**ASP.NET CORE**

1. **Khái Niệm**

* Đây là một bản thiết kế lại hoàn toàn của .NET Framework
* .net core là Open Source (Mã nguồn mở hoàn toàn) và dc phát triển chính là Microsoft và cộng đồng mã nguồn đặt trên GitHub
* Webforms,Windows Forms,WPF chúng không có trong .net Core tương có thể có
* Sao này vào năm 2020 sẽ có .Net 5 (là sự kế hợp của .net framework và .net core) và chỉ phát triển chính là .net core
* .net Standard là thư viện chuẩn khi ta Bulid phiên bản .net Standard thì cái DLL này chạy cho cả .net framework và .net core
  + Tiêu chuẩn .NET là một đặc tả chính thức của các API .NET nhằm mục đích có sẵn trên tất cả các triển khai .NET.
  + Nó xác định một tập hợp các quy tắc thống nhất cần được tuân theo trên tất cả các triển khai .NET
* Sự khác nhau giữa .net core và Famework:
  + .net frame là môi trường thực thi cũ hơn chỉ chạy dc trên windows , tr dây ta có .net Mono là phiên bản mini chạy dc trên linux mà k phổ biến
  + .core có thể chạy đa nền tảng
* Tính năng quan trọng của ASP.NET Core
  + Chạy trên mọi hệ điều hành như mac window, linux và là open source
  + Thống nhất giữa mvc va api
    - MVC là: xây dựng WEB APP có giao diện
    - API là tạo ra giao diện lập trình ứng dụng các api giúp tương tác vs DB và nó k tương tác vs người dung
  + ASP core nó có thể tự host dc
  + Cơ chế Denpendency Injection dc cung cấp sẵn dễ dàng tích hợp vs client-side framworks như Angular,Knockout etc.
  + Cơ chế request pipeline mới gọn nhẹ
  + Dung chung nutgetpaket

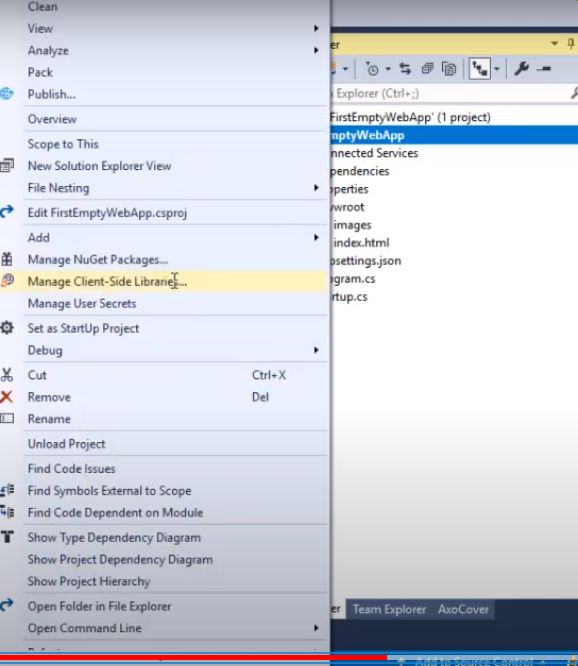
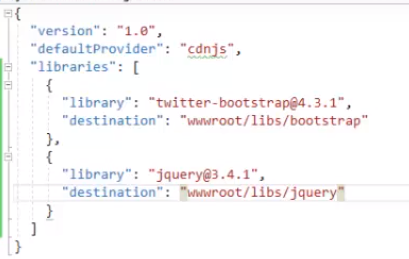
1. **Dotnet command line trong .NET**

Công cụ Donet command dung để khi ta làm việc trên mac hay linux mà ta k có sử dụng visual studio

1. Đầu tiên ta vào forlder nơi sẽ lưu trữ dự án sao đó ta gõ cmd và dung lệnh (dot net new web) nó sẽ lấy tên thư mục hiện tại đặt tên cho dự án và tạo ra những file cần thiết
2. **ASP.NET Core Application Structure (cấu trúc dự án ASP.net Core)**

* WWWroot
* Dependencies
* New project system (hệ thống dự án mới)
  + TargetFramework(mục tiêu framework)
  + Nuget Packages

khi tạo một project đầu tiên ta sẽ có như sao

* Connected services : dung để kết nối đến 1 web service bên ngoài vào (Azure Storage ,WCF)
* Properties: Chứa Các **launchSettings.json** (chứa các profiles thực thi ứng dụng vd như ta thay Ctrl F5 or F5 thì chạy các **CommanName**: như IISExpress or Project) và trong mỗi profiles trong **launchSetting** thì nó sẽ chạy 1 cái môi trường trong **appsettings.json** và ta có thể tạo them nhìu môi trường trong appsettings để chạy đa nền tảng , **launchSettings.json chỉ chạy trên local còn trên server sẽ chạy appsetting.json**
* Wwwroot : là nơi chứa css ,html , file tĩnh , ảnh ,….
* Dependencies
  + Chứa các nugget mạc định sẽ appnetcore app(phiên bản)
  + SDK: là gói môi trường liên quan
* Cơ chế tải Bootrap rồi jquery
  + Gọi là Libraner client side
  + Cách sử dụng
    - Ta chuột phải vào project và chọn Manage Client side Libraner 
    - Tiếp đến nó sẽ sinh ta 1 file libman.json ta chỉ cần cấu hình cho libraries và ctrl F là tự động tải về
* **appsettings.json :** là nơi chứa các file cấu hình chung dung để thiết lập đa môi trường
* **Program.cs và Statup.cs: là 2 file chính để chạy ứng dụng dc (sẽ tìm hiểu kỹ sau)**

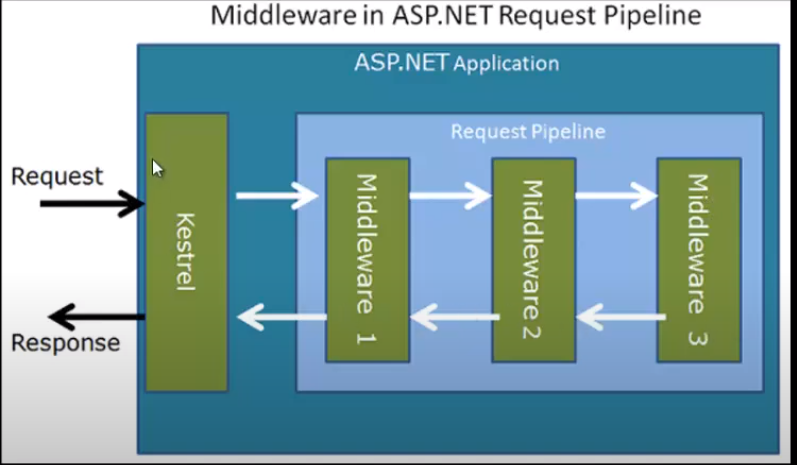
1. **Cơ Chế Application Startup**
2. Program.cs :tạo 1 web server với cái Main Method của nó
   * Tất các application (ứng dụng) trong .net Core điều là application (ứng dụng) Console
   * Application Console là gì?
     + Là ta sẽ chạy và cho kết quả trên màn hình đen (cửa số dòng lệnh)
     + Khác vs chạy console bt vs web tạo cái hosting để nó chạy web side còn lại khac j console bt
     + Điều chứa phương thức void Main
   * Trong Program ta có pt **BuildWebHost** là 1 pt tĩnh nó cấu hình build và trả về 1 đối tượng Host,
     + Phương thức **BuildWebHost** sử dụng class **WebHost**. Method **CreateDefaultBuilder**của class **WebHost** có trách nhiệm khởi tạo thể hiện **WebHostBuilder**với các cấu hình được yêu cầu.
     + **CreateDefaultBuilder dc thực hiện như sao**
       - Cấu hình Kestrel như là web server
       - Đặt thư mục gốc của ứng dụng sử dụng **Directory.GetCurrentDirectory()**
       - Load cấu hình từ
         * Appsettings.json
         * Appsettings.{Environment}.json
         * Load các cấu hình riêng (user secrets) khi chạy ứng dụng ở môi trường Development
         * Load các biến môi trường (Environment Variables)
         * Đọc các tham số truyền vào ứng dụng qua Command line argument
       - Bật logging
       - Tích hợp **Kestrel**chạy với IIS
3. Startup.cs : Configures(cấu hình) services cần thiết cho ứng dụng để nó có thể xử lý các request
   * Là một class đơn giản k kế thừa hay triển khai bất cứ class hay interface nào có 2 chức năng chính:
     + Cấu hình request pipeline để xử lý các request gửi đến ứng dụng
     + Cấu hình các services cho phần denedency injection
     + Trong startup.cs ta có 2 phương thức chính là :
       - ConfigureServices : cho phép ta thêm hoặc đăng ký các service vào ứng dụng
         * Phần khác của ứng dụng có thể cần đến các service này cho dependency injection(là 1 tính năng mới trong asp.core)
         * Vd như ta services.AddMvc(); services.AddSession();
       - Configure: cho phép ta cấu hình cho HTTP Request Pipeline chỉ ra cách mà ứng dụng cần phải hồi đáp các HTTP Request và các thành phần tạo nên request pipeline dc gọi là Middleware
4. **Kestrel Web Server**
5. **Khái niệm:**

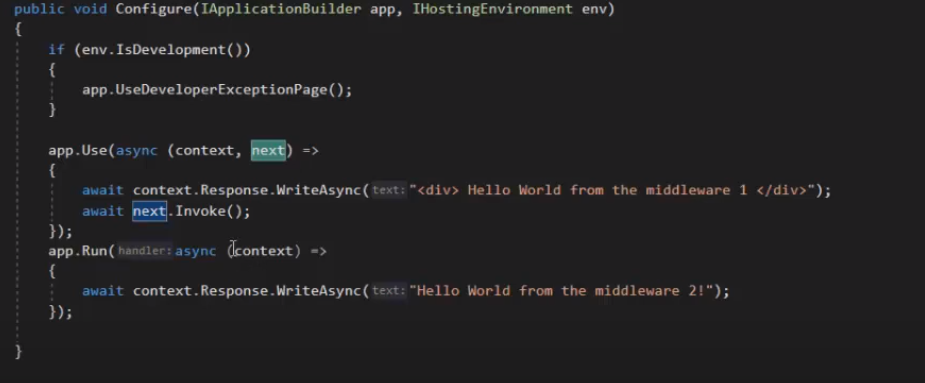
* Kestrel lá 1 web server là máy chủ web lắng nghe HTTP Server
* Lá web server open source , chạy đa nển tảng , hướng sự kiện và chạy bất đồng bộ
* Và nó dc phát triển là host chính cho asp core

1. **Tại sao dùng kestrel**

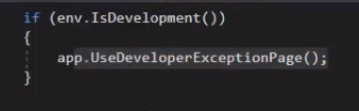
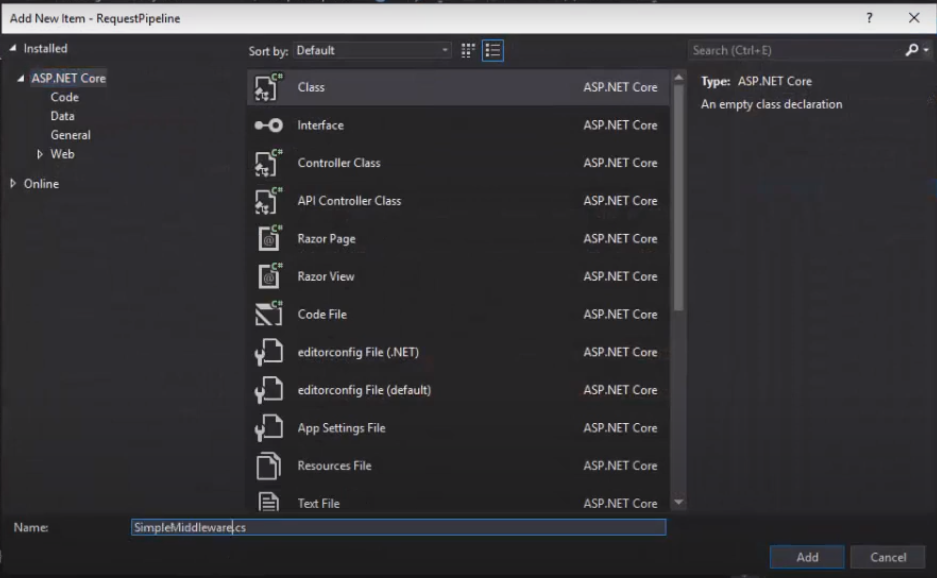
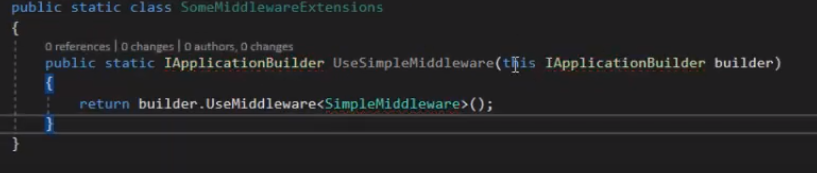
* Tr đây ta sử dụng .net cũ sử dụng IIS cũng là 1 web server đầy đủ tính năng và cũng vì thế mà IIS rất nặng và chậm , đặc biệt là IIS dc gắn trặc vs windows mà ta asp.net core đang hướng theo đa nền tảng
* Nên Ta cần 1 web server mới cho việc chạy đa nền tảng Từ Đó Kestrel xuất hiện
  + Kestrel có 2 cách để sử dụng
    - Self Hosting: tức là khi ta chạy chương trình ta chạy IIS press thì t có thể chọn cái host có tên cùng vs tên dự án
    - Behing another Web Server : cách 2 đó là t thông qua 1 máy chủ khác
      * Vd trên windows là IIS
      * **Trên linux Nginx , Apache**
* **Vì kerstrel**  là web server cơ bản nên thiếu rất nhiều tính năng vd như bảo mật, chức chỉ bảo mật HTTps,… vì thiếu khá nhìu tính năng cần thiết cho websever nên ta thường dùng **Cách thông qua 1 máy chủ để kết nối vs Kerstrel (Behing another Web Server)**
* Ngoài kerstrel ra asp core còn có hổ trợ 1 webserver khác là **HTTP.sys** giành cho window và ta có thể cầu hình trong PT CreateHostBuilder
  + webBuilder.UseStartup<Startup>().UseHttpSys();
  + <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/httpsys?view=aspnetcore-3.1>

# Middleware và Request Pipeline trong ASP.NET Core

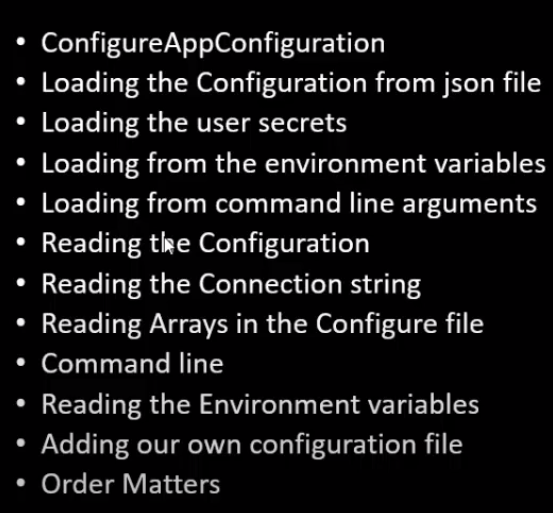
1. Middleware
   1. Là 1 thành phần ở giữa nó can thiệp vào request chạy qua request pipeline và can thiệp vs cả Response quay về nữa
   2. Mỗi middleware thao tác vs request và nhận kết quả từ các middleware trước vd ở tấm lọc nước 1 sẽ lọc ra than từ đó tấm lọc nước 2 sẽ nhận vào dòng nước k có than và tiếp tục làm nv của mình vả trả về tấm lọc nước kế tiếp khi chạy đến tấm lọc nước cuối cùng thì nó sẽ trả ngược về tấm lọc nước tr đó cứ như vậy mỗi request sẽ đi qua MiddleWare 2 lần
   3. MiddleWare cuối gọi là terminating the pipeline
2. Request Pipeline : là cơ chế giúp request nhận request cho đến xử lý và trả về response Request Pipeline chỉ ra cách dể ứng dụng làm việc vs các request gồm những bước nào
   1. do ta cấu hình các request pipeline từ request đến thông qua trình duyệt cho đến trã dữ liệu
   2. vd như đường ống nước ,khi ta nhận vào ở giữa sẽ có những tấm lọc nước (lọc cát , lọc các chất bẩn trong nước ) và mỗi thành phần trong tấm lọc nước sẽ ảnh hưởng đến dòng nước(nhiệm vụ) chạy qua và cho ra kết quả
3. Luồn đi khi nhận request đến khi reponse
   * 1. Nhận request s đó đi qua server Kesterl
     2. Tiếp đến ta đi vào Requset PiPeLine (trong Pipeline sẽ có những Middle Ware ) và sao khi xử lý xog ta sẽ đi ngược về Server Kestrel và trả Respone
     3. 
4. Thực hành
   * 1. Cách tạo và sử dụng app.use va app.run
        1. app.use



Trong app.use có parameter là next đó là cách t tiếp tục chạy middler ware 2 phía dưới trong app.use ta phải có dòng await next.Invoke(); thì nó mới chạy middler kế tiếp

* + 1. App.run giống như là midder ware cuối cùng vậy
    2. **Tạo Middleware bên ngoài :**nếu để ý thì tất cả app.Use đều là middleware isline(tức là những dòng code trong mỗi middleware điều viết cả vào trong File Startup.cs) điều này ảnh hưởng khi mỗi middleware có chục dòng code như vậy rất khó quản lý thay vì viết 1 nùi code trong Startup.cs thì ta tạo ra middleware bên ngoài như trong if từ đó ta chỉ cần gọi nó ra thôi
* Đầu tiên ta tạo forder đăt tên là Middlewares và từ đó khi ta tạo bất kỳ middleware nào cũng bỏ vào trong thư mục này
* Tiếp đến ta tạo file .cs đặt tên middleware mà mình muốn 
* Cách viết 1 middleware :
* Để làm 1 midd khai báo public cho contructor k khai báo static va ít nhất 1 tham số
* Class midd phải dinh nghĩa pt public là Invoke nhận 1 tham số HTTPcontext trả vế 1 Task
* Khi tạo xog ta dk cho nó
* Trên là cách 1 còn có 1 cách nữa là ta tạo Extensions
* Ta cũng tạo folder đặt tên là Extensions rồi s đó ta tạo 1 file .cs
* Extensions ta phải cho hàm đó là static 
* S đó t dk lại do là static method nên việc dk cũng khá dễ 

1. **FILE cấu hình trong ASP.net Core**

* **H**ệ thống cấu hình đã thay đổi rất nhiều trong ASP.net Code mới
* **Vs ASP.NET cũ ta sử dụng Web.config**
* **Còn ASP.NET CORE mới này cung cấp thẳng API để ta làm việc Cấu hình hổ trợ nhiều định dạng như Json,XML,INI, Command line ,Environment varibles (các biến môi trường cấu hình từ window) khuyên dùng để trong appsetting là 1 file Json**
* **Định dạng phổ biến là Json ,lưu trong json dùng Configuration để đọc json**
* **Loading the Configuration** 

1. **Static Files in ASP.NET CORE**

* **ASP.NET CORE** cho phép ta duyệt các file Tĩnh ,File tĩnh là những File nội dung không thay đổi vd HTML,CSS , JS.Images cho phép ta k cần di qua cả Request pepiline mà chỉ cần di qua Middle ware Static
* **Wwwroot(Web root:** là thư mục gốc của ứng dụng web là thư mục để file tĩnh **)**: là thư mục mặc định chứa các file tĩnh và khi ta bỏ các file tĩnh vào trong root tức là bạn đang public cho mọi người có thể truy cập dc
* **Vấn đề bảo mật:** nếu ta muốn bảo mật 1 file ta phải đi qua cơ chế Controller Action tức là trả về từ Action Controller

1. **MVC Design Pattern**
2. **Design Pattern là gì ?**

* **Design Pattern là :** là giải pháp tổng thể dc tối ưu hóa , dc tái sử dụng cho các vấn đề phổ biến trong thiết kế phần mềm má chúng ta thường gặp phải thằng ngày
* **Design Pattern Có Tác Dụng Gì ?**

Những lập trình viên có thể áp dụng giải pháp này để giải quyết các vấn đề tương tự. Các vấn đề bạn có thể sẽ nghĩ ra các giải quyết như nó chưa phải tối ưu nhất

Bạn cần phải hiểu rõ nó không phải là ngôn ngữ cụ thể nào cả. Design patterns có thể thực hiện được ở phần lớn các ngôn ngữ lập trình. Nó giúp bạn giải quyết vấn đề một cách tối ưu nhất, cung cấp cho bạn các giải pháp trong lập trình hướng đối tượng (OOP).

## **Tại sao phải sử dụng Design Pattern?**

Giúp sản phẩm của chúng ta linh hoạt, dễ dàng thay đổi và bảo trì hơn.

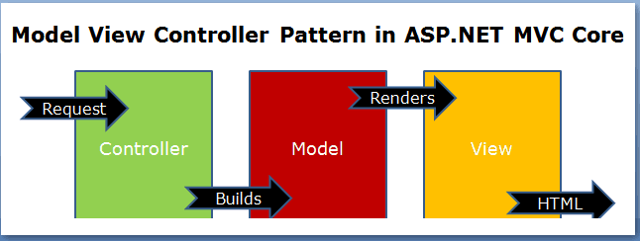
Có một điều luôn xảy ra trong phát triển phần mềm, đó là sự thay đổi về yêu cầu. Lúc này hệ thống phình to, các tính năng mới được thêm vào trong khi performance cần được tối ưu hơn.

Design pattern cung cấp những giải pháp đã được tối ưu hóa, đã được kiểm chứng để giải quyết các vấn đề trong software engineering. Các giải pháp ở dạng tổng quát, giúp tăng tốc độ phát triển phần mềm bằng cách đưa ra các mô hình test, mô hình phát triển đã qua kiểm nghiệm.

Những lúc khi bạn gặp bất kỳ khó khăn đối với những vấn đề đã được giải quyết rồi, design patterns là hướng đi giúp bạn giải quyết vấn đề thay vì tự tìm kiếm giải pháp tốn kém thời gian.

Giúp cho các lập trình viên có thể hiểu code của người khác một cách nhanh chóng (có thể hiểu là các mối quan hệ giữa các module chẳng hạn). Mọi thành viên trong team có thể dễ dàng trao đổi với nhau để cùng xây dựng dự án mà không tốn nhiều thời gian.

1. **MVC Design Pattern là gì?**

* **Khái Niệm:** Trong ASP.NET Core MVC dc áp dụng triệt để MVC viết tắt của Model View Controller , MVC là 1 partter chia để trị mỗi thành phần điều có nv riêng k ảnh hướng đến các phần khác (phụ thuộc vào nhau quá nhiều)
* **Model :** là chứa những dữ liệu cần có hiển thị cho khách hang đơn giản là những class c# bt gồm tt pt dùng để chuyên chỡ dữ liệu , nắm bắt dữ liệu
* **View :**thành phần giúp cho tạo ra giao diện cho người dùng và nhận những request từ người dùng rồi đẩy cho Controller xử lý vd hiển thị html , css,..
* **Controller:** là 1 thành phần chính nó đứng giữa view và Model controller sẽ nhận request thông qua view sao đó xử lý
* **Luồng đi của MVC như sao** 

1. **Basic Controller (controller có 3 nv chính )**

* Handle(xử lý) the request
* Build(xây dựng) a model
* Send(Gửi) the Response

**Action method** có những điều cần nhớ sao:

* Action method phải là một public method
* Action không thể là một Static method và một extension method
* Contructor getter or setter không dc sử dụng
* Inherited(kết thừa) method k dc sử dụng cho action method
* Action không thể chứa tham số ref và out
* Action không thể chứa attribute(NonAction)
* Action không dc overloaded

**Routing (cơ chế routing)**

* Là 1 phần rất quạng trong ASP.NET CORE
* Có 1 phần riêng về routing Hiểu nom na là nó sẽ quyết định action nào xử lý request đến

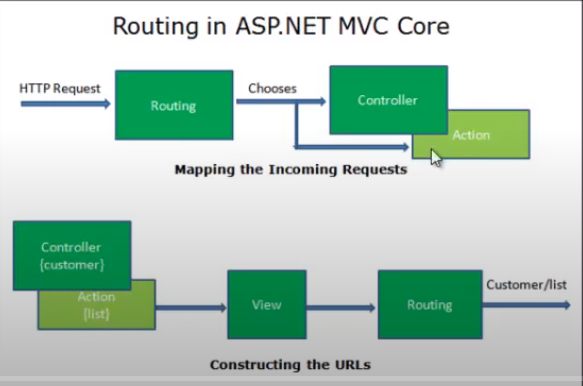
**Return Types (các kiểu trả về)**

* ViewResult trả về trang HTML
* EmptyResult trả vê Empty
* RedirectReult chuyển hướng đến 1 URL mới
* JsonResult trả về json Object
* javaScroptResult trả về Java script
* ContentResult trả về 1 bản text
* FileContentResult trả về 1 file có thể download dc (với dạng nhị phân)
* FilePathResult trả về 1 file có thể download dc (dạng đường dẫn)
* FileStreamResult trả về 1 file stream

1. **Routing mechanism(cơ chế)**

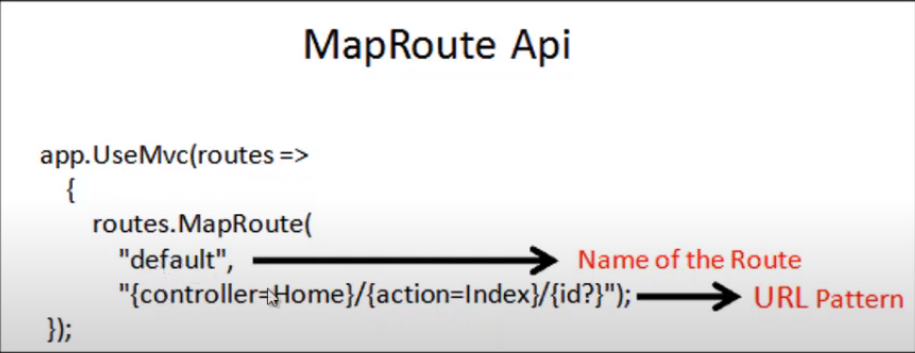
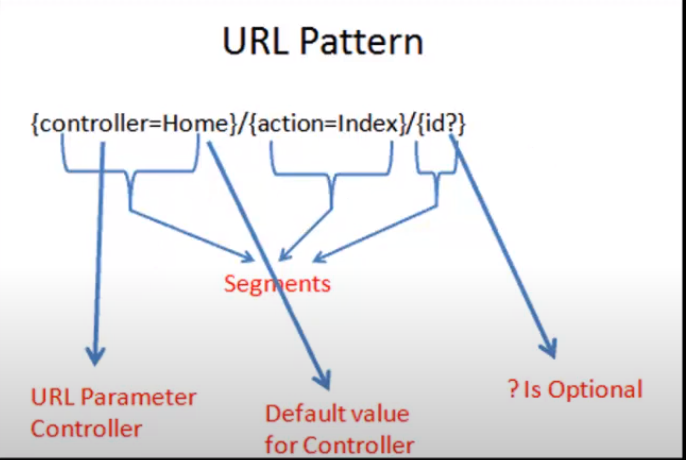
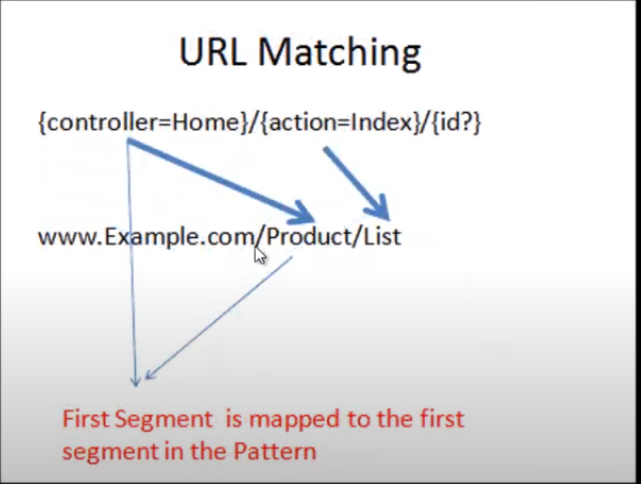
* **Một trong các thành phần quan trọng nhất của kiến trúc MVC là cơ chế routing (định tuyến). Nó là cơ chế quyết định xem Controller nào sẽ được xử lý request nào**
* **Routing là một quá trình khi ASP.NET Core xem xét các URL request gửi đến và "chỉ đường" cho nó đến Controller Actions. Nó cũng được sử dụng để tạo ra URL đầu ra. Quá trình này được đảm nhiệm bởi Routing Middleware. Routing Middleware có sẵn trong thư viện Microsoft.AspNetCore.Routing**
* Routing có 2 trách nhiệm chính:

Nv1: là maps cái yêu cầu cho Controller

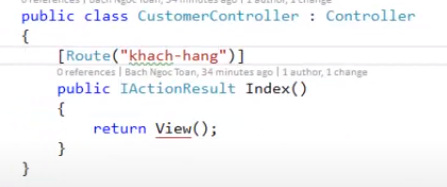
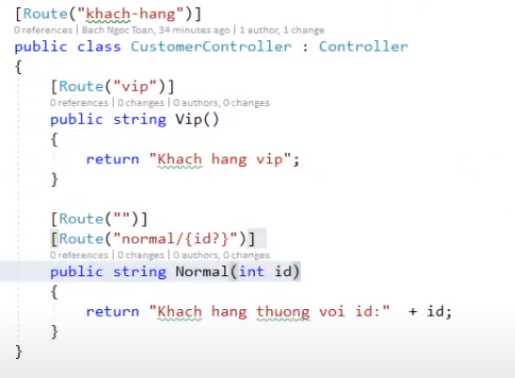
Nv2: Generate(tạo ra) một URL đầu ra correspond to controller action (tương ứng vs controller action) 

* **Routing làm việc ra sao** :
* Đầu tiên nó sẽ nhận HTTP request và Parse(phân tích) URLs
* Tiếp đến nó xem coi có cái matching(phù hợp) Route trong Routes Collection không
* Nếu có chạy đến RouteHandler
* Nếu không thì chạy tiếp Middleware kế ,trong trường hợp k có middleware nào phù hợp thì nó sẽ trả ra trang 404
* **Route là gì**
* Đơn giản là một roadmap(lộ trình) để đi đến đích
* ASP.NET CORE APP sử dụng Route đi đến Controller action
* The Each(Mỗi) Route contains(chứa đựng) một name , URL Pattern(Template)
* Mỗi Route bao gồm các thông tin như tên, mẫu URL (URL pattern) hay còn gọi là template url, thông tin controller action mặc định và ràng buộc (constraints). URL pattern được so sánh với URL đến xem có đúng mẫu không. Một ví dụ của URL pattern là: {controller=Home}/{action=Index}/{id?}
* Route được định nghĩa trong Microsoft.AspNetCore.Routing.

## **Route Collection là gì?**

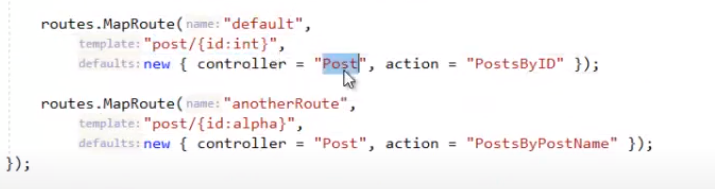
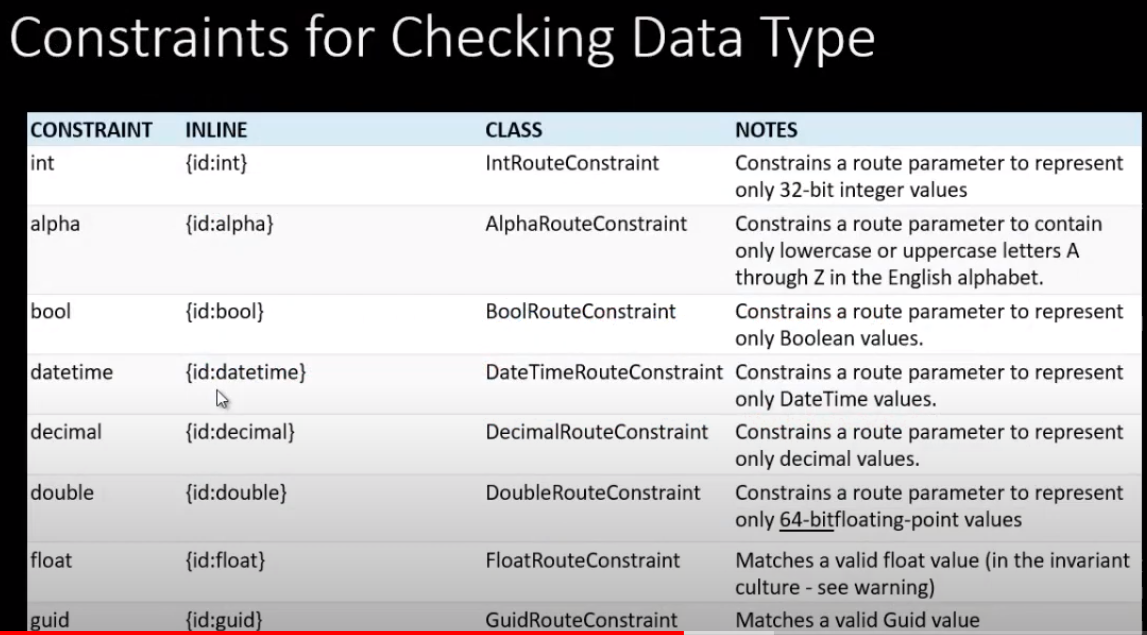
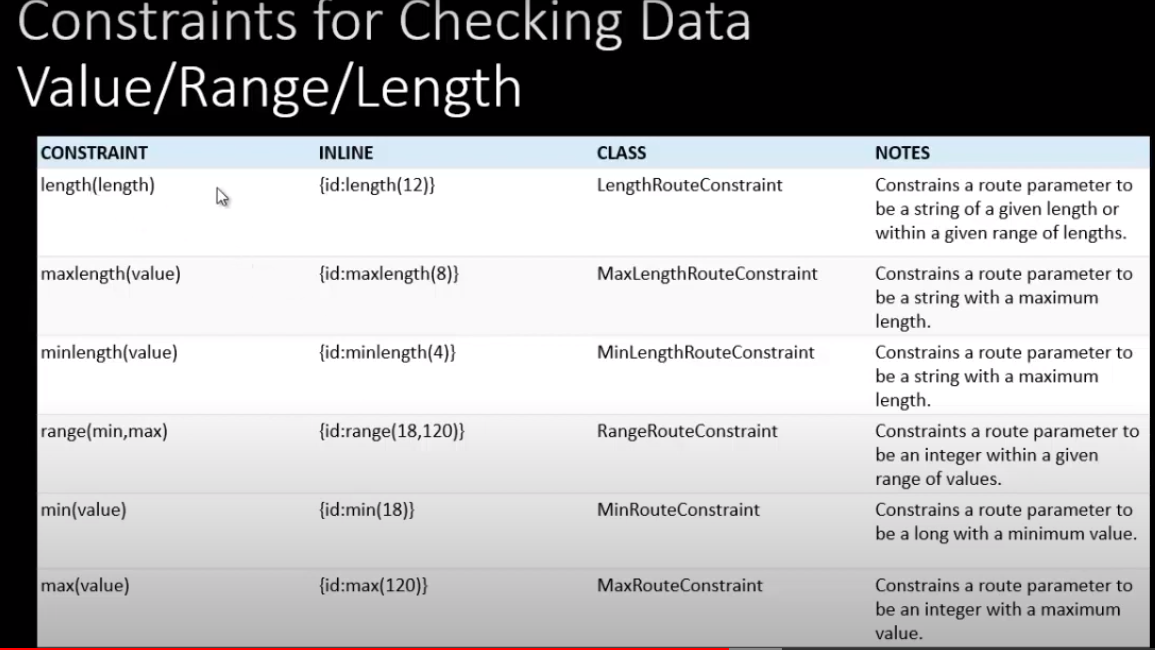
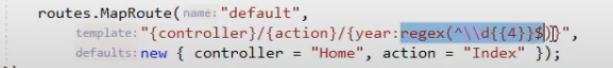
* Route Collection là một tập hợp tất cả ác Route trong ứng dụng. Một ứng dụng sẽ lưu một tập hợp các route ở một nơi duy nhất trong bộ nhớ. Các Route này sẽ thêm vào collection khi ứng dụng khởi động. Route Module sẽ tìm kiếm một Route match với URL request đến trong mỗi một Route của Route Collection. Route Collection được định nghĩa trong **Microsoft.AspNetCore.Routing.**
* **Route Handler là gì?**
* Route Handler là một thành phần quyết định sẽ làm gì với Route. Khi cơ chế routing tìm được một Route thích hợp cho một request đến, nó sẽ gọi đến **RouteHandler**và gửi route đó cho **RouteHandler**xử lý. Route Handler là class triển khai từ interface **IRouteHandler**. Trong ASP.NET Core thì Route được xử lý bởi **MvcRouteHandler**.
* **MVCRouteHandler**
* Đây là Route Handler mặc định của ASP.NET Core MVC Middleware. **MVCRouteHandler**được đăng ký khi đăng ký MVC Middleware. Bạn có thể ghi đè việc này bằng cách tự tạo cho mình một custom implementation của Route Handler.
* MVCRouteHandler được định nghĩa trong namespace: **Microsoft.AspnetCore.Mvc**
* MVCRouteHandler có trách nhiệm gọi Controller Factory, sau đó nó sẽ tạo ra một thể hiện của Controller được ghi trong Route. Controller sẽ được nhận và nó sẽ gọi một Action Memthod và tạo ra View. Vậy là hoàn thành request.
* **URL Pattern**
* Mỗi Route phải chứa một URL Pattern. Pattern này sẽ được so sánh với URL requét. Nếu pattern đúng với URL thì nó sẽ được sử dụng bởi hệ thống routing để xử lý URL đó. Mỗi một URL Pattern bao gồm một hoặc nhiều phần. Các phần chia tách bởi dấu gạch chéo.
* Mỗi phần có thể là một hằng số (constant) hoặc một Route Parameter.
* Route Parameter được bao gọc bởi một cặp dấu ngoặc nhọn ví dụ {controller}, {action}.
* Route Parameter có thể có giá trị mặc định như {controller=Home} khi Home là giá trị mặc định của controller. Một dấu = sẽ gán giá trị cho tên parameter.
* Bạn có thể có một thành phần dạng hằng số. Ví dụ: admin/{controller=Home}/{action=Index}/{id?}. Ở đây thì "admin" là một hằng tức là một chuỗi cố định phải tồn tại trên URL.
* Dấu ? trong {id?} chỉ ra là tham số này không bắt buộc. Một dấu ? sau tên tham số chỉ ra tham số đó không yêu cầu phải có giá trị.
* URL Pattern {controller=Home}/{action=Index}/{id?}. Đăng ký một route có thành phần đầu tiên trên URL là một controller, phần thứ 2 là Action method trong controller đó. Và phần cuối là dữ liệu thêm vào tên là id.
* URL Matching
* Mỗi phần trong URL request đến sẽ match tương ứng với thành phần của URL Pattern. Route {controller=Home}/{action=Index}/{id?} có 3 thành phần. Phần cuối là tùy chọn
* 
* 
* URL MATCHING

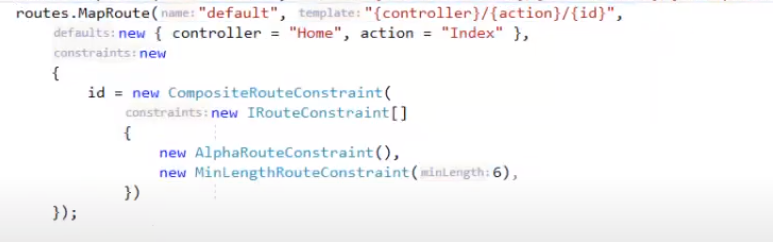
# Cơ chế Attribute Based Routing

* Là cơ chế định nghĩa routing trực tiếp trong Controller Action thay vì trong Startup.cs
* Các routing dc cấu hình trong strartup.cs sẽ không trực quan
* Vd: 
* Ta có thể thiết kế ra nhiều Route khác nhau
* Vd: ta thiết lập Routing chung cho class Customer thay vì ta ghi nhiều Routing khach-hang cho nhiều route bên dưới thì ta thiết lập chung 1 routing cho cả class lúc này khi ta chạy trang web mà k gõ route “Khach-hang” thì nó sẽ chẳng hiện route Normal mặc dù nó đang để route (“”)

1. **Route Constraints(Ràng buộc)**

Route Constrains giúp chúng ta lọc và giới hạn các giá trị không mong muốn truyền vào controller action. Nó được kiểm tra bởi các ràng buộc áp dụng cho giá trị truyền vào URL Parameter.

* Đầu tiên ta xem vd về rang buộc route  ở chỗ id:int? Ta có :int đây chính là cách ta ràng buộc route ở đây dc hiểu là khi ta truyền id vào index thì nó chỉ nhận là kiểu value là int còn nếu value là kiểu khác vd như string thì nó để giá trị mặc định đây ra cách ra tường minh hơn cách trên như khá dài dòng
* Vd thứ 2: ta sẽ thiết lập controller và sử dụng action method để test ta tạo 2 route có controller là Post Nhưng khác Action 1 cái sẽ có kiểu ràng buộc là int:int(nhận giá trị số) action là PostByID, còn cái kia sẽ nhận vào int:alpha(là chữ) action là PostByPostName tiếp đến ta tạo controller  khi ta chạy nếu id = số nó sẽ chạy vào hàm PostByID và ngược lại sẽ chạy PostByPostName và nhận tham số tương ứng
* 
* 
* 1 vd về rang buộc vd này sẽ làm cho bắt buộc giá trị id phải là số và phải là 4 số ,if id=1số,2 số điều k dc 
* Ta có thể kết hợp nhiều cách rang buộc 

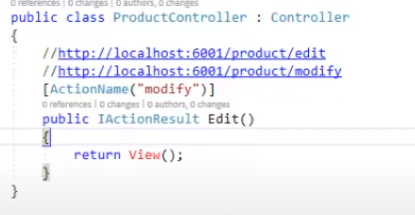
ở đây vừa là chữ vừa chỉ dc 6 ký tự (6 ký tự chỉ là chữ nó mới nhận) vd này chỉ là cách ta viết tường minh thôi chứ tác dụng k khác j vd trên

1. **Action Seletor(Bộ chọn) và Action Verbs(những động từ)**

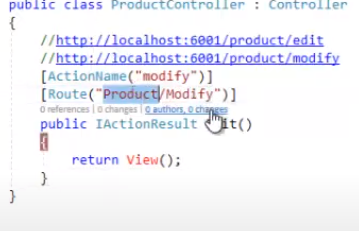
**Action Selector types**

* **Action Name** **:** tức là ta muốn thay đổi name của URL khác k muốn nó giống vs action method

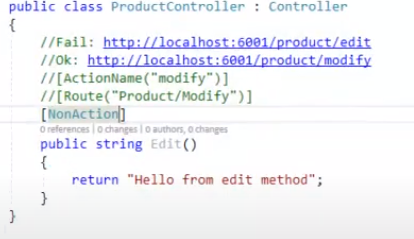
**Cách 1:** ta tạo một controller đặt tên là Product trong product có action method là Edit vậy đường dẫn mặc định là (localhost:5555/product/edit) lúc này ta k muốn để thành edit mà muốn để là modify thì ta sử dụng acctionName để làm việc này

 khi đó dg dẫn sẽ khác đi

**Cách 2: ta dùng route**



* **Non(không) Action:**là đánh dấu(một attiubiu đánh dấu) khi mà method k thỏa mản dk là public k muốn public ra ngoài(tức là tuy nó là public action như nó k dc chia sẽ bất kỳ đâu)

cho dù có gọi cũng fail

* **Action Verbs:** gồm các push ,post, get ,delete dùng trong API

1. **Action Results**
2. **Khái Niệm Action Result:**thì trong action method Client có thể nhận về rất nhìu dử liệu khác nhau có thể là 1 chuỗi json, hay 1 kiểu string, hoặc view html, có đôi khi là 1 file cho người dùng dowload về

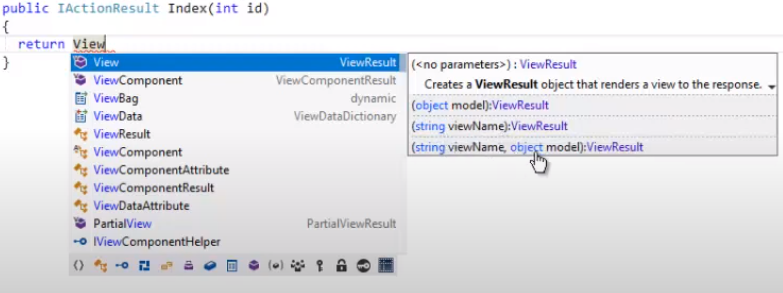
Controller trong ASP.NET Core chỉ đơn giản là các class C#. Nó không cần phải kế thừa từ bất cứ base class nào. Nhưng ASP.NET Core cung cấp một class Controller nó kế thừa từ một ControllerBase class. Điều này giúp class cung cấp rất nhiều các method hữu ích, nó giúp Controller làm việc dễ dàng hơn. Vì thế thông thường các controller của chúng ta đều kế thừa từ Controller class.

Controller Base class triển khai các loại Action Result khác nhau sẵn có giúp xây dựng các loại kết quả trả về khác nhau cho client. Ví dụ, ViewResult trả về một HTML response. Một RedirectResult chuyển hướng đến URL khác. Content Result trả về một chuỗi văn bản. Các kiểu trả về này được biết đến là Action Result.

1. **Phân biệt IActionResult và ActionResult**

* IActionResult là 1 interface và ta sử dụng Iaction hay actionresult trả về điều đc bởi vì acctionresult cũng kế thừa từ IactionResult
* Cho nên actionResult có nhiều con vd như: viewResult , JsonResult,FileResult,…
* Tại sao lại dùng nó: thì nếu ta k dùng ActionResult thì controller của chúng ta tự làm hết
* IActionResult là một interface nó định nghĩa một khuôn mẫu cho toàn bộ các Action Result của một action method. ActionResult là một abstract base class triển khai interface IActionResult. Action result như ViewResult, PartialViewResult hay JsonResult...đều kế thừa từ ActionResult base class.

1. **Action Result Sets(bộ)**

* **Producing(sản xuất) the HTML**
* **ViewResult** 
* **PartialViewResult(Xem 1 phần kết quả):** PartialView Result sử dụng model để tạo ra một phần của View. Chúng ta sử dụng ViewResult để tạo ra một view hoàn chỉnh còn PartialView trả về một phần của View. Kiểu trả về này hữu ích với Single Page Application (SPA) ki bạn muốn cập nhật một phần của View thông qua AJAX.
* **Redirecting(chuyển hướng) the users**
* Redriect result được dùng khi bạn muốn chuyển hướng người dùng đến một URL khác. Có 4 loại redirect result có sẵn. RedirectResult, LocalRedirectResult, RedirectToActionResult và RedirectToRouteResult.
* Mỗi một redirect này có thể trả về bất cứ mã trạng thái (status code) dưới đây:

[302 Found (Chuyển tạp thời)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302)

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302>

[301 Moved Permanently](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301)

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301>

[307 Temporary Redirect](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/307)

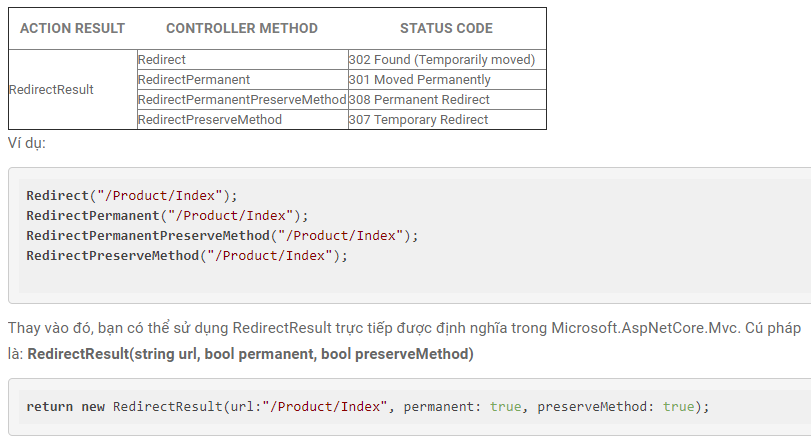
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/307>

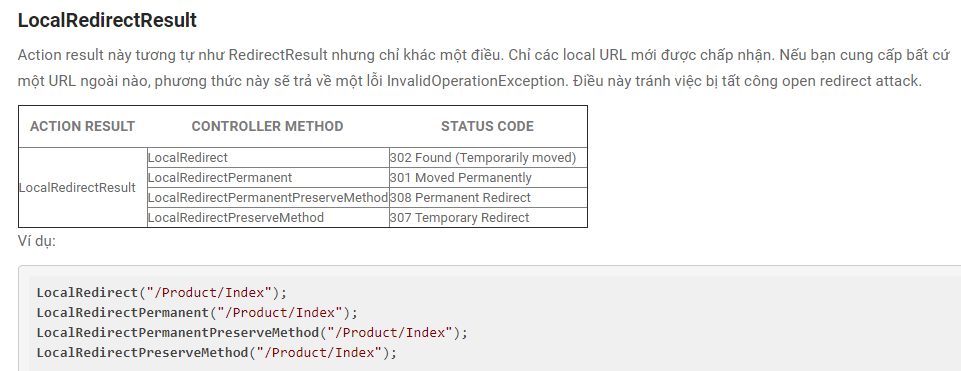
[308 Permanent Redirect](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/308)

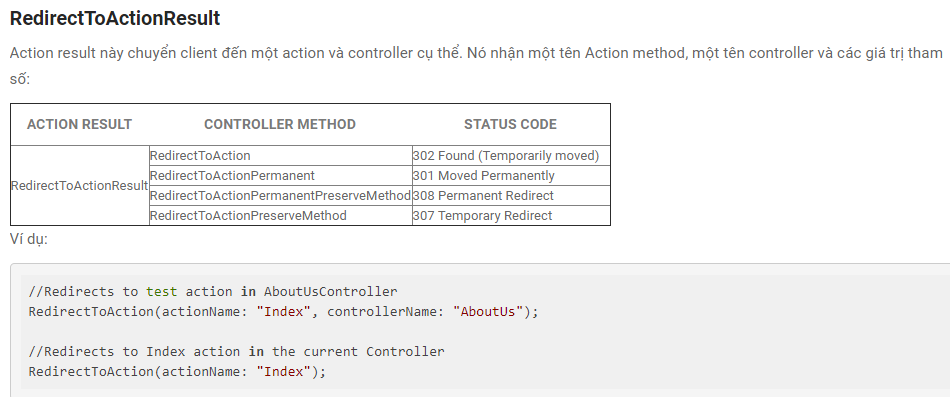
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/308>

### RedirectResult

### RedirectResult sẽ trả về cho user bằng cách cung cấp đường dẫn tuyệt đối hoặc tương đối:





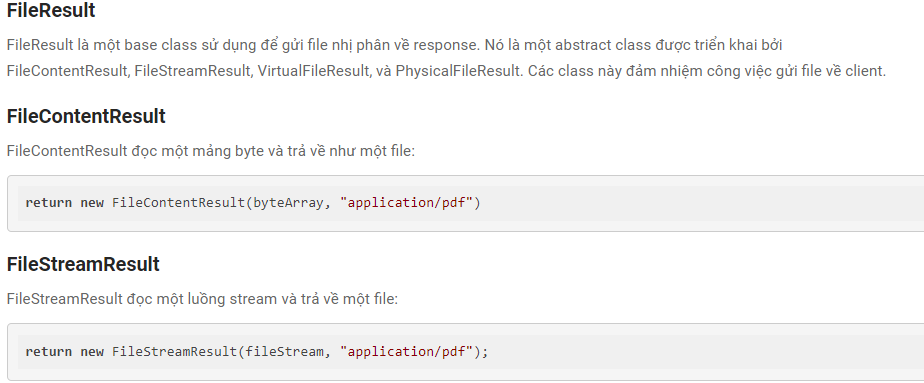




* **Returning(Trả về )Files**

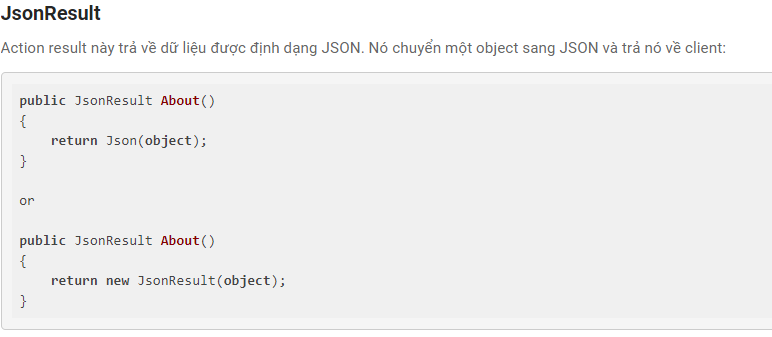
FileResult là một Action result sử dụng bởi Controller action để trả về file cho người dùng.

* **FildeResult**

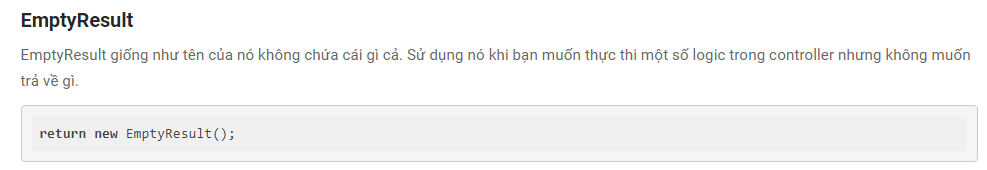




* **Content(nội dung) Result**







* **Returning Error and HTTP codes**

Loại Action result này được dùng trong Web API Controller. Kết quả sẽ được gửi về kèm HTTP Status Code. Một trong số chúng thì có thể gửi một đối tượng vào response.

### StatusCodeResult

StatusCodeResult gửi kết quả và chỉ ra một HTTP Status code:

**return** StatusCode(200);

hoặc

**return** **new** **StatusCodeResult**(200);

### ObjectResult

Action result này sẽ trả về một đối tượng kèm một HTTP Status Code là 200. Nó là một overload của method StatusCode

**return** StatusCode(200, **new** { message = "Hello" });

hoặc

**return** **new** **ObjectResult**(**new** { message = "Hello" });

### OkResult

Action result này trả về nguyên chỉ có HTTP Status code 200:

**return** Ok();

hoặc

**return** **new** **OkResult**();

### OkObjectResult

Action result này trả về một HTTP Status code 200:

**return** Ok(**new** {message="Hello"});

hoặc

**return** **new** **OkObjectResult**(**new** { message = "Not found" });

### CreatedResult

CreatedResult sử dụng khi một tài nguyên được tạo ra sau request Post. Nó gửi trạng thái 201 về kèm đối tượng vừa được tạo:

**return** Created(**new** Uri("/Home/Index", UriKind.Relative), **new** {message="Hello World"});

hoặc

**return** **new** CreatedResult(**new** Uri("/Home/Index", UriKind.Relative), **new** { message = "Hello World" });

### CreatedAtActionResult

Cái này tương tự CreatedResult nhưng nó nhận vào Controller và Action thay vì URL:

**return** CreatedAtAction("Index", **new** {message="Hello World"});

### CreateAtRouteResult

Action Result này nhận vào gái trị route và tương tự như CreatedResult và CreatedAtActionResult

CreatedAtRoute("default", **new** { mesage = "Hello World" });

### BadRequestResult

Action result này gửi về một HTTP Status code 400 cho client. Sử dụng response status code này khi chỉ ra cú pháp không đúng hoặc một request không được rõ ràng.

return BadRequest()

### BadRequestObjectResult

Action result này tương tự BadRequestResult. Khác nhau là bạn có thể gửi về một ModelStateDictionary (chứa chi tiết lỗi) và cũng là status 400:

**var** modelState = **new** ModelStateDictionary();

modelState.AddModelError("message", "errors found. Please rectify before continuing");

**return** BadRequest(modelState);

Phương thức BadRequest có một overload thứ 2, trả về một BadRequestObjectResult

### NotFoundResult

Action result này trả về lỗi HTTP 404 cho client:

return NotFound();

### NotFoundObjectResult

Action result này tương tự nuhw NotFoundResult nhưng trả về một đuối tượng kèm lỗi 404. Overload thứ 2 của NotFound giúp nhận một đối tượng làm tham số để trả về NotFoundObjectResult.

**return** NotFound( **new** { message = "Not found" });

### UnsupportedMediaTypeResult

Action result này gửi về lỗi HTTP 415. Sử dụng action result này khi request với định dạng không được hỗ trợ bởi server.

**return** **new** UnsupportedMediaTypeResult();

### NoContentResult

Action result này gửi lỗi HTTP 204 về. Sử dụng NoContentResult này khi request thành công nhưng không có nội dung nào được trả về

return NoContent();

* **Security(bảo vệ) Related(Liên Quan) Results**

### **SignInResult**

SignInResult là kết quả của hành động đăng nhập. SignInManager.SignInAsync hoặc PasswordSignInAsync trả về một SignInResult. Nó có 4 thuộc tính là Succeeded, IsLockedOut, IsNotAllowed và RequiresTwoFactor

Bạn có thể tham khảo về ASP.NET Identity Core.

### **SignOutResult**

SignOutResult là kết quả của hành động logout

### **ForbitResult**

ForbitResult trả về lỗi 403 tức là người dùng không được cấp quyền để thực hiện một hành động nào đó trên tài nguyên nào đó. ForbitResult không có nghĩa là người dùng chưa chứng thực. Người dùng chưa chứng thực nên trả về ChallengeResult hoặc UnAuthorisedResult.

**return** **new** ForbidResult();

hoặc

**return** **Forbid**();

Forbit là phương thức của Controller base class trả về thể hiện của ForbidResult. Thay vào đó bạn có thể trả về Status Code:

return StatusCode(403);

### **ChallengeResult**

ChallengeResult trả về khi chứng thực thất bại. Kết quả sẽ không gọi đến baatscuws middleware nào để tạo response. Ví dụ trả về lỗi 401 (Unauthorized) hoặc 403 (Forbidden) hoặc chuyển hướng người dùng đến trang đăng nhập.

### **UnauthorizedResult**

UnauthorizedResult trả về lỗi “401 – Unauthorized”. Controller sử dụng phương thức Unauthorized để trả về thể hiện của UnauthorizedResult.

**return** Unauthorized();

hoặc

**return** **new** **UnauthorizedResult**();

Khác nhau giữa UnauthorizedResult và ChallengeResult là một thằng trả về Status Code còn 1 thằng là không làm gì với nó.

1. **View**

## **View là gì?**

View có trách nhiệm tạo ra giao diện cho người dùng từ model. Controller trong ASP.NET Core sẽ nhận request sau đó thực thi với logic tương ứng với dữ liệu đầu vào từ request. Sau đó nó trả về Model cho View.

## **Trách nhiệm của View**

Render ra giao diện và hiển thị model lên là trách nhiệm của View. View không nên chứa bất cứ logic nào và không được xử lý logic. View có thể sử dụng bất cứ định dạng nào để trả về cho user. Định dạng có thể là HTML, JSON, XML hay là bất cứ định dạng nào khác.

## Ví dụ

Mở project sau đó tạo một ứng dụng ASP.NET Core. Project có chứa một Controller và một Index view.

Tham khảo: [Xây dựng ứng dụng ASP.NET Core MVC đầu tiên](https://tedu.com.vn/lap-trinh-aspnet-core/xay-dung-ung-dung-aspnet-core-mvc-dau-tien-225.html)

## **Làm sao để tạo View**

Project ví dụ ở trên đã có một Index Action method. Giờ chúng ta sẽ tạo một Action method khác:

**public** IActionResult **About**()

{

**return** View();

}

Các method trong Controller trả về một Action **Result**(hoặc một class kế thừa từ ActionResult). Nên tham khảo: [Action Result trong ASP.NET Core](https://tedu.com.vn/lap-trinh-aspnet-core/action-result-trong-aspnet-core-237.html)

ActionResult trả về HTML là một **ViewResult**. Phương thức View của Controller base class trả về một **ViewResult**. Chuột phải vào bất cứ đâu trong phương thức và chọn**Add View**

### https://tedu.com.vn/uploaded/images/072019/Adding-View-in-ASP.NET-MVC-Core.png

Sau đó sẽ thấy cửa sổ Add MVC View như sau:

### https://tedu.com.vn/uploaded/images/072019/Add-View-Dialog.png

### **Tên View**

Nhập tên View tại đây. Theo quy tắc, tên phải cùng tên với Action Method name. Bạn có thể ghi đè phương thức này bằng cách đặt tên khác đi. Nếu bạn làm điều này thì bạn sẽ cần phải chỉ ra tên của View trong tham số đầu tiên của phương thức View: View(“ViewName”)

#### **Template**

Template có vài tùy chọn như Create, Delete, Details, Edit, List, Empty (without Model). Các template ở trên cần một model, ngoại trừ Empty Template. Visual Studio tạo ra một view để create, edit, delete...trên model được cung cấp phụ thuộc vào template mà bạn chọn. Mỗi Template trên đều có giải thích phía dưới.

#### **Create**

Tạo một HTML Form để tạo mới một model. Nó tạo ra một label và một input cho mỗi thuộc tính trong model.

#### **Delete**

Tạo một HTML Form cho việc xóa model. Nó tạo ra một label và giá trị heienj tại cho mỗi thuộc tính của model.

#### **Details**

Tạo mới một view để hiển thị model. Nó tạo ra một label và giá trị của mỗi thuộc tính trong model.

#### **Edit**

Nó tạo ra một HTML form để sửa model. Nó tạo ra một label và một input cho mỗi thuộc tính của model.

#### **Empty**

Tạo ra một view trống

#### **List**

Tạo ra một HTML Table hiển thị danh sách model. Nó tạo ra một cột cho mỗi thuộc tính của Model. Bạn cần trả về một IEnumerable<Model> cho View. View cũng chứa danh sách các hành động để thực hiện như create/edit/delete.

### **Model class**

Dropdown hiển thị tất cả các Model class trong project. Tùy chọn này được loại bỏ nếu bạn chọn Empty project.

### **Tạo một Partial View**

Tùy chọn này là Create a Partial View. Partial View tạo một phần của view và không phải view hoàn chỉnh. PartialViewResult sử dụng model để tạo ra một phần của View. Kết quả của partial view nhìn giống hư một view bình thường ngoại trừ nó không có thẻ <html> hoặc không có thẻ <head> ở trên của View. Chọn tùy chọn này chỉ ra view cảu bạn sẽ tạo không phải view đầy đủ vì thế tùy chọn Layout sẽ bị bỏ đi.

### **Thư viện Scripts**

Chọn tùy chọn này sẽ thêm jquery.validate.min.js và jquery.validate.unobtrusive.min.json vào thưu viện JavaScript. Các thư viện này cần thiết cho triển khai client-side validation. CÁc thư viện này đòi hỏi khi tạo view chứa một đầu vào cho dữ liệu như một Form Edit view hoặc một form Create.

### **Sử dụng Layout Page**

Tùy chọn này cho phép bạn chọn một Layout page cho View. Layout page được dùng để chia sẻ các thành phần dùng chung trong trang của bạn và cung cấp một giao diện đồng nhất trong toàn bộ hệ thống.

Ví dụ chọn tên view là About, và template là Empty (without model) không chọn vào Create a partial view và Use a layout page. View được tạo sẽ nằm trong thưu mục Views/Home/About.cshtml. Mở file About.cshtml ra và thêm dòng <h1>About Us</h1> vào sau thẻ tiêu đề:

@{

Layout = null;

}

<!DOCTYPE html>

<**html**>

<**head**>

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

<**title**>About</**title**>

<**h1**>About Us</**h1**>

</**head**>

<**body**>

</**body**>

</**html**>

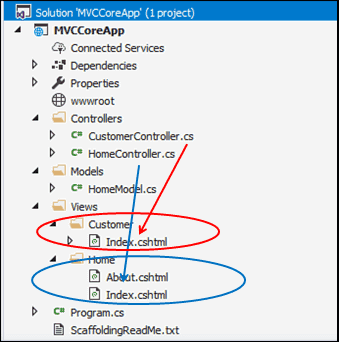
Chạy ứng dụng và bạn nhìn thấy dòng chữ "About us" trên ứng dụng.

## **Vị trí của View**

Theo quy tắc thì tất cả View được lưu trong thư mục Views của ứng dụng. Mỗi Controller sẽ có một thư mục con trong thư mục Views với tên trùng tên controller nhưng sẽ không có hậu tố Controller. Vì thế sẽ có một **HomeController**thư mục tên Home trong thư mục Views.

Mỗi Action method trong Controller sẽ lấy một file cho nó, tên trùng với tên của Action method. Vì thế phương thức Index của HomeController có một file với tên Index.cshtml trong Views/Home.

Dưới đây là hình minh họa:



Mỗi Action method trong **HomeController**sẽ có một file View tương ứng như **About.cshtml** và **Index.cshtml** và lưu trong thưc mục **Views/Home**. Tương tự như thế, action method của **CustomerController**sẽ có thư mục **Views/Customer**.

## **ASP.NET Core tìm View như thế nào?**

Project Example có một Controller tên là **HomeController**. Giờ hãy thêm **CustomerController**vào. Chọn thư mục **Controllers**chuột phải và add thêm Controller. Tên của nó là **CustomerController**.

**using** Microsoft.AspNetCore.Mvc;

**namespace** **MVCCoreApp.Controllers**

{

**public** **class** **CustomerController** : **Controller**

{

**public** IActionResult **Index**()

{

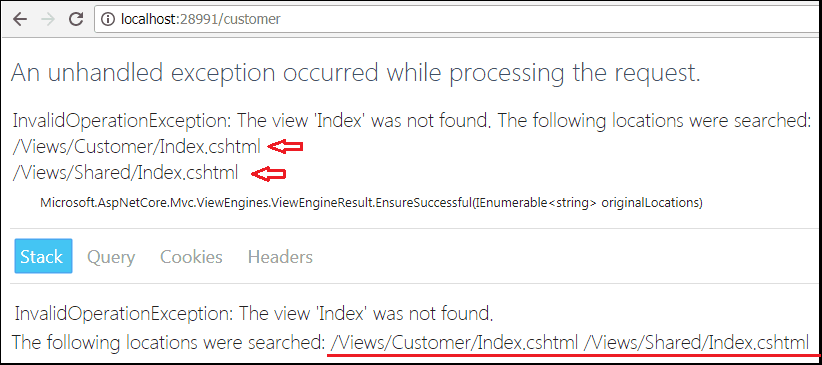
**return** View();

}

}

}

Giờ chúng ta không tạo View thay vào đó chạy luôn URL /Customer. Chúng ta nhìn lỗi như sau:



Màn hình ở trên hiển thị chính xác lỗi khi ASP.NET Core MVC middleware tìm kiếm View trong 2 thư mục:

* /Views/Customer/Index.cshtml
* /Views/Shared/Index.cshtml

Đầu tiên là thư mục Customer với tên của controller. Nếu không thấy nó sẽ tìm tiếp trong thư mục Shared trước khi đưa ra lỗi InvalidOperationException.

Thư mục Shared là thư mục đặc biệt chứa Views, Layouts hoặc Partial view dùng chung cho nhiều view.

Mở CustomerController và click bất cứ chỗ nào trong Index method. Chuột phải chọn Create an Empty View. File Index.cshtml sẽ tự động tạo ra trong thư mục Views/Customer. Giờ hãy mở Index.cshtml:

@{

Layout = null;

}

<!DOCTYPE html>

<**html**>

<**head**>

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

<**title**>Index</**title**>

<**h1**>Customer List</**h1**>

</**head**>

<**body**>

</**body**>

</**html**>

Giờ hãy chạy ứng dụng và chúng ta không thấy lỗi này nữa.

1. **Razor**

# Razor View Engine trong ASP.NET Core MVC

## Razor View Engine là gì?

Razor View Engine là View Engine mặc định của ASP.NET Core. Nó lấy mã Razor trong file View và chuyển sang HTML response.

## Razor Markup

Controller trong MVC gọi View bằng cách gán dữ liệu để tạo giao diện. View phải có khả năng xử lý dữ liệu và tạo response. Điều này được xử lý bằng cách dùng Razor, nó cho chúng ta sử dụng C# code trong file HTML. Razor View Engine xử lý các lệnh này và tạo ra HTML.

Các file chứa Razor có đuôi .cshtml. Cú pháp Razor thường ngắn hơn và đơn giản hơn cũng dễ học như C# hoặc VB. Visual Studio IntelliSense cũng hỗ trợ cú pháp Razor.

## Ví dụ

Để hiểu Razor làm việc ra sao? Hãy tạo Empty Project và thêm MVC Middleware vào. Bạn có thể tham khảo: [Xây dựng ứng dụng ASP.NET Core MVC đầu tiên](https://tedu.com.vn/lap-trinh-aspnet-core/xay-dung-ung-dung-aspnet-core-mvc-dau-tien-225.html). Bạn có thể [download mã nguồn](https://github.com/teduinternational/aspnetcore_samples) từ Github.

Xem Index method của HomeController:

**public** IActionResult **Index**()

{

**return** View();

}

Mở Index.cshtml và copy:

<**h1**>Razor Example</**h1**>

Chạy ứng dụng và bạn nhìn thấy ví dụ: Giờ bạn hãy mở thư mục Models ra và tạo một class Customer.cs

**namespace** **MVCCoreApp.Models**

{

**public** **class** **Customer**

{

**public** **string** name { **get**; **set**; }

**public** **string** address { **get**; **set**; }

}

}

## Cú pháp Razor

Razor sử dụng ký tự @ để chuyển HTML sang C#.  Nos cos 2 cacsh ddeer khai baso:

* Sử dụng Razor expression
* Sử dụng khối lệnh Razor

Các biểu thức này được xử lý bởi Razor View Engine và được gán vào response.

## Khối lệnh

Khối lệnh Razor bắt đầu bởi @ và nằm trong cặp ngoặc nhọn. Bạn có thể sử dụng khối lệnh bất cứ đâu trong file. Một khối Razor có thể sử dụng để thao tác Model, khai báo biến, đặt thuộc tính của View. Tuy nhiên nó không nên sử dụng cho việc xử lý logic.

Mở file Index.cshtml ra và copy đoạn này:

<h3>Code Block</h3>

@{

**var** greeting = "Welcome to our site!";

**var** weekDay = DateTime.Now.DayOfWeek;

}

@{

**var** cust = **new** MVCCoreApp.Models.Customer()

{

name = "Rahul Dravid",

address = "Bangalore"

};

}

Đầu tiên chúng ta tạo một khối lệnh Razor bắt đầu với @ và cặp {}. Trong cặp ngoặc này chúng ta có một đoạn C# thông thường với lệnh khai báo hai biến greeting và weekDay

## Biểu thức Razor

Biểu thức Razor bắt đầu với @ và theo sau là code C#. Biểu thức này có thể là ngầm định hoặc tường minh.

### Implicit Razor Expressions

Implicit Razor Expressions không cho phép dấu cách vì nó dùng để kết tuhcs biểu thức. Các biểu thức được xử lý bởi Razor View Engine và kết quả được thêm ngay vào vị trí nó đặt.

<**h3**>Code Expression</**h3**>

<**p**>@greeting</**p**>

<**p**>@DateTime.Now</**p**>

<**p**>Today is : @WeekDay thank you </**p**>

Đoạn Razor dưới đây hiển thị tên và địa chỉ trong thẻ <p>

<**p**>Name : @cust.name</**p**>

<**p**>Address : @cust.address</**p**>

### Explicit Razor Expressions

Explicit Razor Expressions bắt đầu bằng dấu @ và theo sau bởi cặp (). Bất cứ nội dung nào trong cặp ngoặc đơn này đều được xử lý bởi Razor và tạo ra output.

@{ var ISBNNo = "10001200"; }

<**p**>ISBN : @ISBNNo</**p**>

@\*//<**p**>ISBN@ISBNNo</**p**> //Does not work\*@

<**p**>ISBN@(ISBNNo)</**p**>

## Hỗ trợ nhận diện cú pháp

Hình dưới đây hiển thị cách Razor làm việc được sử dụng trong HTML và Intellisense hỗ trợ từ Visual Studio:

### https://tedu.com.vn/uploaded/images/072019/Intellisense-Support-for-Razor-in-Visual-Studio.png

### Sử dụng Directive

Khai báo @using làm việc tương tự như C# vậy. Nó cho phép chúng ta import namespace. Ứng dụng có thể import namespace MVCCoreApp.Models theo cách sau:

@**using** MVCCoreApp.Models;

Và bạn đơn giản là dùng nó var cust=new Customer() thay vì var cust = new MVCCoreApp.Models.Customer()

### Khai báo biến

Các biến được khai báo sử dụng từ khóa var hoặc sử dụng kiểu dữ liệu C#. CÁc kiểu như int, float, decimal, bool, DateTime hay string có thể được dùng để lưu trữ kiểu dữ liệu tương ứng. Để khai báo biến bạn vẫn có thể dùng dấu @ trực tiếp:

<**h3**>Variables </**h3**>

<!-- Storing a string -->

@{ var message = "Welcome to our website"; }

<!-- Storing a date -->

@{ DateTime date = DateTime.Now; }

<**p**>@message</**p**>

<**p**> The current date is @date</**p**>

Các chuỗi được bao trong một cặp nháy kép. Để sử dụng nháy kép trong chuỗi, bạn cần phải nhân đôi dấu nháy.

@{ var helloWorld = @"Hello ""World"""; }

<p>@helloWorld</p>

Tương tự như vậy thì dấu gạch chéo ngược cũng được làm như thế:

@{ **var** Path = @"C:\Windows\"; }<p>

The path **is**: @Path</p>

Bạn có thể in ký tự @ trong HTML bằng cách lặp 2 ký tự:

@{ var symbol = "You can print @ in html"; }

<p>The @@symbol is: @symbol</p>

Dấu @ trong email thì bạn có thể chỉ đích danh nó và không cần phải double:

<**a** href="mailto:admin@tektutorialsHub.com">admin@tektutorialsHub.com</**a**>

### Comment

Sử dụng @\*\*@ để đặt comment:

@\*This **is** comment\*@

### HTML trong khối lệnh

Bất cứ HTML Element nào trong khối lệnh Razor được nhận dạng như bình thường:

@{

<**p**>Hello from the Code block</**p**>

}

### Dòng lệnh đơn

Bạn có thể xuất ra giá trị mà không cần thẻ HTML bằng dấu @:

@{

@:Hello from the Code block

}

### Đa dòng lệnh

Đối với nhiều dòng bạn cần phỉa có thẻ <text></text>

@{

<**text**>Hello from the multiline text </**text**>

}

### Lệnh điều kiện

Razor cho phép xử lý lệnh điều kiện như if hoặc switch.

#### If else

Điều kiện if được dùng để tạo ra một nội dung dựa trên điều kiện đó:

@{**int** **value** = 200;}

@**if** (**value** > 100)

{

<p>Value **is** greater than 100.</p>

}

**else**

{

<p>Value **is** less than 100.</p>

}

Hoặc đoạn code:

@{

**var** **value** = 200;

**if** (**value** > 100)

{

<p>The **value** **is** greater than 100 </p>

}

**else**

{

<p>This **value** **is** less than 100.</p>

}

}

Và lệnh if else if:

@**if** (SomeCondition)

{

}

**else** **if**(SomeOtherCondition)

{

}

**else**

{

}

#### Switch

Một lệnh switch có thể thêm nội dung vào HTML trên một số điều kiện:

@**switch** (value)

{

**case** 0:

@: value is Zero

**break**;

**case** 100:

<**p**>Value is 100 </**p**>

**break**;

**case** 200:

<**p**>Value is @value </**p**>

**break**;

**case** 300:

<**text**>Value is 300</**text**>

**break**;

**default**:

<**p**>Invalid Value </**p**>

**break**;

}

### Vòng lặp

#### Lặp foreach

Vòng lặp được sử dụng để lặp khối lệnh

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

{

<span> @i </span>

}

Trường hợp tốt nhất là dùng lặp qua các tập hợp đối tượng và hiển thị danh sách  trong bảng:

@{

var custList = new List<**Customer**>()

{

new Customer() { name = "Rahul", address = "Bangalore" },

new Customer() { name = "Sachin", address = "Mumbai" }

};

}

<**table**>

<**thead**>

<**tr**><**td**>Name</**td**><**td**>Address</**td**></**tr**>

</**thead**>

@foreach (Customer custvar in custList)

{

<**tr**>

<**td**>@custvar.name</**td**>

<**td**>@custvar.address</**td**>

</**tr**>

}

</**table**>

#### Vòng lặp While

<h3>While **loop**</h3>

@{

var r = 0;

**while** (r < 5)

{

r += 1;

<span> @r</span>

}

}

@{ var s = 0; }

@while (s < 5)

{

s += 1;

<span> @s</span>

}

}

### Mã hóa HTML

Tất cả biểu thức Razor được tự động mã hóa HTML

Ví dụ:

@{

string encodedMessage = "<**script**>alert('You are under cross-site script injection attack');</**script**>";

}

<**span**>@encodedMessage </**span**>

Đoạn code trên sẽ không có kết quả ra một alert mà thay vào đó bạn sẽ thấy dòng chữ: “<script>alert(‘You are under cross-site script injection attack’);</script>” trên trình duyệt.

Razor sẽ tự động mã hóa < thành &lt; và > thành &gt;

Nếu bạn nhìn thấy HTML Response sẽ như sau:

<span>&**lt**;script&**gt**;alert(&#x27; You are under cross-site script injection attack&#x27;) ;&lt;/script&gt; </span>

Nếu bạn muốn script thực thi, bạn bạn cần sử dụng @Html.Raw để int chuỗi không mã hóa ra.

<**span**>@**Html**.**Raw**(**encodedMessage**)</**span**>