảo ta bổ sung phương thức action Create() cho phép thêm một thực thể Loai. Action này cần 2 phiên bản để hiển thị form nhập mới (GET) và để nhận dữ liệu và thêm vào CSDL (POST). Sau đây là giao diện form thêm mới, mã view và action.

Thêm vào LoaiController action Create() phương thức GET:

```
// GET: Loai/Create  
public ActionResult Create()  
{  
    return View();  
}
```



Mã code của view Create.cshtml

```
@model CodeFirstDB.Models.Loai  
  
@{  
    ViewData["Title"] = "Create";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>Create Loai</h2>  
  
<hr />
```

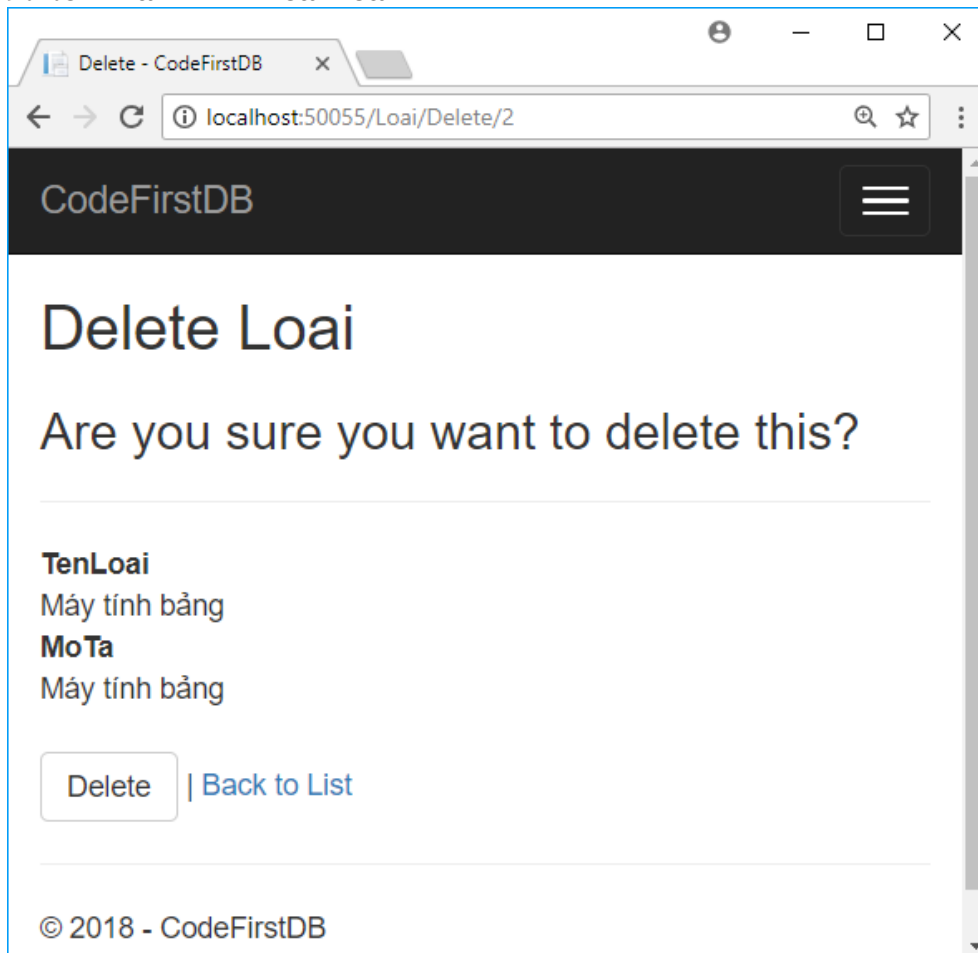


```
<div class="row">
  <div class="col-md-4">
    <form asp-action="Create" asp-controller="Loai">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      <div class="form-group">
        <label class="control-label">Tên loại</label>
        <input asp-for="TenLoai" class="form-control" />
        <span asp-validation-for="TenLoai" class="text-danger"></span>
      </div>
      <div class="form-group">
        <label class="control-label">Mô tả</label>
        <input asp-for="MoTa" class="form-control" />
        <span asp-validation-for="MoTa" class="text-danger"></span>
      </div>
      <div class="form-group">
        <input type="submit" value="Tạo mới" class="btn btn-default" />
      </div>
    </form>
  </div>
</div>
<div>
  <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Action xử lý giao thức POST:

```
// POST: Loai/Create
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind("MaLoai,TenLoai,MoTa")] Loai loai)
{
    if (ModelState.IsValid)
    {
        db.Add(loai);
        db.SaveChanges();
        return RedirectToAction(nameof(Index));
    }
    return View(loai);
}
```

## 9.7.6 Màn hình xóa Loại



Thêm các action Delete() cho phương thức GET – để xác định loại cần xóa và phương thức POST – để tiến hành xóa Loại.

```
// GET: Loại/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var loai = db.Loais.SingleOrDefault(m => m.MaLoai == id);
    if (loai == null)
    {
        return NotFound();
    }

    return View(loai);
}

// POST: Loại/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    var loai = db.Loais.SingleOrDefault(m => m.MaLoai == id);
```

```
db.Loais.Remove(loai);  
db.SaveChanges();  
return RedirectToAction(nameof(Index));  
}
```

Sau khi xóa xong sẽ chuyển đến action Index() để hiển thị danh sách Loai bằng lệnh: `return RedirectToAction(nameof(Index));`

## 9.8 LINQ

### 9.8.1 Giới thiệu

LINQ (Language Integrated Query, tạm dịch là ngôn ngữ truy vấn tích hợp) đưa ra 1 mô hình bền vững để hoạt động với các dạng nguồn dữ liệu và định dạng dữ liệu khác nhau. Trong LINQ, bạn phải làm quen với chuyện làm việc với các đối tượng (objects). LINQ cho phép dùng các đoạn code đơn giản để truy vấn và chuyển đổi dữ liệu trong các tài liệu XML, cơ sở dữ liệu SQL, các tập hợp .NET, và bất kỳ định dạng nào mà LINQ provider hỗ trợ.

Tất cả các hoạt động truy vấn LINQ đều bao gồm 3 tác vụ:

- Kết nối với nguồn dữ liệu (data source)
- Tạo truy vấn
- Thực thi truy vấn

Các biến truy vấn LINQ được biểu diễn dưới dạng `IEnumerable<T>` hoặc là 1 dạng nào đó kế thừa `IEnumerable<T>`, ví dụ như `IQueryable<T>`. Khi chúng ta gặp 1 biến truy vấn có dạng `IEnumerable<KhachHang>`, điều này có nghĩa là, khi được thực thi, truy vấn sẽ sinh ra một chuỗi các đối tượng `KhachHang` hoặc không có đối tượng nào (rỗng).

```
// Tìm các khách hàng ở thành phố Nha Trang  
IEnumerable<KhachHang> customerQuery =  
    from cust in KhachHangs  
    where cust.ThanhPho == "Nha Trang"  
    select cust;
```

### 9.8.2 Kỹ thuật truy vấn dữ liệu

#### Lọc dữ liệu (Filter)

Lọc dữ liệu là câu lệnh truy vấn phổ biến ở dạng diễn giải Boolean (đúng hoặc sai). Câu truy vấn chỉ trả về các phần tử nếu diễn giải là đúng (true). Để lọc dữ liệu, chúng ta dùng mệnh đề `where`, trong đó mô tả các điều kiện lọc.

#### Sắp xếp (order)

Mệnh đề `orderby` cho phép sắp xếp các phần tử theo thứ tự nào đó trong dữ liệu trả về. Để sắp xếp trường `HoTen` theo thứ tự alphabet với các khách hàng ở Đà Lạt, chúng ta có thể làm như ví dụ sau:

```
var queryNTCustomers = from cust in KhachHangs  
    where cust.City == "Nha Trang"  
    orderby cust.HoTen ascending  
    select cust;
```

#### Gom nhóm (group)

Mệnh đề group cho phép gom nhóm kết quả dựa trên 1 khóa được mô tả. Ví dụ, chúng ta muốn gom nhóm các khách hàng từ Nha Trang theo thành phố (ThanhPho), trong trường hợp này cust.ThanhPho được gọi là khóa.

```
var queryCustomersByCity = from cust in KhachHangs
                           group cust by cust.ThanhPho; // gom theo thành phố
```

### Kết hợp (join)

Tương tự như SQL, kết hợp (join) dữ liệu xảy ra giữa các tập đối tượng dữ liệu mà chưa được mô hình rõ ràng trong nguồn dữ liệu. Ví dụ, chúng ta tìm tất cả khách hàng (KhachHangs) và các nhà phân phối (NhaPhanPhois) ở cùng thành phố. Mệnh đề join trong LINQ cho phép kết hợp dữ liệu trên các tập đối tượng theo vì dùng bảng cơ sở dữ liệu trực tiếp.

```
var innerJoinQuery = from cust in KhachHangs
                    join dist in NhaPhanPhois
                      on cust.ThanhPho equals dist.ThanhPho
                    select new
                    {
                        HoTenKhachHang = cust.HoTen,
                        TenNhaPhanPhoi = dist.Ten
                    };
```

### Các truy vấn không chuyển đổi dữ liệu nguồn (Source Data)

Hình sau đây mô tả 1 truy vấn chuyển dữ liệu từ LINQ sang đối tượng mà không có thay đổi dữ liệu. Nguồn dữ liệu là 1 danh sách chứa các chuỗi và đầu ra của truy vấn cũng là 1 danh sách các chuỗi.

```
List<string> nameList = new List<string>{"Dammio", "John", "Aden", "Dammio"};
IEnumerable<string> nameQuery = from name in nameList
                                where name == "Dammio"
                                select name;
foreach(string str in nameQuery)
{
    Console.WriteLine(str);
}
```

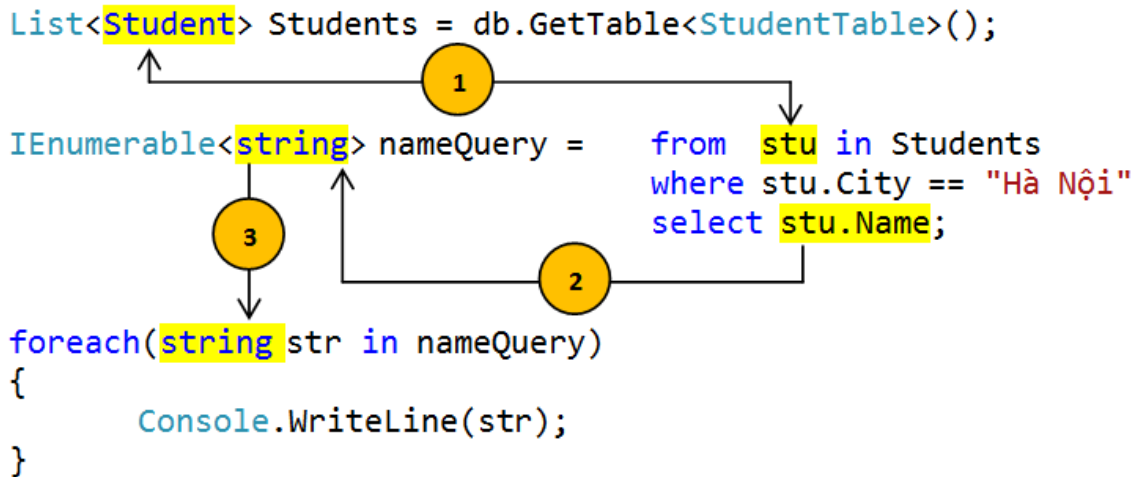
1. Đổi số kiểu của nguồn dữ liệu định nghĩa kiểu của biến phạm vi.
2. Kiểu của đối tượng được chọn (select name) dùng để xác định kiểu của biến truy vấn (**IEnumerable<string>**). Trong hình, biến name có kiểu là 1 chuỗi, vì vậy biến truy vấn là 1 IEnumerable<string>.
3. Biến truy vấn được lặp trong mệnh đề foreach. Bởi vì biến truy vấn là 1 danh sách chuỗi, biến lặp cũng là 1 chuỗi.

Trong truy vấn trên, kết quả không làm thay đổi dữ liệu từ nguồn dữ liệu, rõ ràng chúng ta thấy đầu vào là 1 danh sách chuỗi nameList (mỗi phần tử kiểu string) và đầu ra cũng là 1 danh sách chuỗi tên nameQuery (mỗi phần tử kiểu string).

### Truy vấn chuyển đổi nguồn dữ liệu (Source Data)

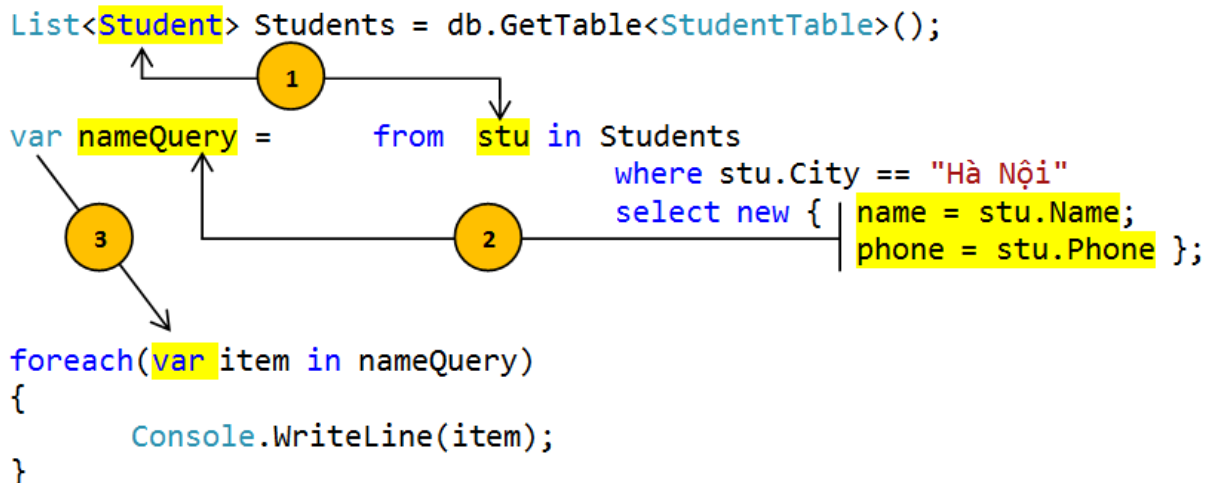
Tiếp theo, chúng ta có ví dụ về một truy vấn từ LINQ sang SQL với sự thay đổi đơn giản trên

nguồn dữ liệu.



1. Đổi số kiểu của nguồn dữ liệu định nghĩa kiểu của biến phạm vi.
2. Mệnh đề select trả về thuộc tính Name thay vì đối tượng Student. Vì Name là 1 chuỗi cho nên kiểu đổi số của nameQuery là 1 chuỗi, chứ không phải là 1 đối tượng Student.
3. Do nameQuery là 1 danh sách chuỗi vì vậy biến vòng lặp foreach cũng phải là 1 chuỗi (string)

Hình tiếp theo mô tả sự chuyển đổi phức tạp hơn chút ít. Dữ liệu trả về là 1 kiểu không xác định với 2 thành viên có mặt trong đối tượng Student ban đầu.



1. Đổi số kiểu của nguồn dữ liệu định nghĩa kiểu của biến phạm vi.
2. Bởi vì mệnh đề select sinh ra 1 kiểu không xác định, kiểu biến truy vẫn phải hiểu ngầm bằng cách dùng từ khóa var.
3. Vì kiểu biến truy vẫn không rõ ràng, cho nên biến lặp trong vòng foreach cũng không rõ ràng (kiểu var).

### 9.8.3 Truy vấn đối tượng

Phương thức	Mô tả	Ví dụ
.Where(e => điều kiện)	Lọc	Students.Where(s => s.Marks > 9)
.GroupBy(e => biểu thức)	Nhóm	Students.GroupBy(s => s.Class)

.OrderBy(e => biểu thức)	Sắp xếp	Students.OrderBy(s => s.Name)
.OrderByDescending(e => biểu thức)		
.Select(e => đối tượng)	Chọn	Students.Select(s => new {s.Name, s.Marks})

#### 9.8.4 Truy vấn phân trang

Phương thức	Mô tả	Ví dụ
.Take(số lượng)	Lấy các phần tử đầu	Students.Take(5)
.Skip(số lượng)	Bỏ các phần tử đầu	Students.Skip(5)
.TakeWhile(e => điều kiện)	Lấy các phần tử thỏa điều kiện	Students.TakeWhile(s => s.Marks < 4)
.SkipWhile(e => điều kiện)	Bỏ qua các phần tử thỏa điều kiện	Students.SkipWhile(s => s.Marks < 4)

#### 9.8.5 Truy vấn 1 thực thể

Phương thức	Mô tả	Ví dụ
.Single(e => điều kiện)	Lấy 1 phần tử thỏa điều kiện. Ngoại lệ nếu không tìm thấy hoặc nhiều hơn một	Students.Single(s => s.Id = 1)
.First()	Lấy phần tử đầu	Students.First()
.Last()	Lấy phần tử cuối	Students.Last()

#### 9.8.6 Tổng hợp số liệu

Phương thức	Mô tả	Ví dụ
.Sum(e=>biểu thức số học)	Tính tổng	Students.Sum(s => s.Marks)
.Count(e=>biểu thức số học)	Đếm số lượng	Students.Count(s => s.Id)
.Min(e=>biểu thức số học)	Giá trị nhỏ nhất	Students.Min(s => s.Marks)
.Max(e=>biểu thức số học)	Giá trị lớn nhất	Students.Max(s => s.Marks)
.Average(e=>biểu thức số học)	Giá trị trung bình	Students.Average(s => s.Marks)

#### 9.8.7 Phương thức kiểm tra

Phương thức	Mô tả	Ví dụ
.Contains(phần tử)	Tập có chứa phần tử?	Students.Contains(sv)
.Any(e=>điều kiện)	Ít nhất một phần tử trong tập thỏa điều kiện	Students.Any(s => s.Marks < 3)
.All(e=>điều kiện)	Tất cả các phần tử trong tập thỏa điều kiện	Students.All(s => s.Marks >= 5)

#### 9.8.8 Ứng dụng LINQ

Sau đây là các truy vấn LINQ trên CSDL MyeStore. Các truy vấn này sẽ được sử dụng trong các bài thực hành sau này và cả trong project cuối.

##### Tìm kiếm hàng hóa

Đoạn mã sau cung cấp 3 lệnh truy vấn hàng hóa với điều kiện tên chứa chuỗi "x", mã loại là 1001 và giá từ 5 đến 10.

```
var items1 = db.Products.Where(p => p.Name.Contains("x"));
var items2 = db.Products.Where(p => p.CategoryId == 1001);
var items3 = db.Products.Where(p => p.UnitPrice >= 5 && p.UnitPrice <= 10);
```

##### Phân trang hàng hóa

Đoạn mã sau truy vấn các hàng hóa ở trang số 4 với mỗi trang là 10 mặt hàng.

```
var pageNo = 3;
var pageSize = 10;
var items4 = db.Products.Skip(pageNo * pageSize).Take(pageSize);
```

##### Truy vấn hàng hóa theo mã

Truy vấn một mặt hàng với Single. Chú ý nếu không tìm thấy hoặc tìm thấy nhiều hơn một mặt hàng thì sẽ xảy ra ngoại lệ.

```
var items5 = db.Products.Single(p => p.Id == 1001);
```

### Truy vấn hàng cùng loại

Truy vấn sau đây sẽ cho các mặt hàng cùng loại với mặt hàng có mã số là 1001.

```
var items6 = db.Products.Single(p => p.Id == 1001).Category.Products;
```

### Thông kê hàng hóa theo loại

Nhóm các mặt hàng theo loại sau đó tổng hợp số liệu trên các nhóm. Chú ý trong mỗi nhóm có thuộc tính Key chính là loại và các mặt hàng của loại này. Các hàm tổng hợp mở rộng như Sum(), Count()... sẽ tổng hợp số liệu trên mỗi nhóm.

```
var items7 = db.Products.GroupBy(p => p.Category)
.Select(g => new ReportInfo
{
    Group = g.Key.Name, //--tên loại
    Sum = g.Sum(p => p.UnitPrice), //--tổng đơn giá hàng hóa của loại
    Count = g.Count(), //--số hàng hóa của loại
    Min = g.Min(p => p.UnitPrice), //--giá hàng hóa thấp nhất
    Max = g.Max(p => p.UnitPrice), //--giá hàng hóa cao nhất
    Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```

### Thông kê doanh số bán hàng từng mặt hàng

Nhóm các chi tiết hóa đơn đã bán theo từng mặt hàng sau đó tổng hợp số liệu trên các nhóm. Thông tin tổng hợp gồm tên hàng hóa, tổng giá trị đã bán, tổng số lượng đã bán, giá bán cao nhất, giá bán thấp nhất, giá trung bình.

```
var items8 = db.OrderDetails.GroupBy(d => d.Product)
.Select(g => new ReportInfo
{
    Group = g.Key.Name, //--tên hàng hóa
    Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị đã bán
    Count = g.Sum(p => p.Quantity), //--tổng số lượng đã bán
    Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
    Max = g.Max(p => p.UnitPrice), //--giá cao nhất
    Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```

### Thông kê doanh số bán hàng từng loại hàng

```
var items9 = db.OrderDetails.GroupBy(d => d.Product.Category)
.Select(g => new ReportInfo
{
    Group = g.Key.Name, //--tên loại hàng
    Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị hàng hóa đã bán
    Count = g.Sum(p => p.Quantity), //--tổng số lượng đã bán
    Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
    Max = g.Max(p => p.UnitPrice), //--giá cao nhất
    Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```

### Thông kê doanh số bán hàng từng khách hàng

```
var items10 = db.OrderDetails.GroupBy(d => d.Order.Customer)
.Select(g => new ReportInfo
{
    Group = g.Key.Fullname, //--họ và tên khách hàng
```

```
Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị hàng hóa đã mua
Count = g.Sum(p => p.Quantity), //--tổng số lượng đã mua
Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
Max = g.Max(p => p.UnitPrice), //--giá cao nhất
Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```

### Thông kê doanh số bán hàng từng tháng

```
var items11 = db.OrderDetails.GroupBy(d => d.Order.OrderDate.Month)
.Select(g => new ReportInfo
{
    Group = g.Key, //--tháng
    Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị hàng hóa đã bán
    Count = g.Sum(p => p.Quantity), //--tổng số lượng đã bán
    Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
    Max = g.Max(p => p.UnitPrice), //--giá cao nhất
    Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```



## Chương 10: Kỹ thuật AJAX

**Sau khi học xong bài này, học viên có khả năng :**

- Hiểu được cơ chế hoạt động của Ajax.
- Viết mã jquery gọi ajax các action của MVC và xử lý các loại dữ liệu trả về từ các action này như text, html, json
- Tạo các action cho phép gọi ajax và trả về các loại dữ liệu khác nhau
- Ứng dụng ajax vào bài toán thực tế để cải thiện tính hiệu quả của ứng dụng

### 10.1 Giới thiệu Ajax

AJAX là một công nghệ cho phép lập trình bất đồng bộ trong ứng dụng Web. Thông thường người dùng muốn thay đổi thông tin từ trang Web bằng cách nhấp vào các nút lệnh (button) hay các liên kết (link) để submit yêu cầu về Web Server để thay đổi nội dung trang Web (postback). Như vậy toàn bộ trang Web phải được xử lý lại do đó tốn khá nhiều thời gian và gia tăng sự phản hồi các trang Web,... Công nghệ Ajax (**A**synchronous **J**avaScript and **X**ML) cho phép chỉ các thông tin nào cần thay đổi được gửi về Sever xử lý, sau đó Server sẽ xử lý và trả kết quả về cho Client. Sau đây là một vài thông tin chung sẽ giúp chúng ta hiểu hơn về Ajax:

AJAX bắt đầu phổ biến từ năm 2005 bởi Google (với một ứng dụng Google Suggest, Google Maps, Gmail). AJAX không phải là ngôn ngữ lập trình mới, mà nó là một công nghệ mới để tạo ra một ứng dụng web nhỏ hơn, nhanh hơn, tốt hơn và giao diện thân thiện với người dùng hơn.

Ajax dựa trên các thành phần HTML trước đây:

- ✓ HTML
- ✓ CSS
- ✓ JavaScript (chủ chốt)
- ✓ XML

AJAX là một công nghệ được hỗ trợ bởi trình duyệt (browser) và nó độc lập với các ứng dụng Web server. Với Ajax, Javascript của bạn có thể liên lạc trực tiếp với Web server bằng cách sử dụng đối tượng XMLHttpRequest của Javascript. Với đối tượng này Javascript của bạn có thể trao đổi dữ liệu trực tiếp Web server mà không cần đệ trình (submit) toàn bộ dữ liệu đến, do đó trang web của bạn không reload lại.

Ajax sử dụng cơ chế làm việc bất đồng bộ (Asynchronous), tức là trong khi đối tượng XMLHttpRequest thực hiện gửi yêu cầu đến Web server thì Web browser vẫn tiếp tục xử lý các công việc khác mà không cần Web server hoàn thành việc trả lời lại yêu cầu đó. Nhiều công việc được xử lý song song với nhau, điều này khác với cách lập trình web cổ điển trước đây, do đó ứng dụng web sẽ chạy nhanh hơn.

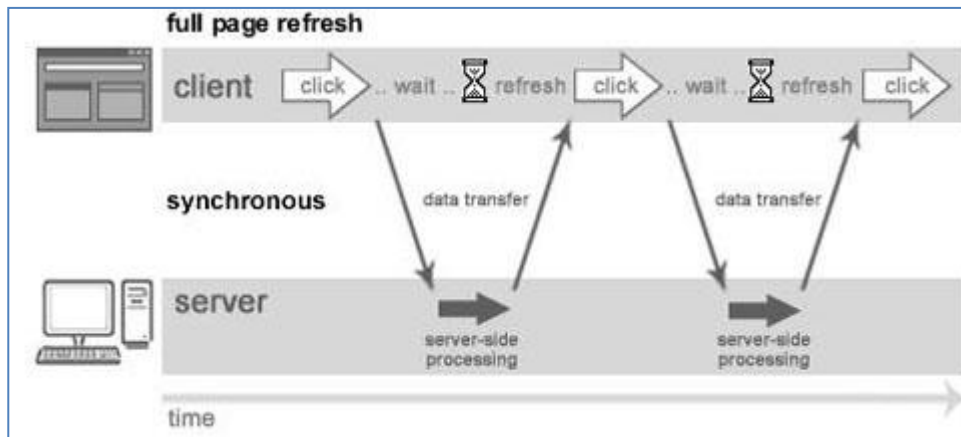
Ajax là một kỹ thuật của Web browser và độc lập với Web server. Tất cả Web có sử dụng Ajax gọi là Web 2.0. Ajax có thể gửi và nhận dữ liệu với nhiều định dạng khác nhau, bao gồm XML, HTML và thậm chí là file text.

### 10.2 Cơ chế làm việc của ajax

Ta hãy phân tích và so sánh cách thức hoạt động của một trang web thông thường và một trang web có ứng dụng Ajax để thấy rõ cách thức thực hiện của Ajax.

### 10.2.1 Cơ chế truyền thông đồng bộ

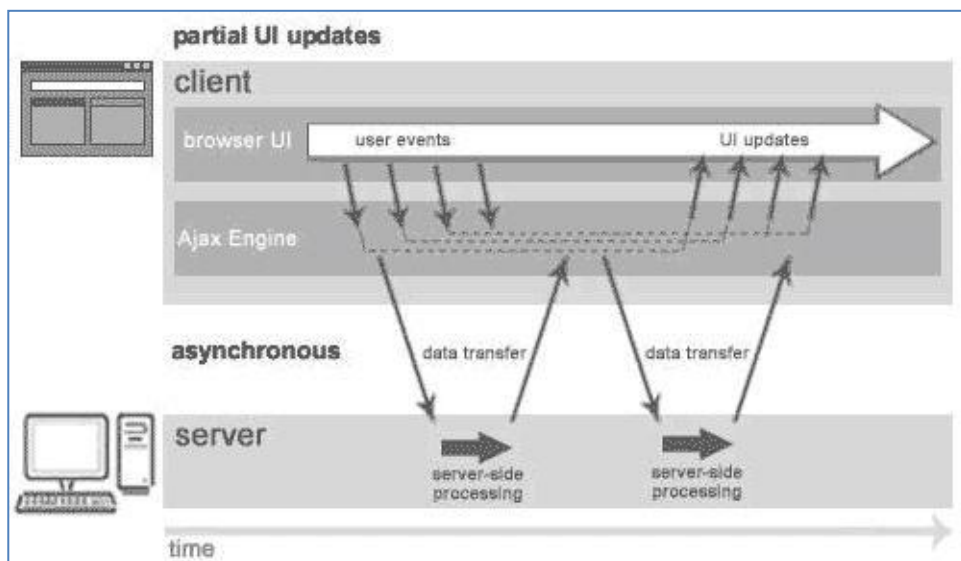
Với cách lập trình Web trước đây (còn gọi là Web 1.0) thì khi người dùng cần cập nhật thông tin (click vào một Button nào đó), thì yêu cầu thay đổi thông tin sẽ được gửi từ phía Client về Server dưới dạng HTTP request, toàn bộ trang web sẽ được gửi chứ không riêng gì một vài thông tin cần thay đổi (dạng này gọi là postback). Lúc đó Client sẽ rơi vào trạng thái chờ (waiting...), trong lúc này phía Client không thể thực hiện một công việc nào khác. Khi Server xử lý hoàn thành các yêu cầu và thì sẽ gửi trả lại cho phía Client một trang web khác thay thế trang cũ (thông tin mà Server response lại ở dạng HTML và CSS). Qui trình này được mô tả như sau :



Như vậy ta thấy cách thức hoạt động của 1 trang web cổ điển là : Click → waiting... → refresh ...→... Do đó cho dù yêu cầu cập nhật một lượng thông tin nhỏ thì trang web cũng phải load lại, do đó cách trang web chạy chậm.

### 10.2.2 Cơ chế truyền thông bất đồng bộ

Với cách lập trình Web có ứng dụng kỹ thuật Ajax thì Ajax cho phép tạo ra một Ajax Engine nằm giữa UI (user interface – giao diện người dùng) và Server, tức là nằm giữa giao tiếp Client – Server, nhưng phần Ajax Engine này vẫn nằm ở phía Client.



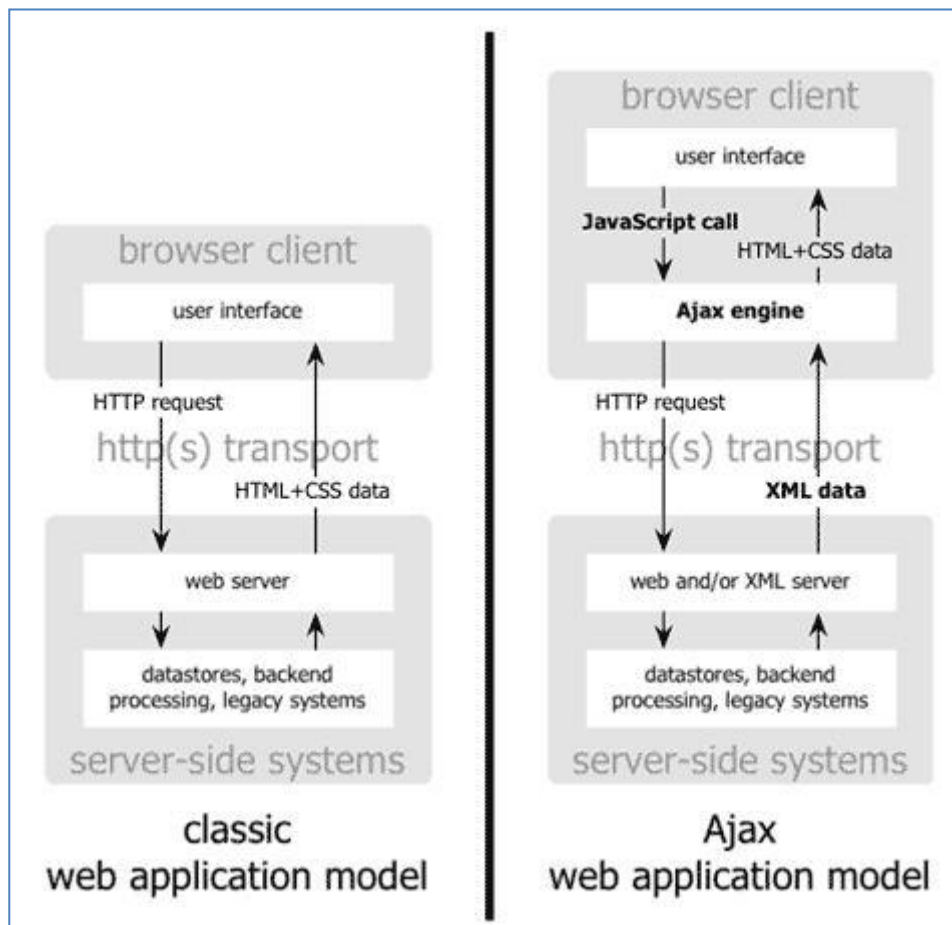
Khi đó công việc công việc gửi request và nhận response đều do Ajax Engine thực hiện. Thay vì trả dữ liệu dưới dạng HTML và CSS trực tiếp cho trình duyệt, Web server có thể gửi trả dữ liệu dạng XML và Ajax Engine sẽ tiếp nhận, phân tích và chuyển đổi thành XHTML + CSS cho trình duyệt hiển thị. Việc phân tích và chuyển đổi này được thực hiện trên Client nên giảm tải rất nhiều cho Server, đồng thời User cảm thấy kết quả xử lý được hiển thị tức thì mà không cần nạp

lại toàn bộ trang. Mặt khác, sự kết hợp của các công nghệ web như CSS và XHTML làm cho việc trình bày giao diện trang web tốt hơn nhiều và giảm đáng kể dung lượng trang phải nạp. Đây là những lợi ích hết sức thiết thực mà Ajax đem lại.

Ajax Engine chỉ gửi đi những thông tin cần thay đổi chứ không phải toàn bộ trang web, do đó giảm được tải qua mạng.

Việc gửi request và nhận response do Ajax Engine thực hiện. Do đó phía Browser UI không rơi vào trạng thái chờ (waiting...), tức là có thể thực hiện nhiều việc cùng lúc (Asynchronous).

Có thể nhìn vào 2 hình sau đây để so sánh hai mô hình ứng dụng Web: truyền thống và sử dụng Ajax.



### 10.3 jQuery Ajax

jQuery cung cấp khá nhiều hàm để làm việc với ajax từ dạng thô cho đến dạng chuyên biệt

- \$.ajax(options)
- \$.post()
- \$.get()
- \$.getJSON()
- \$.getScript()

Trong đó \$.ajax(options) là phương thức thô và là gốc gác của các phương thức khác. Vì vậy chúng ta cần nghiên cứu kỹ phương thức này để từ đó dễ dàng hiểu được các phương thức còn lại. Cú pháp của \$.ajax(options) như sau:

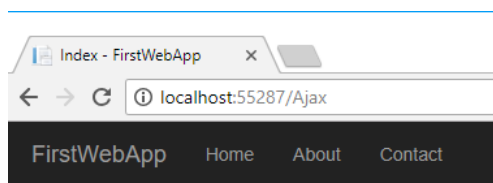
```
<script>
$.ajax({
  url: "", //--địa chỉ server (trang cần tương tác)
  data: {}, //--dữ liệu truyền lên server
  success: function (response) { }, //--hàm xử lý kết quả phản hồi từ
server
  type: "", //--phương thức truyền dữ liệu lên server GET, POST, PUT,
DELETE...
  dataType: "" //--kiểu của dữ liệu nhận từ server text, xml, json,
javascript
});
</script>
```

Ví dụ sau là tương tác với action Search() của AjaxController và truyền các tham số Min, Max cho action. Kết quả phản hồi từ server sẽ được thông báo bằng hộp thoại alert.

```
<script>
$.ajax({
  url: "/Ajax/Search",
  data: { Min: 5, Max: 10 },
  success: function (response) {
    alert(response);
  }
})
</script>
```

Xử lý kết quả phản hồi từ server phụ thuộc hoàn toàn vào kiểu dữ liệu phản hồi là text, html, json hay javascript. Sau đây là các ví dụ cơ sở về nhận và xử lý các loại dữ liệu này.

**Ví dụ 1:** Lấy giờ trên server, cứ 1 giây lấy giờ 1 lần.



Index

15:39:17 PM

Nội dung controller AjaxController:

```
public class AjaxController : Controller
{
  // GET: /Ajax/
  public ActionResult Index()
  {
    return View();
  }
}
```

```
// GET: /Ajax/ServerTime
public ActionResult ServerTime()
{
    var text = DateTime.Now.ToString("HH:mm:ss tt");
    return Content(text);
}
}
```

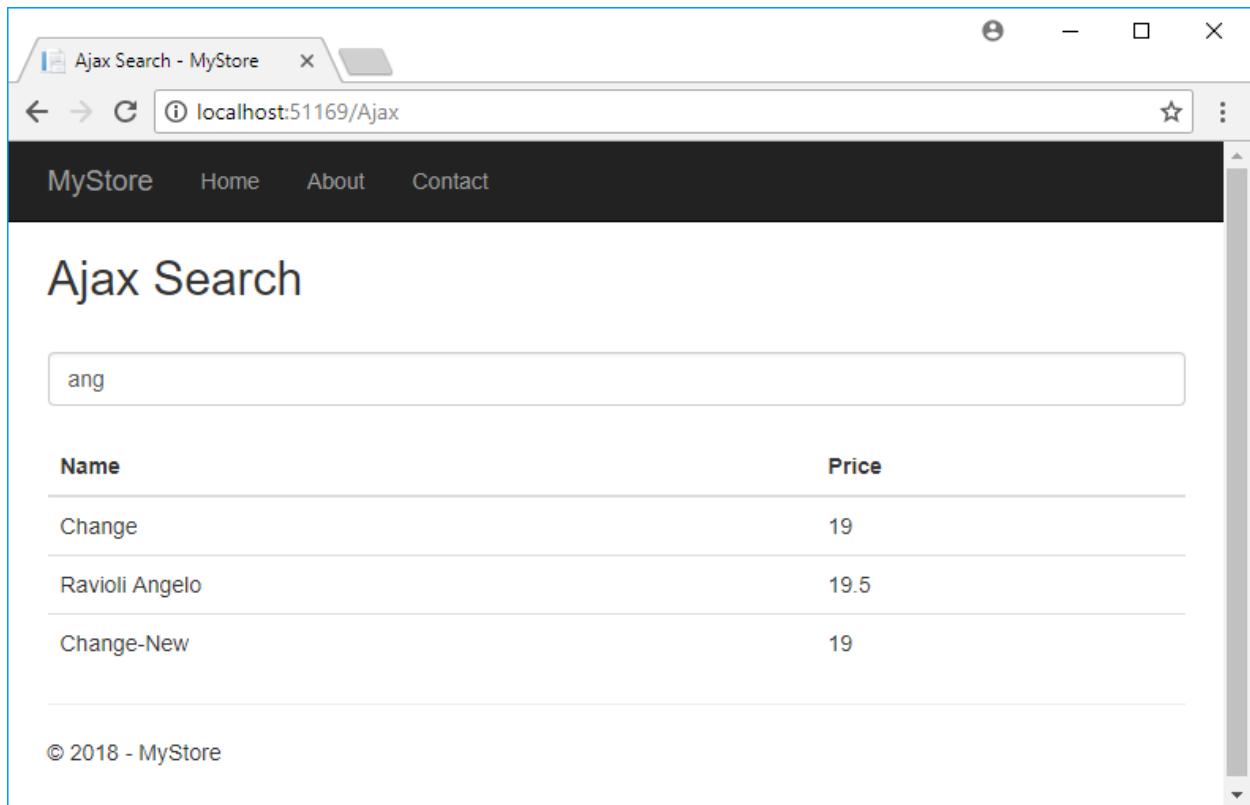
Nội dung view Index.cshtml

```
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>
<h2 id="clock">Clock</h2>

@section scripts{
    <script>
        $(function () {
            setInterval(function () {
                $.ajax({
                    url: "/Ajax/ServerTime",
                    success: function (response) {
                        $("h2#clock").html(response);
                    }
                })
            });
        }, 1000);
    </script>
}
```

**Ví dụ 2:** Tìm kiếm hàng hóa ajax



Người dùng gõ từ cần tìm, dữ liệu sẽ được cập nhật tương ứng.

Nội dung controller AjaxController:

```
public class AjaxController : Controller
{
    MyStoreContext db = new MyStoreContext();
    public IActionResult Index()
    {
        return View();
    }

    // GET: /Ajax/JsonSearch?Name=?
    public ActionResult JsonSearch(String Name = "")
    {
        var model = db.Products
            .Where(p => p.Name.Contains(Name))
            .Select(p => new { Name = p.Name, Price = p.UnitPrice });
        return Json(model);
    }
}
```

Action `JsonSearch()` trả về kết quả dạng JSON chứa thông tin mảng các đối tượng có 2 thành phần `Name` và `Price`.

Trong phần view của action `Index()` sẽ có phép duyệt mảng json này trong jquery là sử dụng hàm `each(function(i, e){})`.

Trong đó i là vị trí của đối tượng hiện tại còn e là đối tượng hiện tại. Việc làm của chúng ta là lấy thông tin từ e để tạo một `<tr><td>tên</td><td>giá</td><tr>` và bổ sung thẻ này vào `<tbody>` của bảng.

Nội dung view Index.cshtml:

```
@{
    ViewData["Title"] = "Ajax Search";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Ajax Search</h2>

<br />
<input id="search" placeholder="Search" class="form-control" />
<br />

<table class="table table-hover">
    <thead>
        <tr>
            <th>Name</th>
            <th>Price</th>
        </tr>
    </thead>
    <tbody id="result"></tbody>
</table>
@section scripts{
    <script>
        $(function () {
            $("#search").keyup(function () {
                var search = $("#search").val();
                $.ajax({
                    url: "/Ajax/JsonSearch",
                    data: { Name: search },
                    success: function (response) {
                        $("#tbody#result").html(""); // xóa nội dung tbody
                        $(response).each(function (i, e) {
                            // duyệt mảng đối tượng
                            var tr = $("<tr />"); // tạo <tr>
                            $("<td />").html(e.name).appendTo(tr); // bổ sung
                            <td> vào <tr>
                            $("<td />").html(e.price).appendTo(tr); // bổ sung
                            <td> vào <tr>
                            tr.appendTo("tbody#result"); // bổ sung <tr> vào
                            <tbody>
                        });
                    }
                });
            });
        });
    </script>
}
```





## Chương 11: **Web API**

**Sau khi học xong bài này, học viên có khả năng :**

- Trình bày được kiến trúc và vai trò Web API trong ứng dụng thương mại điện tử
- Phát triển được ứng dụng Client-Server sử dụng WebAPI
- Thực hiện được mô hình bất đồng bộ sử dụng Web API

### 11.1 Giới thiệu về ASP.NET Core Web API

API là viết tắt của **A**pplication **P**rogramming **I**nterface (giao diện lập trình ứng dụng) phương thức kết nối với các thư viện và ứng dụng khác. Windows có nhiều API, và Twitter cũng có web API, tuy nhiên chúng thực hiện các chức năng khác nhau, với mục tiêu khác nhau. Nó chính là một phần mềm giao tiếp được sử dụng bởi các ứng dụng khác nhau. Nó cũng giống như bàn phím là thiết bị dùng để giao tiếp giữa người sử dụng và máy tính, API là một phần mềm giao tiếp giữa chương trình và hệ điều hành.



API cung cấp khả năng cung cấp khả năng truy xuất đến một tập các hàm hay dùng.

Web API là một trong những công nghệ mới của Microsoft dùng để xây dựng dịch vụ thành phần phân tán. Web API là mô hình dùng để hỗ trợ MVC bao gồm: Routing, Controller, Action Result, Filter, IoC Container, Model binder, Unit Test, Injection. Bên cạnh đó nó còn hỗ trợ restful đầy đủ các phương thức: GET/POST/PUT/DELETE dữ liệu.

Những điểm nổi bật của API.

- Đây là một trong những framework mới sẽ giúp ít cho bạn trong việc xây dựng các HTTP service một cách rất đơn giản và nhanh chóng.
- Mã nguồn mở nên bạn có thể được sử dụng bởi bất kì một client nào hỗ trợ XML, JSON.

- Nó cũng có khả năng hỗ trợ đầy đủ các thành phần HTTP: URI, request/response headers, caching, versioning, content forma.
- Bạn có thể sử dụng các host nằm trong phần ứng dụng hoặc trên IIS.
- Một kiểu kiến trúc vô cùng phù hợp dành cho các thiết bị trang bị bằng thông giới hạn như smartphone, tablet.
- Thường nó có định dạng dữ liệu là JSON, XML hoặc một kiểu dữ liệu bất kỳ.

Ưu điểm:

- Cấu hình đơn giản khi được so sánh với WCF
- Khả năng trình diễn cao
- Hỗ trợ chức năng RESTful một cách đầy đủ
- Hỗ trợ đầy đủ các thành phần MVC như: routing, controller, action result, filter, model binder, IoC container, dependency injection, unit test
- Mã nguồn mở.

## 11.2 Các loại API Action:



Hình 11-1 : Các loại Web API action verb

HTTP verb là một thành phần của request gọi từ client tới server để yêu cầu server thực hiện một việc gì đó như là lấy dữ liệu từ server về, gửi dữ liệu lên server để xử lý, cập nhật hoặc xóa dữ liệu trên server...

### Sử dụng các phương thức theo chuẩn RESTful

Để các web api tuân thủ theo chuẩn restful chúng ta cần sử dụng đúng các HTTP verb tương ứng với ý nghĩa của chúng. Đơn giản có thể hiểu từng HTTP verb tương ứng với một thuật ngữ rất quen thuộc với chúng ta đó là CRUD viết tắt của:

- POST – Create: Tạo dữ liệu mới
- GET – Read: Lấy dữ liệu về
- PUT – Update: Cập nhật dữ liệu
- DELETE – Delete: Xóa dữ liệu

Trong 4 HTTP verb trên mặc dù POST có thể thực hiện tất cả các action nhưng với RESTful service thì cần sử dụng tất cả các verb trên bởi vì:

- 3 verb (GET, PUT, DELETE) được gọi là các phương thức không thay đổi giá trị (idempotent), tức là bạn có thể gọi GET/PUT/DELETE nhiều lần cũng không có lỗi hay gây bất kỳ ảnh hưởng nào đến ứng dụng.
- Nhưng POST lại là một phương thức làm thay đổi giá trị, tức là nếu gọi POST nhiều lần thì sẽ tạo ra nhiều dữ liệu giống nhau.

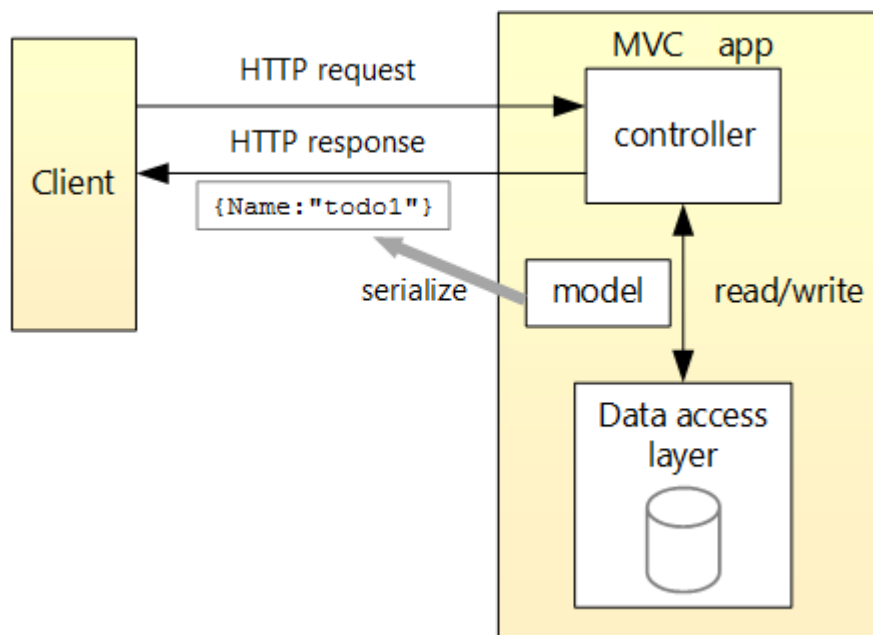
## 11.3 Xây dựng Web API với Entity Framework

### 11.3.1 Xây dựng API dùng data local

Bài này hướng dẫn tạo các API như sau:

API	Description	Request body	Response body
GET /api/todo	Get all to-do items	None	Array of to-do items
GET /api/todo/{id}	Get an item by ID	None	To-do item
POST /api/todo	Add a new item	To-do item	To-do item
PUT /api/todo/{id}	Update an existing item	To-do item	None
DELETE /api/todo/{id}	Delete an item	None	None

Sơ đồ hoạt động của ứng dụng:



Để chuẩn bị nguồn dữ liệu, bạn tạo model TodoItem như sau:

```
public class TodoItem
{
    public long Id { get; set; }
    public string Name { get; set; }
    public bool IsComplete { get; set; }
}
```

Xây dựng data Context:

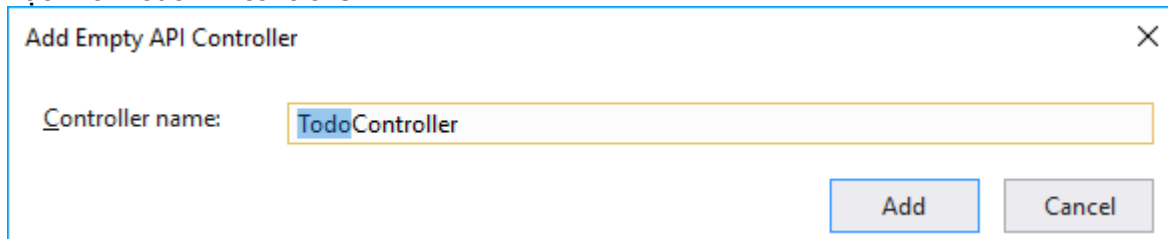
```
public class TodoContext : DbContext
{
    public TodoContext(DbContextOptions<TodoContext> options)
        : base(options)
    {
    }

    public DbSet<TodoItem> TodoItems { get; set; }
}
```

Đăng ký DbContext ở Startup:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddMvc();
}
```

Tạo mới TodoAPI controller:



Bổ sung hàm tạo:

```
[Produces("application/json")]
[Route("api/ToDo")]
public class TodoController : Controller
{
    private readonly TodoContext _context;

    public TodoController(TodoContext context)
    {
        _context = context;

        if (_context.TodoItems.Count() == 0)
        {
            _context.TodoItems.Add(new TodoItem { Name = "Item1" });
            _context.TodoItems.Add(new TodoItem { Name = "Item2" });
            _context.SaveChanges();
        }
    }
}
```

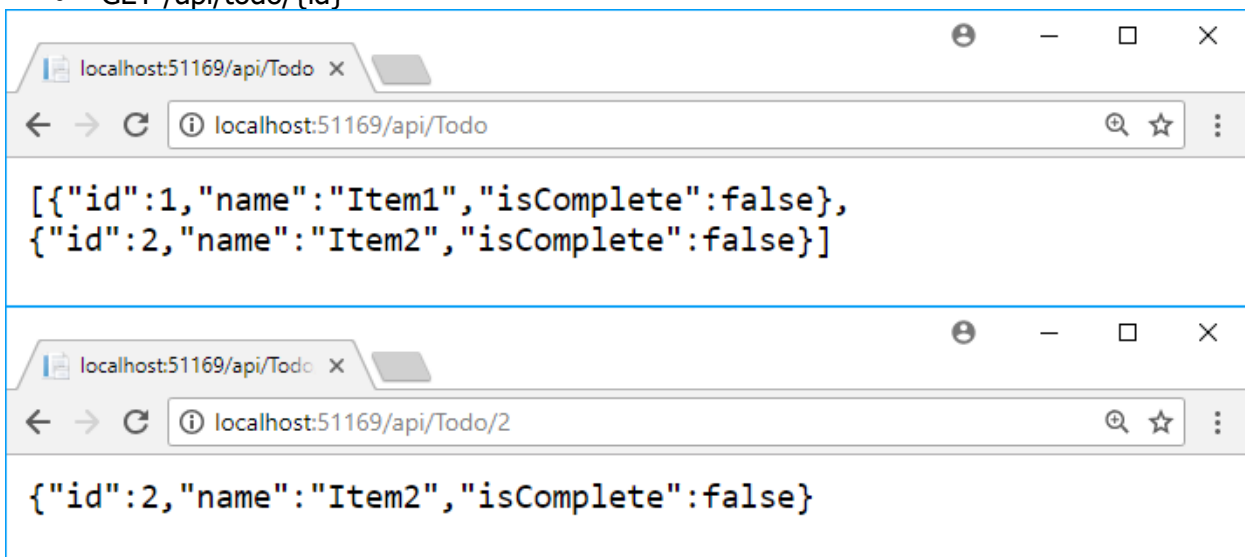
Định nghĩa phương thức HTTP GET dùng để lấy 1 và nhiều TodoItem.

```
[HttpGet]
public List<TodoItem> GetAll()
{
    return _context.TODOItems.ToList();
}

[HttpGet("{id}", Name = "GetTodo")]
public IActionResult GetById(long id)
{
    var item = _context.TODOItems.Find(id);
    if (item == null)
    {
        return NotFound();
    }
    return Ok(item);
}
```

Truy xuất dữ liệu 2 phương thức GET:

- GET /api/todo
- GET /api/todo/{id}



Để ý route name là GetAll() dành cho tất cả, Get<Item>() dành cho việc lấy một Item cụ thể. Nếu tên action khác với tên mặc định buộc phải khai báo chỉ định [HttpGet("{id}", Name = "GetTodo")] trước action đó.

Cài đặt thêm các phương thức Create(), Update(), Delete() cho API controller.

```
[HttpPost]
public IActionResult Create([FromBody] TodoItem item)
{
    if (item == null)
    {
        return BadRequest();
    }

    _context.TODOItems.Add(item);
    _context.SaveChanges();
}
```

```
        return CreatedAtRoute("GetTodo", new { id = item.Id }, item);  
    }  
}
```

Sau khi thêm (Create) xong sẽ chuyển đến GET để hiển thị thông tin vừa thêm `api/Todo/<id>`.

Phương thức Cập nhật (Update):

```
[HttpPut("{id}")]  
public IActionResult Update(long id, [FromBody] TodoItem item)  
{  
    if (item == null || item.Id != id)  
    {  
        return BadRequest();  
    }  
  
    var todo = _context.TODOItems.Find(id);  
    if (todo == null)  
    {  
        return NotFound();  
    }  
  
    todo.IsComplete = item.IsComplete;  
    todo.Name = item.Name;  
  
    _context.TODOItems.Update(todo);  
    _context.SaveChanges();  
    return NoContent();  
}
```

Phương thức Xóa (Delete):

```
[HttpDelete("{id}")]  
public IActionResult Delete(long id)  
{  
    var todo = _context.TODOItems.Find(id);  
    if (todo == null)  
    {  
        return NotFound();  
    }  
  
    _context.TODOItems.Remove(todo);  
    _context.SaveChanges();  
    return NoContent();  
}
```

## Thử nghiệm gọi API bằng POSTMAN:

### Thử nghiệm GET:

The screenshot shows a Postman interface for a GET request. The URL is `http://localhost:51169/api/ToDo`. The response status is 200 OK and the time taken is 113 ms. The response body is displayed in JSON format:

```
[
  {
    "id": 1,
    "name": "Item1",
    "isComplete": false
  },
  {
    "id": 2,
    "name": "Item2",
    "isComplete": false
  }
]
```

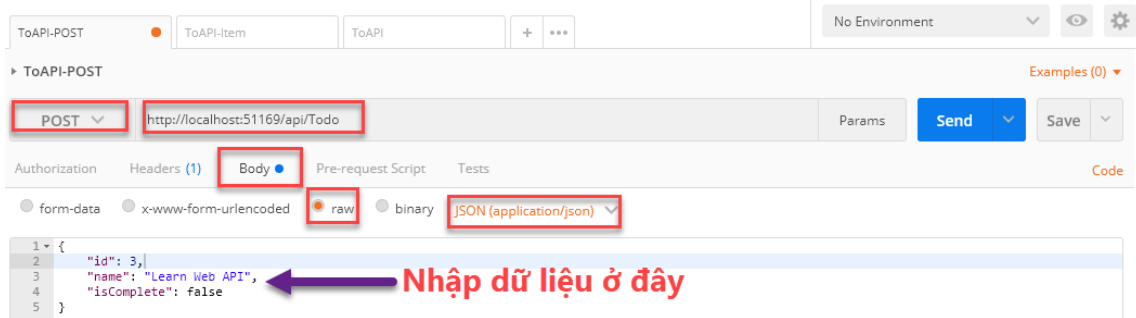
Hình 11-2 : Thử nghiệm POSTMAN với GET N

The screenshot shows a Postman interface for a GET request. The URL is `http://localhost:51169/api/ToDo/2`. The response status is 200 OK and the time taken is 139 ms. The response body is displayed in JSON format:

```
{
  "id": 2,
  "name": "Item2",
  "isComplete": false
}
```

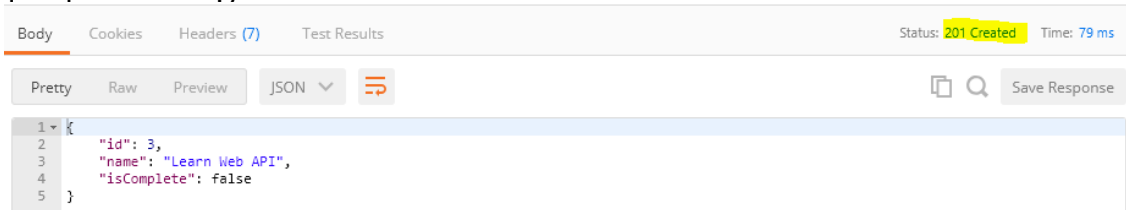
Hình 11-3: Thử nghiệm POSTMAN với GET 1

### Thử nghiệm POST:



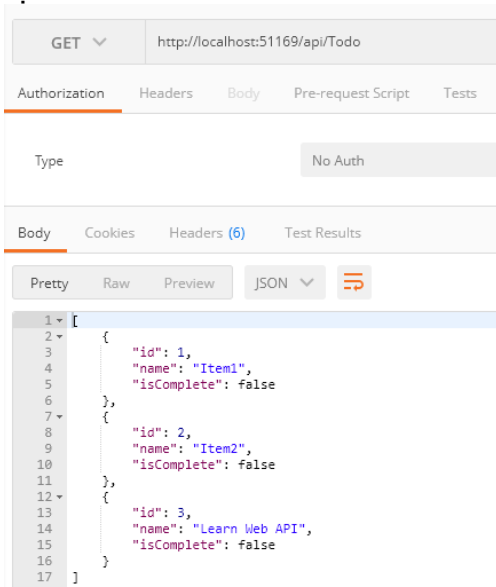
Hình 11-4: Thử nghiệm POSTMAN với POST action

### Kết quả quá trình chạy:



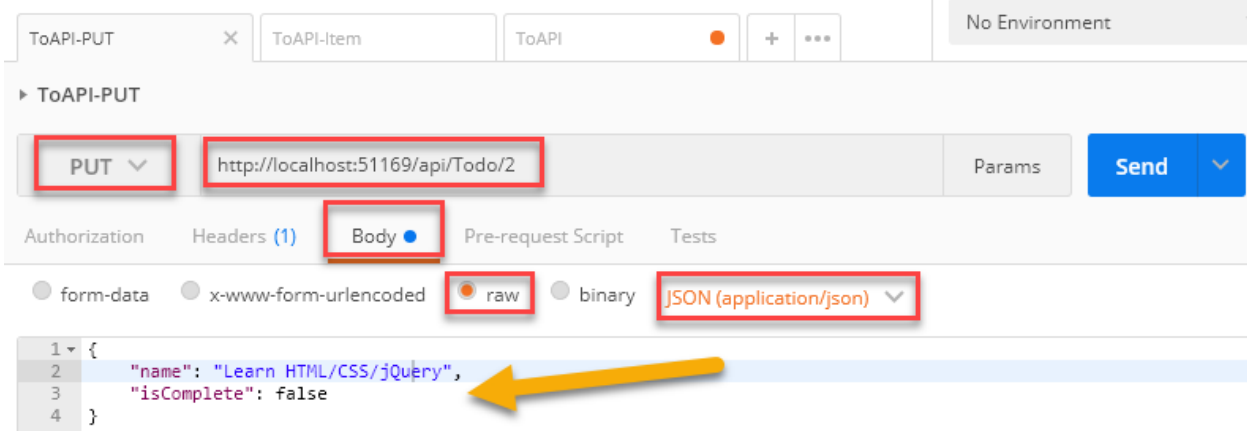
Hình 11-5: Kết quả chạy POSTMAN POST action

Sau khi chạy xong, test GET lại:



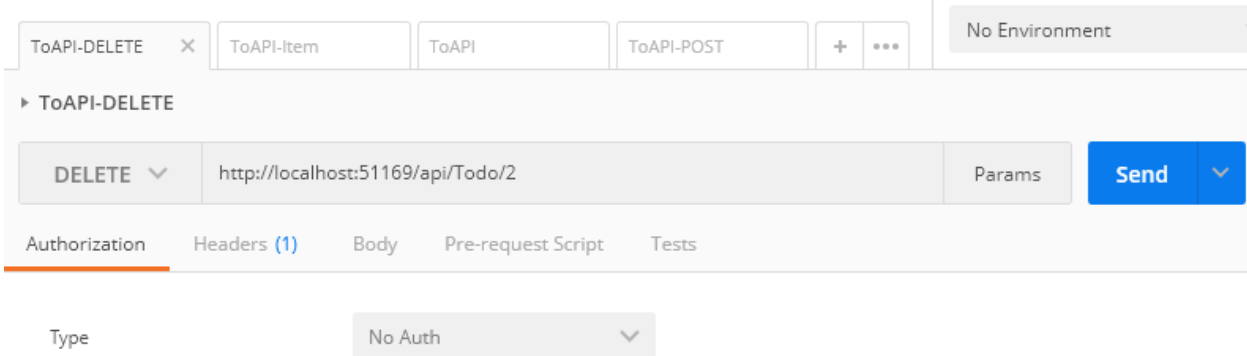


Sử dụng POSTMAN cho phương thức PUT – cập nhật:



Hình 11-6 : Sử dụng POSTMAN với PUT action

Sử dụng POSTMAN cho phương thức DELETE:



Hình 11-7 : Sử dụng POSTMAN với DELETE action

**Xây dựng trang web tĩnh gọi API:**

Tạo trang api.html trong wwwroot với nội dung sau:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>To-do CRUD</title>
</head>
<body>
  <h1>To-do CRUD</h1>
  <h3>Add</h3>
  <form action="javascript:void(0);" method="POST" onsubmit="addItem()">
    <input type="text" id="add-name" placeholder="New to-do">
    <input type="submit" value="Add">
  </form>

  <div id="spoiler">
    <h3>Edit</h3>
    <form class="my-form">
      <input type="hidden" id="edit-id">
      <input type="checkbox" id="edit-isComplete">
```

```
<input type="text" id="edit-name">
<input type="submit" value="Edit">
<a onclick="closeInput()" aria-label="Close">&#10006;</a>
</form>
</div>

<p id="counter"></p>

<table>
  <tr>
    <th>Is Complete</th>
    <th>Name</th>
    <th></th>
    <th></th>
  </tr>
  <tbody id="todos"></tbody>
</table>

<script src="https://code.jquery.com/jquery-3.3.1.min.js"
  integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
  crossorigin="anonymous"></script>
<script src="js/site.js"></script>
</body>
</html>
```

Mở file site.js định nghĩa các sự kiện:

```
const uri = 'api/todo';
let todos = null;
function getCount(data) {
  const el = $('#counter');
  let name = 'to-do';
  if (data) {
    if (data > 1) {
      name = 'to-dos';
    }
    el.text(data + ' ' + name);
  } else {
    el.html('No ' + name);
  }
}

$(document).ready(function () {
  getData();
});

function getData() {
  $.ajax({
    type: 'GET',
    url: uri,
    success: function (data) {
      $('#todos').empty();
      getCount(data.length);
      $.each(data, function (key, item) {
        const checked = item.isComplete ? 'checked' : '';
```

```
                $('<tr><td><input disabled="true" type="checkbox" ' + checked
+ '></td>' +
                '<td>' + item.name + '</td>' +
                '<td><button onclick="editItem(' + item.id +
')">Edit</button></td>' +
                '<td><button onclick="deleteItem(' + item.id +
')">Delete</button></td>' +
                '</tr>').appendTo($('#todos'));
        });
        todos = data;
    }
});
}

function addItem() {
    const item = {
        'name': $('#add-name').val(),
        'isComplete': false
    };

    $.ajax({
        type: 'POST',
        accepts: 'application/json',
        url: uri,
        contentType: 'application/json',
        data: JSON.stringify(item),
        error: function (jqXHR, textStatus, errorThrown) {
            alert('here');
        },
        success: function (result) {
            getData();
            $('#add-name').val('');
        }
    });
}

function deleteItem(id) {
    $.ajax({
        url: uri + '/' + id,
        type: 'DELETE',
        success: function (result) {
            getData();
        }
    });
}

function editItem(id) {
    $.each(todos, function (key, item) {
        if (item.id === id) {
            $('#edit-name').val(item.name);
            $('#edit-id').val(item.id);
        }
    });
}
```

```
        $('#edit-isComplete').val(item.isComplete);
    }
});
$('#spoiler').css({ 'display': 'block' });
}

$('.my-form').on('submit', function () {
    const item = {
        'name': $('#edit-name').val(),
        'isComplete': $('#edit-isComplete').is(':checked'),
        'id': $('#edit-id').val()
    };

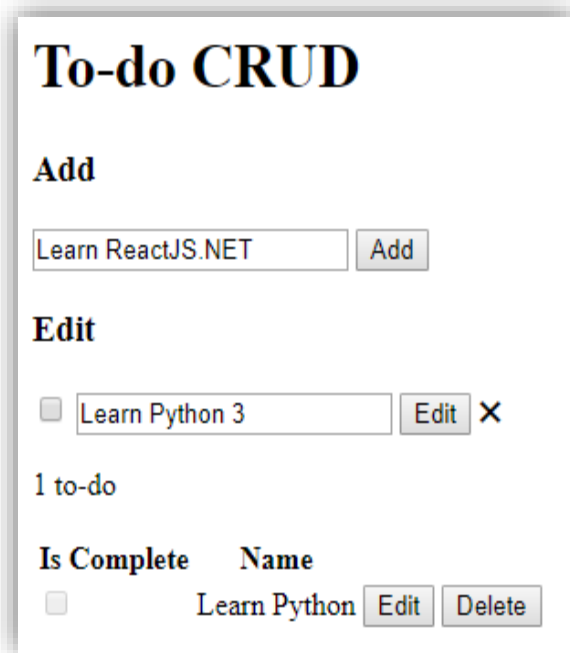
    $.ajax({
        url: uri + '/' + $('#edit-id').val(),
        type: 'PUT',
        accepts: 'application/json',
        contentType: 'application/json',
        data: JSON.stringify(item),
        success: function (result) {
            getData();
        }
    });

    closeInput();
    return false;
});

function closeInput() {
    $('#spoiler').css({ 'display': 'none' });
}
```

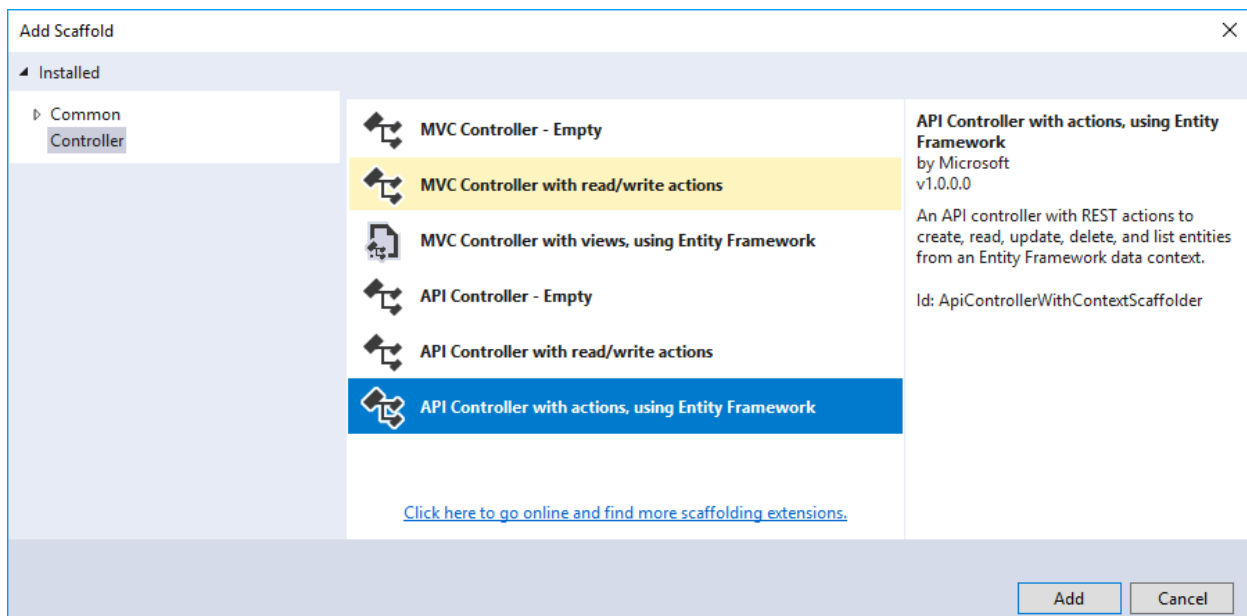
- Hàm addItem() tiến hành tạo mới TodoItem, gửi lên server theo phương thức POST, kiểu JSON.
- Hàm getData() để lấy toàn bộ các TodoItem, sau đó duyệt qua từng phần tử để thêm vào bảng.
- Hàm deleteItem() tiến hành xóa một TodoItem cụ thể, gửi lên server theo phương thức DELETE và cập nhật lại danh sách các TodoItem.

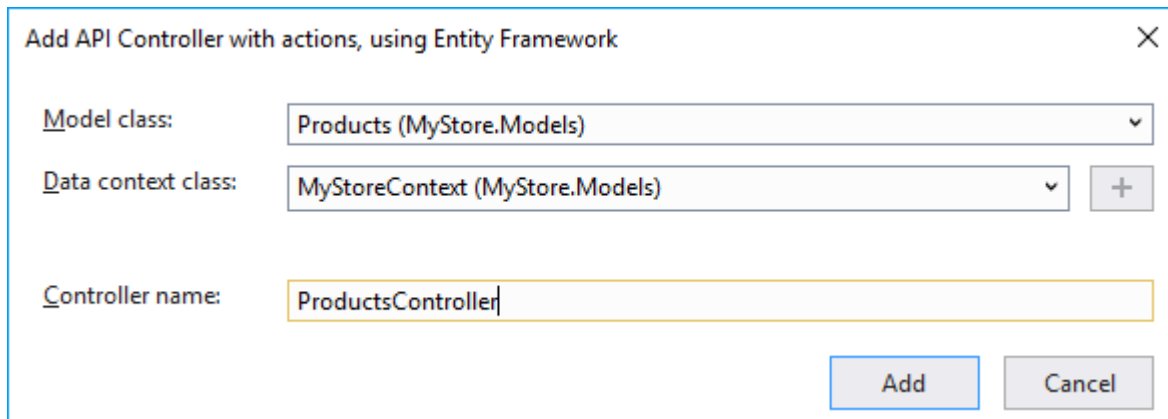
Một số màn hình thể hiện:



### 11.3.2 Xây dựng API dùng EF kết nối SQL Server

Tạo Controller dạng API sử dụng Entity Framework.





Add API Controller with actions, using Entity Framework

Model class: Products (MyStore.Models)

Data context class: MyStoreContext (MyStore.Models)

Controller name: ProductsController

Add Cancel

Chọn Model class và DataContext class tương ứng.

Mã nguồn controller code sinh ra:

```
[Produces("application/json")]
[Route("api/Products")]
public class ProductsController : Controller
{
    private readonly MyStoreContext _context;

    public ProductsController(MyStoreContext context)
    {
        _context = context;
    }

    // GET: api/Products
    [HttpGet]
    public IEnumerable<Products> GetProducts()
    {
        return _context.Products;
    }

    // GET: api/Products/5
    [HttpGet("{id}")]
    public async Task<IActionResult> GetProducts([FromRoute] int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var products = await _context.Products.SingleOrDefaultAsync(m =>
m.Id == id);

        if (products == null)
        {
            return NotFound();
        }

        return Ok(products);
    }
}
```

```
// PUT: api/Products/5
[HttpPut("{id}")]
public async Task<IActionResult> PutProducts([FromRoute] int id,
[FromBody] Products products)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != products.Id)
    {
        return BadRequest();
    }

    _context.Entry(products).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!ProductsExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}

// POST: api/Products
[HttpPost]
public async Task<IActionResult> PostProducts([FromBody] Products
products)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    _context.Products.Add(products);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetProducts", new { id = products.Id },
products);
}
```

```
    }

    // DELETE: api/Products/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteProducts([FromRoute] int id)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var products = await _context.Products.SingleOrDefaultAsync(m =>
m.Id == id);
        if (products == null)
        {
            return NotFound();
        }

        _context.Products.Remove(products);
        await _context.SaveChangesAsync();

        return Ok(products);
    }

    private bool ProductsExists(int id)
    {
        return _context.Products.Any(e => e.Id == id);
    }
}
```



## Chương 12: Bảo mật

### 12.1 Identity

ASP.NET Core Identity là một thành phần (built-in) của ASP.NET Core, cung cấp các tính năng đầy đủ và đa dạng về authentication. Bao gồm: Tạo tài khoản, login với username và password, cập nhật profile, quản lý vai trò (Role) người dùng. Ngoài ra, nó hỗ trợ xác thực nhiều lớp, đăng nhập bằng mạng xã hội với Google, Facebook cũng như kết nối tới dịch vụ OAuth 2.0 và OpenID.

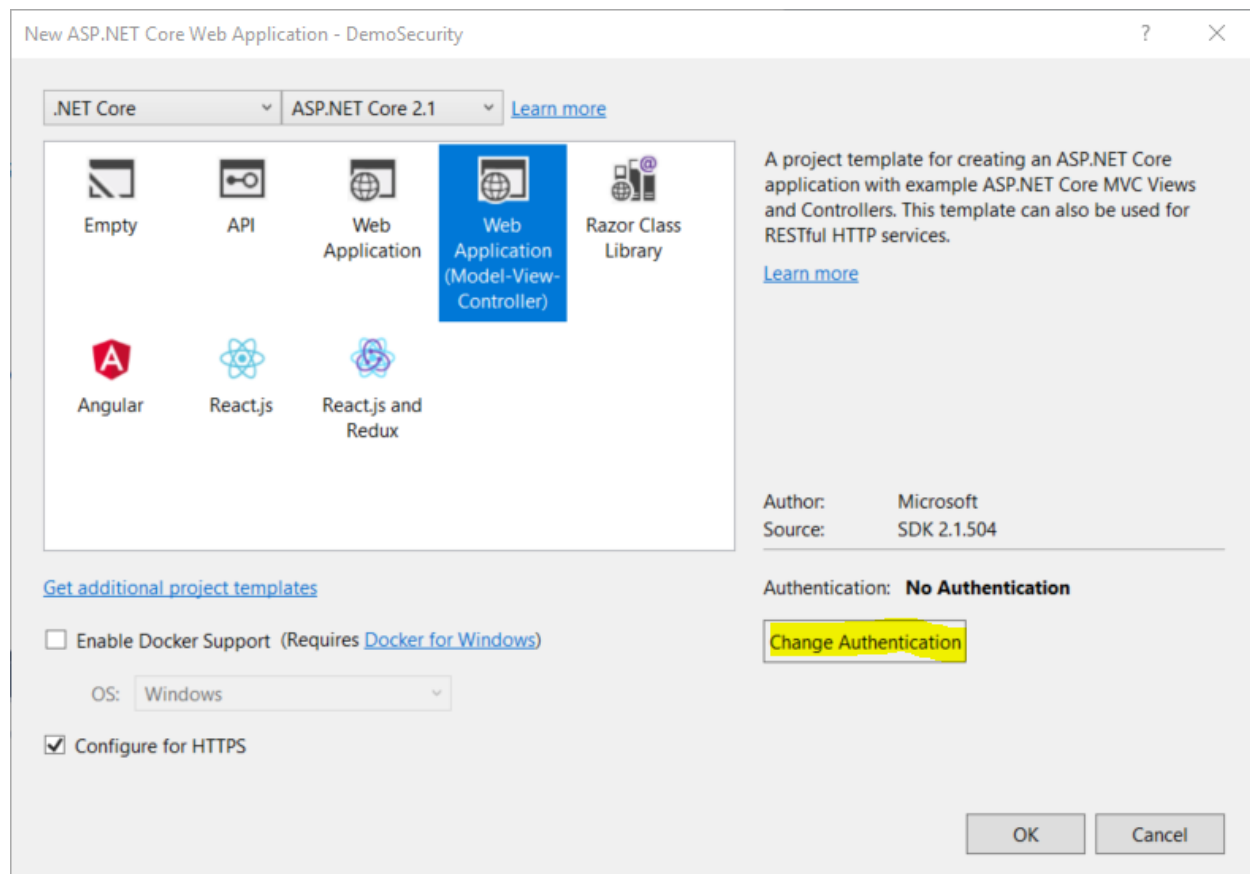
Bạn có thể cấu hình ASP.NET Core để sử dụng với SQL Server nhằm lưu trữ username, password và dữ liệu profile.

### 12.2 Authentication

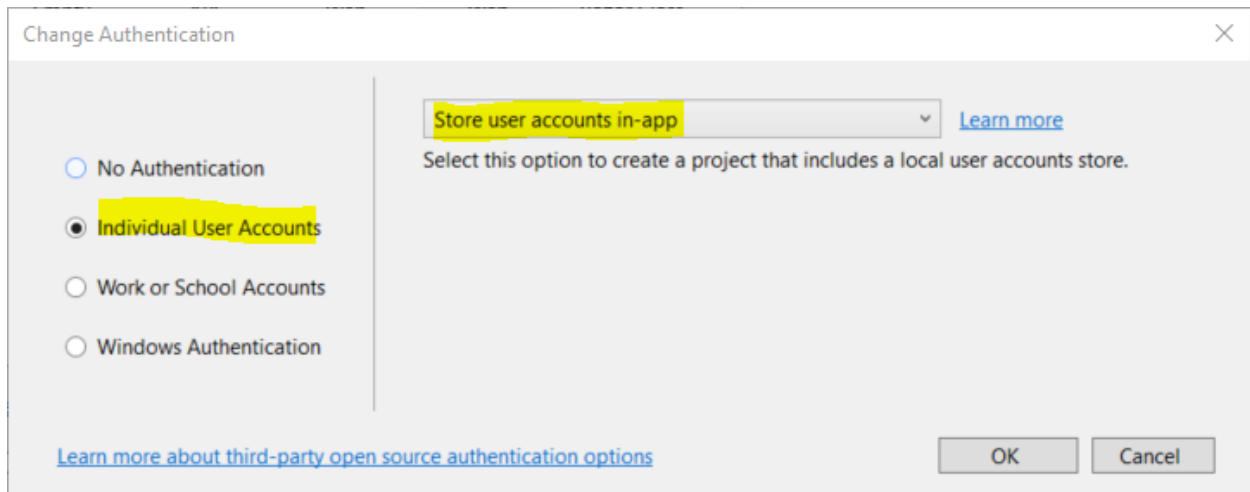
Để dễ hiểu, trước hết chúng ta tìm hiểu các chức năng security sự hỗ trợ sẵn trong MVC, sau đó chúng ta sẽ tìm hiểu sâu hơn.

#### 12.2.1 Tạo dự án có hỗ trợ security

Để tận dụng các chức năng security được hỗ trợ sẵn, lúc tạo dự án bạn phải chọn **Individual User Accounts**.



Hình 12-1: Màn hình chọn loại security

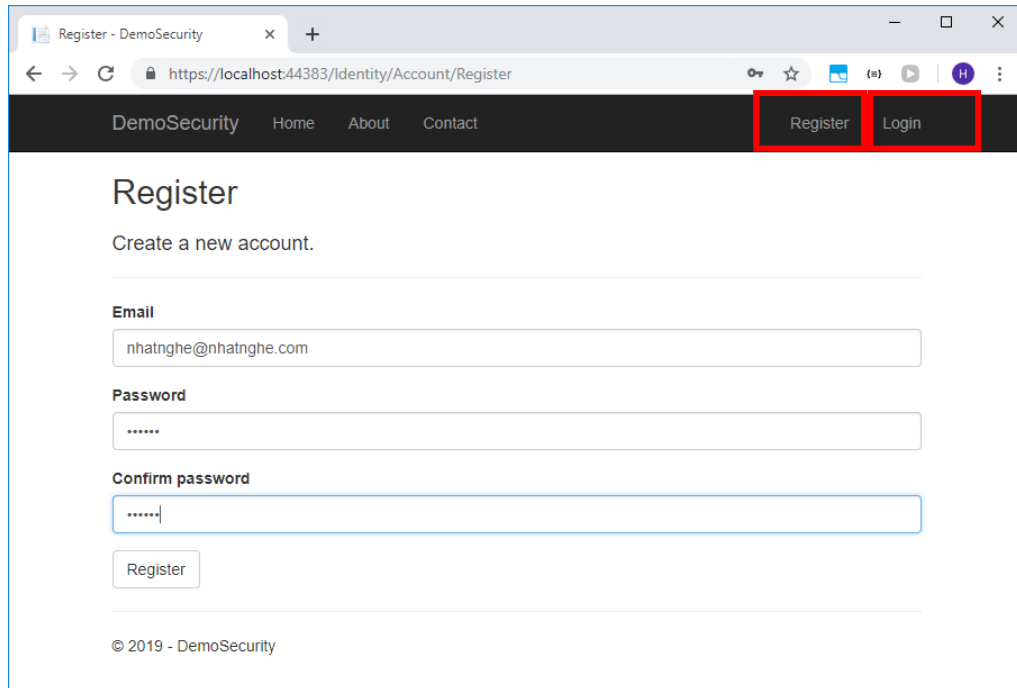


Hình 12-2: Các loại security

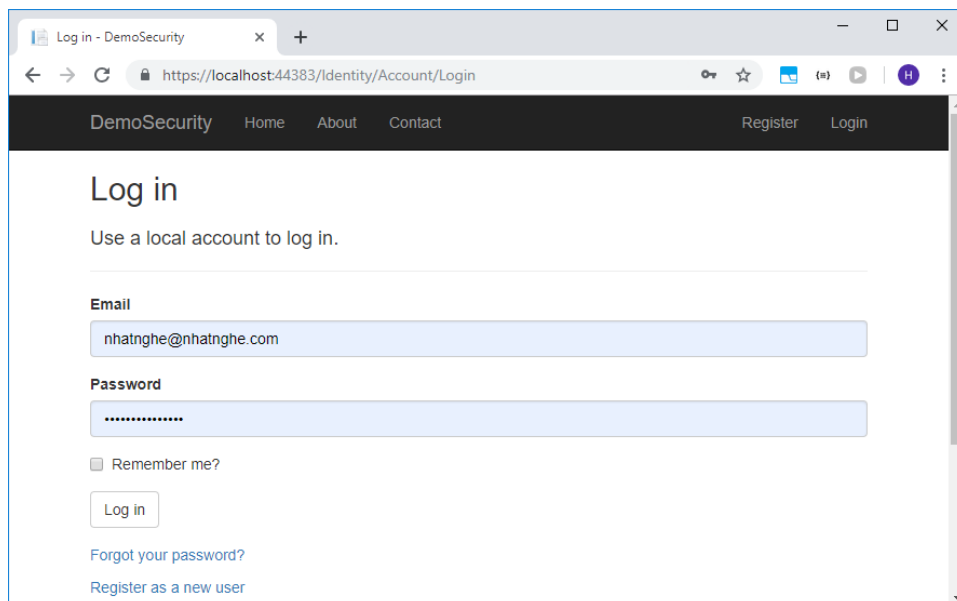
<i>Loại</i>	<i>Mô tả</i>
<b>No Authentication</b>	Ứng dụng không hỗ trợ Security
<b>Individual User Accounts</b>	Ứng dụng sử dụng tài khoản được quản lý bởi SQL Server hoặc từ gmail, facebook...
<b>Work or School Accounts</b>	Ứng dụng sử dụng tài khoản được cung cấp cho tổ chức hoặc trường học
<b>Windows Authentication</b>	Ứng dụng chạy trên intranet tức sử dụng tài khoản windows để đăng nhập.

Mỗi thể loại security khác nhau sẽ phù hợp với các ứng dụng khác nhau. Trong trường hợp này, chúng ta chọn Individual User Accounts vì ứng dụng của chúng ta chạy trên Internet và sử dụng SQL Server để lưu trữ thành viên hoặc đăng nhập từ các hệ thống khác như gmail, facebook...

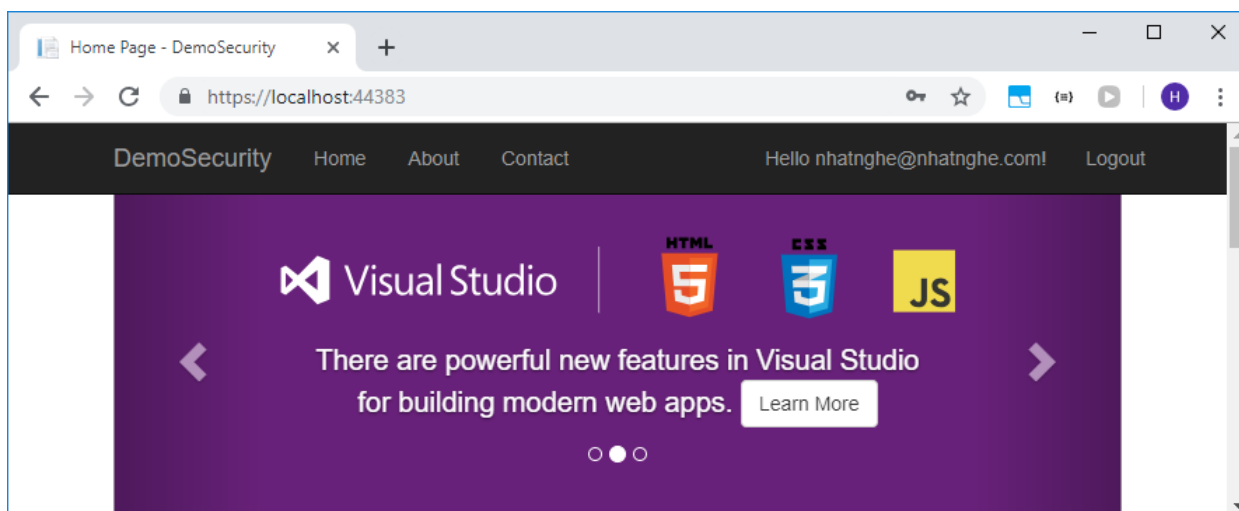
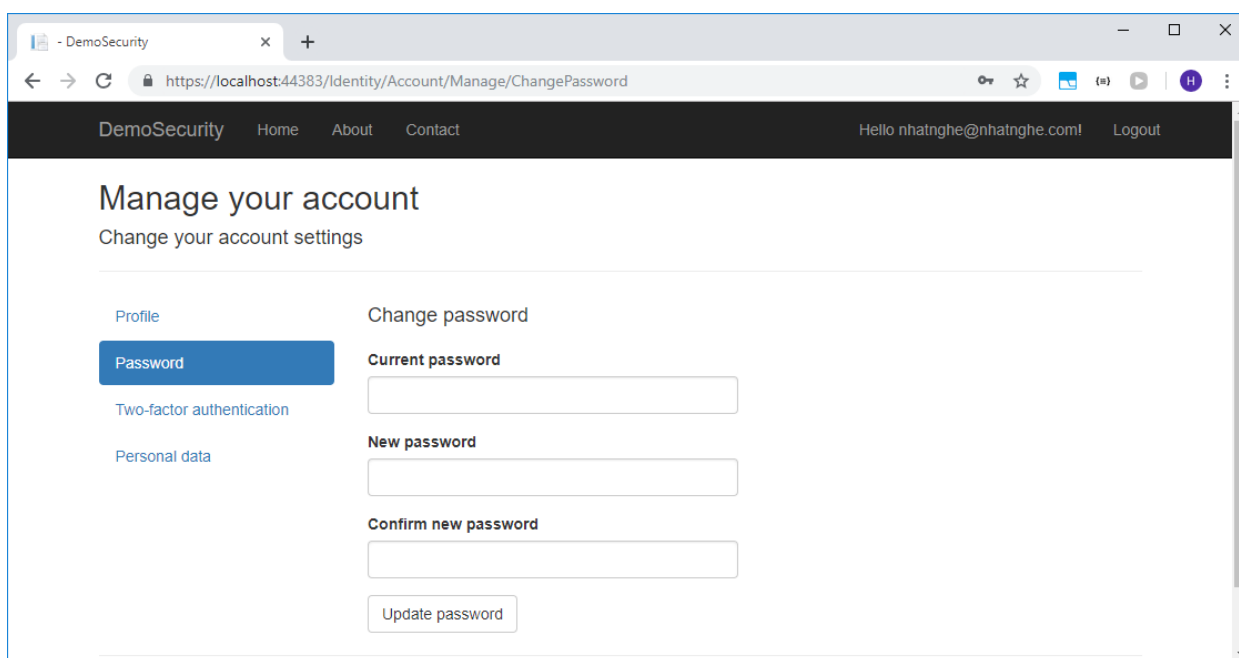
Sau khi dự án loại này được tạo, bạn đã có thể đăng ký, đăng nhập, đổi mật khẩu, đăng xuất với tài khoản cục bộ hoặc bên ngoài (gmail, facebook...).



**Hình 12-3: Đăng ký Thành viên**

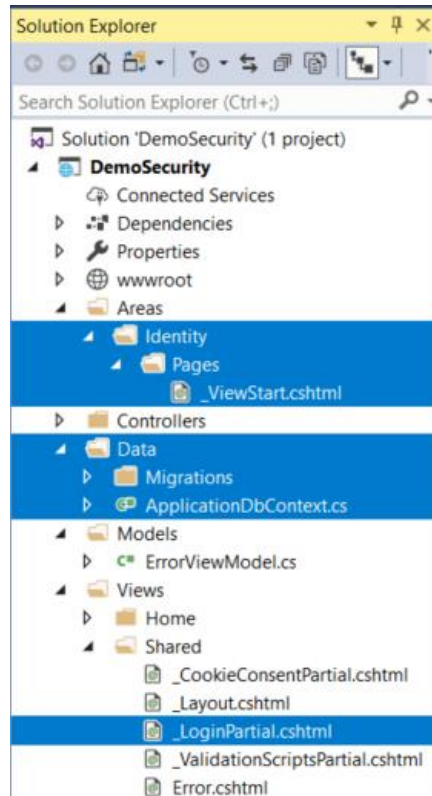


**Hình 12-4: Màn hình đăng nhập**

**Hình 12-5: Đăng nhập thành công****Hình 12-6: Màn hình đổi mật khẩu**

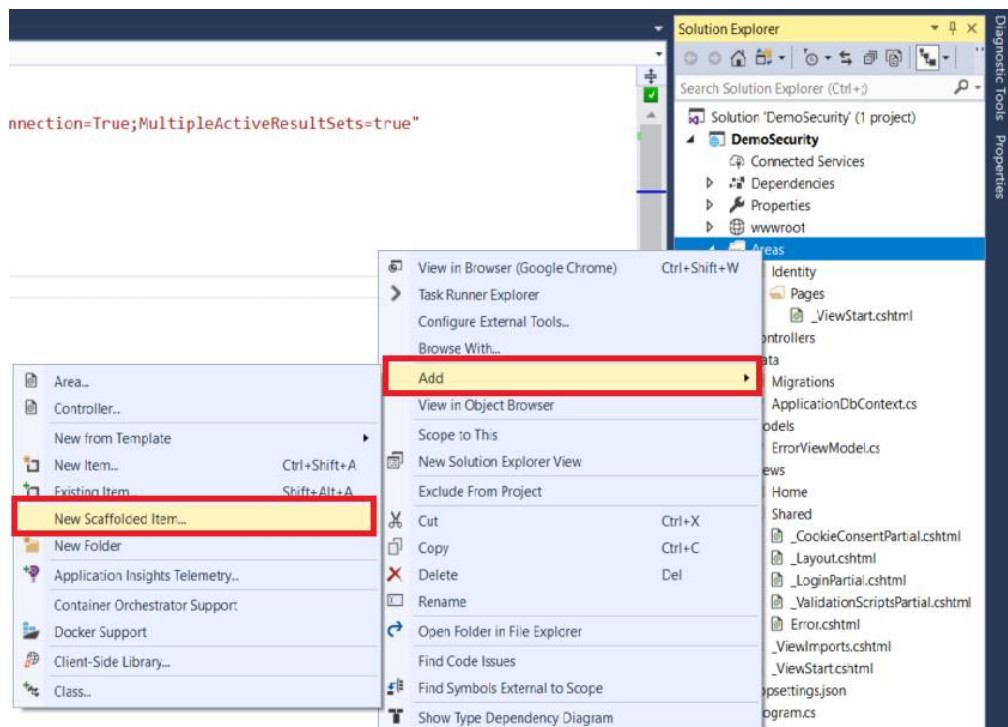
Một câu hỏi lớn là tại sao khi mới tạo dự án đã có sẵn các chức năng security ở trên? Đó là động lực giúp bạn tìm hiểu sâu hơn về điều này của dự án.

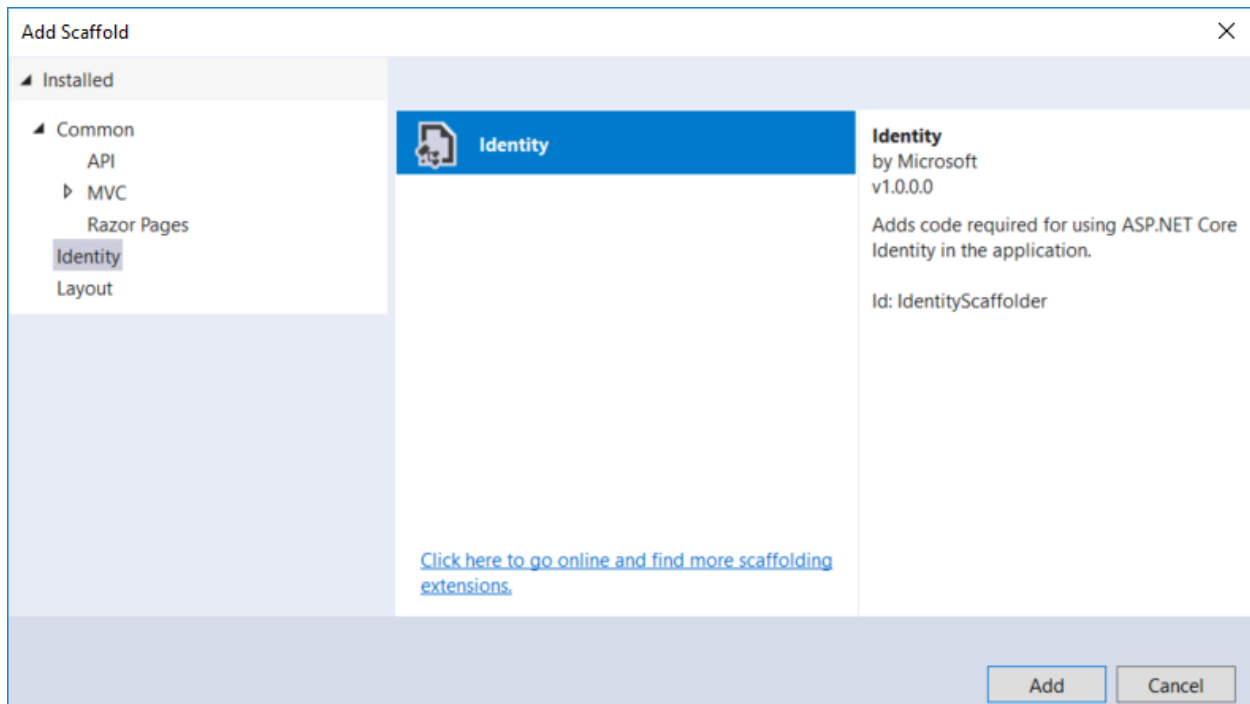
Sau khi dự án Individual User Accounts được tạo ra thì một số thành phần của dự án liên quan đến security đã được cài đặt sẵn. Nhiệm vụ của chúng ta là cần tìm hiểu để nâng cấp các thành phần này hoàn thiện hơn theo ý riêng của mình đồng thời bổ sung thêm các thành phần/chức năng còn thiếu.



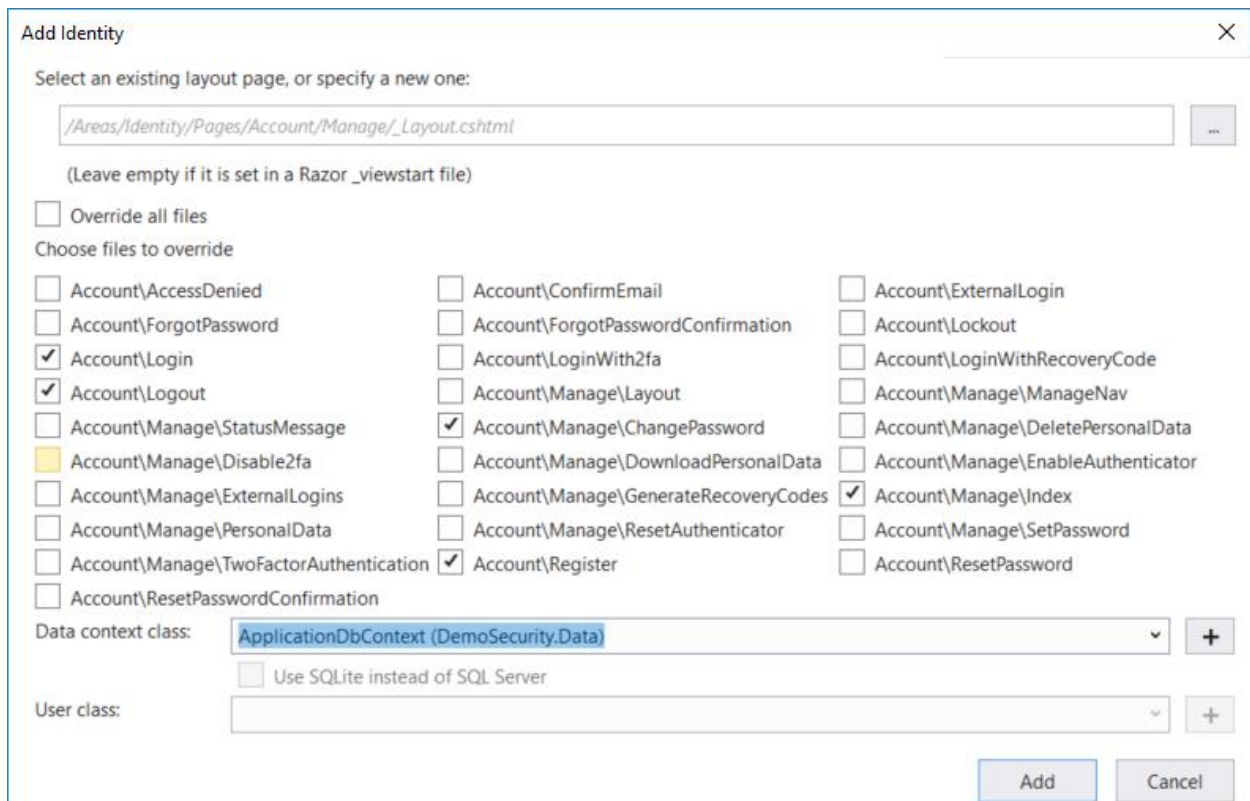
Hình 12-7: Cấu trúc project lúc tạo với Individual Identity

Muốn custom lại các màn hình chức năng, bạn cần right click lên mục **Area**, bấm chọn **Add**, chọn **New Scaffolded Item**, ...





Chọn màn hình muốn chỉnh sửa:



### 12.2.2 Cấu hình Cookie authentication

Nếu người dùng chọn loại dự án là No Authentication thì phải tự cấu hình phần Authentication.

#### 12.2.2.1 Đánh dấu [không] cần Authorize

Để đánh dấu controller hoặc action nào cần authorize thì thêm Annotation [Authorize] phía trước class hoặc method đó. Ví dụ:

```
namespace DemoSecurity.Controllers
{
    [Authorize]
    public class CustomerController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Logout()
        {
            return View();
        }

        public IActionResult Profile()
        {
            return View();
        }
    }
}
```

Để đánh dấu action đó được truy xuất ở bất kỳ nơi đâu, cần đánh dấu với annotation **[AllowAnonymous]**.

```
[Authorize]
public class CustomerController : Controller
{
    [AllowAnonymous]
    public IActionResult Login()
    {
        return View();
    }

    [HttpPost, AllowAnonymous]
    public IActionResult Login(string username, string password)
    {
        return View();
    }
}
```

### 12.2.2.2 Cấu hình Startup.cs

Trước tiên, bạn cần tạo service Authentication Middleware trong method **ConfigureServices** trong class Startup.cs.

```
services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie();
```

AuthenticationScheme được truyền tới method AddAuthentication thiết lập giá trị default authentication scheme cho ứng dụng. AuthenticationScheme là hữu ích khi có nhiều thể hiện của cookie authentication và bạn muốn xác thực với một scheme nhất định. Thiết lập AuthenticationScheme với enum CookieAuthenticationDefaults.AuthenticationScheme cung cấp một giá trị của Cookies cho scheme. Bạn có thể cung cấp bất kỳ giá trị string nào cho scheme. Có rất nhiều option khác mà có thể sử dụng trong các trường hợp cụ thể, bạn có thể tham khảo đầy đủ tại đây [CookieAuthenticationOptions](#).

Trong method `Configure`, sử dụng method `UserAuthentication` để triệu gọi `Authentication Middleware`. Lưu ý cần gọi phương thức này trước khi gọi `UseMvcWithDefaultRoute` hoặc `UseMvc`

```
app.UseAuthentication();
```

### 12.2.2.3 Hàm xử lý login

Giả sử đã định nghĩa model login đơn giản dung kiểm tra và hiển thị view như sau:

```
public class LoginViewModel
{
    public string Username { get; set; }

    public string Password { get; set; }

    public string RequestPath { get; set; }
}
```

Để tạo một cookie năm giữ thông tin người dùng, bạn phải xây dựng một `ClaimsPrincipal`. Thông tin người dùng được serialized và lưu trữ trong cookie. Sau đó, các bạn sẽ gọi `SignInAsync` để thực hiện login với thiết lập scheme đã được cấu hình tại `Startup`, `principal` và `option` cho việc lưu trữ cookie.

```
[HttpPost]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (!IsAuthenticated(model.Username, model.Password))
        return View();

    // create claims
    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.Name, "Cookie authentication demo"),
        new Claim(ClaimTypes.Email, model.Username)
    };

    // create identity
    ClaimsIdentity identity = new ClaimsIdentity(claims, "cookie");

    // create principal
    ClaimsPrincipal principal = new ClaimsPrincipal(identity);

    // sign-in
    await HttpContext.SignInAsync(
        scheme: "DemoSecuritySchemeNN",
        principal: principal,
        properties: new AuthenticationProperties()
    );

    return Redirect(model.RequestPath ?? "/");
}

private bool IsAuthenticated(string username, string password)
{
    //check in database
}
```



```
    return (username == "nhatnghe" && password == "Nh@tNgh3");  
}
```

#### 12.2.2.4 Hàm xử lý logout

Sử dụng hàm `SignInAsync` của `HttpContext` với đúng `scheme name` đã cấu hình trước đó và đồng thời xóa cookie của người dùng.

```
public async Task<IActionResult> Logout(string requestPath)  
{  
    await HttpContext.SignOutAsync(scheme: "DemoSecuritySchemeNN");  
  
    return RedirectToAction("Login");  
}
```

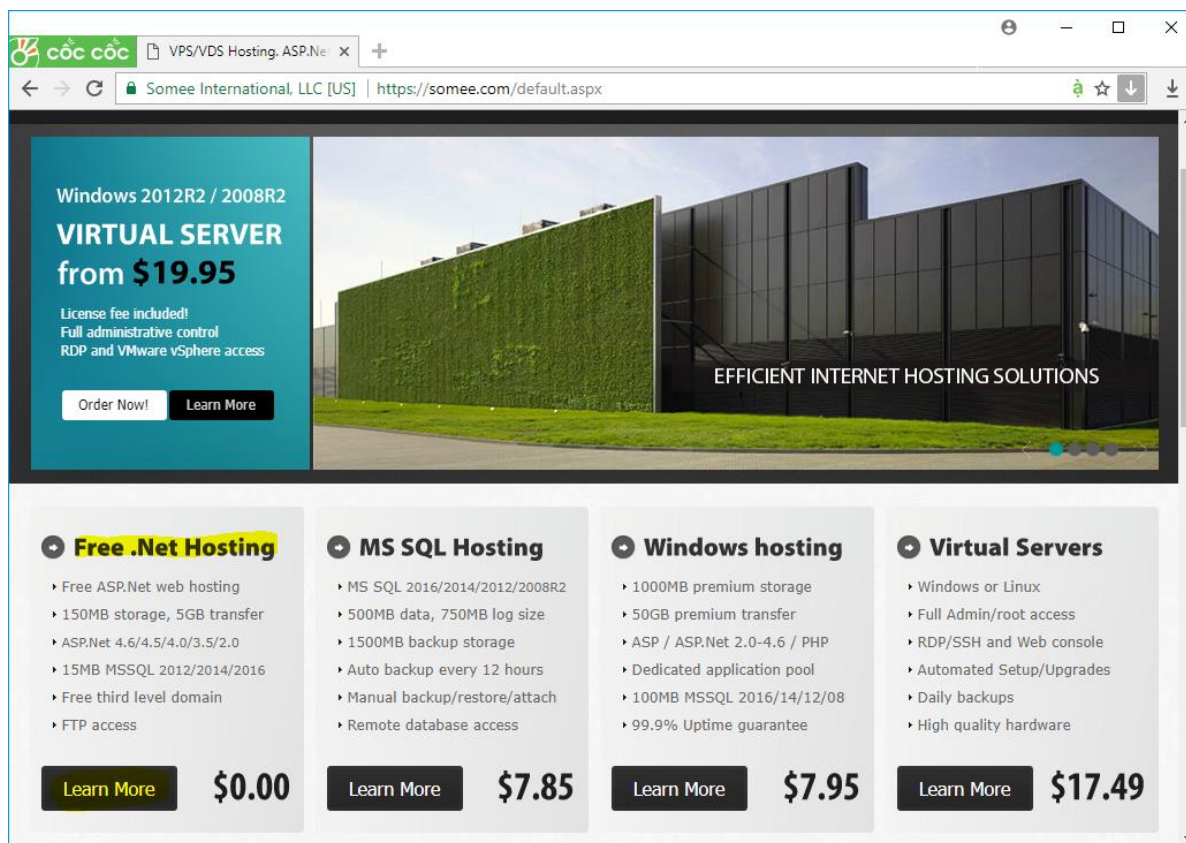
#### 12.2.2.5 Kiểm tra/Lấy thông tin đăng nhập trên View

Ở View, để kiểm tra đã đăng nhập chưa dùng thuộc tính `User.Identity.IsAuthenticated` và lấy thông tin đăng nhập dùng: `@User.Identity.Name`

## Chương 13: **UPLOAD FILE LÊN HOST**

### \_ Các bước thực hiện

**Bước 1** : Truy cập trang <https://somee.com>, chọn mục **Free. Net hosting**. Sau đó click vào **Learn more**



The screenshot shows a web browser window with the URL <https://somee.com/default.aspx>. The page features a large banner for 'VIRTUAL SERVER from \$19.95' with a 'Learn More' button. Below the banner, there are four columns of hosting plans, each with a 'Learn More' button and a price:

Plan Name	Price
<b>Free .Net Hosting</b>	<b>\$0.00</b>
<b>MS SQL Hosting</b>	<b>\$7.85</b>
<b>Windows hosting</b>	<b>\$7.95</b>
<b>Virtual Servers</b>	<b>\$17.49</b>

**Free .Net Hosting** features include: Free ASP.Net web hosting, 150MB storage, 5GB transfer, ASP.Net 4.6/4.5/4.0/3.5/2.0, 15MB MSSQL 2012/2014/2016, Free third level domain, and FTP access.

**MS SQL Hosting** features include: MS SQL 2016/2014/2012/2008R2, 500MB data, 750MB log size, 1500MB backup storage, Auto backup every 12 hours, Manual backup/restore/attach, and Remote database access.

**Windows hosting** features include: 1000MB premium storage, 50GB premium transfer, ASP / ASP.Net 2.0-4.6 / PHP, Dedicated application pool, 100MB MSSQL 2016/14/12/08, and 99.9% Uptime guarantee.

**Virtual Servers** features include: Windows or Linux, Full Admin/root access, RDP/SSH and Web console, Automated Setup/Upgrades, Daily backups, and High quality hardware.

**Bước 2** : Click vào **Order now**

**Free Hosting package**

- 1 x Hosting plan "Freebie"
- Forced advertising
- Storage capacity: **150MB**
- Monthly transfer: **5GB/Month**
- Web domains: **1**
- ASP.Net 4.6/4.5/4.0/3.5/2.0
- AJAX 3.5/1.0
- Silverlight
- MS Access 2007, 2010
- Dedicated web application pool

1 x Email plan "Forwarder"

1 x MS SQL Plan "Novice"

- MS SQL database size: **15MB**
- MS SQL log size: **20MB**
- Backup storage size: **40MB**

**Order now**

### Free Windows ASP.Net hosting

Free web hosting on Windows 2012R2. Supported features:  
IIS 8.5; ASP; ASP.Net 4.6/4.5/4.0/3.5/2.0; MVC 1.0/2.0/3.0/4.0/5.0; PHP 5; MS SQL Express 2014/2012/2008R2 and other standard components.

#### Free Windows web hosting with fast registration

Since 2004 Somee.com provides free Windows ASP.Net web hosting. This is an absolutely free Windows hosting offer. There is no credit card or other payment information required to pass the registration. We welcome people all around the globe to join our free web hosting offering. You'll find us the best choice for ASP.Net hosting, ASP, PHP hosting, MS SQL and Windows VPS hosting solutions.

#### Instant Hosting setup

You can get your free Windows hosting account up and running within the next few minutes. Try us risk free and start building powerful websites and save money at the same time.

#### Limitations and restrictions of the free web hosting package

In order to cover up the costs of hosting we insert a small advertisement on bottom of every page of your website. The ads are inserted automatically during web response and the source files are not modified. If you are looking for no ads hosting please consider our paid [hosting packages](#). Restrictions: No adult, extremist or hate content. No hacking, illegal software distribution or phishing websites. We also do not tolerate hot linking and proxy website. Please read our [terms of service](#) for the list of all restrictions.

#### Keeping your site active

We have an automated verification system which checks if your free website or database is active. The websites are removed automatically if not visited once during last 30 days. MS SQL databases will be removed if not accessed once with SQL select, insert, update or delete commands during last 30 days. This does not apply to our [paid hosting](#).

**Bước 3:** Điền thông tin để đăng ký tài khoản nếu chưa có tài khoản. Hoặc đăng nhập nếu đã có tài khoản

SOME.com Login or register  
Click here if already registered

Create your account

First name: Nhat  
Last Name: Nghe  
User ID: aspnhatnghe  
Password: .....  
Confirm password: .....  
Email address: aspnhatnghe@gmail.com  
Additional email address:

Prove you're not a robot  
q b s d k 1  
Type the text: qbsdk1

I agree to the [Terms of service](#) and [Privacy policy](#)

**Register new account**

Điền mã code xác nhận:

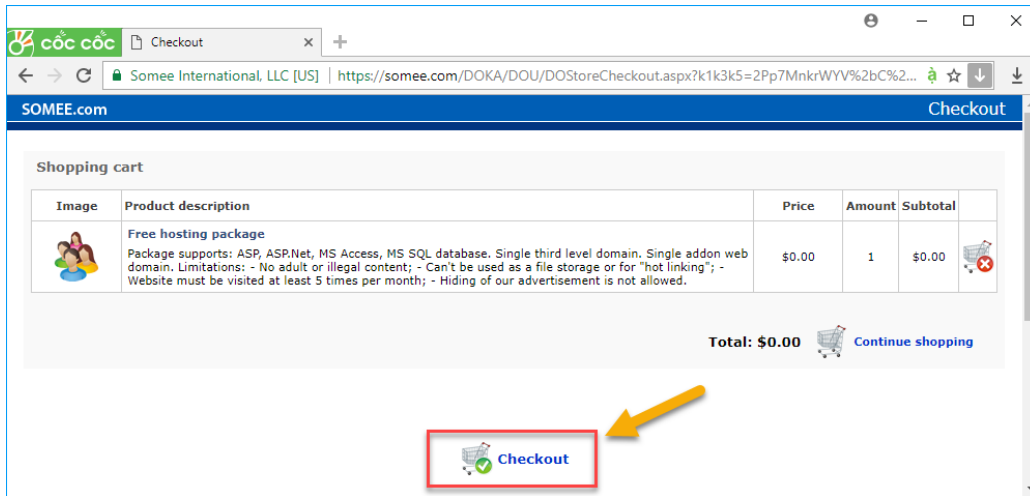
We just sent you an email with validation code. You should receive it within next 10 minutes.  
In order to proceed you need to enter in the field below.  
If you don't get it within 10 minutes [Click here](#) to try again.

Email validation code: 1218707390

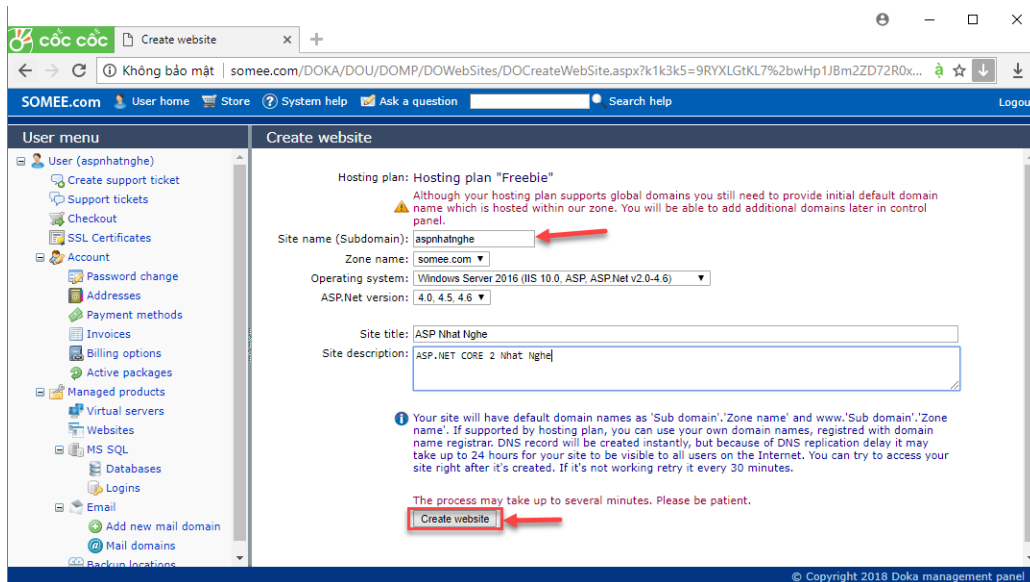
Confirm

© Copyright 2018 Doka management panel

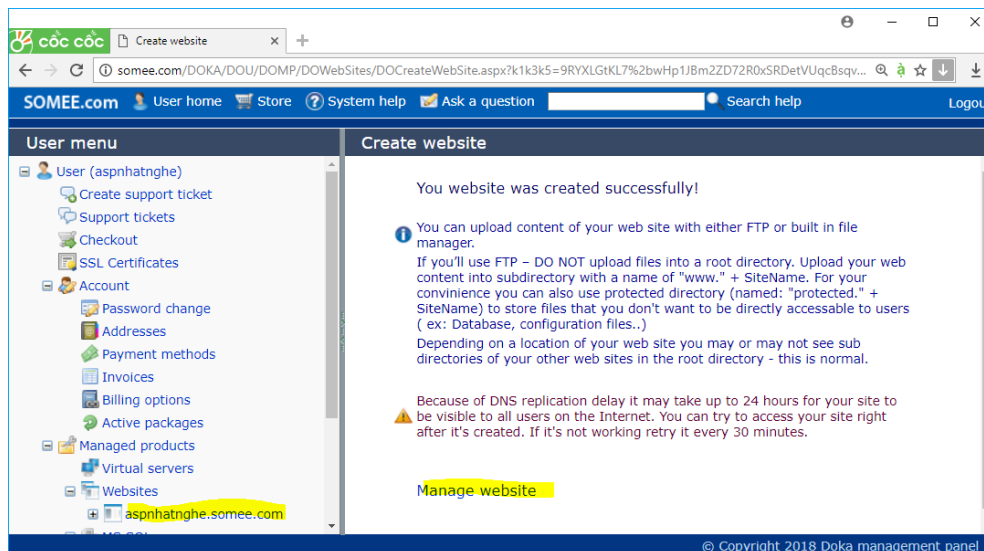
**Bước 4 :** Click chọn **Checkout**



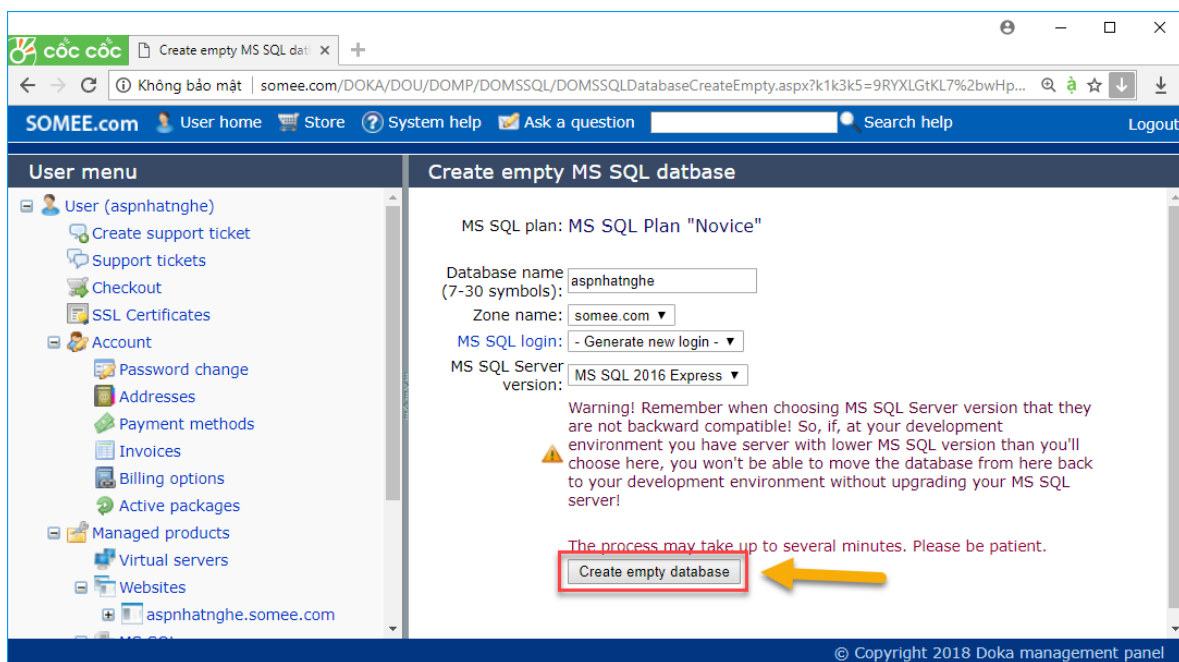
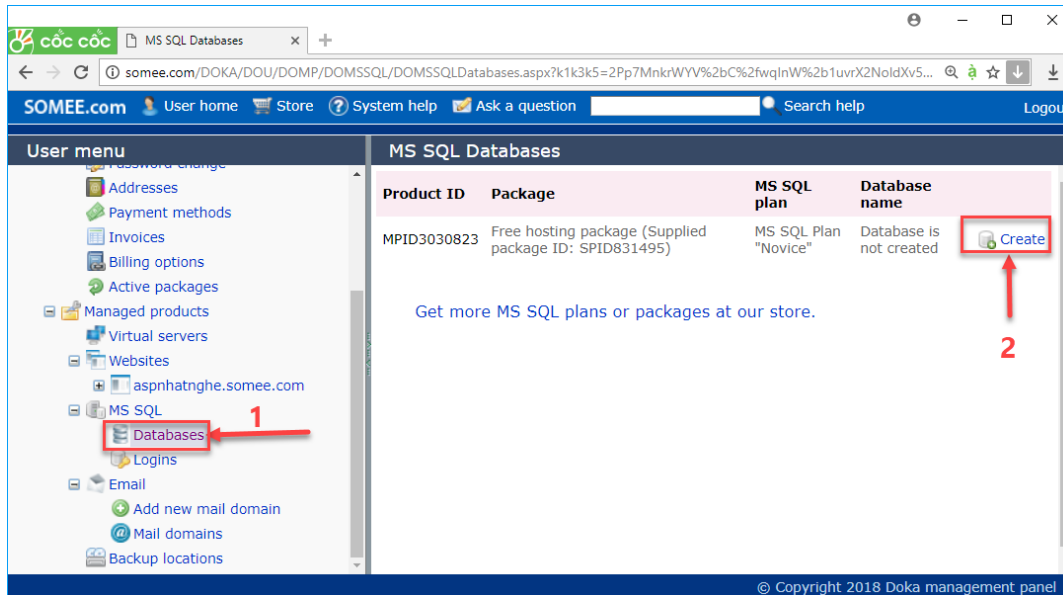
**Bước 5 : Đặt tên Site name và click Create website**



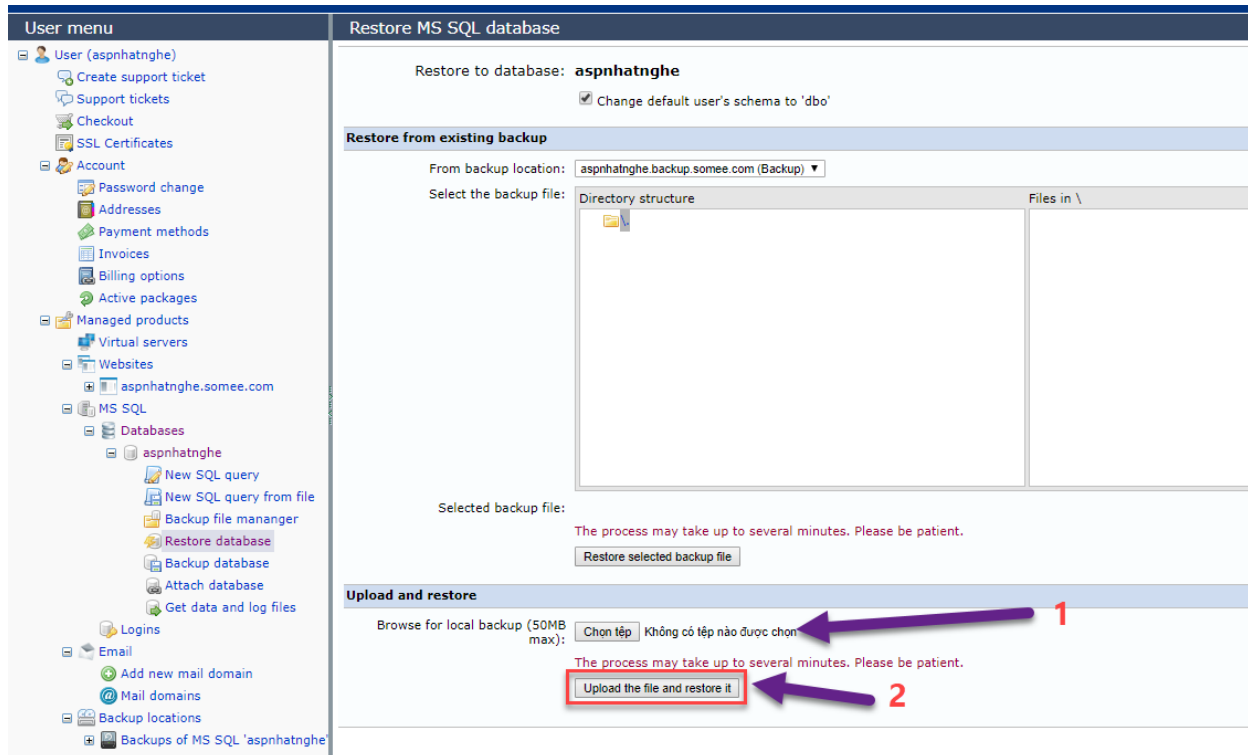
**Kết quả đăng ký hosting thành công:**



**Bước 6 : Chọn mục Database. Sau đó đặt tên database, chọn phiên bản SQL và click Create empty database**



## Bước 7 : Restore database



## Bước 8 : Sửa đoạn code sau ở appsettings.json

```
{
  "ConnectionStrings": {
    "MyStore": "Server=.; Database=MyStore; Integrated Security=SSPI;"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

Thành

```
{
  "ConnectionStrings": {
    "MyStore": "Server=.; Database=MyStore; Integrated Security=SSPI;"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

**Lưu ý :** đoạn code được bôi vàng lấy từ

MS SQL Server address: **aspnhatnghe.mssql.somee.com**  
Login name: **aspnhatnghe\_SQLLogin\_1**  
Login password: **hi31zjzbg5**  
Connection string: **workstation id=aspnhatnghe.mssql.somee.com;packet size=4096;user id=**  **;** **pwd=**  **;** **data source=aspnhatnghe.mssql.somee.com;persist security info=False;initial catalog=aspnhatnghe**

**Copy nguyên chuỗi để thay**

### **Bước 9 : Upload file lên host**

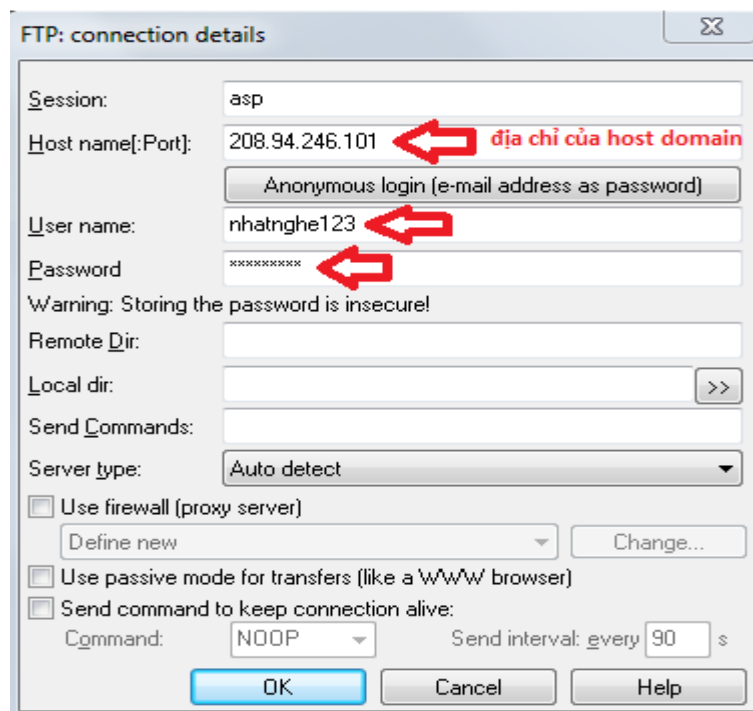
Dùng chương trình **Total Commander**

Điền thông tin

**Host name :** địa chỉ IP hoặc tên domain

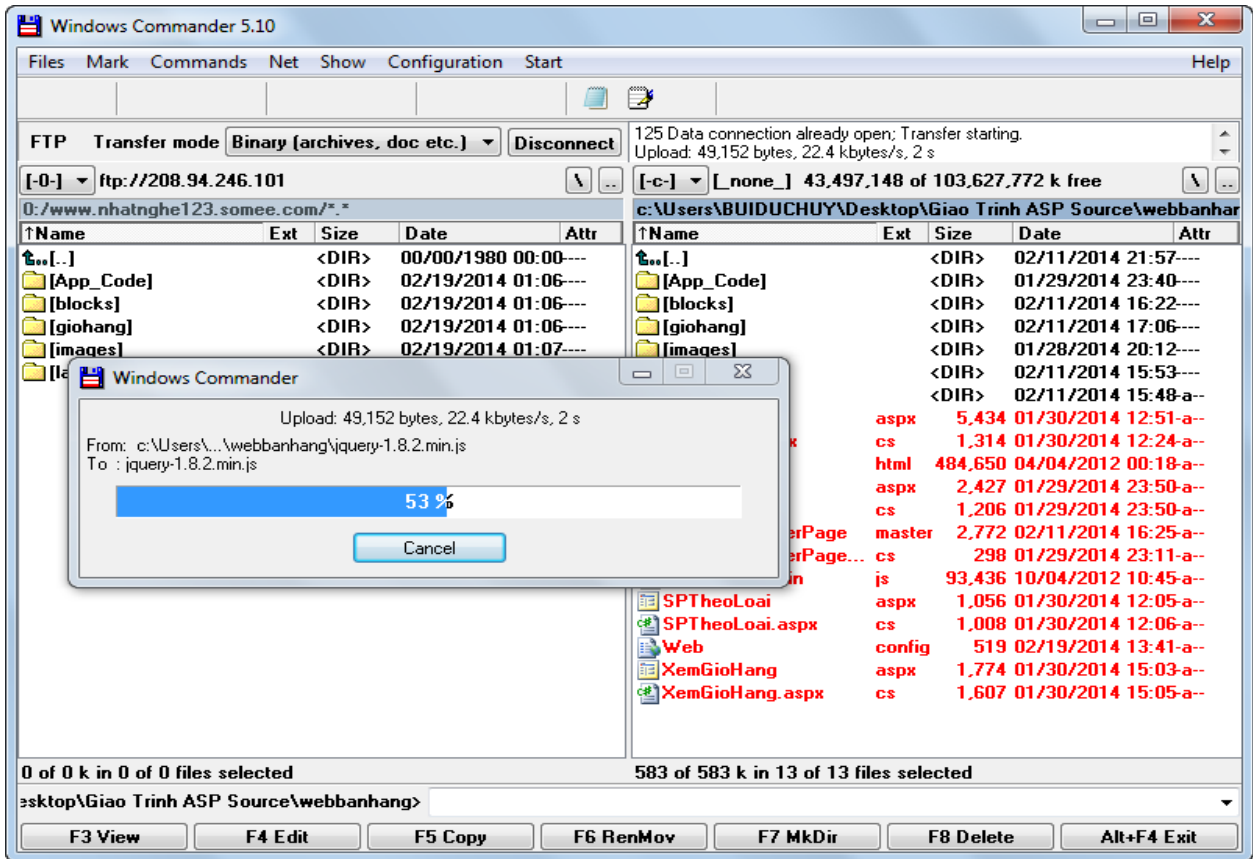
**User name :** user name mà domain đã cấp cho mình

**Password :** password mà domain đã cấp cho mình

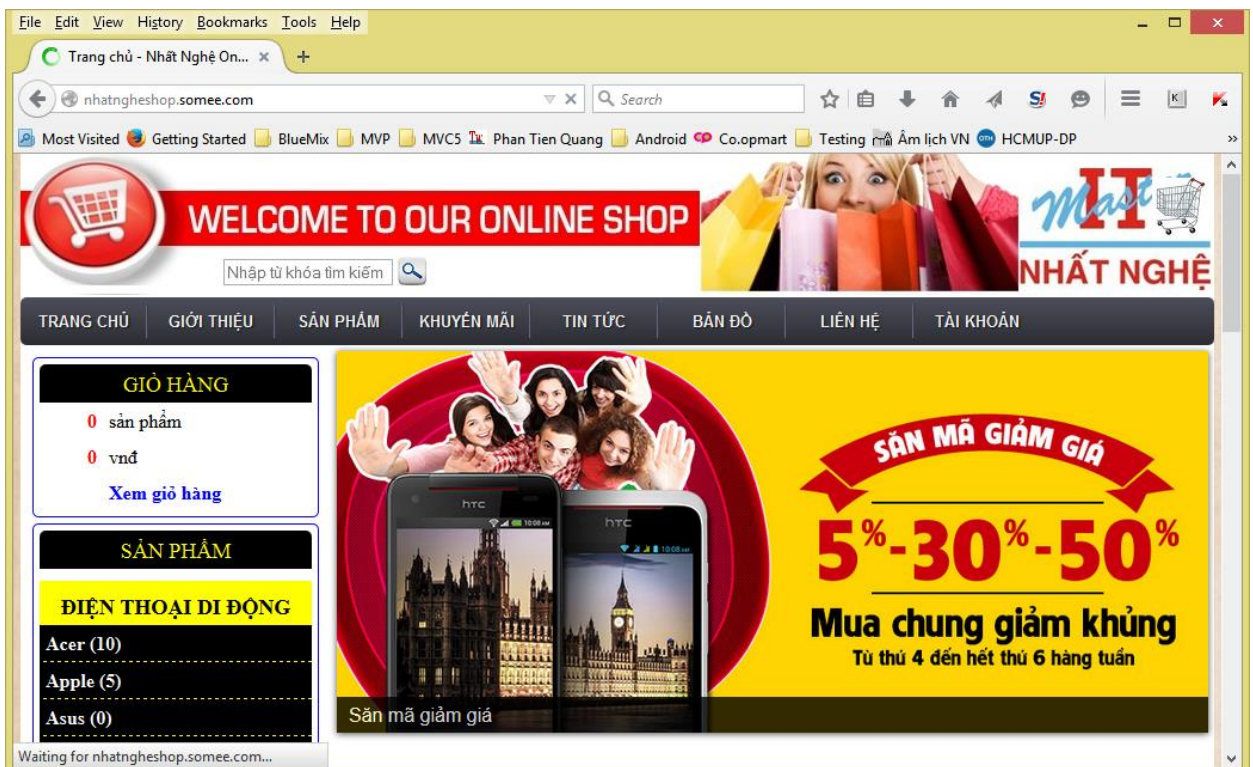




**Kéo những trang, những thư mục qua host**



**Bước 10 : Truy cập vào host vừa đăng ký để kiểm tra**





## Chương 14: Thực hiện dự án

Kết thúc khóa học bằng việc thực hiện một dự án website bán hàng. Bạn sẽ cùng giảng viên thực hiện từng bước để hoàn thành website này.

Thực ra công việc làm dự án của khóa học mang một số ý nghĩa lớn là

- ✓ Hệ thống các bước thực hiện từ bắt đầu cho đến kết thúc.
- ✓ Củng cố lại kiến thức đã học trong quá trình của môn. Với website này, bạn sẽ được ôn tập và vận dụng phần lý thuyết cơ sở đã học để củng cố vững chắc kiến thức
- ✓ Xây dựng một thư viện kỹ thuật lập trình để tra cứu khi cần thiết

### CÁC CÔNG VIỆC CẦN LÀM

- Phân tích chức năng, CSDL
- Xây dựng layout trang chủ, module hóa thành phần giao diện
- Xây dựng EF Code First
- Xây dựng Trang chủ
- Xây dựng Trang hàng hóa
  - o Lọc, phân loại hàng hóa.
  - o Phân trang hàng hóa với ajax.
  - o Tải bổ sung hàng hóa với ajax (lazy loading).
- Xây dựng Trang chi tiết hàng hóa
  - o Thông tin chi tiết.
  - o Hàng cùng loại.
  - o Hàng cùng nhà cung cấp.
  - o Hàng đã xem
- Quản lý Thành viên
  - o Đăng ký
  - o Đăng nhập/Đăng xuất
  - o Quên mật khẩu
  - o Quản lý tài khoản
  - o Quản lý đơn hàng
- Xử lý Giỏ hàng và Thanh toán
  - o Chọn hàng hóa
  - o Quản lý (thêm/xóa/sửa) giỏ hàng
  - o Đặt hàng
  - o Thanh toán kết nối với ngân hàng (áo).
- Phần Admin - Quản lý thêm, xóa, sửa Hàng hóa
- Phần Admin - Quản lý thêm, xóa, sửa Hóa đơn
- Phần Admin – Quản lý thêm, xóa, sửa Khách hàng
- Phần Admin – Quản lý Doanh thu
- Phần Admin – Quản lý tài khoản người dùng, Vai trò (Role), Phân quyền
- Phần Admin – Thống kê
  - o Hàng tồn kho
  - o Doanh số từng mặt hàng
  - o Doanh số theo nhà cung cấp/khách hàng
  - o Doanh số từng tháng/quý/năm
- Phần Admin – Rewrite URL trong ASP.NET Core/SEO

---

## TÀI LIỆU THAM KHẢO

Freeman, A. (2017). *Pro ASP.NET Core MVC 2*. Apress.

Joseph, A., & Ben, A. (2018). *C# 7.0 in a Nutshell*. O'Reilly.

Philip, J., Kevin, G., & Ben, D. (2017). *Building Web Applications with Visual Studio 2017*. Apress.

Rouleau, D. J. (2018). *Beginning Entity Framework Core 2.0*. Apress.

Smith, S. (2017). *Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure*. Microsoft Corporation.

<https://dotnet.github.io/>

<https://docs.microsoft.com/en-us/aspnet/core/>

<https://docs.microsoft.com/en-us/ef/core/>

TRUNG TÂM ĐÀO TẠO CNTT ỦY QUYỀN CỦA MICROSOFT  
105 Bà Huyện Thanh Quan, Quận 3, TP. Hồ Chí Minh.  
TEL: (028) 3 9322 735 - 0913 735 906  
FAX: (028) 3 9322734  
[www.nhatnghe.com](http://www.nhatnghe.com)

# ASP.NET Core 2.0