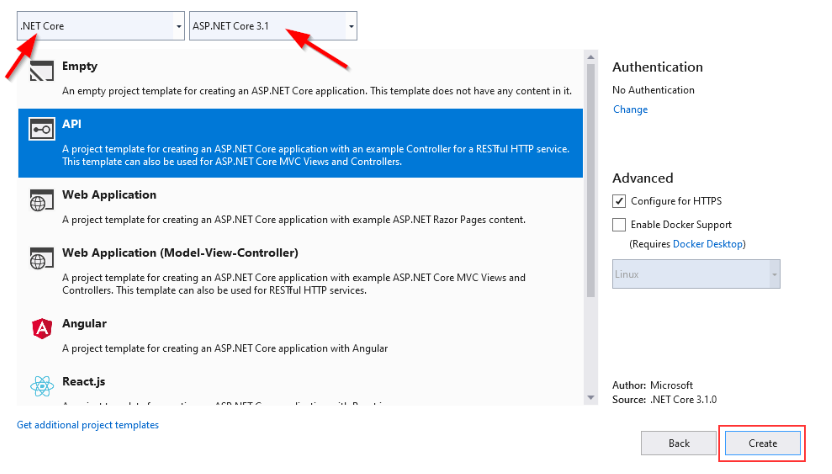
# Web API

## ASP.NET Core Web API - .NET Core Service Configuration



Tạo project như trên, sau khi tạo xong chúng ta sẽ tiến hành chỉnh sửa launchSetting.json, đây là file khá quan trọng trong .NET Core configuration.

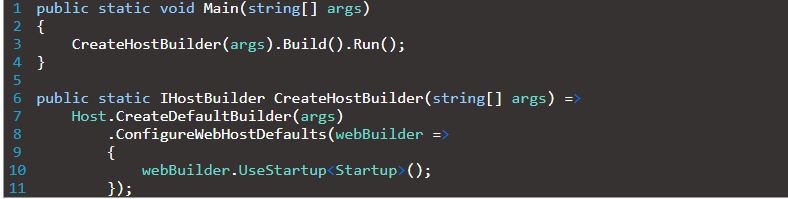
Property cần thay đổi là:

* applicationUrl - đây là property thiết lập địa chỉ đường dẫn của API
* launchBrowser- property này có nhận giá trị là true/false, nếu true thì khi chạy project thì sẽ tự động bật Browser truy cập vào địa chỉ của API



## Program.cs and Startup.cs

Trong Program.cs chúng ta có thể thấy được đoạn code:



Code này nhìn khá giống version .NET Core 2.2 nhưng thực chất có nhiều cái đã thay đổi.

Trong version 3.0 .NET Core đã rebuilt to use a Generic Host, thay vì sử dụng a separate Web Host

Về cơ bản, the CreateHostBuilder method bị phá vỡ thành 2 thành phần logic:

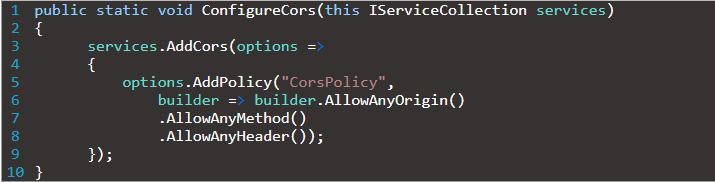
* CreateDefaultBuilder(args) – ASP.NET Core sử dụng cái này để config the app configuration, logging, and dependency injection container.
* ConfigureWebHostDefaults – Cái này add mọi thứ được yêu cầu bởi ASP.NET Core application điển hình (kestrel configuration, sử dụng Startup class, middleware pipeline...)

Chúng ta chú ý đến phần code: UserStartup<Startup>() . The Startup class là bắt buộc, trong class này chúng ta nhúng các config hoặc custom services những cái mà application của chúng ta cần.

## Extension Methods and CORS Configuration

Một extension method bản chất của nó là một static method. Chúng đóng vai trò lớn trọng .NET Core configuration bởi vì chúng làm dễ đọc source. Sự khác biệt giữa extension method và static method là nó chấp nhận “this” như là first parameter, và “this” thể hiện kiểu dữ liệu của đối tượng cái sử dụng extension method này. Một extension method phải nằm trong một static class

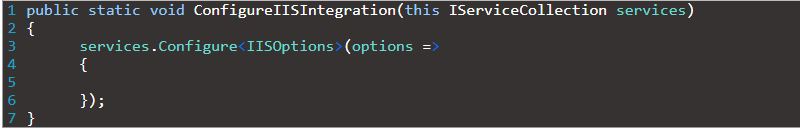
Chúng ta sẽ tiến hành config CORS trong application của chúng ta. CORS (Cross-Origin Resource Sharing) là một cơ chế, cái mà trao quyền tới user để access tài nguyên từ server trên một domain khác. Domain khác là như thế nào, ví dụ như chúng ta sử dụng Angular như là một client side trên một domain khác với domain của server, việc config CORS là bắt buộc.



Chúng ta đang sử dụng basic setting cho adding CORS policy bởi vì với project này cho phép bất cứ origin, method và header là khá đủ. Nhưng chúng ta có thể viết nhiều hơn hạn chế với setting nếu muốn. Thay vì AllowAnyOrigin() method cho phép request từ bất cứ nguồn nào, chúng ta có thể sử dụng WithOrigins(<http://dsads>) cái này sẽ chỉ cho phép request từ nguồn xác định. Cũng tương tự, thay vì sử dụng AllowAnyMethod() cái cho phép tất cả HTTP method, chúng ta có thể sử dụng WithMethod(“POST”,”GET”) cái sẽ chỉ cho phép methods HTTP xác định.....

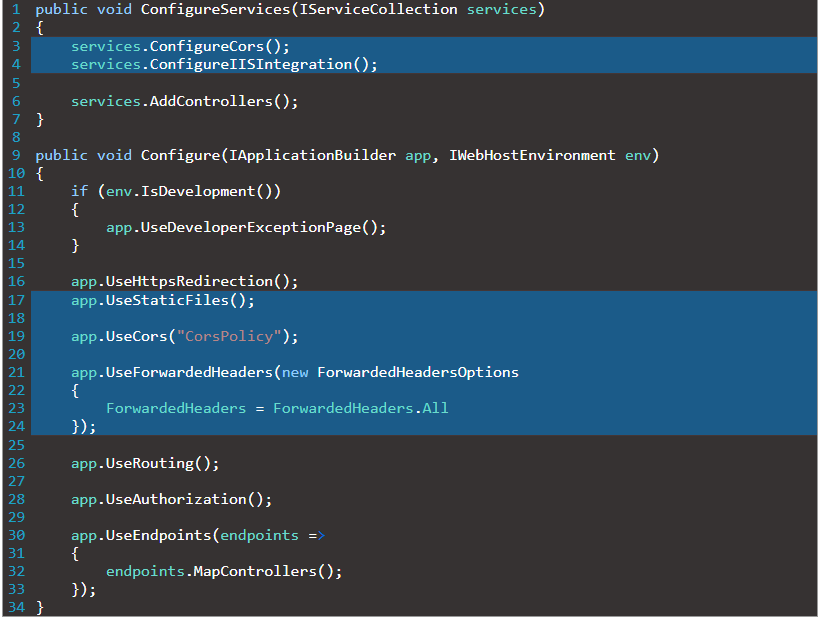
## IIS Configuration

Chúng ta cần config IIS intergration cái này sẽ giúp chúng ta với IIS deployment



Chúng ta không init bất cứ properties nào bên trong options bởi vì chúng ta chỉ cần default values.

Trong Startup class chúng ta sẽ thay đổi ConfigureServices và Configure methods:



* App.UseForwardedHeaders sẽ forward proxy headers tới the current request. Cái này sẽ giúp chúng ta trong Linux deployment.
* App.UseStaticFiles() cho phép sử dụng static files cho request. Nếu chúng ta không có cái này, nó sẽ sử dụng wwwroot folder trong our solution mặc định.

# Logging

Tại sao logging rất quan trọng? Trong khi mà application của ta trong deployping stage nó rất dễ để debug code và tìm ra lỗi, nhưng chúng ta làm sao có thể debug trong production environment?

Chính xác là không thể.

Đây là lý do tại sao logging mesage là một cách tuyệt vời để tìm ra lỗi sai và nơi mà error xảy ra vấn đè trong code của ta ở môi trường production. Chúng ta sẽ tạo một logger service

## Createing Required Projects

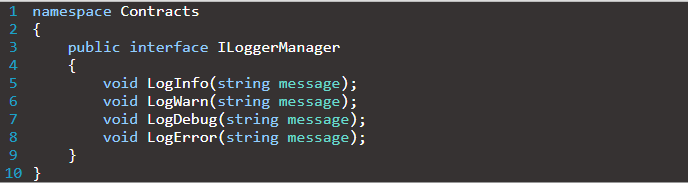
Chúng ta tạo 2 project mới. Project đầu tên là Contracts. CHúng ta sẽ chứa interfaces bên trong project này. Cái thứ hai là LoggerService. Chúng ta sẽ sử dụng nó cho logger logic

## Creating the Interface and Installing Nlog

Logger service của chúng ta chứa 4 method cho logging

* Info messages
* Debug messages
* Warning messages
* And error messages

Thêm nữa chúng ta sẽ tạo interface IloggerManager bên trong Contracts project chứa 4 định nghĩa của thằng này.

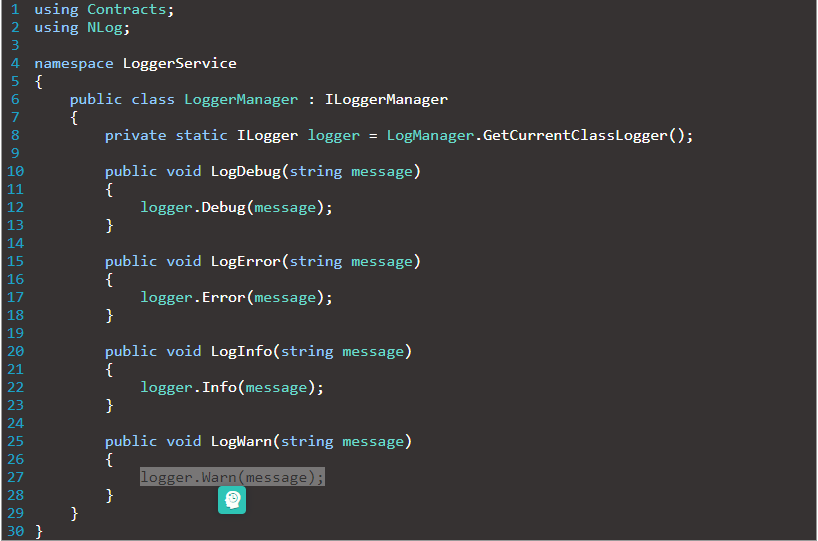


Trước khi mà thực thi IloggerManager bên trong LoggerService, chúng ta cần install Nlog library tới LoggerService project

Install-Package NLog.Extensions.Logging -Version 1.6.1

## Implement Interface and Nlog.Config file

Trong LoggerService project tạo một class mới là LoggerManager

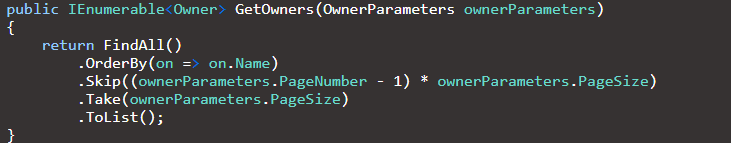


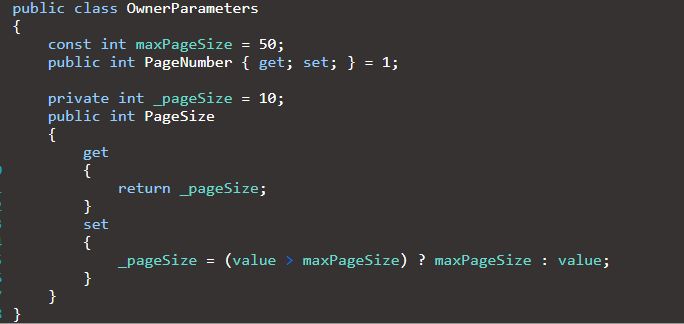
Chúng ta cần configure và inject nó vào bên trong startup class trong ConfigureServices method

Nlog cần thông tin về folder để tạo log files ở đó. Tên của các file sẽ là một minimum level of logging. Vì vậy chúng ta cần tạo một text file trong main project với tên là nlog.config. Chúng ta cần thay đổi path của internal log và file name parameters tới paths của chúng ta.

# Paging

Trong thực tế, ta không bao giờ sẽ trả về hết toàn bộ resources khi mà query API. Nếu trả về hết sẽ dẫn tới vấn đề hiệu năng và nó không được tối ưu cho việc public hoặc private APIs





Đơn giản khi ta request đến server, ta sẽ đi kèm là số lượng mỗi page và page nào được chỉ định. Vì vậy ta sẽ bỏ qua toàn bộ giá trị trước page và sau page đó.

