**Dot Net Core Notes**

- Ở môi trường local thì setting trong file **launchSetting.json,**trên môi trường production thì sẽ là file **appsetting.json.**

- Để quản lý các packages của frontend thì configure trong file **libman.json (*manage client-site libraries*)** như sau

***{***

***"version": "1.0",***

***"defaultProvider": "cdnjs",***

***"libraries": [***

***{***

***"library": "twitter-bootstrap@3.3.7",***

***"destination": "wwwroot/lib/bootstrap"***

***},***

***{***

***"library": "jquery@3.4.1"***

***"destination": "wwwroot/js/"***

***}***

***]***

}

----------

**if (env.IsDevelopment())**

**{**

**app.UseDeveloperExceptionPage();**

**}**

- Ở đoạn code phía trên nếu là môi trường development thì bật báo lỗi lên, còn nếu không phải thì chuyển qua trang error đế ẩn lỗi đối với user.

---------

- Cách viết sau:

***public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>***

***WebHost.CreateDefaultBuilder(args)***

***.UseStartup();***

Sẽ tương đương với:

***public static IWebHostBuilder CreateWebHostBuilder(string[] args)***

***{***

***return WebHost.CreateDefaultBuilder(args)***

***.UseStartup();***

***}***

*-------*

**Programe.cs**sẽ chạy trước, nó sẽ gọi đến method **Configure**của thằng**Startup.cs ,**rồi tiếp theo nó sẽ gọi đến **ConfigureServices**cũng của **Startup.cs .**

---------

**NGUYÊN TẮC HOẠT ĐỘNG CỦA MIDDLEWARE**

- **MiddleWare** được khai báo trong method **Configure**của thằng **Startup.cs**

**-**Sự khác biệt giữa việc thiết lập middleware **app.Run()** và **app.Use()**

**+ app.Run():**

*--Chạy xong là quay trả lại kết quả không gọi middleware tiếp theo*

**+app.Use():**

***app.Use(async (context, next) =>***

***{***

***await context.Response.WriteAsync(text: "***

Hello World!

");

***// Gọi middleware tiếp theo.***

***await next.Invoke();***

***});***

*-- Nếu passed middleware này thì nó sẽ gọi middleware tiếp theo khi ta khai báo next (kiểu như là lông middleware khác vào nó.*

**Ví dụ như bên dưới:**

// **middleware 1**

app.Use(async (context, next) =>

{

await context.Response.WriteAsync(text: "

Middleware 1.1

");

await next.Invoke();

await context.Response.WriteAsync(text: "

Middleware 1.2

");

});

// **middleware 2**

app.Use(async (context, next) =>

{

await context.Response.WriteAsync(text: "

Middleware 2.1

");

await next.Invoke();

await context.Response.WriteAsync(text: "

Middleware 2.2

");

});

// **middleware 3**

app.Run(async (context) =>

{

await context.Response.WriteAsync("Middleware 3.1");

});

Ta sẽ có kết quả như sau:

Middleware 1.1

Middleware 2.1

Middleware 3.1

Middleware 2.2

Middleware 1.2

**Do vậy việc sắp xếp middleware nào trước sau cũng sẽ ảnh hưởng đến kết quả.**

**TẠO MỚI MỘT MIDDLEWARE**

Khai báo một middleware.

public class SimpleMiddleWare

{

private readonly RequestDelegate \_next;

public SimpleMiddleWare(RequestDelegate next)

{

\_next = next;

}

public async Task Invoke(HttpContext context)

{

await context.Response.WriteAsync(text: "<div>Middleware 1.1</div>");

await \_next(context);

await context.Response.WriteAsync(text: "<div>Middleware 1.2</div>");

}

}

Đăng ký một middleware do user định nghĩa, trong **Startuup.cs** ta khai báo như sau:

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

// middleware 1

app.Use(async (context, next) =>

{

await context.Response.WriteAsync(text: "<div>Middleware 1.1</div>");

await next.Invoke();

await context.Response.WriteAsync(text: "<div>Middleware 1.2</div>");

});

// middleware 2

app.Use(async (context, next) =>

{

await context.Response.WriteAsync(text: "<div>Middleware 2.1</div>");

await next.Invoke();

await context.Response.WriteAsync(text: "<div>Middleware 2.2</div>");

});

**// Call a custome middleware.**

**app.UseMiddleware<SimpleMiddleWare>();**

// middleware 3

app.Run(async (context) =>

{

await context.Response.WriteAsync("Middleware 3.1");

});

}

Khi chạy chương trình ta sẽ được kết quả như sau:

Middleware 1.1

Middleware 2.1

Custome Middleware 1.1

Middleware 3.1

Custome Middleware 1.2

Middleware 2.2

Middleware 1.2

**TẠO MIDDLE THEO KIỂU EXTENSION METHOD**

Khai báo một class như sau:

public static class SomeMiddlewareExtensions

{

public static IApplicationBuilder UseSimpleMiddleware(this IApplicationBuilder builder)

{

return builder.UseMiddleware<SimpleMiddleWare>();

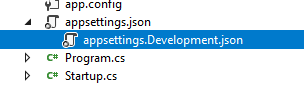
}

}

Lúc gọi ra trong file Startup.cs như sau:

app.UseSimpleMiddleware();

**CONFIGURATION IN ASP.NET CORE**



Quy tắc đặt tên cho file configuration.

Appsettings.**Tên\_môi\_trường**.json

**Ví dụ:**

appsettings.**Development**.json

appsettings.**Staging**.json

appsettings.**UAT**.json

Trong Program.cs

public static IWebHostBuilder CreateWebHostBuilder(string[] args)

{

var builder = WebHost.CreateDefaultBuilder(args).ConfigureAppConfiguration((hostingContext, config) =>

{

var env = hostingContext.HostingEnvironment;

// Đọc từ file appsettings.json

config.AddJsonFile(path: "appsettings.json", optional: true, reloadOnChange: true)

// Đọc từ các file mà user thêm vào.

.AddJsonFile(path: $"appsettings.{env.EnvironmentName}.json", optional: true,

reloadOnChange: true);

if (env.IsDevelopment())

{

var appAssembly = Assembly.Load(new AssemblyName(env.ApplicationName));

if (appAssembly != null)

{

config.AddUserSecrets(appAssembly, optional: true);

}

// Đọc cá biến môi trường từ máy đang chạy project.

config.AddEnvironmentVariables();

if (args!=null)

{

config.AddCommandLine(args);

}

}

})

.UseStartup<Startup>();

return builder;

}

Ví dụ trong file **appsettings.json** ta định nghĩa như sau:

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"SQLServerConnectionString": "SQL Connection String Simple",

"OracleServerConnectionString": "Oracle Connection String Simple"

},

"Message": "Hello from appsetting.json",

"Students": [

{

"Name": "Student 1",

"Age": 10

},

{

"Name": "Student 2",

"Age": 16

}

]

}

Giờ ta muốn đọc giá trị của **Message**  thì ta làm như sau:

Trong **Startup.cs**

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// Để đọc gia trị ra

Configuration.GetSection(key: "Message").Value;

// Để đọc giá trị từ thằng con, cha con phân cách với nhau bằng dấu :

Configuration.GetSection(key: "ConnectionStrings:SQLServerConnectionString").Value;

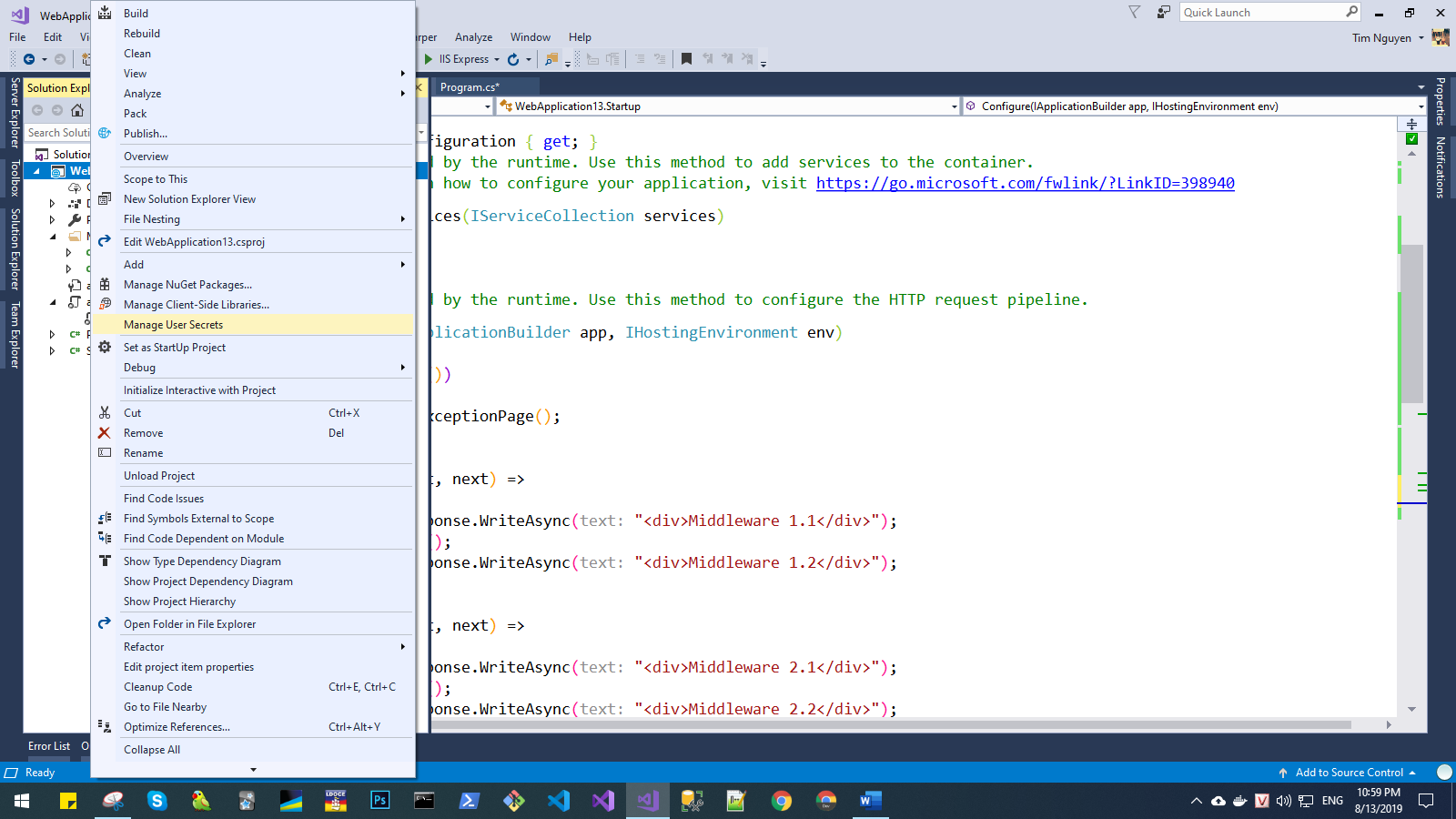
// Đọc từ một mảng giá trị

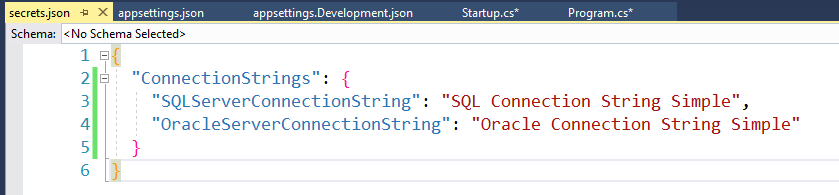
// Lấy ra tên của thằng số 0

// Kết quả là Student 1

Configuration.GetSection(key: "Students:0:name").Value;

Đọc giá trị từ **usersecret** trong môi trường Production.

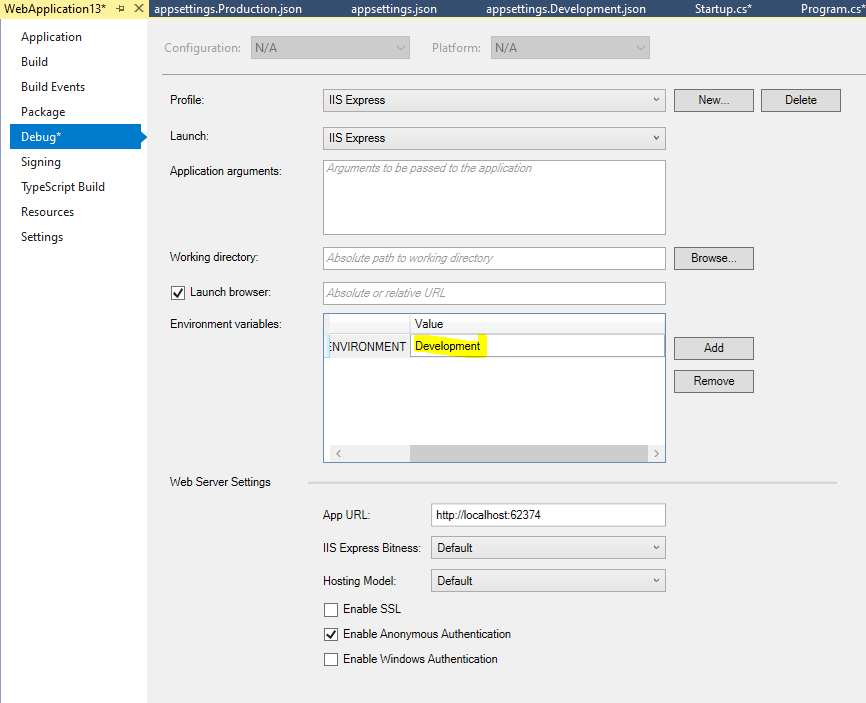




File secrets.json sẽ không nằm trong source code mà sẽ nằm trong folder user trên máy chạy code đó.

Đọc giá trị từ file appsettings. **Production**, hay appsettings.**Development**

Ta thiết đặt ở đây, tùy vào giá trị ta khai báo mà nó sẽ đọc giá **configuration** từ file **Production** hay **Development**



Ngoài ra ta có khai báo config.AddCommandLine(args); nên nếu ta dùng lệnh dotnet run với argument thì sẽ nhận giá trị từ argument ta truyền vào.

Ngoài ra ta còn có thể đọc tư các biến setup môi trường cho máy đang chạy ứng dụng dot net core.

**STATIC FILE TRONG DOTNET CORE**

public static IWebHostBuilder CreateWebHostBuilder(string[] args)

{

return WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>()

.UseContentRoot(Directory.GetCurrentDirectory())

.UseWebRoot("Resources");// Thay vì lấy thư mục wwwroot.

}

**CẤU HÌNH ROUTE TRONG CHO ASP.NET CORE MVC.**

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.Run(async (context) =>

{

await context.Response.WriteAsync("Hello World!");

});

// Sử dụng MVC với default route (như phần bên dưới).

app.UseMvcWithDefaultRoute();

// Khai báo với custome route.

app.UseMvc(routes =>

{

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

});

}

Cấu hình route trong dot net core để giống pattern nhưng nhận tham số đầu vào khác kiểu.

app.UseMvc(routes =>

{

// id ở đây là kiểu int

routes.MapRoute(name: "default", template: "post/{id:int}",

defaults: new { Controllers = "Post", action = "PostById" });

// id ở đây là kiểu string

routes.MapRoute(name: "anotherRoute", template: "post/{id:alpha}",

defaults: new { Controllers = "Post", action = "PostByName" });

});

// Parameter sẽ được validate bằng chuỗi regex

routes.MapRoute(name:"regexRoute",

template:"{controller}/{action}/{year:regex(^\\d{{4}}$)",

defaults:new {controller="Home",action="Index"});

public class PostController : Controller

{

public IActionResult PostById(int id)

{

return View();

}

public IActionResult PostByName(string id)

{

return View();

}

}