Buổi 1
Tổng quan về ASP.NET Core và mô hình MVC

I. Giới thiệu về ASP.NET Core

ASP.NET Core là 1 framework mã nguồn mở (*open-source*) và đa nền tảng (*cross-platform*), dùng cho việc xây dựng các ứng dụng hiện đại và có thể tích hợp công nghệ điện toán đám mây, ví dụ các ứng dụng web, ứng dụng mobile, IoT…

Ứng dụng ASP.NET Core có thể chạy trên .NET Core hoặc trên phiên bản đầy đủ của .NET Framework. ASP.NET Core bao gồm các thành phần theo hướng module nhằm giảm thiểu tài nguyên và chi phí phát triển nhưng vẫn giữ lại được sự mềm dẻo, linh hoạt trong việc xây dựng giải pháp CNTT. Những ứng dụng xây dựng bằng ASP.NET Core có thể chạy trên cả Windows, MacOS và Linux.

ASP.NET Core kế thừa từ ASP.NET nhưng có sự thay đổi lớn về kiến trúc do học hỏi từ các framework module hóa khác. ASP.NET Core không còn dựa trên `System.Web.dll` nữa mà dựa trên tập hợp các gói, các module được gọi là *NuGet Package*. Điều này cho phép lập trình viên tối ưu ứng dụng để chỉ bao gồm những package cần thiết, giúp cho ứng dụng nhỏ gọn hơn, bảo mật chặt chẽ hơn, giảm sự phức tạp, tối ưu hiệu suất và giảm chi phí, thời gian cho việc phát triển.

ASP.NET Core cũng được thiết kế để tích hợp nhiều framework front-end, ví dụ AngularJS, KnockoutJS và Bootstrap.

Các tính chất quan trọng của ASP.NET Core:

- Có thể xây dựng và chạy ứng dụng đa nền tảng trên Windows, MacOS và Linux.
- Hợp nhất ASP.NET MVC và ASP.NET Web API.
- Có thể host trên IIS hoặc tự host.
- Có sẵn Dependency Injection.
- Dễ dàng tích hợp với các framework front-end như AngularJS, KnockoutJS, Bootstrap...
- Hỗ trợ cấu hình cho nhiều môi trường.
- Cơ chế HTTP Request pipeline mới.
- Dùng chung toàn bộ NuGet Package.

II. Các công cụ, phần mềm

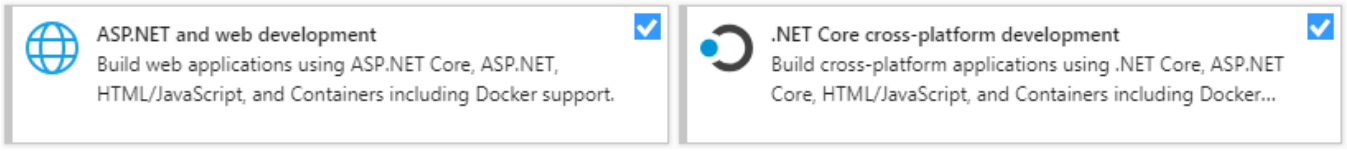
Trong khuôn khổ môn học này, **khuyến cáo** sử dụng Visual Studio[1] phiên bản mới nhất (hiện tại là phiên bản 2019).

Địa chỉ tải VS chính thức từ Microsoft: https://visualstudio.microsoft.com/vs (miễn phí đối với phiên bản Community).

Để có thể phát triển ứng dụng với ASP.NET Core, khi cài đặt VS, cần chọn vào 2 workload sau:

- ASP.NET and web development
- .NET Core cross-platform development

[1] Gọi tắt là VS

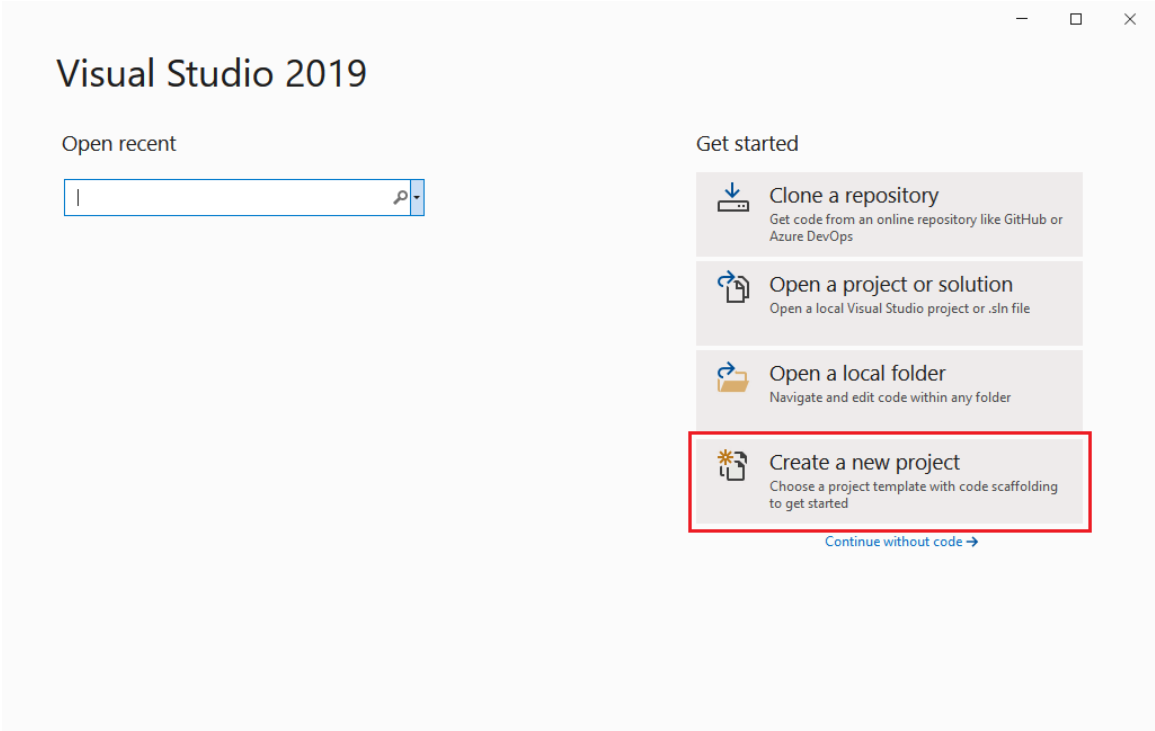


III. Tạo project ASP.NET Core MVC

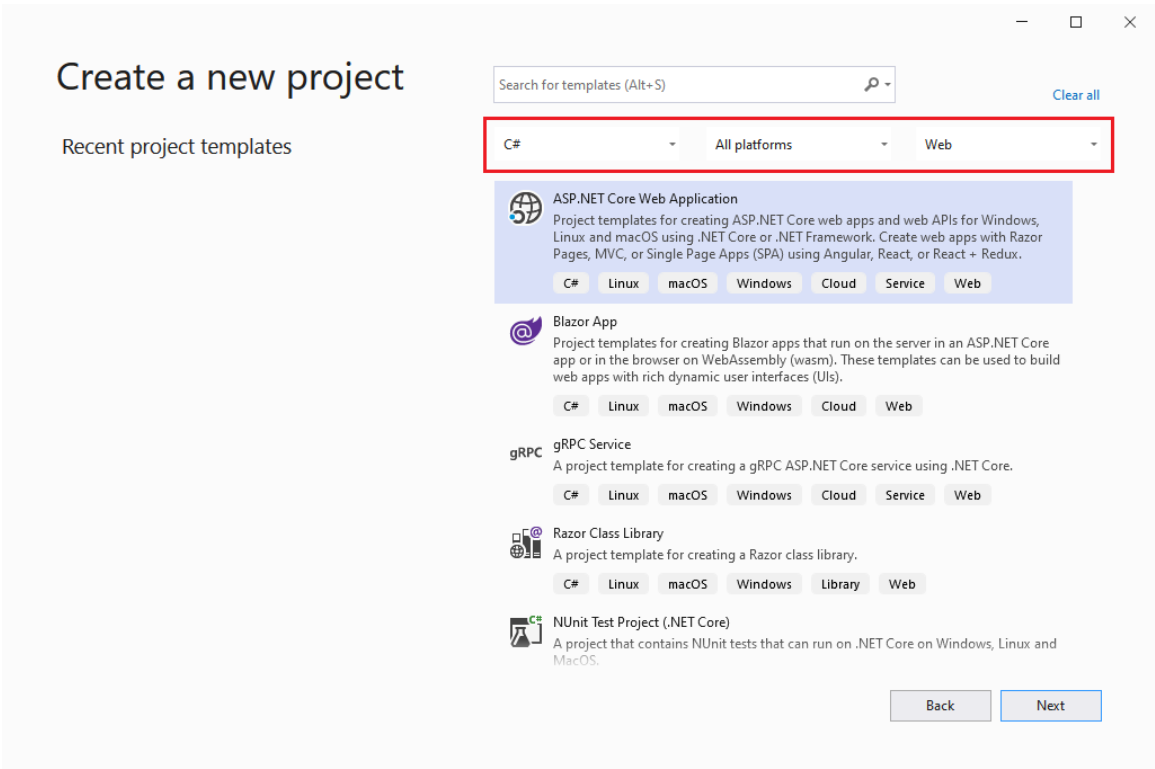
Trong khuôn khổ môn học này, chúng ta sẽ phát triển ứng dụng với ASP.NET Core bằng mô hình *MVC*, *Entity Framework* và tính năng *Code-First*[2].

Các bước để tạo 1 project ASP.NET Core MVC:

- Trong giao diện VS 2019, chọn *Create a new project*:



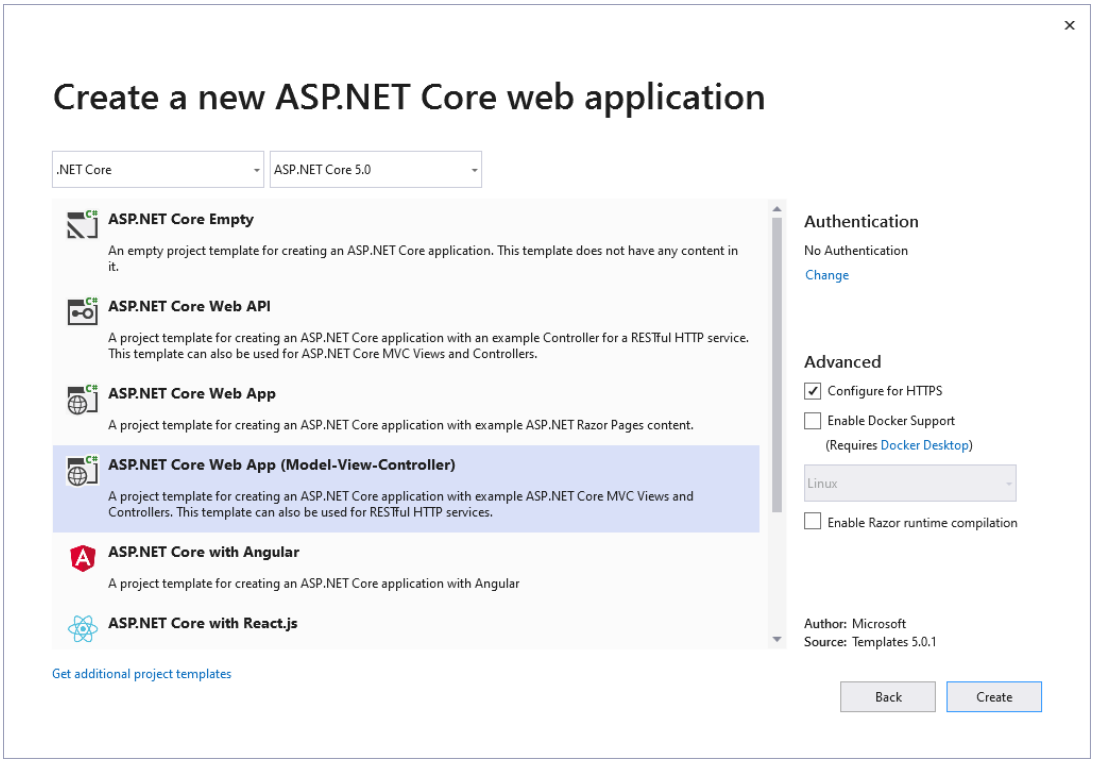
- Chọn loại project là *ASP.NET Core Web Application*[3], sau đó nhấn *Next*. Có thể tìm nhanh bằng cách điền vào khung tìm kiếm hoặc filter như sau:



[2] Entity Framework và Code-First sẽ được trình bày ở tài liệu buổi 3

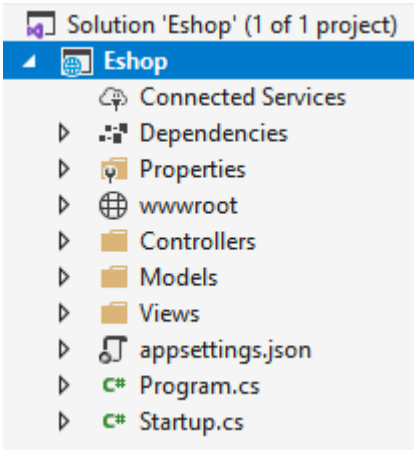
[3] Lưu ý: Tránh nhầm với project loại ASP.NET Web Application (.NET Framework)

– Đặt tên cho project và solution (ví dụ *Eshop*), chọn nơi lưu, sau đó nhấn nút *Create*.
– Chọn mô hình *ASP.NET Core Web App (Model-View-Controller)*, sau đó nhấn nút *Create*:



Lưu ý: Trong các tài liệu của môn học này, chúng ta minh họa trên ASP.NET Core 5.0.

Cấu trúc 1 project ASP.NET Core MVC như sau:



– Thư mục *wwwroot* được xem như thư mục gốc của website. Tất cả những thành phần tĩnh như file HTML, CSS, JavaScript, các tài nguyên (*asset*, *resource*) như hình ảnh, video, âm thanh… đều phải đặt trong thư mục này[4]. Project ASP.NET Core MVC đã có sẵn Bootstrap và jQuery nằm trong thư mục *wwwroot/lib*.
– Các thư mục *Controllers*, *Models*, *Views* để chứa các thành phần tương ứng của mô hình MVC (sẽ được giải thích ở phần IV và VII).
– File *appsettings.json* chứa các cài đặt, thiết lập cho project, ví dụ như các giá trị toàn cục, chuỗi kết nối… (tương tự file *Web.config* với ứng dụng web ASP.NET hoặc file *App.config* với ứng dụng WinForm). Cú pháp của file này như sau:

```
{
    "option1": "value1",
    "option2": "value2"
}
```

[4] Theo quy chuẩn của web, cần phải chia thư mục riêng cho các file này, ví dụ thư mục css, js, asset, images…

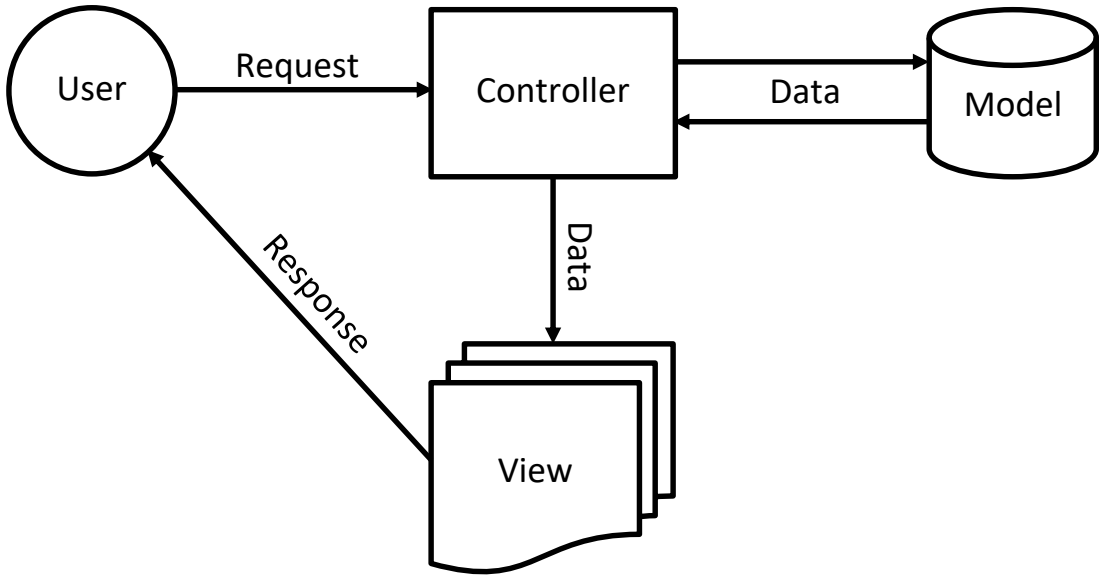
- File *Program.cs* là nơi chứa phương thức `Main()` để chạy ứng dụng.
- File *Startup.cs* là nơi cài đặt class `Startup` chứa các câu lệnh thiết lập cho ứng dụng. Trong đó có đoạn sau, sẽ được giải thích trong phần IV:

```
pattern: "{controller=Home}/{action=Index}/{id?}"
```

IV. Mô hình MVC và cơ chế Routing

1. Mô hình MVC trong ASP.NET Core

Mô hình MVC là 1 mô hình trong xây dựng phần mềm, ứng dụng, đặc biệt là với các ứng dụng web. Mô hình này chia phần mềm thành 3 lớp logic như sau:



- **M**odel: Lớp này chịu trách nhiệm quản lí dữ liệu: giao tiếp với CSDL, lưu trữ, truy vấn dữ liệu và là trung tâm của mô hình MVC.
- **V**iew: Lớp này chính là giao diện của ứng dụng. Sau khi xử lí, truy vấn, các dữ liệu sẽ được hiển thị lên View để người dùng có thể thấy được.
- **C**ontroller: Lớp này đóng vai trò trung gian giữa Model và View, quản lý và điều phối hoạt động của ứng dụng.

Cơ chế hoạt động của 1 ứng dụng web ASP.NET Core MVC như sau:

- Controller nhận *request* từ người dùng dưới dạng *route*.
- Controller xử lí *request* và chuyển yêu cầu dưới dạng *action* sang cho model thực hiện thao tác (xem, thêm, sửa, xóa) dữ liệu.
- Model thực hiện yêu cầu, sau đó trả kết quả về cho controller.
- Controller chuyển dữ liệu kết quả này sang view.
- View hiển thị giao diện kèm theo dữ liệu cho người dùng.

2. Cơ chế Routing

ASP.NET Core MVC sử dụng cơ chế *Routing*[5] để xử lí request từ người dùng.

Cơ chế routing có 2 loại: *Conventional Routing* và *Attribute Routing*. Trong khuôn khổ tài liệu buổi 1, chúng ta chỉ tìm hiểu loại đầu tiên.

Routing được định nghĩa trong mã nguồn startup của ứng dụng, mô tả việc đường dẫn web sẽ được tách ra thành *action* như thế nào. Phương thức `Configure()` của class `Startup` có đoạn sau:

```
pattern: "{controller=Home}/{action=Index}/{id?}"
```

[5] Xem thêm về cơ chế Routing: https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing

Đây chính là cấu trúc của 1 route mặc định của ASP.NET Core MVC.

Giả sử người dùng truy cập vào đường dẫn sau:

http://www.eshop.com/Products/Details/2

Trong đó: *http://www.eshop.com* chính là domain của ứng dụng hiện tại, và phần còn lại là route: */Products/Details/2*.

Đối chiếu với cấu trúc route phía trên, ta thấy:

- Route này sẽ dẫn tới controller *Products* và action *Details*, kèm theo id là 2.
- Nếu request không có action thì action mặc định là *Index*, không có controller thì controller mặc định là *Home*, tức là:
  - Route *~/Accounts* tương đương với *~/Account/Index*.
  - Route *~/* tương đương với *~/Home/Index*.
- Ký tự `?` đặt cạnh id cho biết id có thể không có.
- Controller *Products* sau khi nhận route này sẽ gọi phương thức `Details()` với tham số là 2, và trong phương thức đó sẽ có các câu lệnh yêu cầu model lấy thông tin sản phẩm có ID là 2.

Nhờ cơ chế Routing, đường dẫn trang web sẽ được tổ chức lại theo cách dễ đọc hơn, thân thiện hơn, thay vì phải sử dụng chuỗi GET như sau: *http://www.eshop.com/Products.aspx?id=2*.

V. ASP.NET Razor

Razor là 1 cú pháp lập trình ASP.NET được dùng để nhúng C# hoặc VB.NET trực tiếp vào trang web (HTML), tương tự như PHP với thẻ `<?php … ?>` hoặc ASP.NET WebForm với thẻ `<% … %>`.

Razor được phát triển vào tháng 06/2010, phát hành tháng 01/2011 và hiện tại đã trở thành một phần của ASP.NET Core.

Razor sử dụng kí hiệu `@` để bắt đầu 1 đoạn code C#. Cú pháp của Razor như sau:

```
@{
    // Các câu lệnh theo cú pháp C#
}
```

Với các biểu thức đơn hoặc giá trị biến, có thể lược bỏ cặp ngoặc `{ }` và dấu `;` cuối câu lệnh, ví dụ `@DateTime.Today` hoặc `@username` (với username là 1 biến C# đã khai báo).

Các trang web được viết theo cú pháp Razor có phần mở rộng là *.cshtml*, và sẽ được biên dịch thành HTML trước khi gửi xuống trình duyệt. Mặc định các trang của View trong ASP.NET Core đều được viết theo cú pháp Razor và do đó đều có phần mở rộng là *.cshtml*.

Ví dụ: Xem ví dụ *1.1* hoặc thực hiện các bước sau:

- Với project *Eshop* đã tạo ở phần III, mở file *Views/Home/Index.cshtml*. Mã nguồn file hiện tại như sau:

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="…">building Web apps …</a>.</p>
</div>
```

- Sau thẻ … ở trên, lần lượt bổ sung thêm các câu lệnh sau:
  + Khai báo 1 biến:

```
@{ string name = "Ricky"; }
```

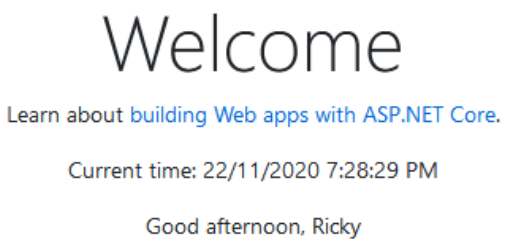
  + Xuất giá trị 1 biểu thức bằng thẻ HTML:

```
<p>Current time: @DateTime.Now</p>
```

  + Đoạn lệnh phức tạp:

```
@{
    if (DateTime.Now.Hour <= 12)
    {
        <p>Good morning, @name</p>
    }
    else
    {
        <p>Good afternoon, @name</p>
    }
}
```

- Kết quả sau khi biên dịch chương trình:



Lưu ý: Razor có thể tự nhận biết phần mã nguồn C# và HTML khi viết xen kẽ nhau. Phần mã nguồn C# sẽ được tô màu nền xám, còn phần mã nguồn HTML không có màu nền.

Lý thuyết về Razor sẽ được trình bày kỹ hơn trong tài liệu buổi 2.

VI. ViewBag và ViewData

ViewData và *ViewBag* được giới thiệu lần lượt ở MVC 1.0 và MVC 3.0, dùng để truyền dữ liệu từ controller sang view. Ở controller, chúng ta có thể gán giá trị cho ViewData/ViewBag, sau đó gọi lại giá trị này khi ở view.

Cả ViewData và ViewBag đều lưu dữ liệu theo kiểu *Dictionary*[6]. Tuy nhiên, chúng ta thao tác với ViewData thông qua từng cặp key-value, còn thao tác với ViewBag thông qua thuộc tính như 1 dynamic object.

Xét file *Index.cshtml* ở phần V (xem ví dụ *1.2*): File này là view do controller *Home* điều khiển. File *Home/HomeController.cs* cài đặt class `HomeController` có phương thức `Index()` như sau:

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

Phương thức `Index()` mặc định sẽ trả về view cùng tên thông qua câu lệnh `return View()`.

Giả sử trong phương thức `Index()` ta có 2 biến `temperature` và `humidity` lần lượt có giá trị là 28 và 89, và cần truyền giá trị 2 biến này sang view để hiển thị lên trang *Index*. Khi đó, chúng ta gán giá trị cho ViewData và ViewBag như sau:

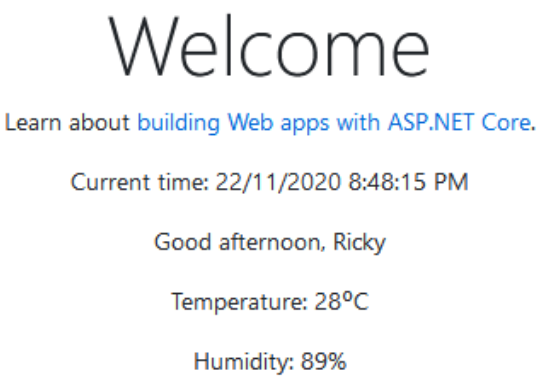
[6] Xem thêm về Dictionary của C#: https://www.tutorialsteacher.com/csharp/csharp-dictionary

```
public IActionResult Index()
{
      ViewData["temperature"] = 28;
      ViewBag.humidity = 89
      return View();
}
```

Quay lại file *Index.cshtml* ở phần V, chúng ta bổ sung các câu lệnh sau để gọi giá trị từ ViewData và ViewBag:

```
<p>Temperature: @ViewData["temperature"]<sup>o</sup>C</p>
<p>Humidity: @ViewBag.humidity%</p>
```

Kết quả sau khi biên dịch chương trình:



Do cả ViewData và ViewBag đều lưu dữ liệu chung vào 1 Dictionary, do đó có thể gán dữ liệu bằng ViewData nhưng lại lấy dữ liệu bằng ViewBag và ngược lại. Thay thế 2 câu lệnh ở trên bằng 2 câu lệnh sau vẫn sẽ cho kết quả như cũ:

```
<p>Temperature: @ViewBag.temperature<sup>o</sup>C</p>
<p>Humidity: @ViewData["humidity"]%</p>
```

So sánh ViewData và ViewBag:

	ViewData	**ViewBag**
Ra mắt ở phiên bản MVC	MVC 1.0	MVC 3.0
Yêu cầu tối thiểu của project	.NET Framework 3.5	.NET Framework 4.0
Kiểu	Dictionary	Dynamic Object
Cách truy cập	Thông qua key-value	Thông qua dynamic property
Tốc độ thực thi	Nhanh hơn	Chậm hơn
Cần phải chuyển đổi kiểu dữ liệu (Type Conversion)	Có, nếu dữ liệu lưu với cấu trúc phức tạp.	Không

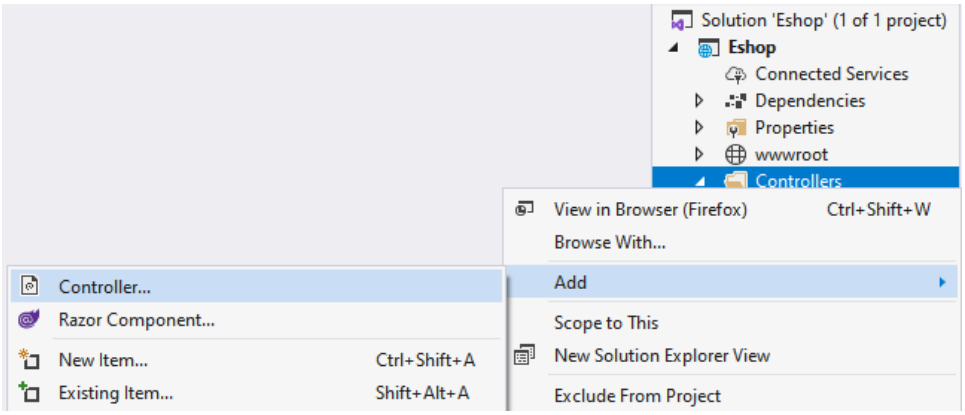
VII. Tạo 1 ứng dụng web MVC đơn giản

Thông thường, mô hình MVC dùng cho những ứng dụng có tương tác với CSDL. Tuy nhiên, trong khuôn khổ nội dung tài liệu buổi 1, chúng ta sẽ không tương tác với CSDL mà chỉ minh họa trên dữ liệu có sẵn. Việc tương tác với CSDL sẽ được trình bày trong tài liệu buổi 3.

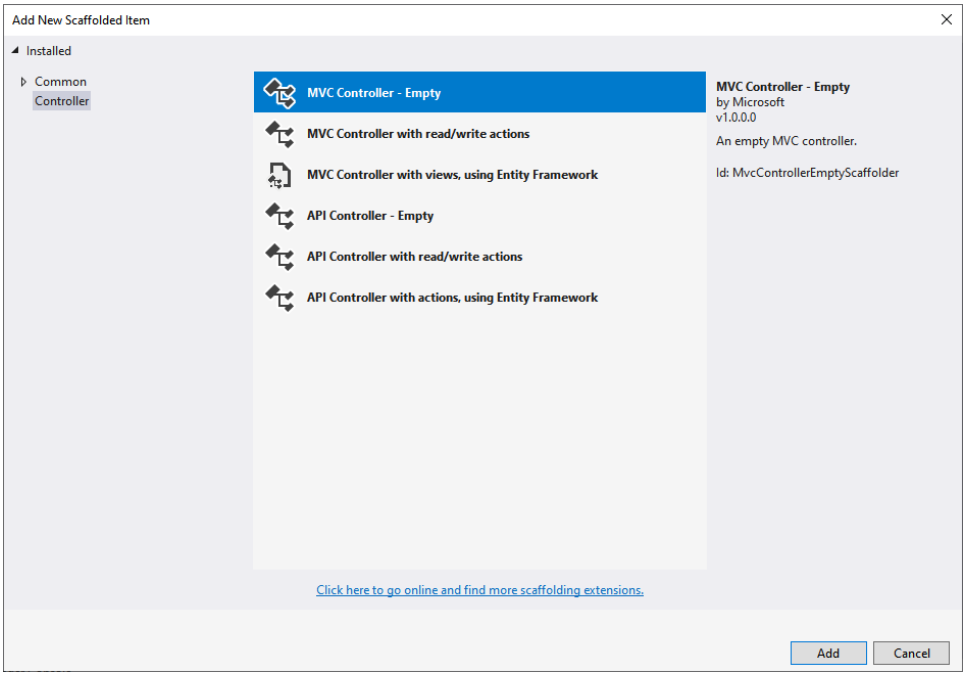
Trong ví dụ sau đây, chúng ta sẽ mô phỏng việc tạo 1 ứng dụng hiển thị danh sách các tài khoản có trong CSDL của 1 trang web thương mại điện tử Eshop (xem ví dụ *1.3*).

1. Tạo Controller

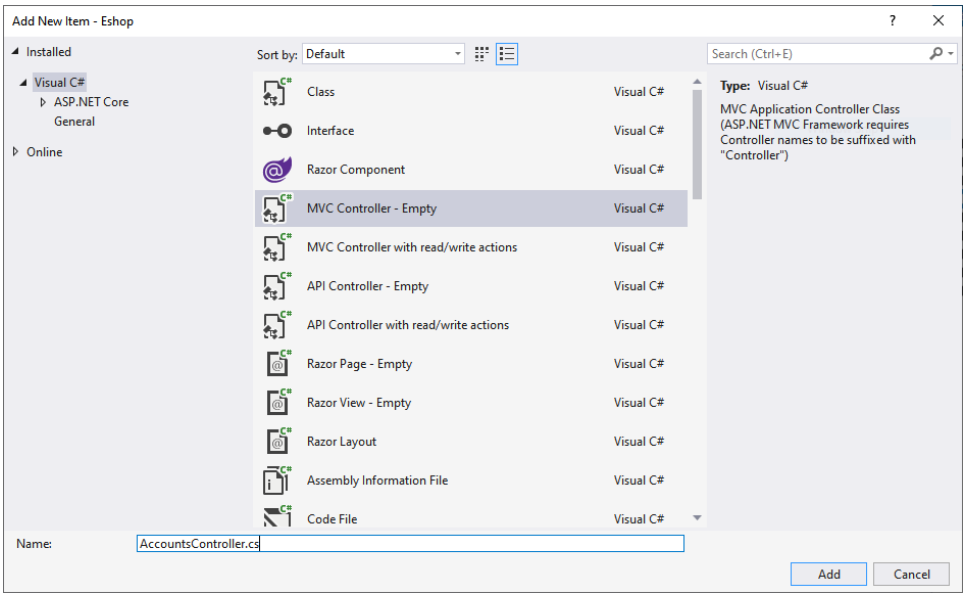
Để tạo 1 controller, click chuột phải vào thư mục *Controllers* và chọn *Add* → *Controller…*:



Trong cửa sổ *Add New Scaffolded Item*, chọn *MVC Controller – Empty*, sau đó nhấn nút *Add*:



Trong cửa sổ *Add New Item*, chọn *MVC Controller – Empty* và đặt tên cho class này là `AccountsController`, sau đó nhấn nút *Add*:



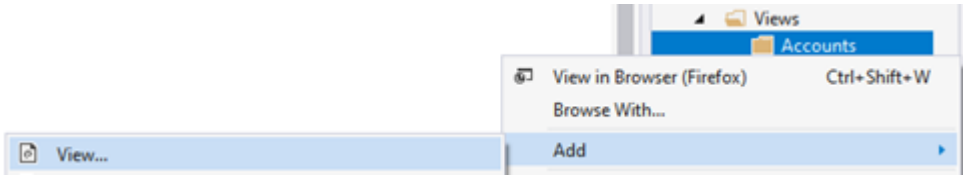
<u>Lưu ý:</u> Mặc định, tên class controller phải có chứa từ *"Controller"* ở cuối.

Class `AccountsController` được tạo ra với phương thức `Index()`.

2. Tạo Model

Trong mô hình MVC, model dùng để định nghĩa các class mô phỏng lại cấu trúc của các bảng trong CSDL. Các model này sẽ lưu trữ dữ liệu từ các bảng dưới dạng object.

Click chuột phải vào thư mục *Models* và chọn *Add* → *Class…*:



Đặt tên class là `Account` và nhấn nút *Add*.

Giả sử 1 tài khoản cần được lưu những thông tin sau:

- Tên đăng nhập.
- Mật khẩu.
- Email.
- Số điện thoại.
- Địa chỉ.
- Họ tên.
- Tài khoản có phải là admin hay không.
- Ảnh đại diện.
- Trạng thái (còn hoạt động hay đã bị khóa).

Do đó, class `Account` được cài đặt như sau:

```
public class Account
{
	public int Id { get; set; }
	public string Username { get; set; }
	public string Password { get; set; }
	public string Email { get; set; }
	public string Phone { get; set; }
	public string Address { get; set; }
	public string FullName { get; set; }
	public bool IsAdmin { get; set; }
	public string Avatar { get; set; }
	public bool Status { get; set; }
}
```

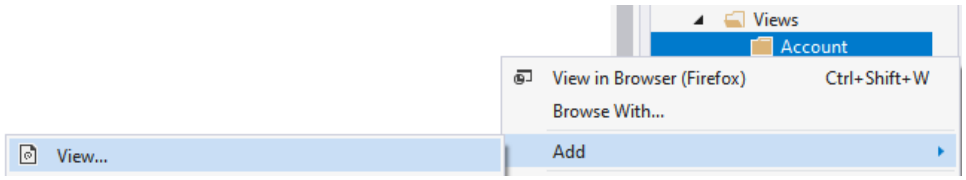
Lưu ý: Để đơn giản, chúng ta sẽ không thao tác với 3 thuộc tính `IsAdmin`, `Avatar` và `Status`.

3. Tạo View

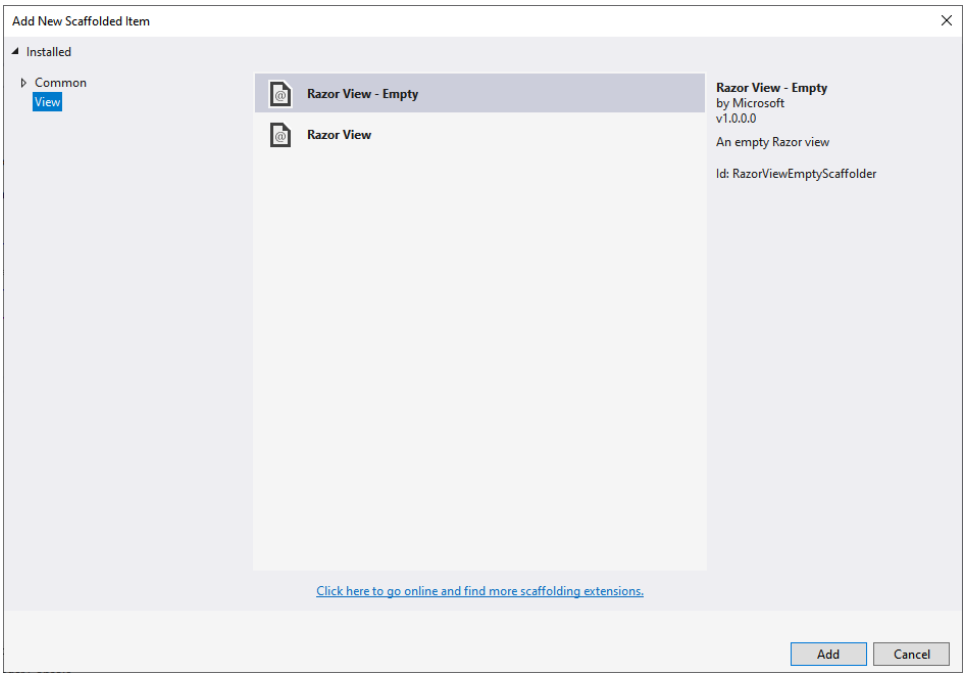
Mỗi controller có thể tương ứng với nhiều view khác nhau tùy thuộc vào action. Do đó, để dễ quản lý, chúng ta sẽ gom nhóm các view của cùng 1 controller vào chung 1 thư mục có tên trùng với tên của controller.

Click chuột phải vào thư mục *Views* và chọn *Add* → *New Folder*, đặt tên thư mục là *Accounts*.

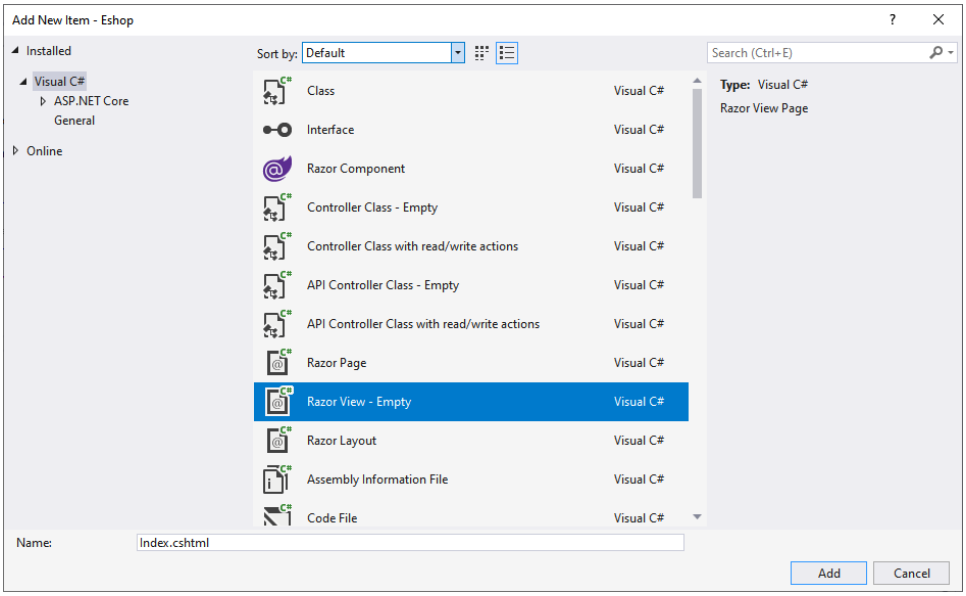
Click chuột phải vào thư mục *Accounts* vừa tạo và chọn *Add* → *View…*:



Trong cửa sổ *Add New Scaffolded Item*, chọn *Razor View – Empty*, sau đó nhấn nút *Add*:



Trong cửa sổ *Add New Item*, chọn *Razor View – Empty* và đặt tên cho view này là *Index.cshtml*, sau đó nhấn nút *Add*:



4. Cài đặt xử lí ở Controller

Trong 1 ứng dụng MVC, controller sẽ đóng vai trò điều phối, đưa dữ liệu từ model lên view để hiển thị cho người dùng.

Quay lại class `AccountsController`, bổ sung câu lệnh sau để có thể thao tác được với các class trong thư mục *Models*:

```
using Eshop.Models;
```

Chúng ta mô phỏng dữ liệu bằng 1 danh sách các tài khoản dưới dạng các object của class `Account`. Với 1 ứng dụng MVC thực sự, dữ liệu này sẽ được lấy từ CSDL.

Cài đặt phương thức `Index()` của `AccountsController` như sau:

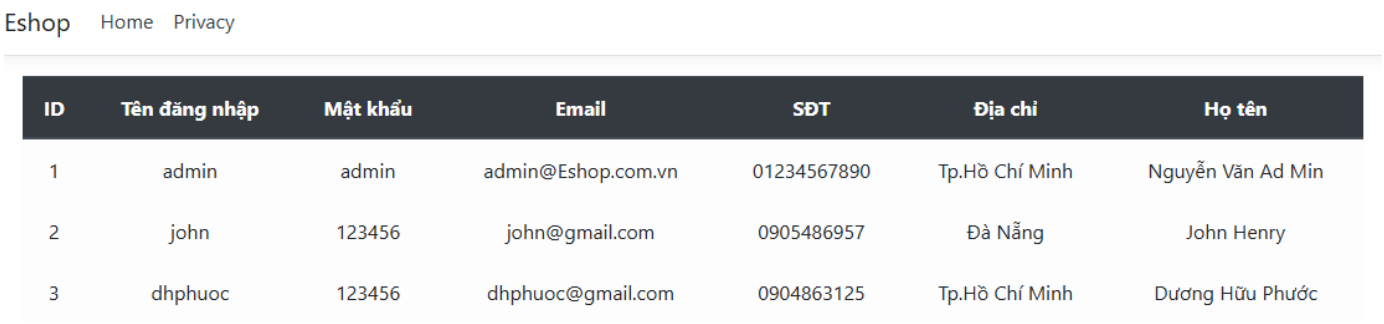
```
public IActionResult Index()
{
    List<Account> lstAccounts = new List<Account>();
    lstAccounts.Add(new Account
    {
        Id = 1,
```

```
                Username = "admin",
                Password = "admin",
                Email = "admin@Eshop.com.vn",
                Phone = "01234567890",
                Address = "Tp.Hồ Chí Minh",
                FullName = "Nguyễn Văn Ad Min",
                Avatar = "",
                IsAdmin = true,
                Status = true
            });
            // Có thể thêm một số tài khoản nữa vào lstAccount

            ViewBag.AccountList = lstAccounts;
            return View();
        }
```

Quay lại file *Index.cshtml* vừa tạo ở phần VII.3, chúng ta lấy danh sách tài khoản từ ViewBag và dùng vòng lặp để hiển thị thông tin từng tài khoản dưới dạng table:

```
@{
    List<Account> lstAccounts = ViewBag.AccountList;
}

<table class="table table-light table-hover text-center">
    <thead class="thead-dark">
        <tr>
            <th>ID</th>
            <th>Tên đăng nhập</th>
            <th>Mật khẩu</th>
            <th>Email</th>
            <th>SĐT</th>
            <th>Địa chỉ</th>
            <th>Họ tên</th>
        </tr>
    </thead>
    <tbody>
        @foreach (Account acc in lstAccounts)
        {
        <tr>
            <td>@acc.Id</td>
            <td>@acc.Username</td>
            <td>@acc.Password</td>
            <td>@acc.Email</td>
            <td>@acc.Phone</td>
            <td>@acc.Address</td>
            <td>@acc.FullName</td>
        </tr>
        }
    </tbody>
</table>
```

Kết quả sau khi biên dịch chương trình và chạy với route *~/Accounts* hoặc *~/Accounts/Index*:



ID	Tên đăng nhập	Mật khẩu	Email	SĐT	Địa chỉ	Họ tên
1	admin	admin	admin@Eshop.com.vn	01234567890	Tp.Hồ Chí Minh	Nguyễn Văn Ad Min
2	john	123456	john@gmail.com	0905486957	Đà Nẵng	John Henry
3	dhphuoc	123456	dhphuoc@gmail.com	0904863125	Tp.Hồ Chí Minh	Dương Hữu Phước

Giải thích:

- Khi người dùng truy cập đường dẫn *~/Accounts* hoặc *~/Accounts/Index*, request được gửi đến controller tương ứng là controller *Accounts* và action *Index*.
- Do đó, phương thức `Index()` của class `AccountsController` sẽ được gọi.
- Phương thức này sử dụng model *Account* để lấy dữ liệu là danh sách tài khoản, sau đó sử dụng ViewBag để truyền dữ liệu sang view. Cuối cùng hiển thị view bằng câu lệnh `return View()`.
- Ở view tương ứng (là file *Index.cshtml*), chúng ta lấy dữ liệu ra bằng ViewBag, sau đó dùng vòng lặp để hiển thị danh sách tài khoản này theo định dạng mong muốn.

Lưu ý:

- Mặc định, action sẽ trả về 1 view cùng tên. Trong ví dụ trên, action *Index* trả về view *Index.cshtml*. Nếu muốn action *A* trả về view *B.cshtml*, chúng ta dùng câu lệnh sau:

```
public IActionResult A()
{
    ……
    return View("B");
}
```

- Với dữ liệu có cấu trúc phức tạp (trong ví dụ là `List<Account>`), nếu muốn dùng ViewData để lấy dữ liệu thì cần phải chuyển đổi kiểu dữ liệu như sau:

```
List<Account> lstAccounts = ViewData["AccountList"] as List<Account>;
```

5. ViewModel

Để truyền dữ liệu từ controller sang view, ngoài cách dùng ViewBag và ViewData như ở trên, chúng ta có thể sử dụng *ViewModel*.

ViewModel cho phép khai báo view hiện tại sẽ sử dụng dữ liệu của model nào bằng cách khai báo câu lệnh sau vào đầu view:

```
@model <kiểu dữ liệu của ViewModel>
```

Ví dụ:

- Trong ví dụ trên, view Index hiển thị danh sách các tài khoản (ViewModel thuộc kiểu `List<Account>`), do đó chúng ta khai báo như sau:

```
@model IEnumerable<Eshop.Models.Account>
```

- Giả sử có view Details dùng để hiển thị thông tin chi tiết của 1 tài khoản (ViewModel thuộc kiểu `Account`), chúng ta sẽ khai báo như sau:

```
@model Eshop.Models.Account
```

Khi đã khai báo ViewModel, chúng ta có thể truyền dữ liệu từ controller sang view bằng cách truyền tham số cho phương thức `View()` thay vì phải dùng ViewBag và ViewData:

```
public IActionResult Index()
{
    // ……
    ViewBag.AccountList = lstAccounts;
    return View(lstAccounts);
}
```

Bên cạnh đó, khi thao tác với dữ liệu này ở view, chúng ta sẽ viết như sau:

```
@model IEnumerable<Eshop.Models.Account>
@{
    List<Account> lstAccounts = ViewBag.AccountList;
}

// ……
@foreach (Account acc in lstAccounts)
@foreach (Account acc in Model)
{
    // ……
}
```

Việc sử dụng ViewModel sẽ giúp code gọn gàng hơn, cũng như tận dụng được các tính năng như Tag Helper, Data Annotation, Data Validation[7]…

Lưu ý: ViewModel chỉ dùng để khai báo dữ liệu ở dạng model mà view dùng để hiển thị. Các dữ liệu khác muốn truyền từ controller sang view vẫn cần phải dùng ViewBag và ViewData.

[7] Tag Helper, Data Annotation, Data Validation sẽ được trình bày lần lượt ở tài liệu buổi 2, buổi 4 và buổi 9