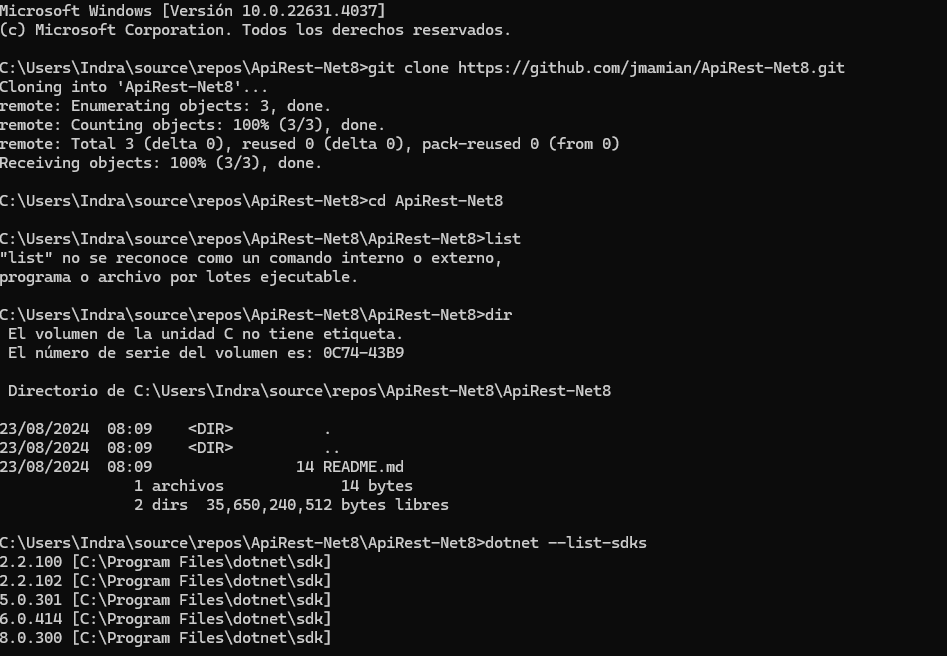
Crear WebAPi con .NET8

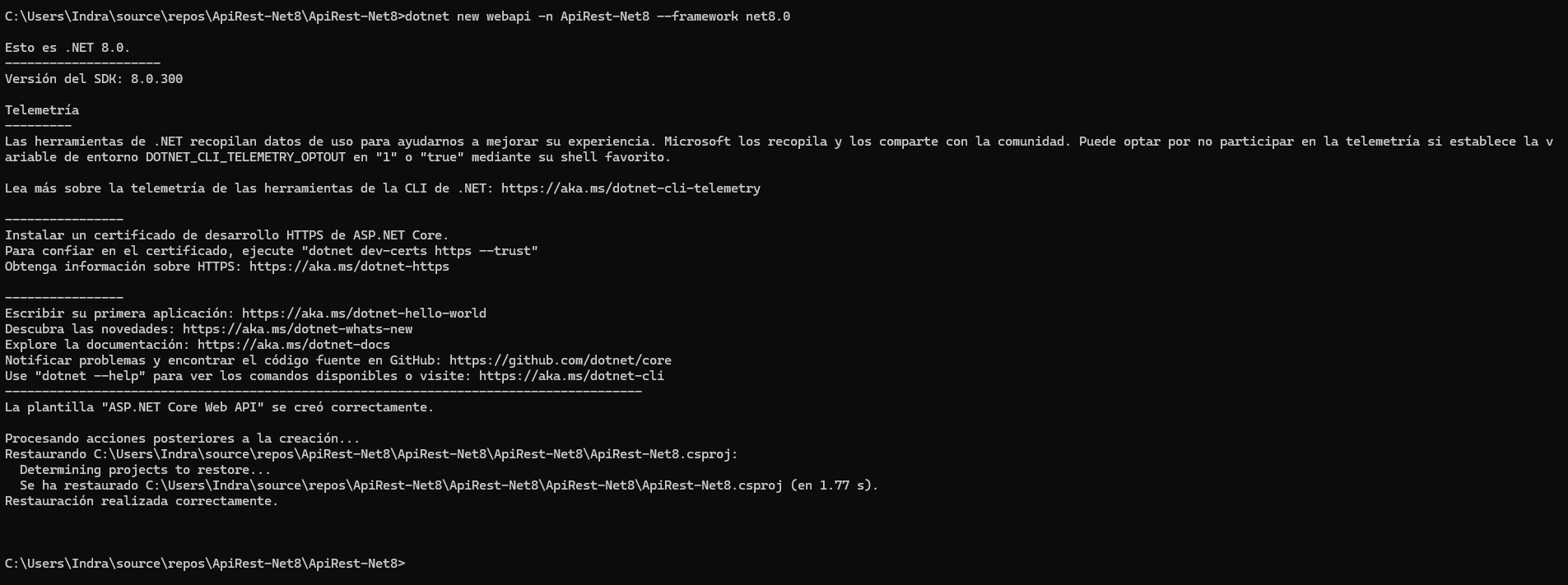
1. Lo primero que tenemos que hacer es comprobar las versiones de los SDK de .NET que tenemos instalados:

dotnet --list-sdks



1. Desde el cmd escribir el siguiente comando para crear la webapi:

dotnet new webapi -n ApiRest-Net8 --framework net8.0



1. Luego nos toca crear el archivo de solución ya que por defecto si creamos la webapi desde la consola no crea el archivo .sln automáticamente debido a un comportamiento predeterminado de la CLI de .NET, ya que esta diseñada para ser minimalista, permitiendo a los desarrolladores crear proyectos rápidamente sin necesidad de una solución, lo cual es útil para proyectos individuales o ejemplos rápidos.

**Cómo Crear un Archivo de Solución Durante la Creación del Proyecto**

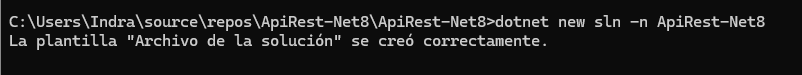
Si prefieres que el archivo de solución sea creado automáticamente al generar tu proyecto de API, puedes hacerlo en un solo paso agregando el comando para crear la solución y añadir el proyecto a ella:

dotnet new sln -n SimpleApi

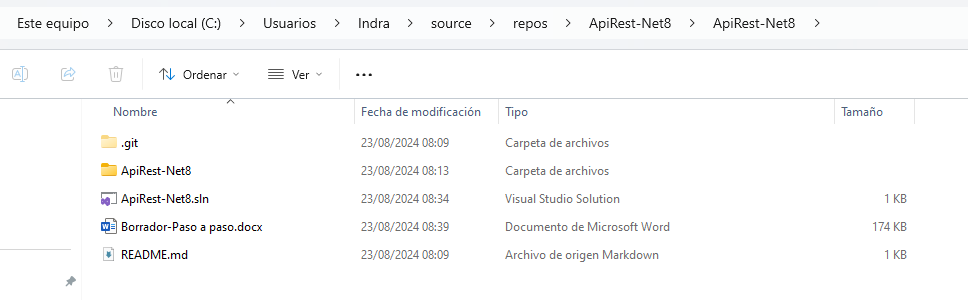
dotnet new webapi -n SimpleApi

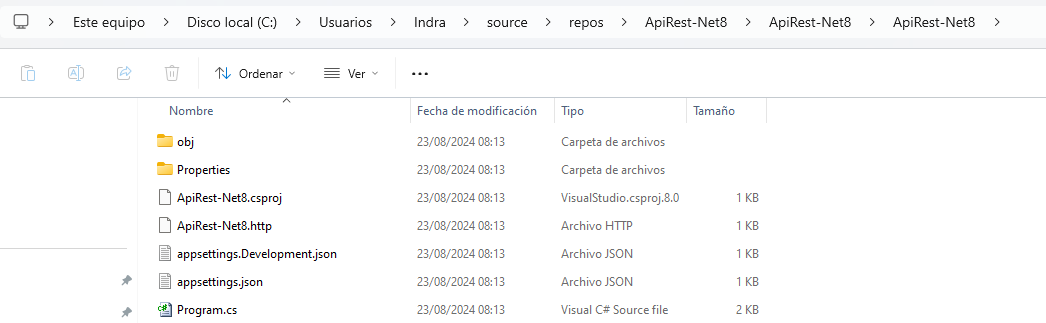
dotnet sln add SimpleApi/SimpleApi.csproj

* Creamos el archivo .sln en el directorio raíz de nuestra webapi

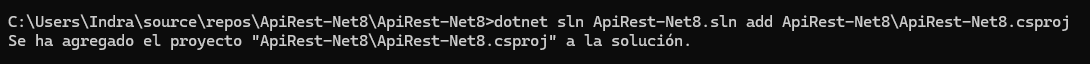


* Debe ya aparecer el archivo de solución

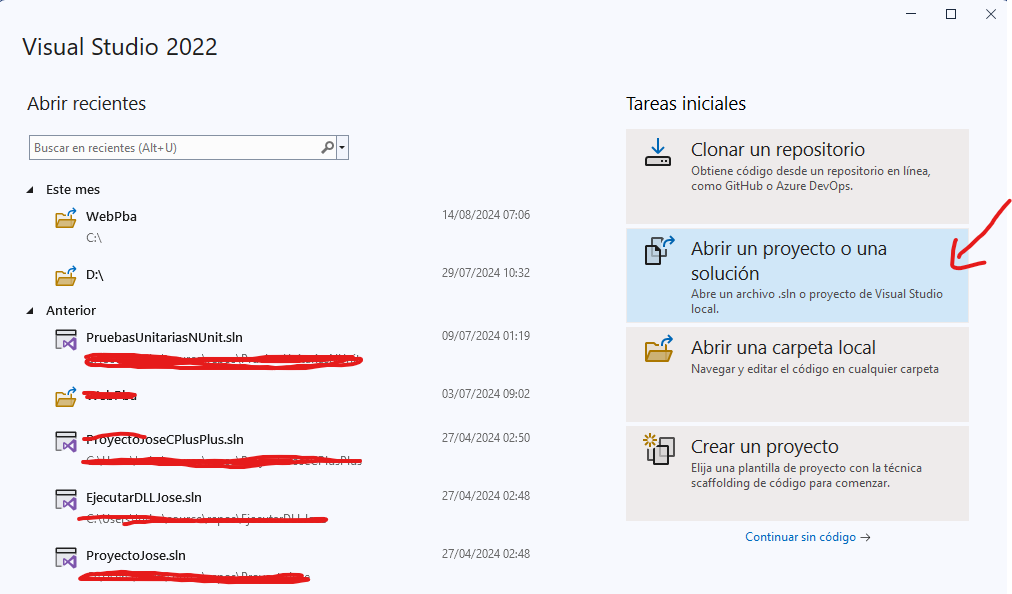




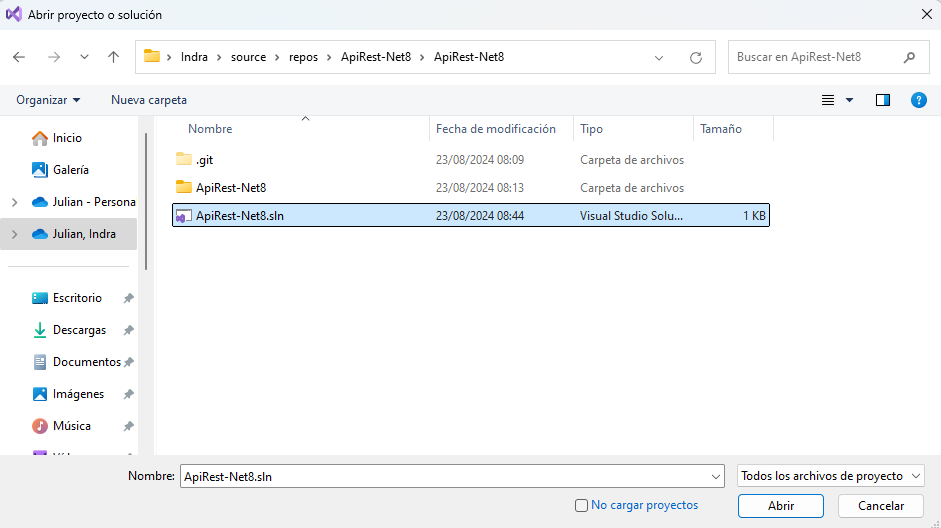
1. Ahora toca incluir el proyecto a la solución:



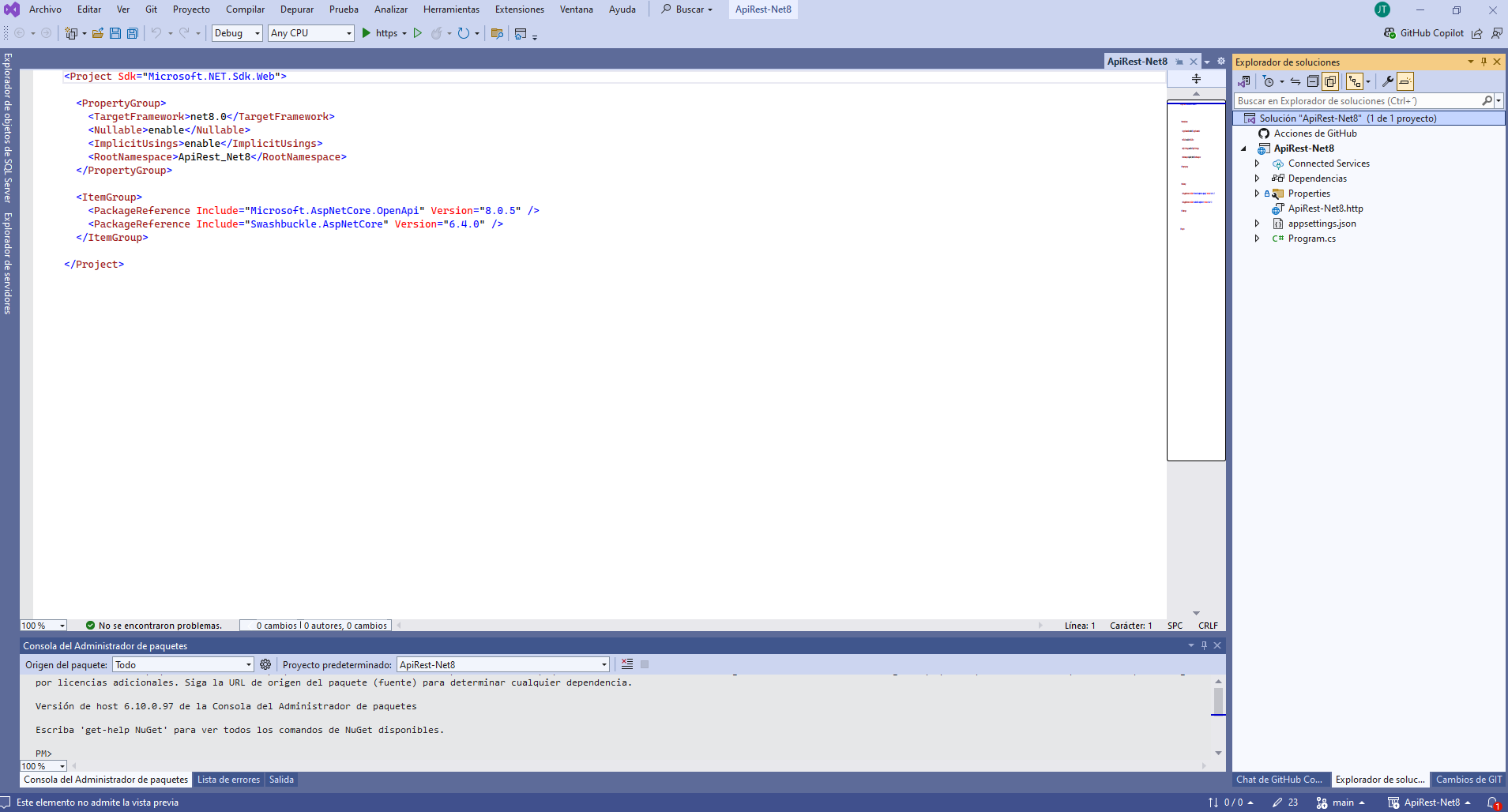
1. El paso a seguir es abrir visual studio 2022 y oprimir en la opción abrir proyecto o solución:



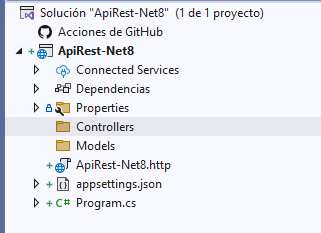
* Luego seleccionamos el archivo (.sln):



* Observamos que se abrió correctamente la solución:



1. Creo las carpetas Models y Controllers dentro del proyecto ya que en .NET8 por tener un enfoque minimalista, no las crea por defecto.



1. Crear un modelo de datos simple: Agrega una clase Product para representar el recurso. Crea un archivo llamado Product.cs en el directorio Models:

// Models/Product.cs

public class Product

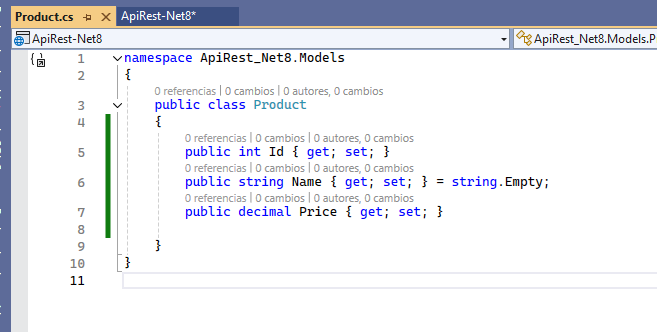
{

public int Id { get; set; }

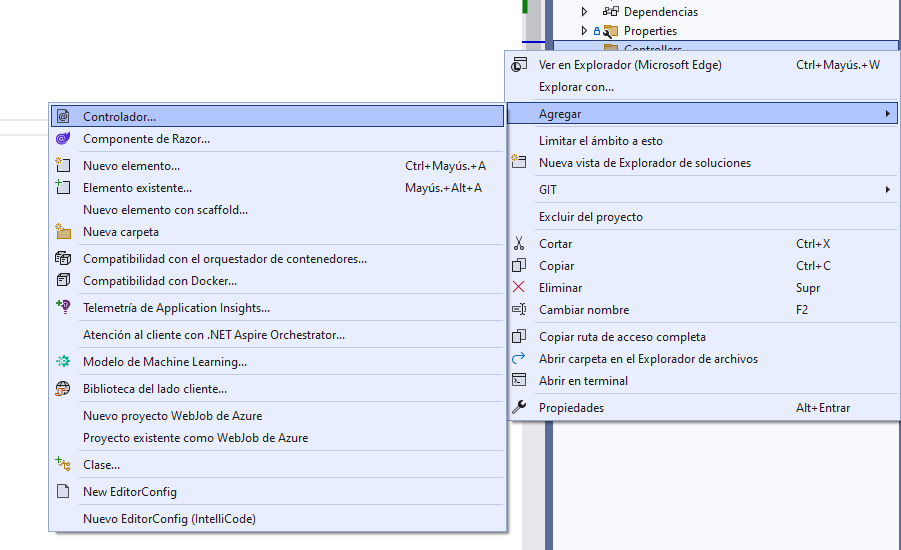
public string Name { get; set; } = string.Empty;

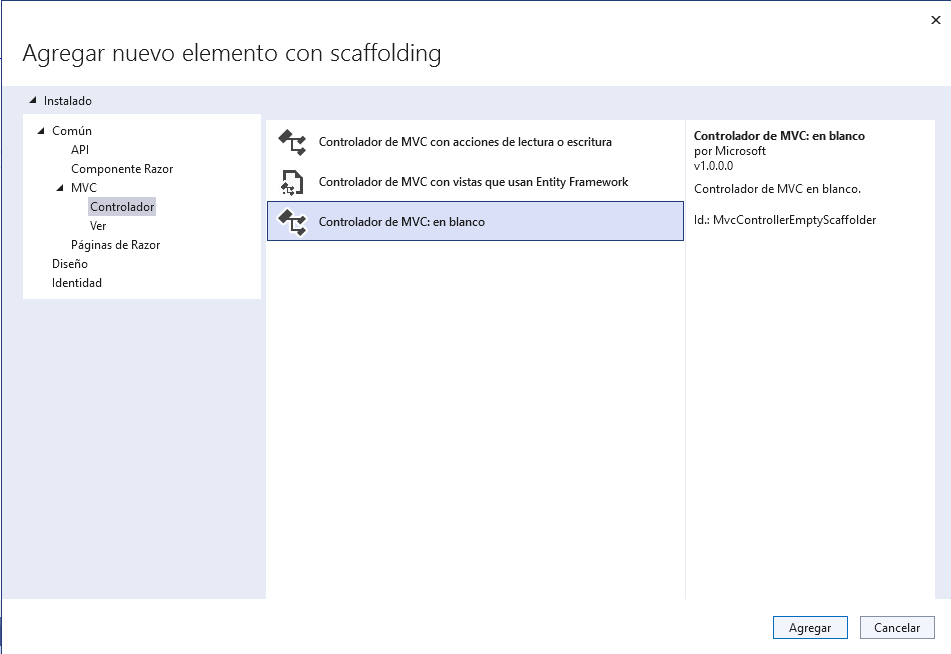
public decimal Price { get; set; }

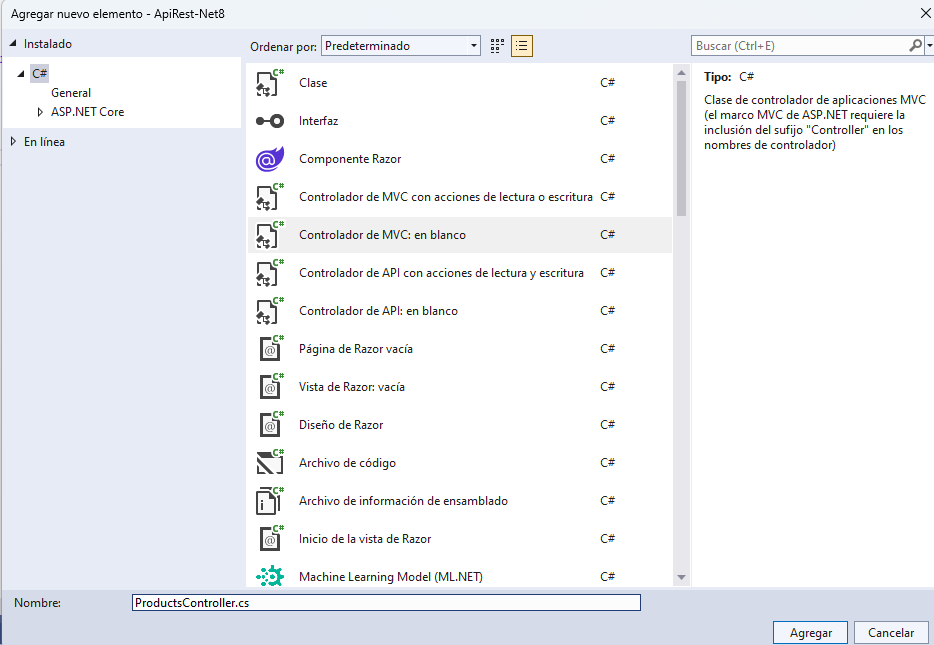
}



1. Ahora creamos el controlador ProductsController.cs







using ApiRest\_Net8.Models;

using Microsoft.AspNetCore.Mvc;

namespace ApiRest\_Net8.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class ProductsController : ControllerBase

{

private static List<Product> Products = new List<Product>

{

new Product { Id = 1, Name = "Product1", Price = 9.99M },

new Product { Id = 2, Name = "Product2", Price = 19.99M }

};

[HttpGet]

public ActionResult<IEnumerable<Product>> GetAll()

{

return Ok(Products);

}

[HttpGet("{id}")]

public ActionResult<Product> GetById(int id)

{

var product = Products.FirstOrDefault(p => p.Id == id);

if (product == null)

{

return NotFound();

}

return Ok(product);

}

[HttpPost]

public ActionResult<Product> Create(Product product)

{

product.Id = Products.Count + 1;

Products.Add(product);

return CreatedAtAction(nameof(GetById), new { id = product.Id }, product);

}

[HttpPut("{id}")]

public IActionResult Update(int id, Product updatedProduct)

{

var product = Products.FirstOrDefault(p => p.Id == id);

if (product == null)

{

return NotFound();

}

product.Name = updatedProduct.Name;

product.Price = updatedProduct.Price;

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var product = Products.FirstOrDefault(p => p.Id == id);

if (product == null)

{

return NotFound();

}

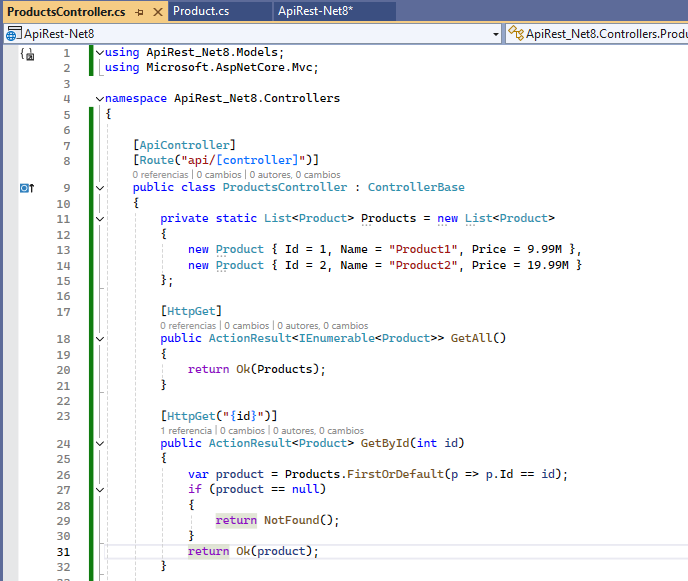
Products.Remove(product);

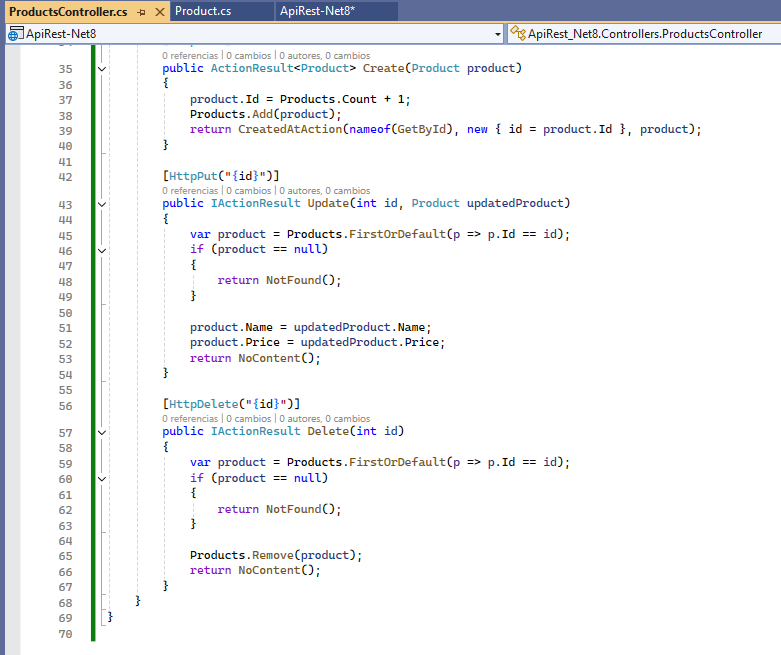
return NoContent();

}

}

}

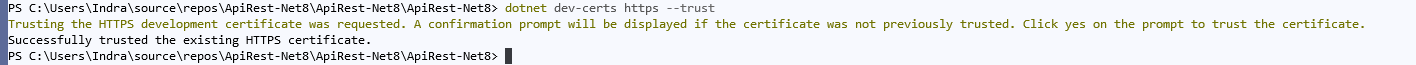




Este controlador define las rutas y los métodos para manejar las operaciones CRUD:

* GET /api/products para obtener todos los productos.
* GET /api/products/{id} para obtener un producto por ID.
* POST /api/products para crear un nuevo producto.
* PUT /api/products/{id} para actualizar un producto existente.
* DELETE /api/products/{id} para eliminar un producto.

1. Procedemos a agregar un certificado ssl (Este paso lo omitimos):



1. Verificamos todos los perfiles que tiene el archivo launchSettings.json:

{

"$schema": "http://json.schemastore.org/launchsettings.json",

"iisSettings": {

"windowsAuthentication": false,

"anonymousAuthentication": true,

"iisExpress": {

"applicationUrl": "http://localhost:14741",

"sslPort": 44317

}

},

"profiles": {

"http": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"launchUrl": "swagger",

"applicationUrl": "http://localhost:5126",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"https": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": true,

"launchUrl": "swagger",

"applicationUrl": "https://localhost:7035;http://localhost:5126",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"launchUrl": "swagger",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

1. Modificamos el archivo Program.cs ya que cuando se creó se cargó información de guía:

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(); // Agrega soporte para controladores

// Add services to the container.

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

//app.UseAuthorization();

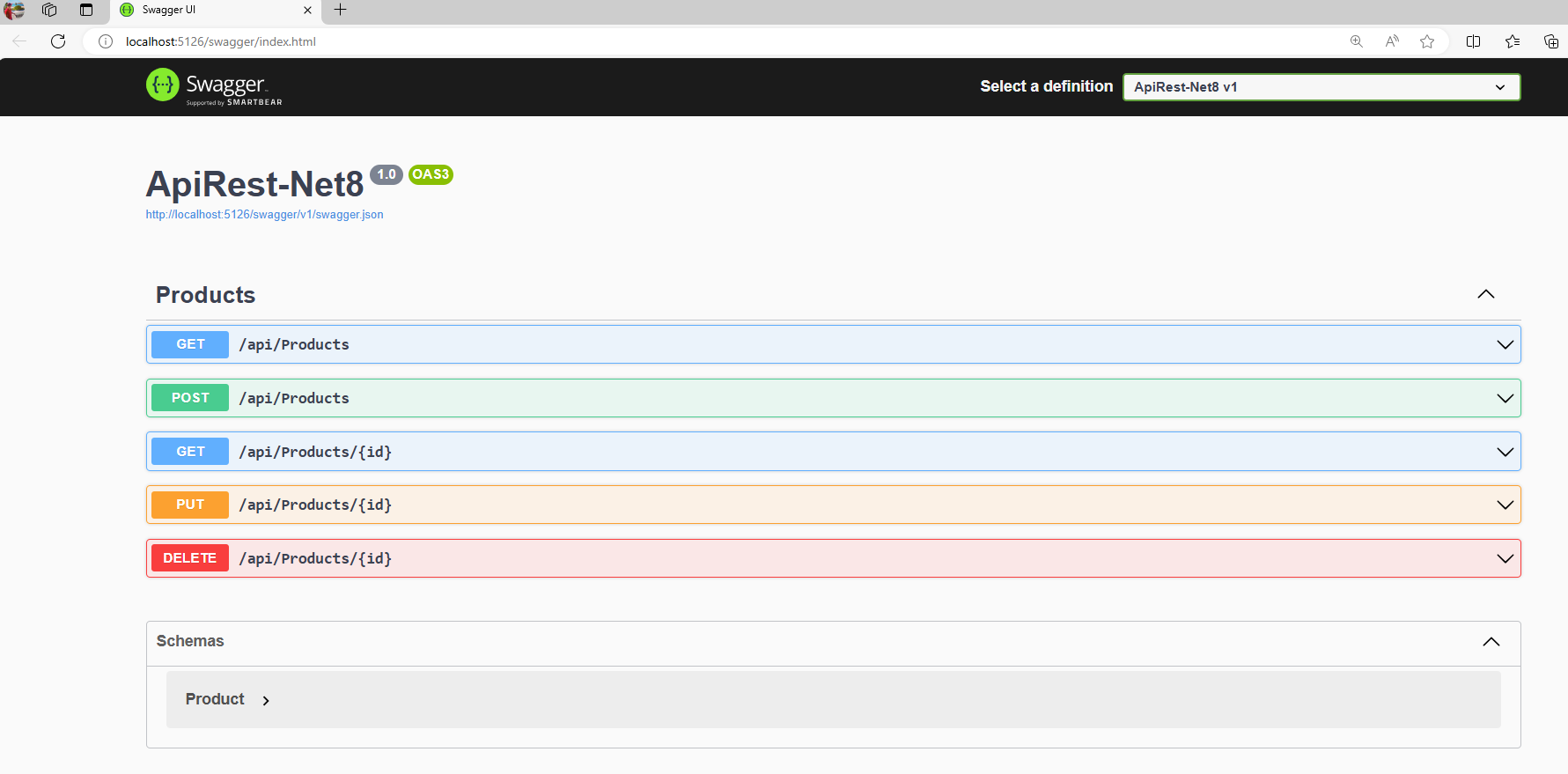
app.MapControllers(); // Mapea los controladores a las rutas

app.Run();

1. Ahora procedemos a ejecutar la aplicación:

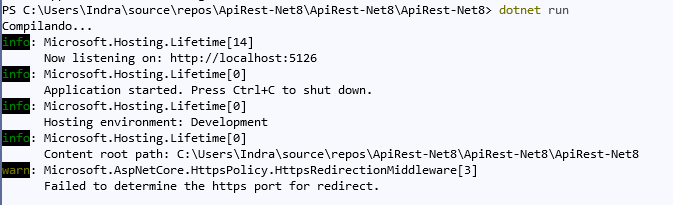


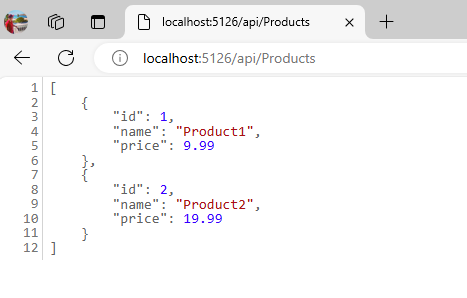
1. Observamos que cargó el swagger y muestra los respectivos endPoints que tiene el controlador Products

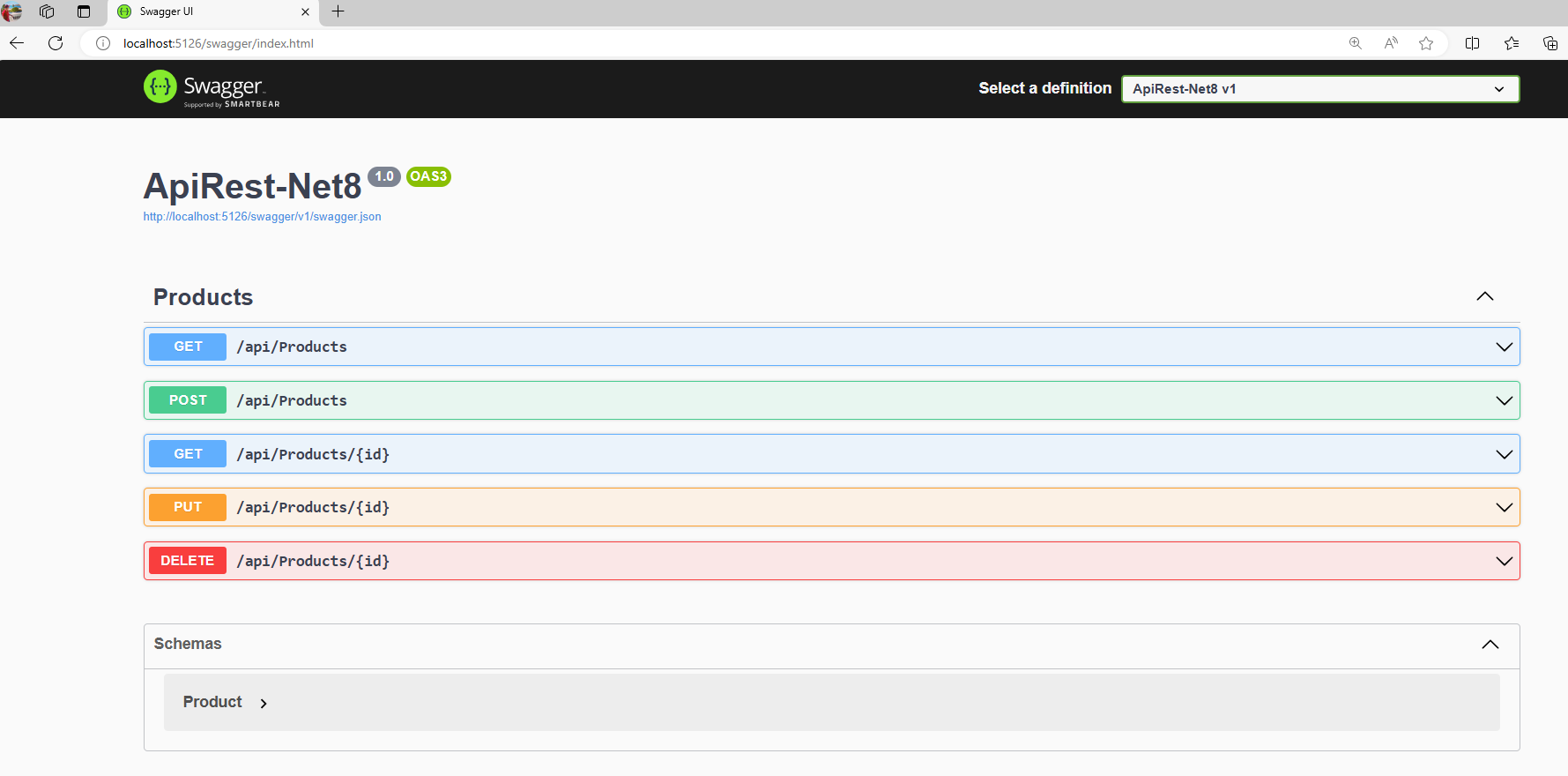


* También podemos ejecutar la aplicación por medio de la consola, con el comando:

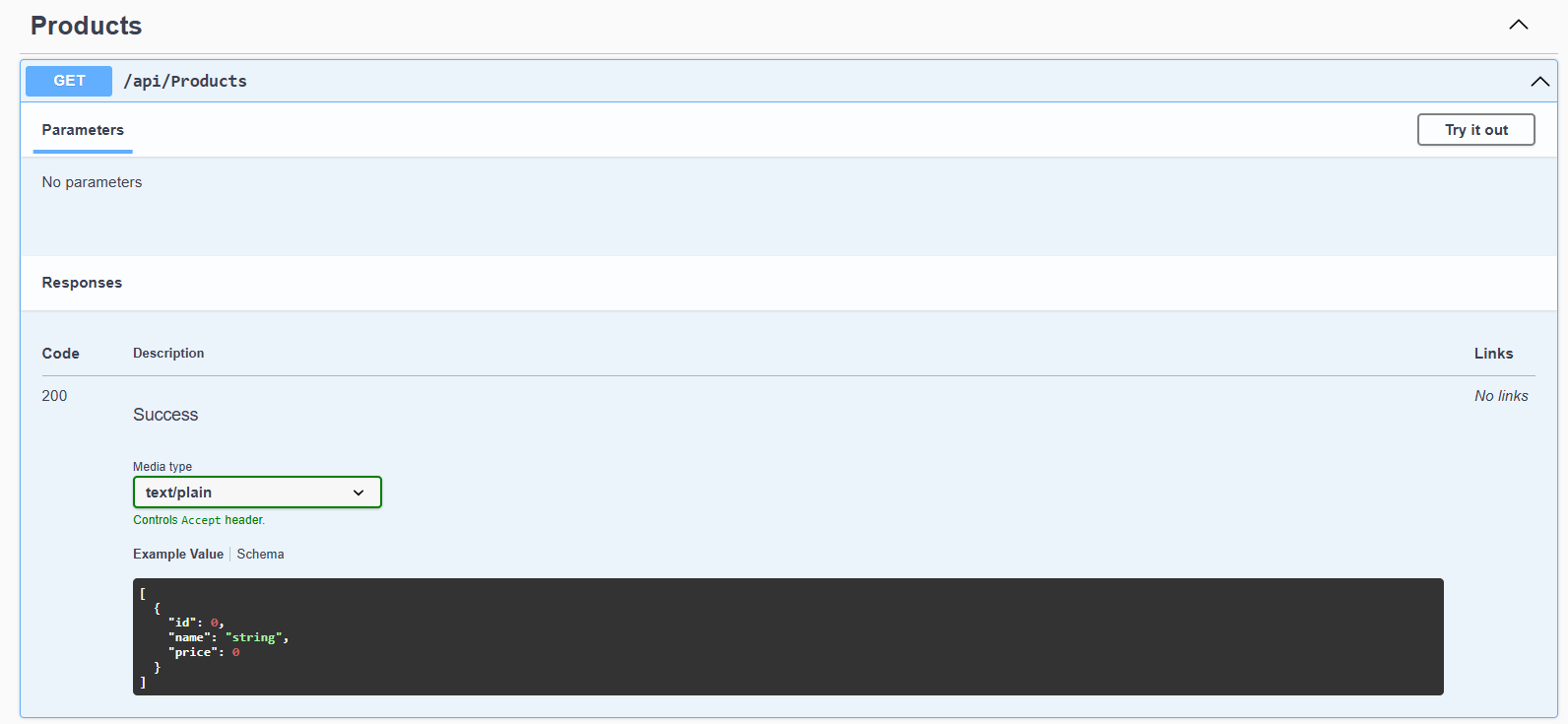
dotnet run

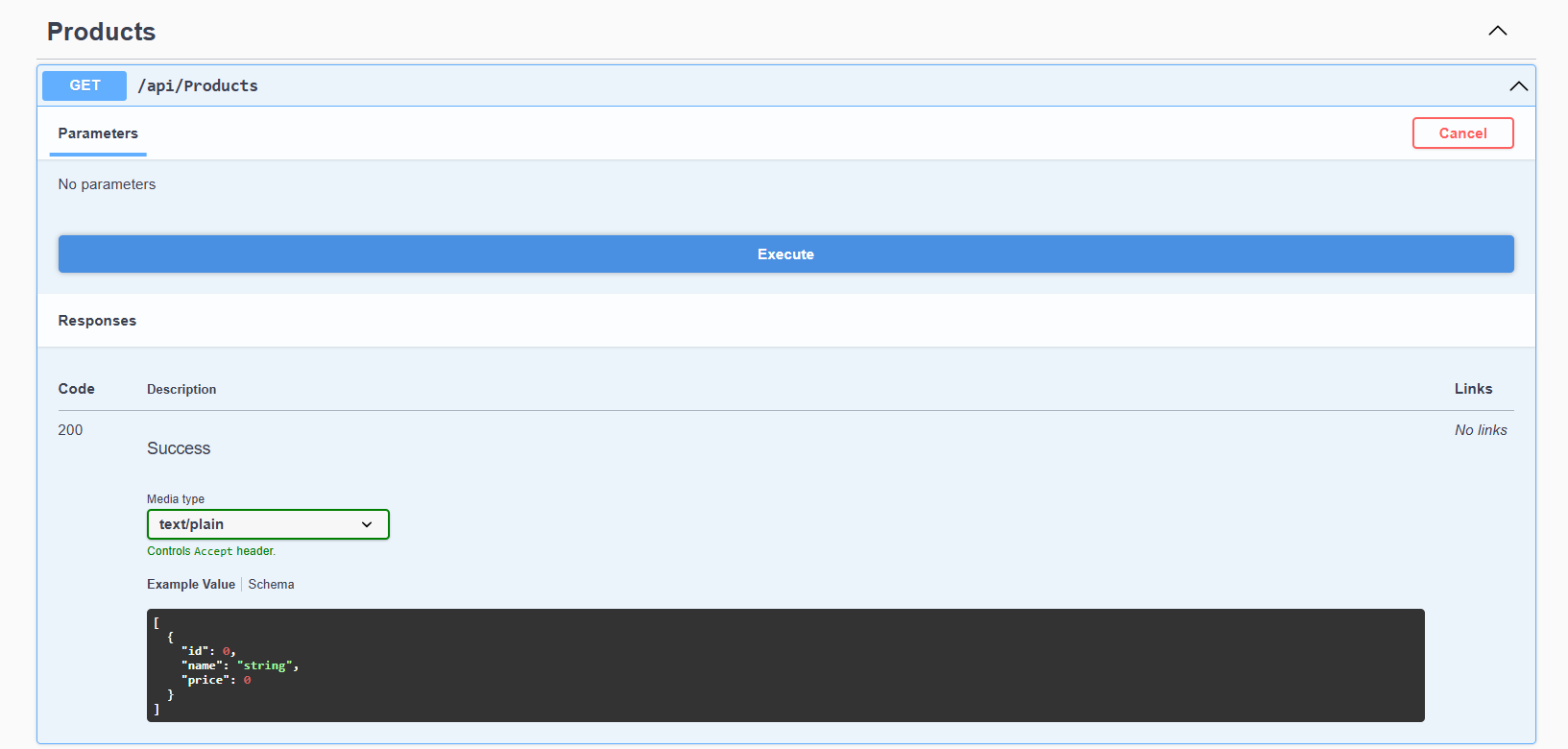


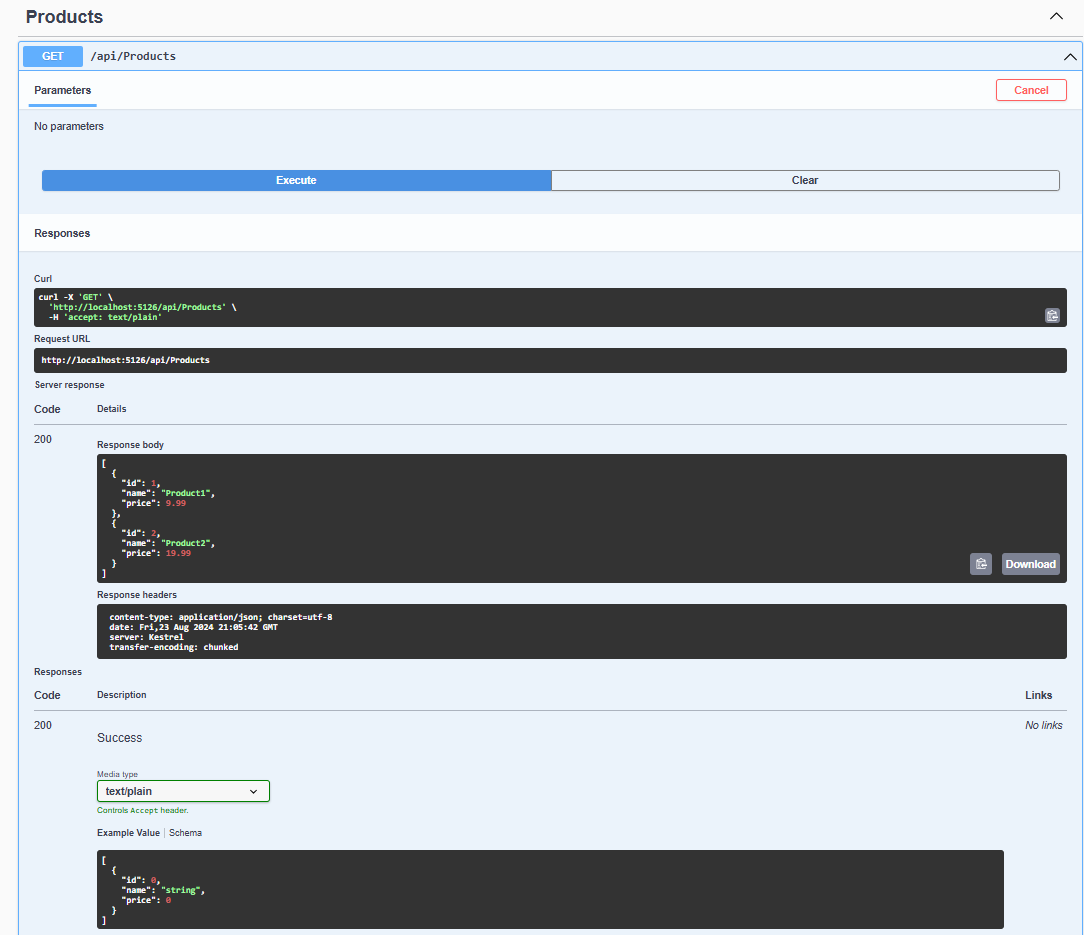




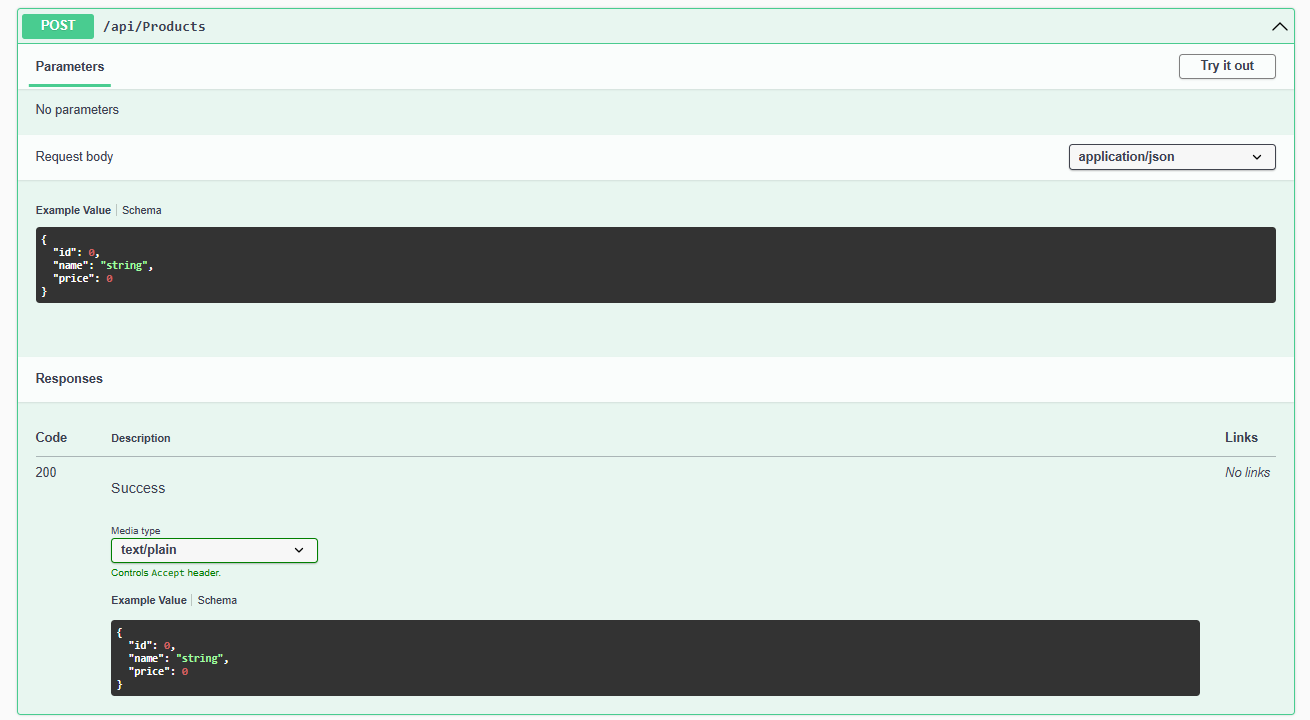
1. Probamos cada uno de los endPoints:
   * Get:

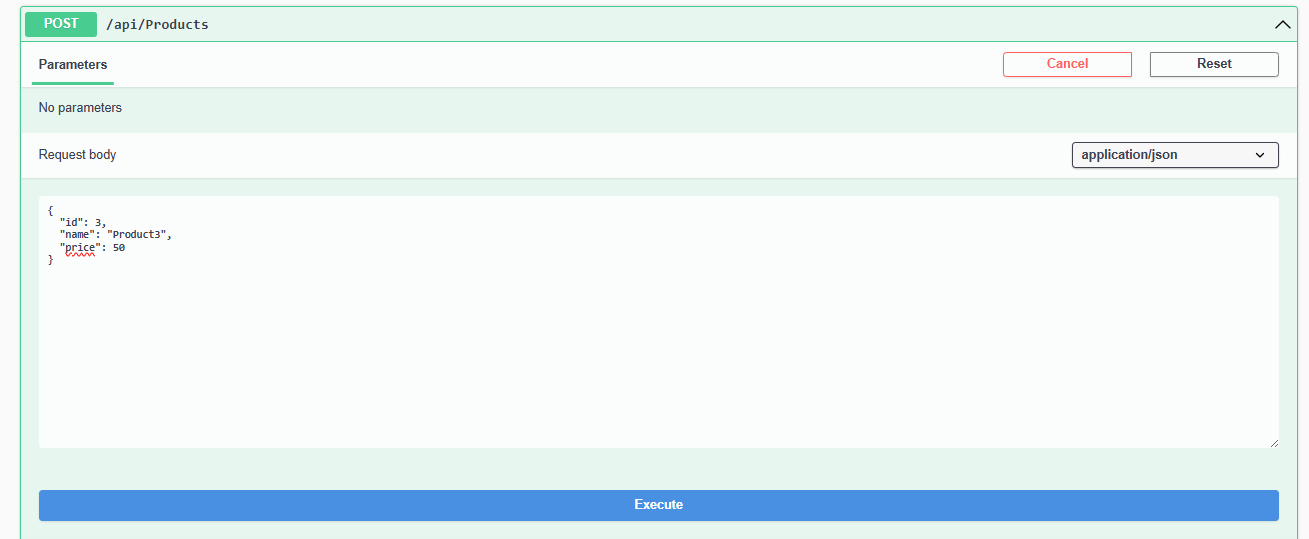


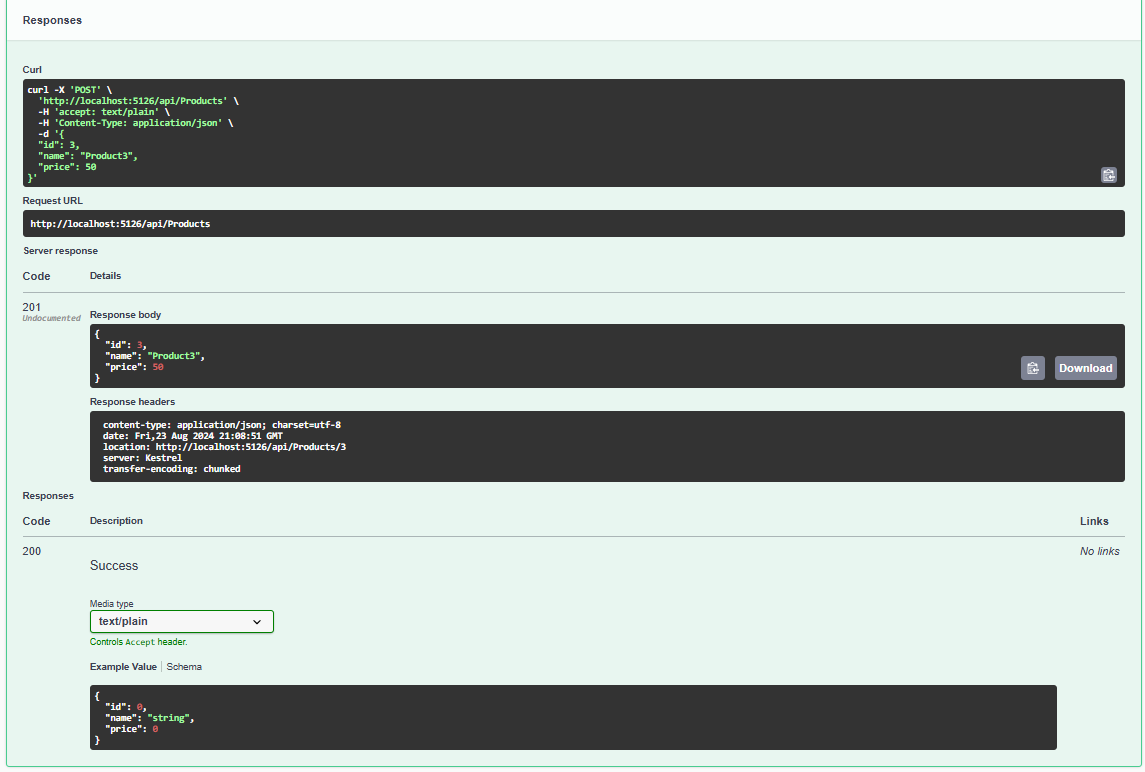




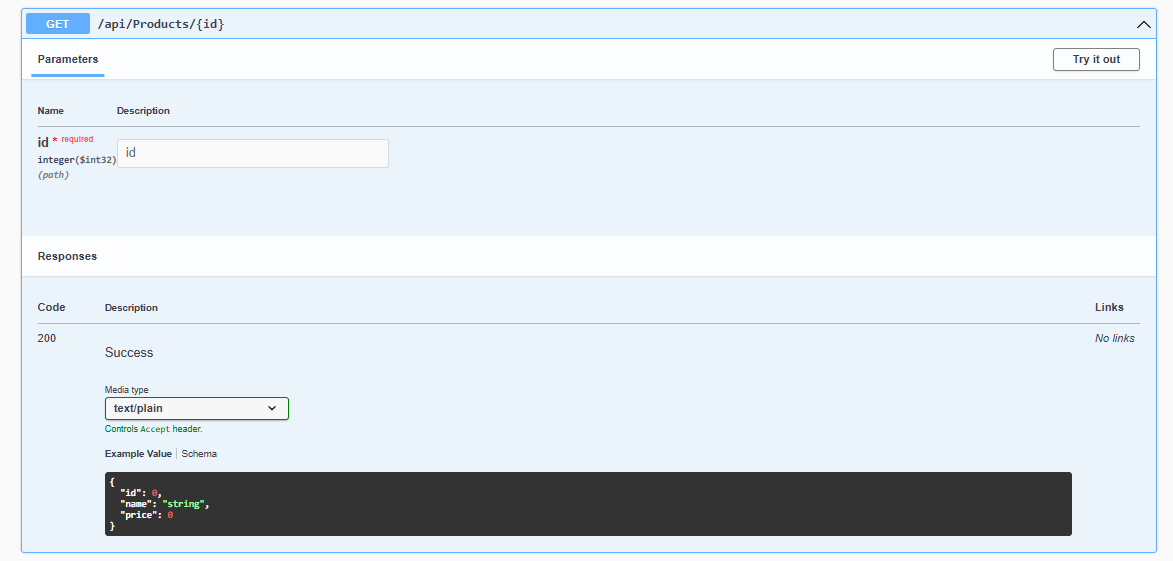
* + POST

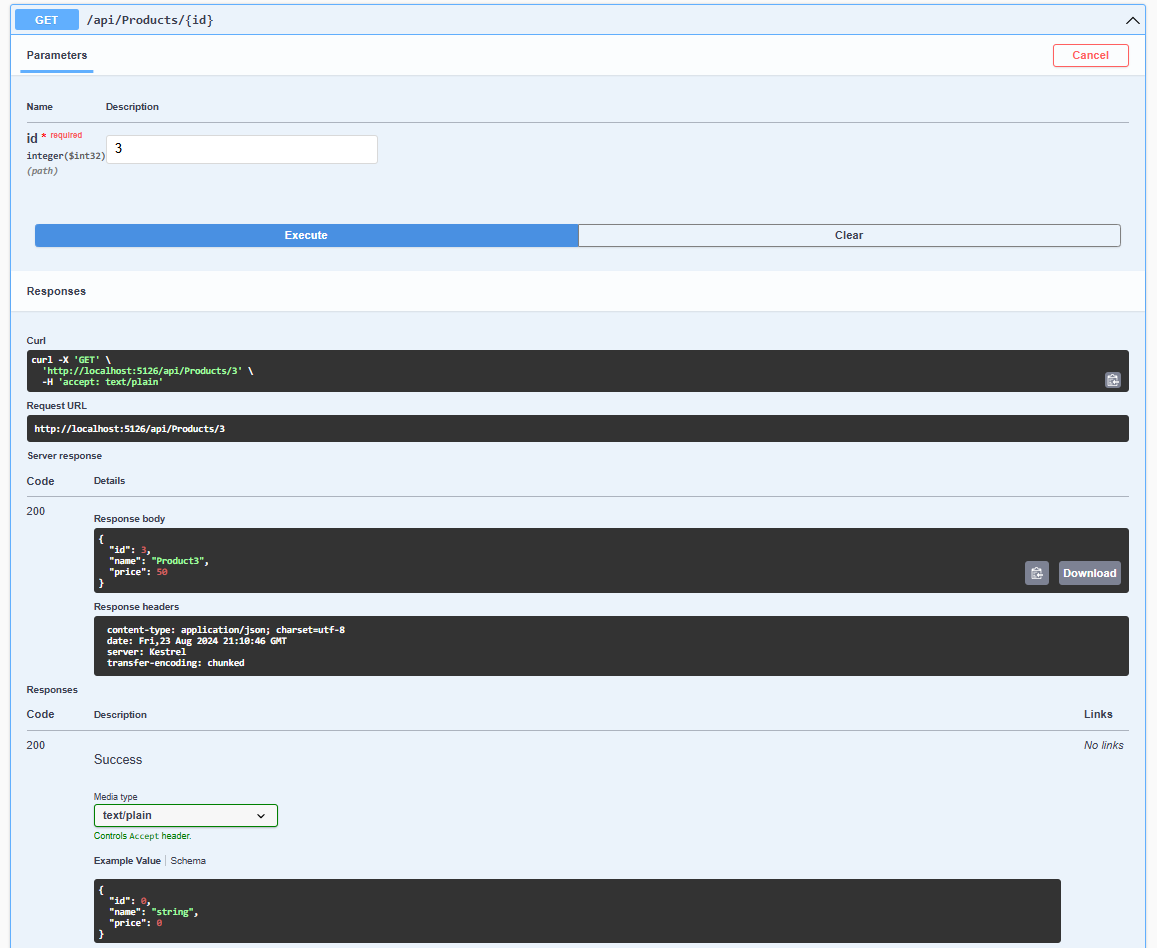




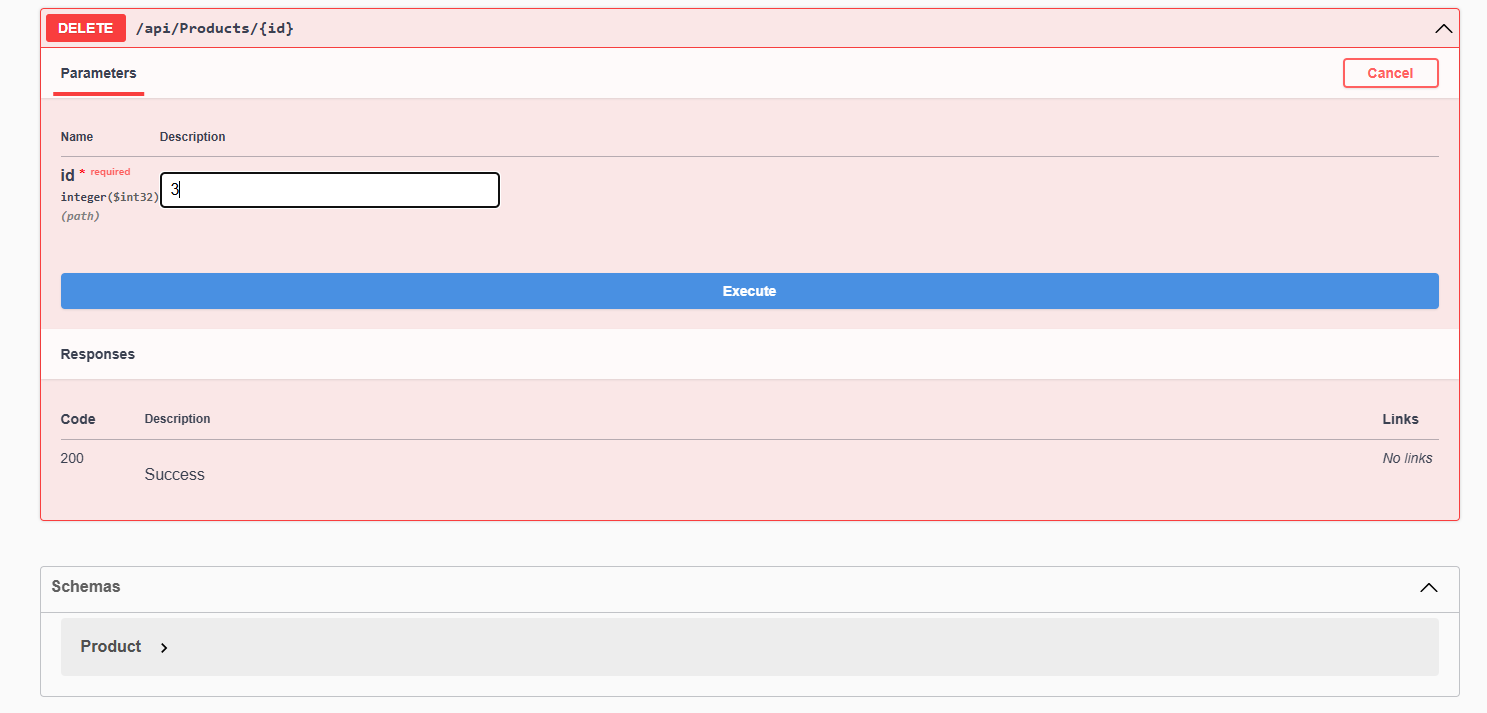


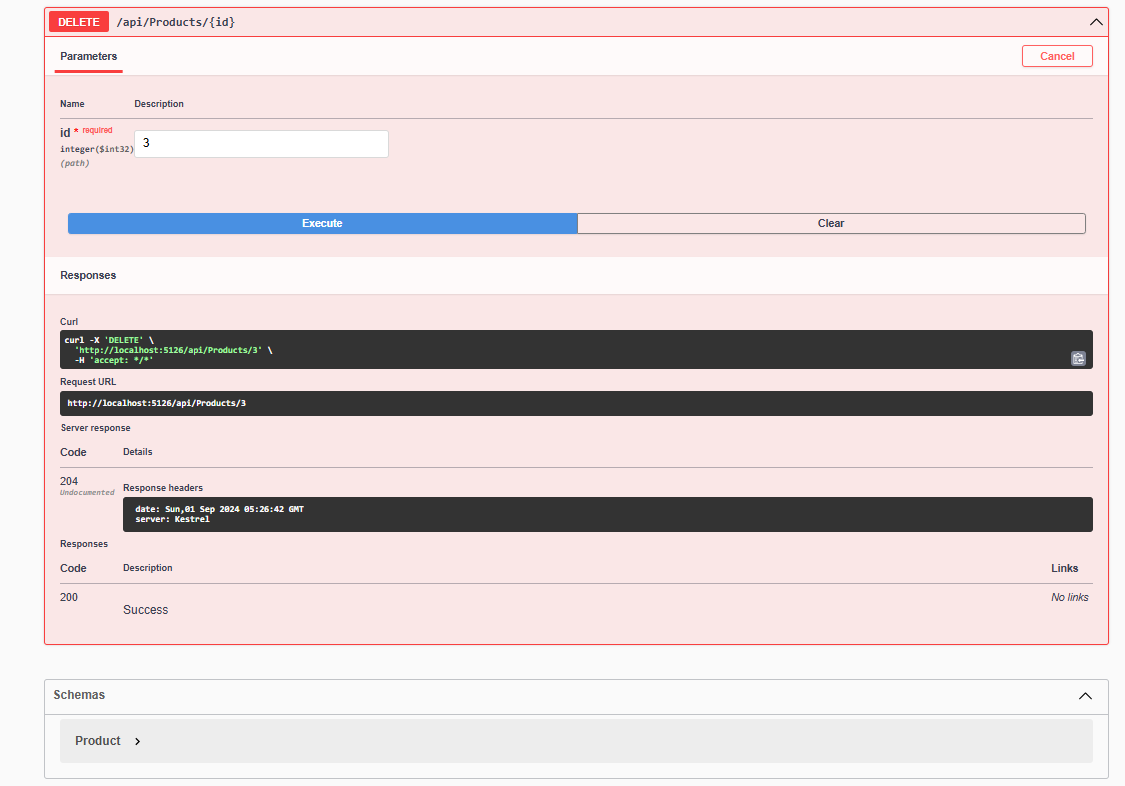
* + GET{}

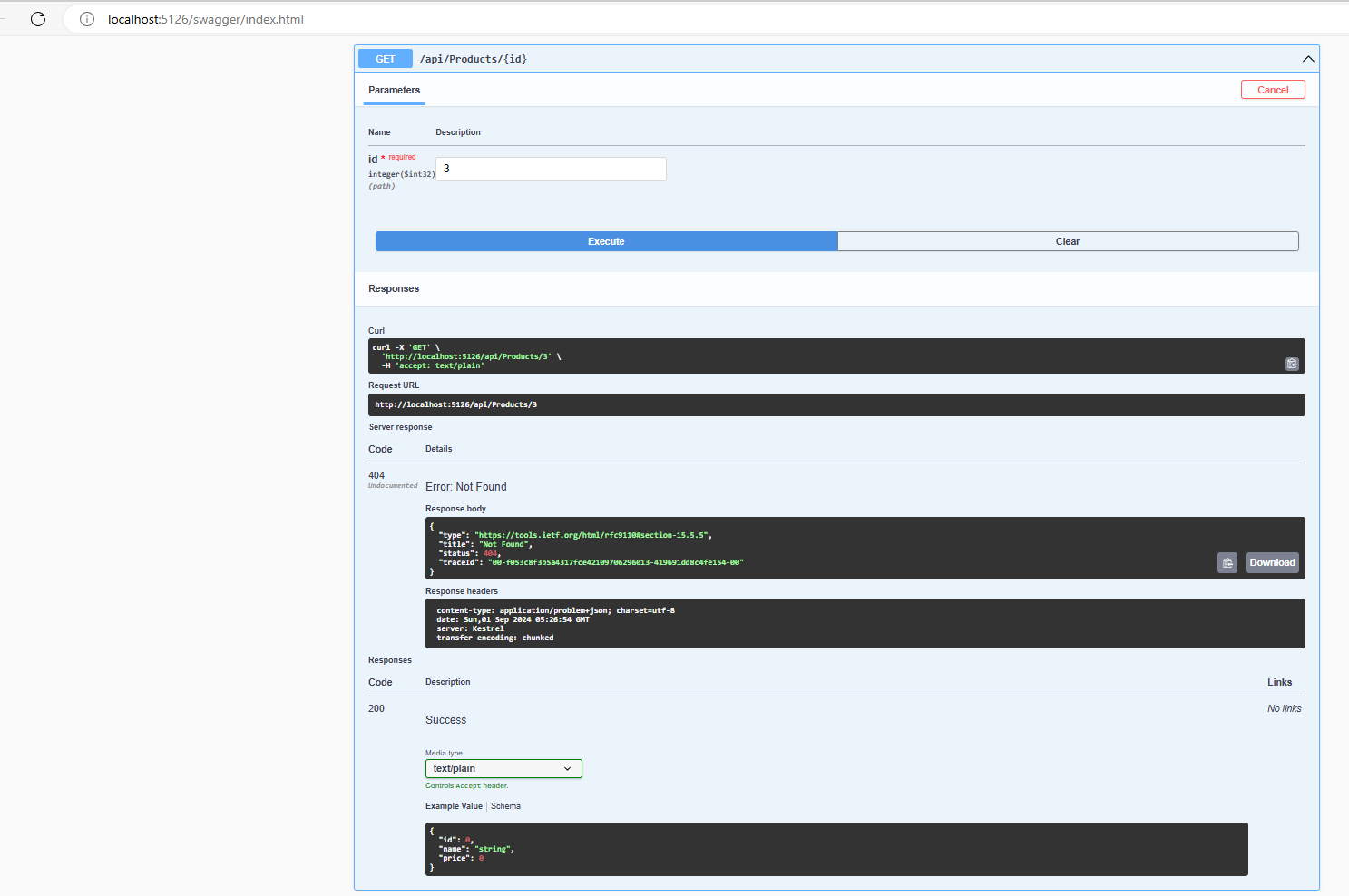




* + Delete:



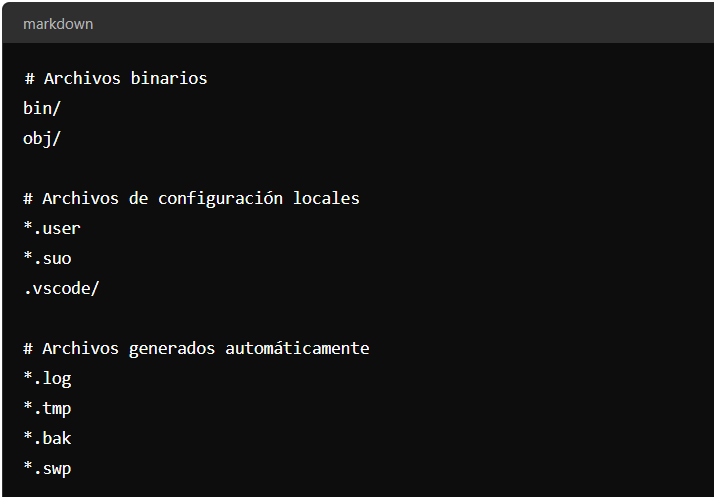




## Hacer el primer commit

Antes de realizar el primer commit es importante crear el archivo .gitignore con el fin de evitar que archivos innecesarios o sensibles se incluyan en tu repositorio. Este archivo debe listar los archivos y directorios que Git debe ignorar.

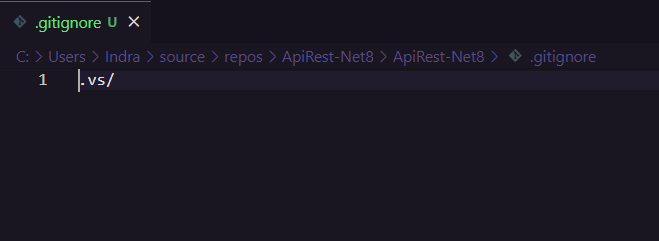
Ejemplo de un archivo (.gitignore) básico para un proyecto en .NET:



En nuestro caso, excluimos únicamente el archivo (.vs/ ), ya que al hacer el (git add .) apareció un error el cuál agregaba ese directorio:







* **Agregar Archivos al Índice**: Luego de crear y configurar tu archivo .gitignore, añade los archivos de tu proyecto al índice de Git usando el comando git add.

Este comando agrega todos los archivos en el directorio actual y subdirectorios, excluyendo aquellos especificados en (.gitignore)