Como Construir un Web Api Restful con ASP.NET Core

Con este documento, se presenta una guía paso a paso acerca de como poder implementar una *Web API RESTful* limpia y mantenible.

A lo largo del desarrollo se introduciran conceptos importantes en el desarrollo de software modular, como es el caso de uso de algunos los patrones de diseño, testeo de aplicaciones web y fundamentos del funcionamiento del protocolo HTTP.

Visión General

El término Restful no es nada nuevo, en realidad este se refiere a un estilo arquitectónico donde los servicios web envian y reciben datos desde y hacia aplicaciones cliente, sean estas aplicaciones web o mobiles. El objetivo de este tipo de aplicaciones es centralizar los datos, para que cualquier aplicación cliente pueda consumirlos o usarlos.

Elegir las herramientas apropiadas para crear o escribir aplicaciones de tipo Restful es crucial, debido a que ud debería pensar en sesalabilidad, mantenibilidad, documentación y otros aspectos importantes. Por ejemplo, el Framework ASP.NET Core se presenta como una herramienta poderosa para crear un API y lograr tales objetivos.

> Puede que de momento ud sea algo nuevo en el desarrollo web, y quizás desconozca algunos de los términos mencionados con anterioridad o próximos a ser mencionados. El consejo al respecto es, mantengase activo a la lectura pues el desarrollo web es quizás uno de los escenarios en los que el desarrollo de software es muy volatil al respecto de las tecnologías. Sin embargo, no hay necesidad de alarmarse los conceptos y tecnologías que surgen a diario surgen sobre los mismos conceptos, y son producto de mejoras u optimizaciones a los mismos.

> Trataré de forma breve dar unas bases acerca de Frameworks, Api, y HTTPs antes de entrar de lleno a los pasos para crear una Web Api.

Prerequisitos

Conceptuales (Knowledge)

> En lo que respecta al desarrollo de Web Apis Restfull y en general para el *Desarrollo Web*, los conocimientos fundamentales son:
>
> - Conocer bien el problema que quiere atacar o aquello que quiere desarrollar.
> - Conocimientos sólidos de la programación orientada a objetos (**OOP**)
> - Conocer acerca de la estructura del formato de representación de texto JSON.
> - Conocimiento y comprensión acerca como trabaja el protocolo de Hipertexto (**HTTP**). Incluso el reconocer e interpretar los diferentes códigos de error HTTP, es una habilidad deseable de cualquier desarrollador web.
> - Conocimientos acerca del enfoque (**REST**) y el significado y propósito que cumplen los API endpoints.

> - Entendimiento de como las bases de datos relacionales (RDMS) trabajan. Finalmente y no menos importante, el conocimiento de acerca de ¿Que son Patrones de Diseño? y ¿Para que sirven?

Habilidades (Skills)

Ests conceptos no estarían en el alcance de este documento, sin embargo puedo sugerir algunas buenas fuentes para lograr el objetivo.

- Conocimiento Intermedio-Avanzado del lenguaje de programación seleccionado o de su preferencia. (C# para nuestro caso). Yo recomendaría la lectura detenidad de Bases de C# y C# Intermedio y OOP, que en uno o dos días lo pondrán en forma sobre el lenguaje.
- El conocimiento sobre ¿Qué es un Marco de Trabajo o Framekwork? y su elección; puede hacer diferencia en la velocidad y calidad del trabajo diario de un desarrollador.

Técnicos - Herramientas (IDEs, Librerías & Tools)

Para el desarrollo de una Web Api bien estructurada, y casi cercana a un escenario real empleando NetCore, haré uso de:

- VsCode, un editor de código multiplataforma que bien puede ser adaptado, para convertirse en una herramienta alternativa de reemplazo para Visual Studio, en cunto al desarrollo de aplicaciones apoyadas en el framework .NetCore.

  > Si por preferencia, decide acoger la sugenrencia de emplear **VsCode** como herramienta de desarrollo,.Recomiendo instalar la extensión de C# , para tener un soporte cómodo en el manejo del lenguaje mientras se codifica el programa.

- .NET Core 2.2 como marco de trabajo (FrameWork), para el desarrrollo con C#. Por medio de este framework, iremos a detalle acerca de patrones de diseño y estrategias comunes de desarrolle, para simplificar el desarrollo de la aplicación.

- Entity Framework Core un marco de traabajo e Microsotf, creado para facilitar del desarrollo y migraciones hacia bases de datos relacionales MsSqlServer, será integrado con el proyecto para emular el comportamiento de una base de datos, desde la memoria RAM de la máquina empleada para el desarrollo.

- AutoMapper es una librería que nos facilita la vida al momento de mapear de manera automática las propiedades de un objeto to deliver the necessary functionalities. Si hay curiosidad acerca de que hablo, puede ver esta corta referencia acerca de su uso.

The Scope

Escribamos una API web ficticia para un supermercado. Imaginemos que tenemos que implementar el siguiente alcance:

> - *Crear un servicio REST que permita a las aplicaciones de los clientes gestionar el catálogo de productos del supermercado. Necesita exponer **endpoints** para crear, leer, editar y eliminar categorías de productos, tales como productos lácteos y cosméticos, y a su vez poder gestionar los productos de estas categorías.*

> - *Para las categorías, necesitamos almacenar sus nombres. Para los productos, necesitamos almacenar sus nombres, unidad de medida (por ejemplo, KG para productos medidos por peso), cantidad en el paquete (por ejemplo, 10 si el producto es un paquete de galletas) y sus respectivas categorías.*

Para simplificar el ejemplo, no manejaré los productos en stock, el envío de productos, la seguridad y cualquier otra funcionalidad. El alcance dado es suficiente para mostrar cómo funciona ASP.NET Core, para el propósito alcance descrito con anterioridad.

Para desarrollar este servicio, necesitamos básicamente dos **endpoints** de la API: uno para gestionar las categorías y otro para gestionar los productos. En términos de comunicación y respuestas, podemos pensar en que las respuestas devueltas desde dichos **endpoints** obedecen a las siguientes estructuras en formato **JSON**:

API endpoint: `/api/categories`

JSON Response (para solicitudes (GET HTTP requests)):

```
{
  [
    { "id": 1, "name": "Fruits and Vegetables" },
    { "id": 2, "name": "Breads" },
    …
  ]
}
```

API endpoint: `/api/products`

JSON Response (para solicitudes (GET HTTP requests)):

```
{
  [
    {
      "id": 1,
      "name": "Sugar",
      "quantityInPackage": 1,
      "unitOfMeasurement": "KG"
      "category": {
        "id": 3,
        "name": "Sugar"
      }
    },
    …
  ]
}
```

Let's get started writing the application.

Antes de iniciar (Ajustes VsCode - Para NetCore)

Considerando que ha optado por trabajar con **VsCode**, es necesario verificar que versión de FrameWork está **Activa** para trabajar.

Si está trabajando desde Visual Studio, puede omitir esta sección.

Verificar versión de .NetCore activa

Para verificar que versión se encuentra, pueden activar la consola de VsCode `Shift + Ñ` y digital el comando:

```
dotnet --version
```

Asumamos que el resultado es `3.1.101`, sin embargo, deseamos trabajar con `2.2.207`; para activar esta versión es necesario:

- Tener instalado el SDK versión `2.2.207`
- Crear el archivo `global.json` en la carpeta del proyecto.
- Ajustar el archivo `global.json` para indicar la versión del framework a utilizar.
- Verificar el ajuste.
- Proceder a crear el proyecto que desea.

Crear el `global.json`

El archivo `global.json` guarda la configuración del proyecto sobre el cual se encuentre ubicado para crear o trabajar en un proyecto NetCore, empleando la consola.

Para crear dicho archivo, se puede valer del comando:

```
dotnet --globaljson
```

Como resultado de la ejecución del comando anterior, verá que el archivo `global.json` ha sido creado a nivel de la carpeta.

Asigne la versión del FrameWork Deseado

Para setear la versión del framework con la cuál va a trabajar, bastará modificar el valor de la llave (key) `version` anidada dentro de la llave `sdk`.

Por ejemplo, si quisiera migrar desde la versión `3.1.101`

```
{
  "sdk": {
    "version": "3.1.101"
  }
}
```

a la `2.2.207`, debería hacer ajustar así

```
{
  "sdk": {
    "version": "2.2.207"
  }
}
```

y finalmente, verificar que el resultado de `dotnet --version` corresponde con el ajuste.

Verificar otras versiones NetCore Instaladas

Puede ser el caso en que tenga varias versiones del SDK instaladas en la misma máquina de desarrollo y desee, saltar entre ellas para trabajar distintos proyectos. En ese caso necesitará verificar la versión exacta de cada SDK, para poder indicarla en el archivo `global.json`.

Para tal propósito, puede consultar los SDKs instalados en la máquina empleando:

```
dotnet --list-sdks
```

Step 1 — Creating the API

First of all, we have to create the folders structure for the web service, and then we have to use the .NET CLI tools to scaffold a basic web API. Open the terminal or command prompt (it depends on the operating system you are using) and type the following commands, in sequence:

Paso 1 - Creación de la API

En primer lugar, tenemos que crear la estructura de carpetas para el servicio web, y luego tenemos que utilizar las herramientas .NET CLI para crear una API web básica.

Abra la terminal o el símbolo del sistema (depende del sistema operativo que esté usando) y escribe los siguientes comandos, en secuencia:

Linux (Terminal)

```
mkdir -p src/Supermarket.API
cd src/Supermarket.API
dotnet new webapi
```

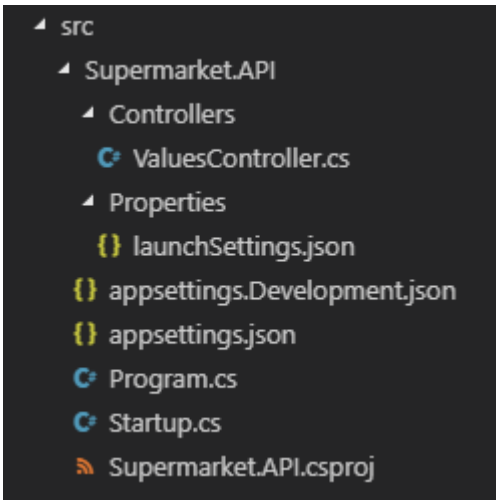
Windows (cmd)

```
mkdir src/Supermarket.API
cd src/Supermarket.API
dotnet new webapi
```

- El primer comando, crea un nuevo directorio para la API `src/Supermarket.API`
- El segundo, cambia la ubicación actual a la nueva carpeta.
- El último, genera un nuevo proyecto siguiendo la plantilla de la API Web (de NetCore), pues es el tipo de aplicación que se desea desarrollar.

🔎📚🏄🌎 Puedes leer más sobre este comando y otras plantillas de proyecto que puedes generar revisando este enlace.

El nuevo directorio ahora tendrá la siguiente estructura:



Estructura del proyecto

Debido a que .NetCore ha sido creado bajo el enfoque de un middleware (es decir, pequeños trozos de la solicitud que llega a la aplicación son procesados por lote mediante el application pipeline, para posteriormente entregar respuestas.

Adicionalmente, todos los módulos de las aplicaciones en .NetCore, deberían ser pensados como servicios acoplables al servidor principal (**`kestrel`**) creado por NetCore desde sus lineas de código definidas en `program.cs`

Vista General

- `Program.cs` Al igual que con las aplicaciones de consola en *NetFramework*, en *NetCore* la clase `Program.cs` es el punto de entrada para cualquier aplicación, mediante el método **Main**, en el cual una sentencia `CreateWebHostBuilder(args).Build().Run();` la creación del servidor `kestrel` en tiempo de ejecución.

```
Namespace Supermarket.API
{
    public class Program
```

```
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
```

> Si, el servidor web que expone o sirve la aplicación, **va embebido junto con la aplicación**, ya no más dolores de cabeza con **IIS** para poder servir aplicaciones .Net.

- `Startup.cs` La configuración del servidor ``kertrel.cs` puede ser manejada por la clase "Startup". Si ya ha trabajado con marcos de trabajo como Express.js antes, este concepto no es nuevo.

```
namespace Supermarket.API
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services
to the container.
        public void ConfigureServices(IServiceCollection services)
        {

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
        }

        // 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();
            }
            else
            {
                // The default HSTS value is 30 days. You may want to change this
for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }
```

```
            app.UseHttpsRedirection();
            app.UseMvc();
        }
    }
}
```

Cuando la aplicación se inicia, se llama al método "Principal", de la clase "Programa". Crea un host web por defecto usando la configuración de inicio, exponiendo la aplicación vía HTTP a través de un puerto específico (por defecto, el puerto 5000 para HTTP y 5001 para HTTPS).

Referencias Para Mantenerse Aprendiendo

- .NET Core Tutorials — Microsoft Docs
- ASP.NET Core Documentation — Microsoft Docs
- how to build RESTful APIs with ASP.NET Core
- Glosary Api
- JWT Tockens
- NetCore Api