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 escalabilidad, 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"
      }
    },
    …
  ]
}
```

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
```

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)

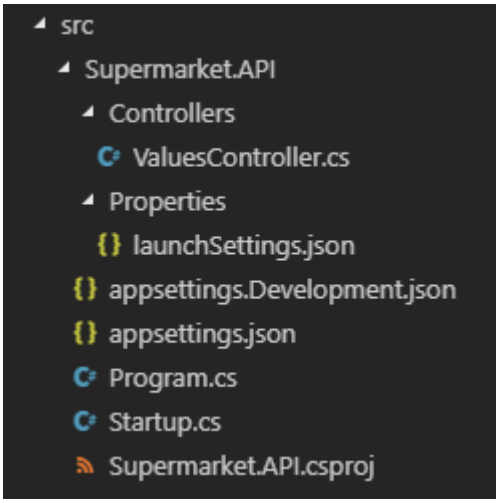
```
mkdir src/Supermarket.API
cd src/Supermarket.API
dotnet new webapi
```

- 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, es un RUNTIME SERVER**, ya no más dolores de cabeza con **IIS** para poder servir aplicaciones .Net.

- `Startup.cs` La configuración del servidor `kestrel.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).

El controlador

Si se da un vistazo al archivo `Controller/ValuesController.cs`, se puede ver que es una clase que define una serie de métodos.

Para comprender mejor el propósito de este archivo y comprender la utilidad que tienen los métodos definidos se considera importante introducir algunos conceptos básicos acerca del funcionamiento del protocolo http, a la par que verificamos la definición de cada método.

El Contexto HTTP

El conocido protocolo de hipertexto - Http, es un conjunto de reglas y procesos, que definen la forma en cómo se puede establecer la comunicación entre distintas máquinas que acceden a internet para poder intercambiar información.

La palabra protocolo no compete sólo al ámbito de las tecnologías y las comunicaciones. Si se hace memoria, acerca de los conocimientos adquiridos sobre gramática y lenguaje en época de la escuela o el colegio, quizá le sea familiar el término proceso de comunicación, el cual explica los factores y/o elementos claves, que permite definir la forma en que nos comunicamos, bien sea de forma oral o escrita.

En aquel entonces se mencionaba que todo proceso de comunicación depende de tres factores fundamentales para poder llevarse a cabo. El primero de ellos era la existencia de un emisor, Que podría ser o no una persona el segundo el receptor, que era un elemento o persona encargado de recibir la información. El tercer factor el mensaje el cual contenía la información que se desea va a transmitir. Finalmente los factores más importantes el código y el protocolo, Los cuales Mediante los cuales se definió una serie de símbolos y reglas con las cuales se podían perfectamente interpretar el mensaje y garantizar que la conversación se desarrollará de forma clara, correcta y coherente.

Puede que ahora sea más sencillo hacer alguna analogía, si se recuerda que los elementos que componen una arquitectura cliente-servidor, perfectamente podemos asociar el emisor como un cliente y al receptor como un servidor, y viceversa.

En cuanto al código y el mensaje que se transmite, este estará asociado a los distintos formatos (JSON, XML entre otros) a través de los cuales se intercambia la información a través del Canal (internet). Tan solo hace falta relacionar un elemento, el protocolo.

Apegado estrictamente al significado de la palabra, protocolo se define como un conjunto de reglas por las cuales se rigen el intercambio de información entre dos personas equipos o cosas conectados entre sí.

Para el caso particular de nuestra analogía, el protocolo corresponde con http; el cual es un protocolo de comunicación asíncrono, que define una secuencia de pasos mediante la cual la información debe ser intercambiada para ser coherente. Adicionalmente, http define un conjunto de códigos de error y/o éxito, entre otros; mediante los cuales se puede establecer el estado de la comunicación y la información, intercambiada entre el cliente y el servidor.

En términos generales se puede decir que la información (el mensaje), está compuesto por un cabecero (header) y un cuerpo (body). El cabecero contiene información relevante para definir la forma en cómo se intercambia la información; mientras que el cuerpo, contiene los datos a ser procesados o presentados; según sea sea su origen o destino; el cliente o el servidor).

Aplicando el concepto (los Controladores y su esencia)

Dentro de la arquitectura cliente-servidor los conceptos de ingeniería del sofá se encuentra la definición de la arquitectura por capas.

Si bien la arquitectura cliente-servidor permite Definir la forma en cómo el software se divide desde el punto de vista de la comunicación. Arquitectura de capas permite Definir la forma en como el sofá se divide en su estructura en código.

Existen dos tipos de arquitectura de casas de dos niveles y tres niveles, El caso abordan en este documento se abordan enfoques de tres capas. Las cuales se dividen en capa de presentación (FrontEnd), capa de lógica de negocio (Backend) y Capa de persistencia (almacenamiento).

Los controladores son un elemento especial dentro de las arquitecturas pues se convierten en la puerta de entrada en que las acciones ejecutadas por el usuario en la capa de presentación (Frontend - Lado cliente), acceden a la lógica de negocio (BackEnd) definida del lado del servidor.

La definición anterior es importante dado que catalogamos a los controladores como un componente intermediario, Es decir su única función es la De servir como componente mediador en el intercambio de de la información, Definiendo la puerta de entrada hacia la lógica de negocio; de manera que sobre él o ellos no se deben definir estructuras de control ni lógica de negocio.

A lo sumo, la lógica definida dentro de los controladores está limitada a servir para validación del estado de la comunicación mediante el protocolo http, o validación en primera instancia de los datos que provienen del cliente.

Es por ello que si miramos detenidamente los métodos definidos dentro de la clase `Controller/ValuesController.cs`, se aprecia que en la parte superior de cada método están definidos los verbos (acciones) a través de las cuales, se define la forma en como se intercambia información con el lado cliente mediante el protocolo http; para el caso puntual los verbos HttpGet, HttpPost, HttpPut y HttpDelete.

```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
    // GET api/values
    [HttpGet]
    public ActionResult<IEnumerable<string>> Get()
    {
        // obtener información de manera grupal, sin necesidad de un parámetro
        return new string[] { "value1", "value2" };
    }

    // GET api/values/5
    [HttpGet("{id}")]
```

```
        public ActionResult<string> Get(int id)
        {
            // obtener información de manera individual recibiendo un parámetro
            return String.Format("HttpGet del elemento {0}", id);
        }

        // POST api/values
        [HttpPost]
        public string Post([FromBody] string value)
        {
            // permitir la creación de un elemento apoyado en los parámetros recibidos
            return String.Format("HttpPost {0}", value);
        }

        // PUT api/values/5
        [HttpPut("{id}")]
        public string Put(int id, [FromBody] string value)
        {
            // permitir la modificación de un elemento apoyado en los parámetros
    recibidos
            return String.Format("HttpPut modificar elmento {0}, con el valor {1}",
    id, value);
        }

        // DELETE api/values/5
        [HttpDelete("{id}")]
        public string Delete(int id)
        {
            // permitir la eliminación de un elemento apoyado en los parámetros
    recibidos
            return String.Format("HttpDelete {0}", id);
        }
    }
```

Para el caso particular de las Api web de tipo Restful, el funcionamiento de los controladores se apoya en las direcciones de recurso (endpoints); que son Expuestos por el servidor, y a las cuales se envían solicitudes http para tener acceso a la lógica de negocio disponible detrás de los controladores.

Es importante mencionar que, Si bien los controladores están en capacidad de interpretar el cuerpo del mensaje recibido mediante la solicitud http; también, a partir de la url mediante la cual se hace la solicitud; estos están en capacidad de extraer información adicional, que puede emplearse como parámetro, para la invocación de los métodos definidos dentro de la lógica del controlador.

En el caso particular del framework .*NetCore*, la ruta del recurso a través de la cual se accede al controlador `ValuesController.cs`, está definida por la sentencia `[Route("api/[controller]")]`. La sentencia anterior indica que el acceso a los métodos del controlador, es posible cuando se hacen solicitudes a la dirección url compuesta por `dominio/api/[controller]`, que cono se verá en futuro corresponderá a `http://localhost:5000/api/values` para el caso en que se usa protocolo http para el intercambio de información.

> ⚠ Con lo anterior, se ha podido dar Claridad acerca del contexto y uso del protocolo http. Adicionalmente se a podido exponer el rol que tienen los controladores dentro de la arquitectura de capas; mediante la cual se definen las responsabilidades de cada componente de software, y el rol que cumplen desde el punto de vista de la arquitectura cliente-servidor; por medio de quien se define la función y los elementos involucrados en el Intercambio de la información.

El Negocio

Un aspecto fundamental antes de adentrarse en el conjunto de reglas que describen la lógica del negocio de cualquier aplicación, son las etapas posteriores a diseño del software, en las cuales se definen las estructuras de datos que representan de la forma más sencilla y clara datos que serán manejados y procesados por la aplicación.

Teniendo una estructura Clara acerca de la información que procesa la aplicación se tiene un punto De partida para iniciar la fase de codificación y desarrollo del sofá allí empieza a ser importante el tener claro que el código que se escribe debe ser lo más claro y simple posible para garantizar que a medida que cada desarrollador aporta lógica al software ésta sea mantenible. .

Los modelos de dominio

Tomando como Punto de partida el análisis de la fase de requerimientos y posiblemente los primeros resultados de la fase de diseño se puede iniciar la codificación de las estructuras de datos dentro del código esto es a lo que denominamos modelos de dominio.

El nombre modelo de dominio, se atribuye a que se trata de el conjunto de estructuras de datos que permiten representar la información en el lado del dominio; es decir el servidor, y que serán procesadas por la lógica de negocio bien sea para retornar un resultado o, para ser almacenadas por la capa de persistencia.

Para el caso de uso presentadom, se definen dos modelos de dominio que permiten representar la información asociada a productos a las categorías; estas últimas permiten representar la forma en como productos pueden ser agrupados, además de permitir establecer una relación entre clases, muy común en aplicaciones de ambito empresarial en las que se emplean bases de datos relacionales.

Para tal propósito dentro del proyecto se crea una carpeta denominada `Dominio` y dentro de ella otra a la que se le define `Modelos`. Dentro de esta última, se definen las clases `Dominio/Modelos/Categoria.cs` y `Dominio/Modelos/Producto.cs`; que representan las estructuras de datos asociadas a la información de categorias y productos respectivamente.

```
public class Categoria
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public IList<Producto> Productos { get; set; } = new List<Producto>();
}
```

```
public class Producto
{
```

```
        public int Id { get; set; }
        public string Nombre { get; set; }
        public short CantidadxPaquete { get; set; }
        public EUnidadDeMedida UnidadDeMedida { get; set; }

        public int CategoriaId { get; set; }
        public Categoria Categoria { get; set; }
    }
```

Además de las clases que permiten representar a los productos y las categorías, es necesario definir un conjunto de clases que permite representar de forma clara y coherente la lógica del negocio manejada por la aplicación.

El patron de repositorios

Cuando se piensa en una arquitectura de software modular, los patrones de diseño son un elemento clave que permite establecer el contexto y delimitación de cada una de las instancias u objetos, que representan la información de un producto o una categoría.

Para nuestro caso en particular, se hace uso del patrón de diseño de repositorios; el cúal involucra (implica) la definición de interfaces a través de las cuales los controladores podrán invocar la lógica de negocio,Representada por un conjunto de clases a las que se denomina repositorios.

Controladores

En la sección de contexto se abordaron los conceptos básicos detrás del protocolo http y el funcionamiento de los controladores. Ahora es el momento de presentar el diseño y construcción de los controladores de Api Web el caso de uso, y definir los direcciones de recurso (endpoints) que expondrá el servidor para tener acceso a la lógica de negocio.

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
- Building an ASP.NET Web API with ASP.NET Core
- Build a Simple CRUD App with Angular 8 and ASP.NET Core 2.2
- Creating Web API in ASP.NET Core 2.0