Framework Core 7

Satya Karki 24 de mayo de 2023 50,1k 0 8

Introducción

Las API mínimas están diseñadas para crear API HTTP con dependencias mínimas. Las API mínimas son adecuadas para microservicios y aplicaciones que incluyen archivos, características y dependencias mínimas con ASP.NET Core. Existieron API mínimas desde .NET 6 y le ayudan a crear API fácilmente. En el momento de redactar este artículo, .NET 7 se lanzó recientemente. Ha incluido varias mejoras en el rendimiento, nuevas funciones para C#11 y F#, .Net MAUI, mejoras de ASP.NET Core/Blazor, API web y muchas más. Además, puede contener fácilmente sus proyectos .NET 7, así como configurar flujos de trabajo de CI/CD para acciones de GitHub. Además, .NET MAUI es parte de .NET 7. .NET 7 cuenta con soporte oficial de Microsoft durante solo 18 meses y está etiquetado como soporte a plazo estándar. Este artículo demuestra cómo crear API mínimas utilizando .NET 7 y Entity Framework Core 7.

En este artículo, cubriremos lo siguiente:

- Cree API mínimas usando .NET 7
- Familiarícese con las API mínimas
- Familiarícese con Entity Framework y utilice Entity Framework en una API mínima
- Crear API para la operación CRUD

Requisitos previos

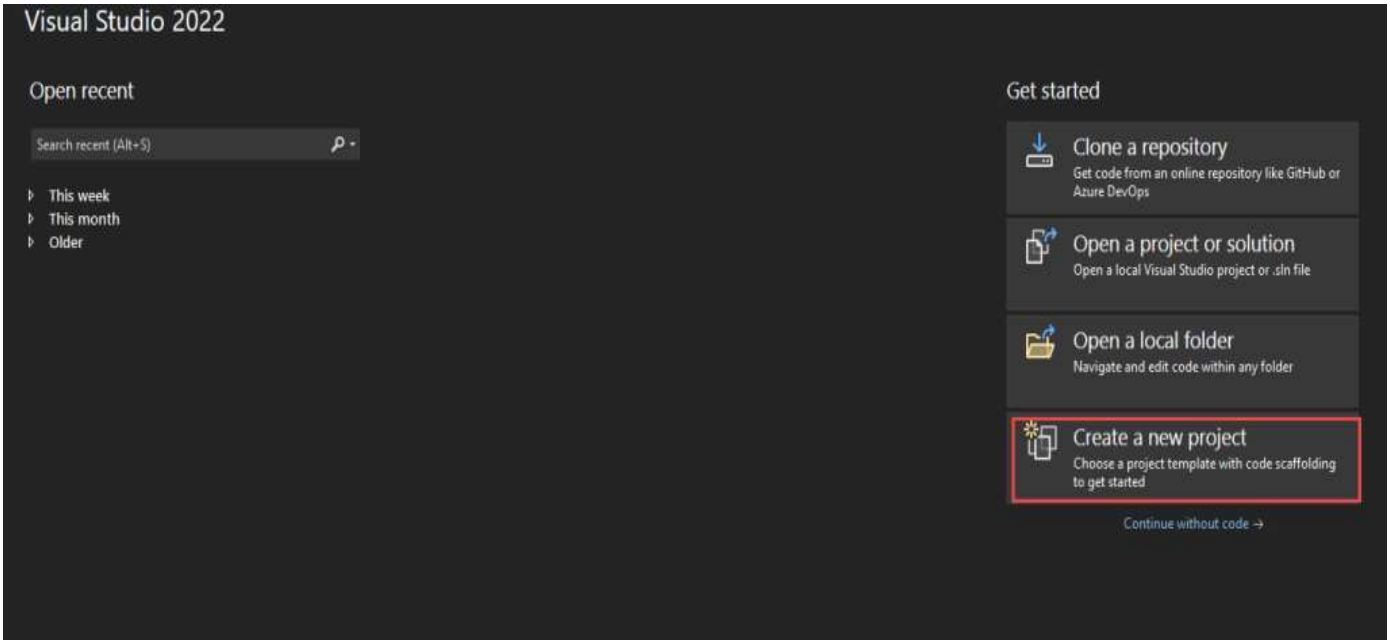
- Estudio visual 2022
- SDK de .NET 7

Puede obtener el artículo anterior Introducción a ASP.NET Core 7 y Algunas características clave de .NET 7 aquí .

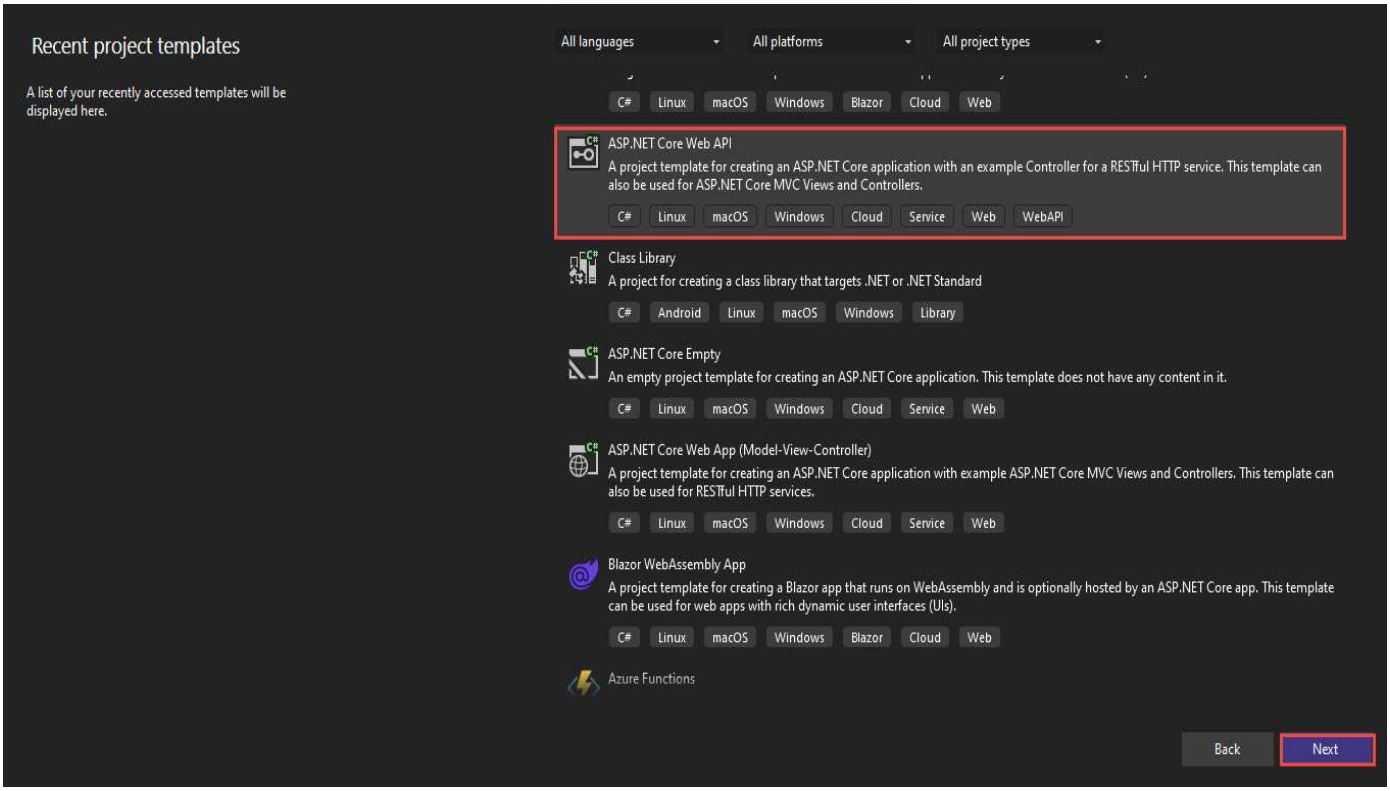
Crear proyecto API mínimo

Siga los pasos a continuación para crear una API mínima.

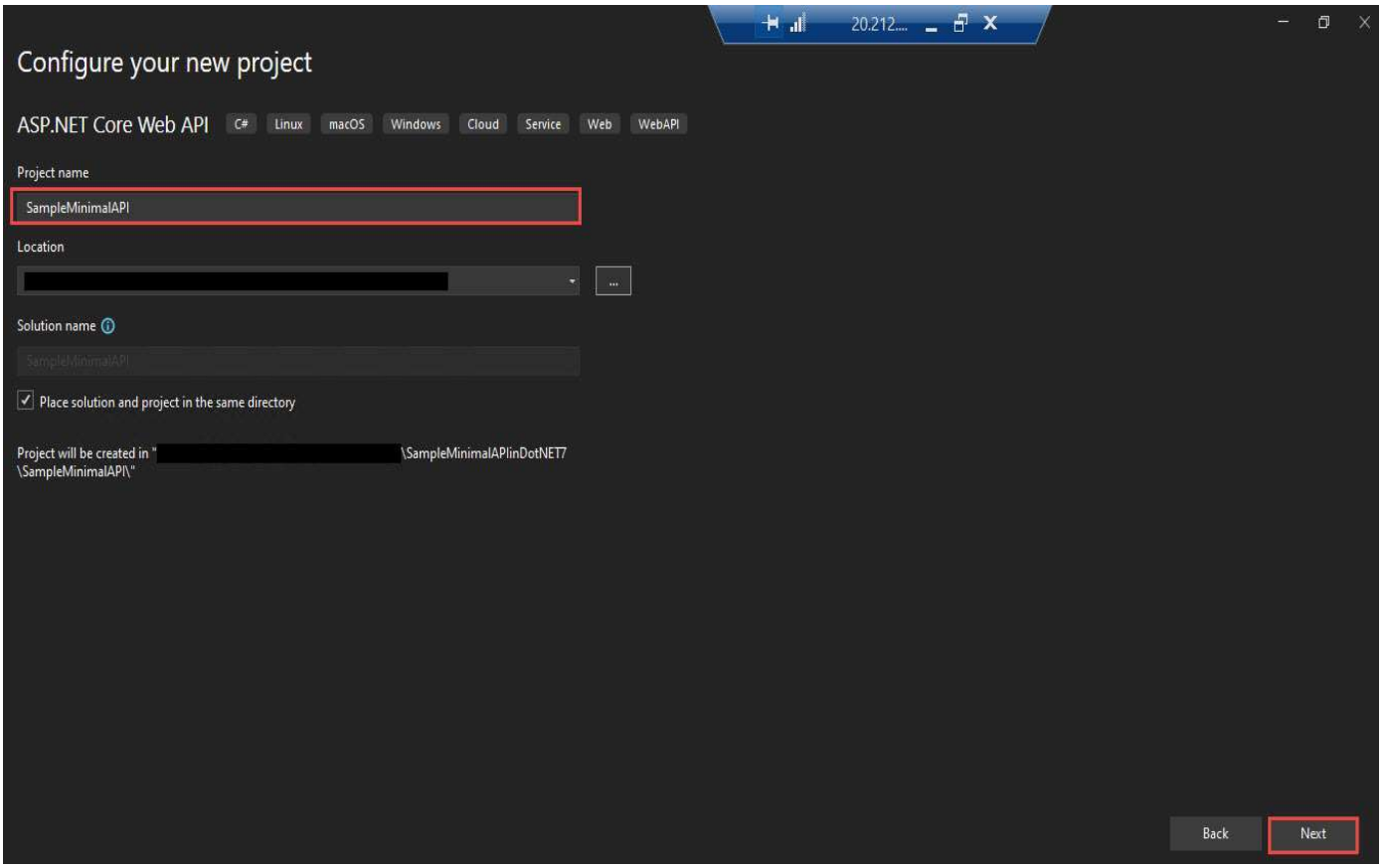
Paso 1: abra Visual Studio y haga clic en crear un nuevo proyecto.



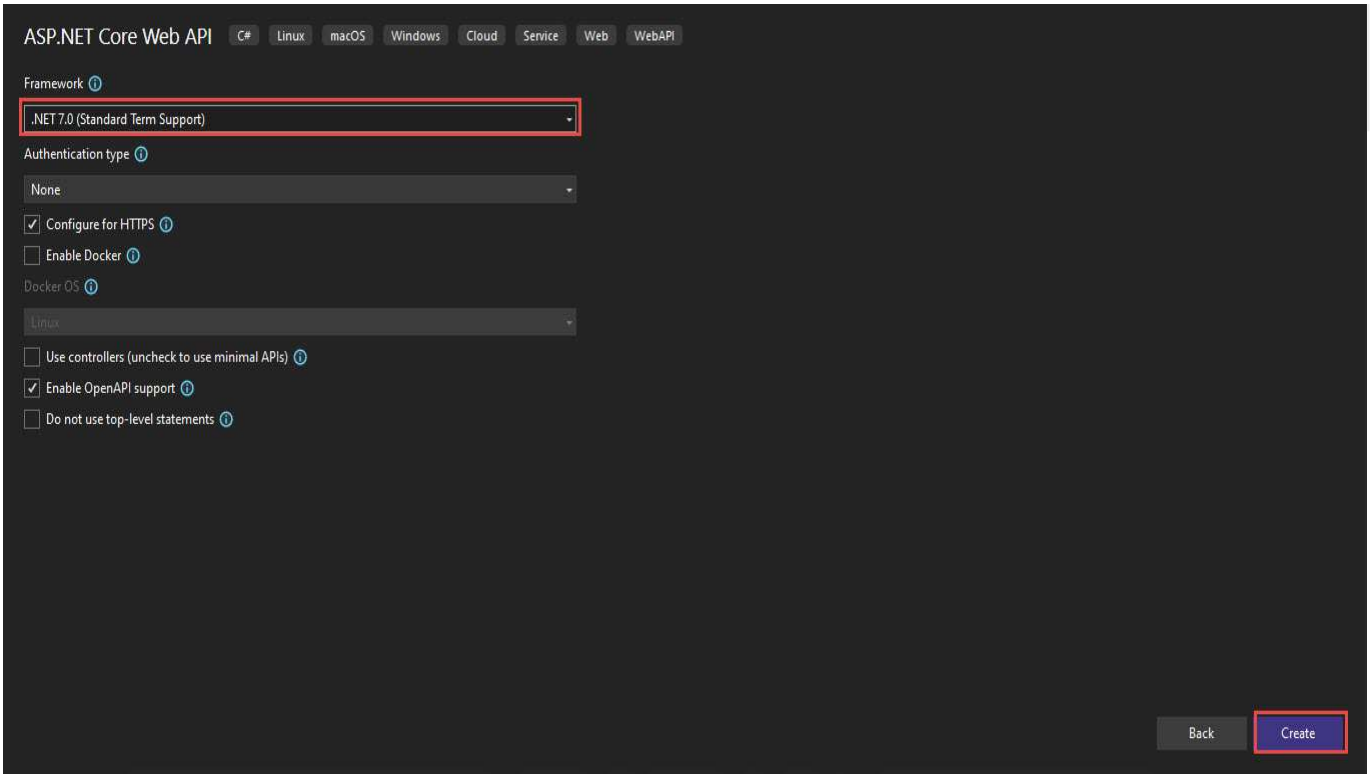
Paso 2: seleccione ASP.NET Core Web API como se muestra a continuación. Puede ver que esta plantilla viene con un ejemplo de un servicio HTTP RESTFul y también se puede usar para vistas y controladores ASP.NET Core MVC (API con controlador).



Paso 3: proporcione el nombre del proyecto y el directorio del proyecto y luego haga clic en Siguiente como se ilustra a continuación.



Paso 4: seleccione Framework: **.Net 7.0 (soporte de término estándar),** desmarque **Usar controlador para usar API mínimas.** desmarque **No utilizar declaraciones de nivel superio** Seleccione otros como se ilustra en la imagen a continuación.



Paso 5: Explorar el código Program.cs predeterminado

Ahora, exploremos el código de program.cs.

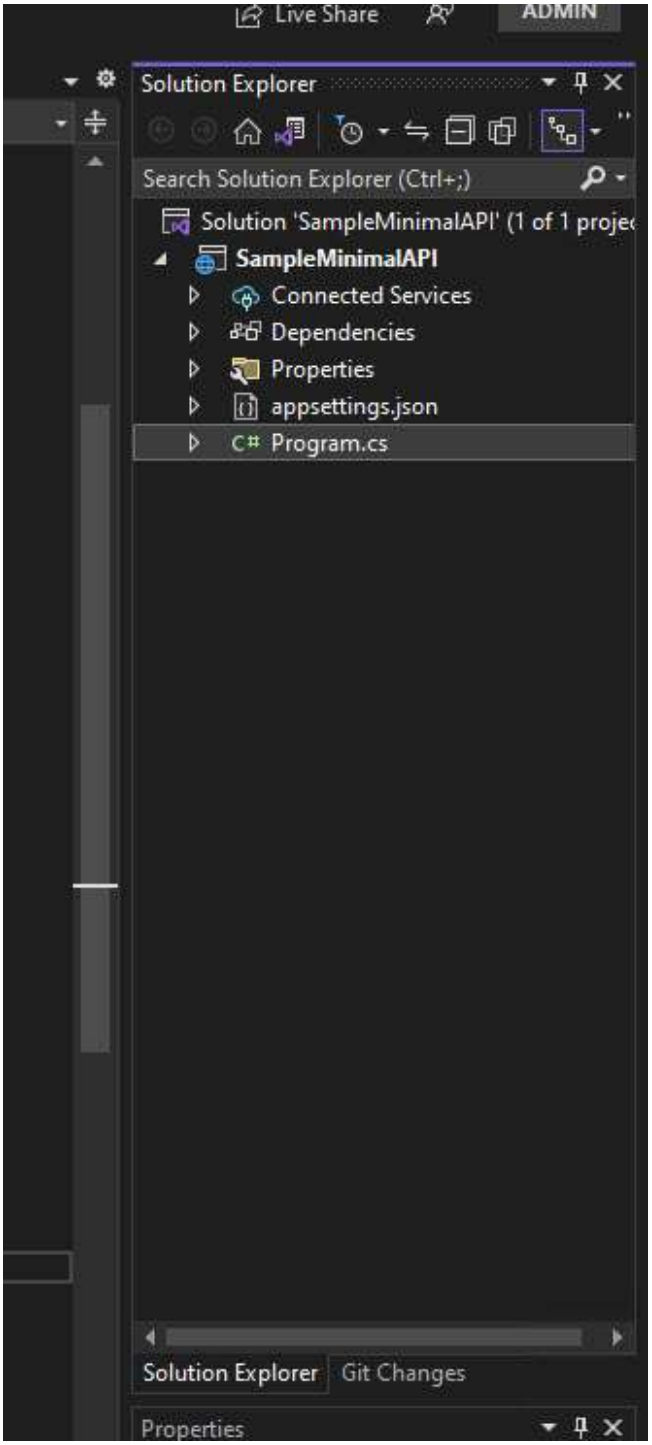
A continuación se muestra el código predeterminado de Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetc
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();

var summaries = new[]
{
```

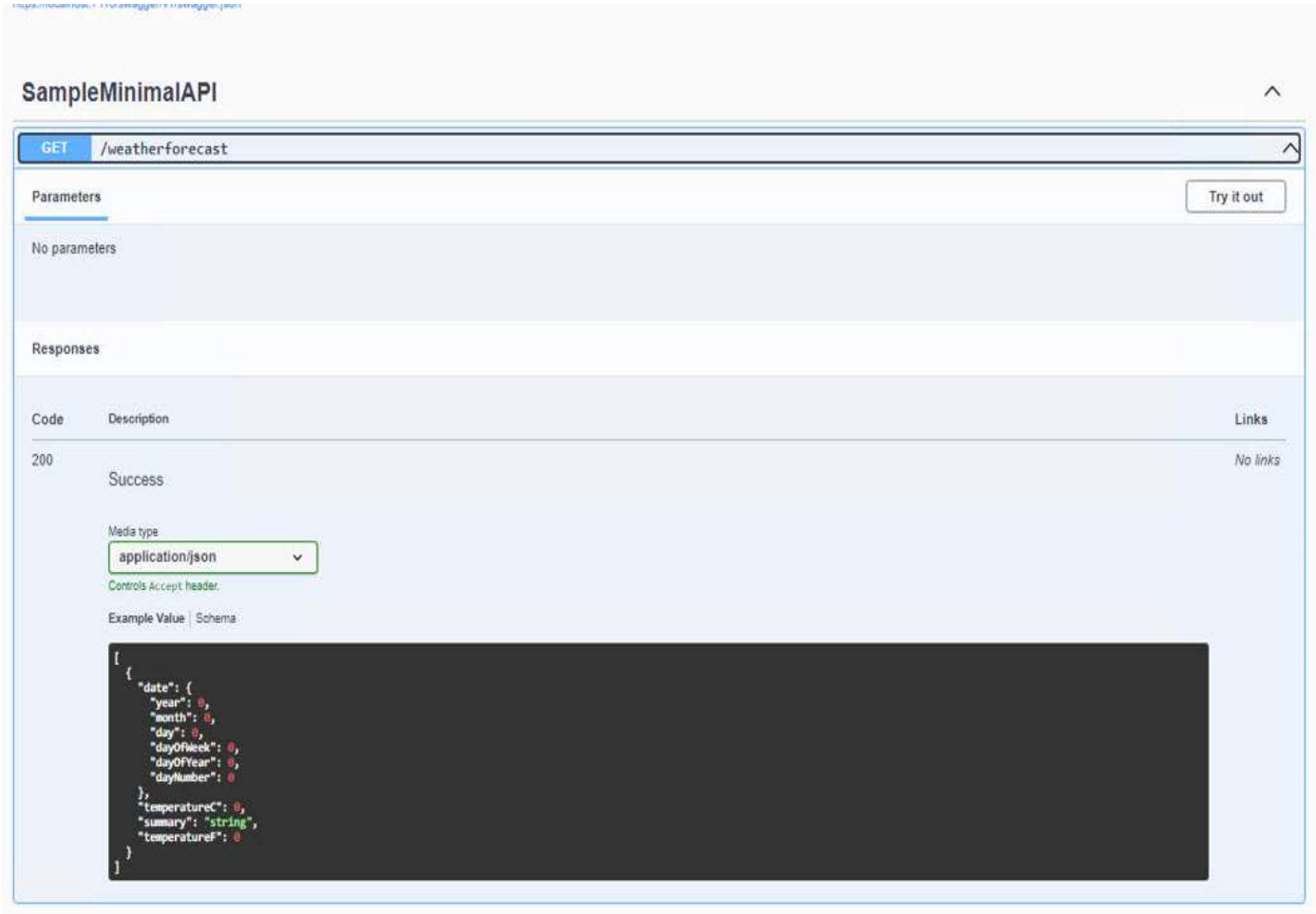
```
24
25  app.MapGet("/weatherforecast", () =>
26  {
27      var forecast = Enumerable.Range(1, 5).Select(index =>
28          new WeatherForecast
29          (
30              DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
31              Random.Shared.Next(-20, 55),
32              summaries[Random.Shared.Next(summaries.Length)]
33          ))
34          .ToArray();
35      return forecast;
36  })
37  .WithName("GetWeatherForecast")
38  .WithOpenApi();
39
40  app.Run();
41
42  internal record WeatherForecast(DateOnly Date, int TemperatureC, string?
43  {
44      public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
    }
```

- El método MapGet en las API mínimas se puede utilizar para manejar la solicitud HTTP Get. Lo que significa que la API de pronóstico del tiempo es un punto final HTTP Get: “/weatherforecast”.
- El pronóstico del tiempo predeterminado muestra la fecha, la temperatura, el resumen y la temperatura.

Proyecto predeterminado Estructura del proyecto API mínima.



Ahora, ejecutemos el proyecto predeterminado. A continuación se muestra la vista de una página de Swagger.



Puede ver en la imagen de arriba que Minimal API viene con un pronóstico del tiempo predeterminado y documentación Swagger para Web API igual que ASP.NET Core Web API con controlador.

Hasta ahora solo hemos creado y explorado la API mínima predeterminada. Ahora, pasaremos a crear nuevas API.

Agregar/crear nuevas API

Paso 1: agregar Entity Framework

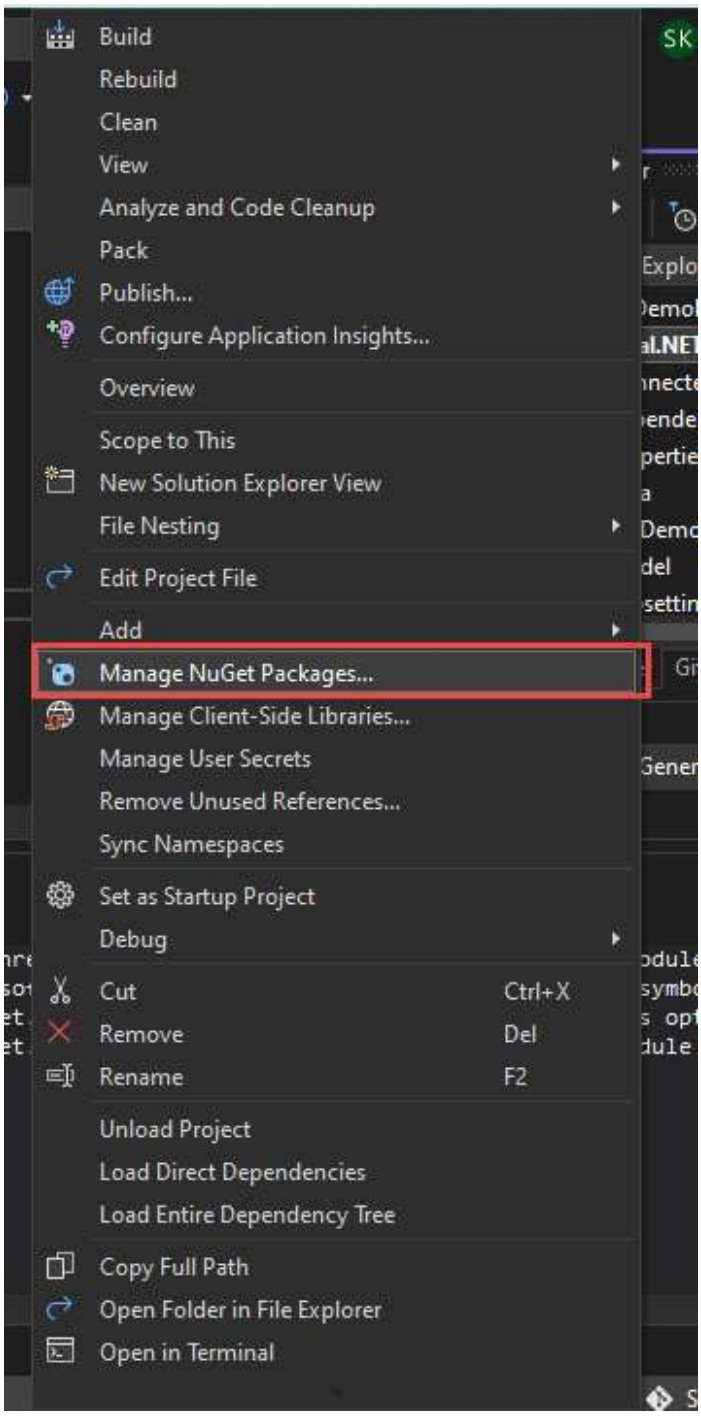
Núcleo de Entity Framework: el núcleo de Entity Framework es parte de Entity Framework. Después de Entity Framework 6.x, está etiquetado como núcleo de Entity Framework. El núcleo de Entity Framework es una versión de código abierto, liviana, extensible y multiplataforma de Entity Framework. Funciona como un asignador relacional de objetos (ORM) que ayuda al desarrollador a trabajar fácilmente con bases de datos y objetos .Net. Con el uso de EF, los desarrolladores no necesitan escribir la mayor parte del código de acceso a datos.

Base de datos en memoria : EF core tiene la capacidad de almacenar y recuperar datos en la memoria. La base de datos en memoria se utiliza para probar la aplicación. No almacena datos en la base de datos física, solo almacena datos en la memoria y es volátil. Podemos usarlo solo para probar si nuestra API funciona perfectamente o no en este proyecto de demostración de API mínima.

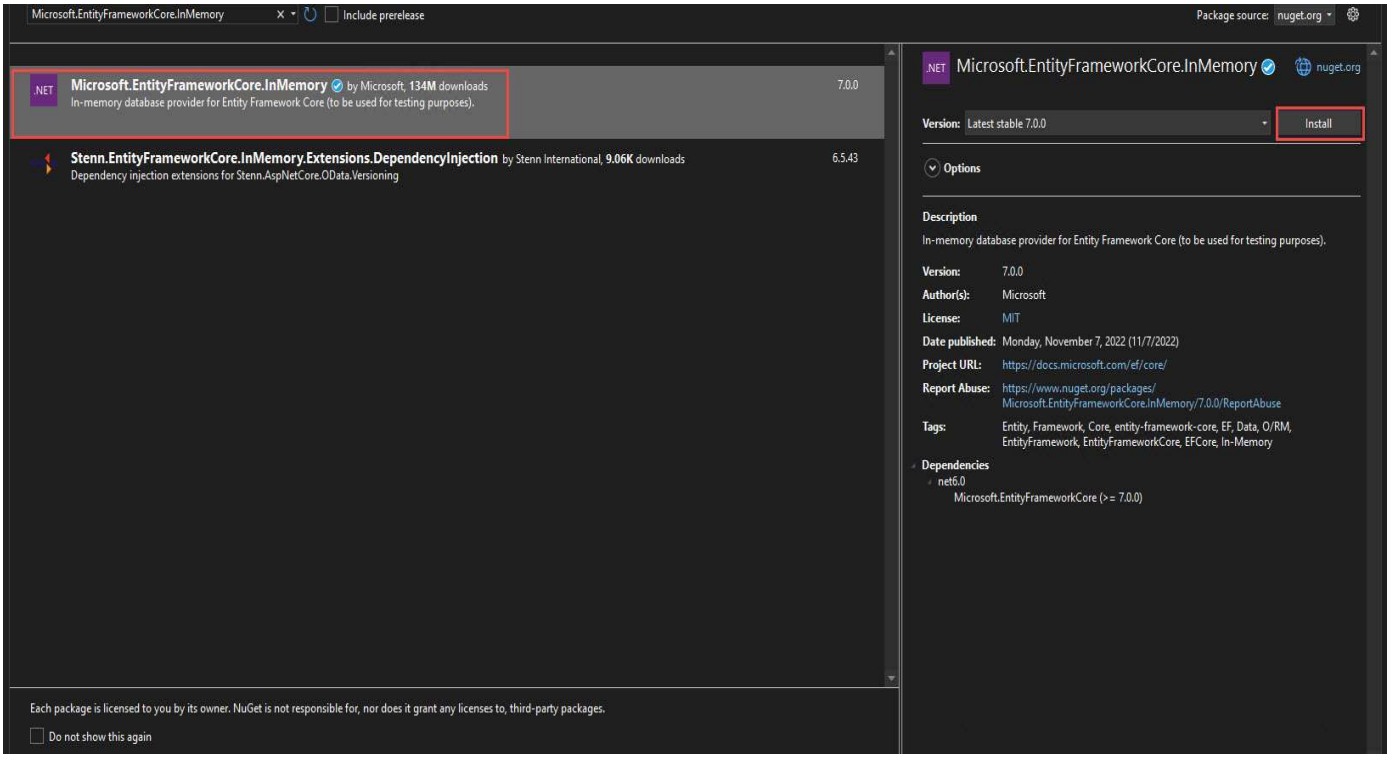
- **Microsoft.EntityFrameworkCore.InMemory**
- **Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore**

Ahora, pasemos a agregar EF Core.

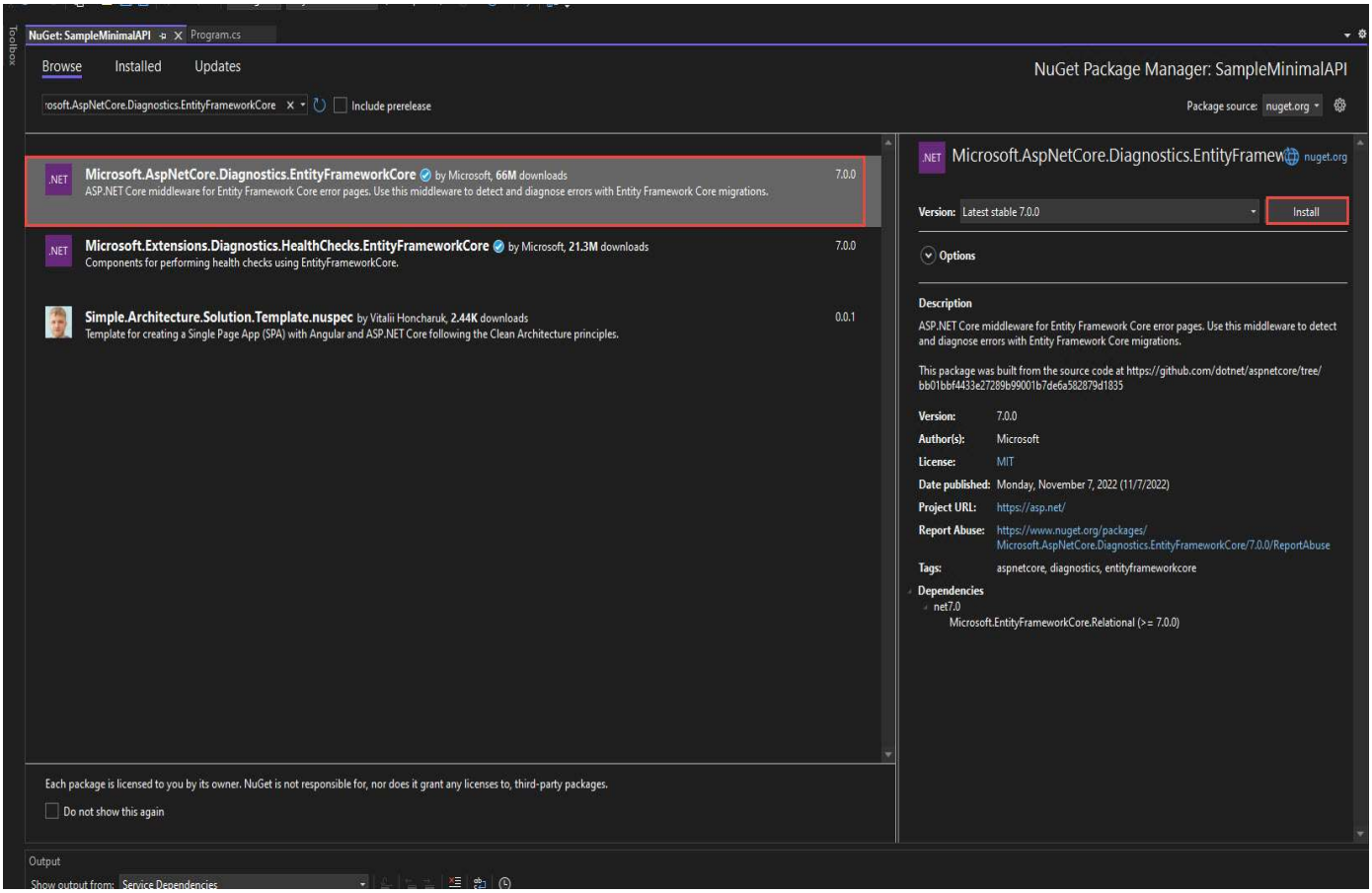
Haga clic derecho en Proyecto y vaya a Administrar paquete NuGet para encontrar la solución como se muestra a continuación.



Luego vaya a la pestaña Explorar. En el cuadro de búsqueda, ingrese **Microsoft.EntityFrameworkCore.InMemory** , seleccione **Microsoft.EntityFrameworkCore.InMemory** y luego instálelo.



De manera similar, vaya a la pestaña Examinar -> En el cuadro de búsqueda, ingrese: **Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore** , luego selecciónelo e instálelo.



Paso 2: agregar la clase de modelo.

Escriba el siguiente código en su clase Student.cs:

```
public class Student
{
    public int Id { get; set; }
    [Required]
    public string? Name { get; set; }
    [Required]
    public string? Email { get; set; }
    public string? Phone { get; set; }
}
```

Paso 3: agregue DbContext.

Luego creemos una carpeta Datos y agreguemos una clase para el contexto de los datos. Digamos la clase **MyDataContext** .

Código de **MyDataContext**

```
public class MyDataContext:DbContext
{
    public MyDataContext(DbContextOptions<MyDataContext> options)
   : base(options) { }

    public DbSet<Student> Students => Set<Student>();
}
```

Luego agregue Datacontext en **Program.cs** como se muestra a continuación.

```
//Add dbContext, here you can we are using In-memory database.
builder.Services.AddDbContext<MyDataContext>(opt => opt.UseInMemoryDatabase("
```

Paso 4: crear nuevas API

En Minimal API escribimos todo el código API dentro del archivo Program.cs. Ahora, vamos a agregar un punto final API básico para una operación CRUD de la clase Estudiante.

Guardar API : escriba el siguiente código para guardar los datos de los estudiantes utilizando el marco de entidad central 7 en API mínima. Aquí la API de guardado es el punto final de publicación HTTP.

```
    app.MapPost("/SaveStudent", async (Student student, MyDataContext db) =>
    {
        db.Students.Add(student);
        await db.SaveChangesAsync();
```

```
});
```

GetAll : a continuación se muestra el código del punto final de GetAll Student API. Esta es la API de obtención HTTP.

```
app.MapGet("/GetAllStudent", async (MyDataContext db) =>
    await db.Students.ToListAsync());
```

Actualización : a continuación se muestra el código de actualización para los detalles del estudiante, que es HTTP Put API.

```
app.MapPut("/UpdateStudents/{id}", async (int id, Student studentinput, M
{
    var student = await db.Students.FindAsync(id);

    if (student is null) return Results.NotFound();

    student.Name = studentinput.Name;
    student.Phone = studentinput.Phone;

    await db.SaveChangesAsync();

    return Results.NoContent();
});
```

A continuación se proporciona un código completo de Program.cs con las API Publicar, Obtener, Colocar y Eliminar para Estudiantes:

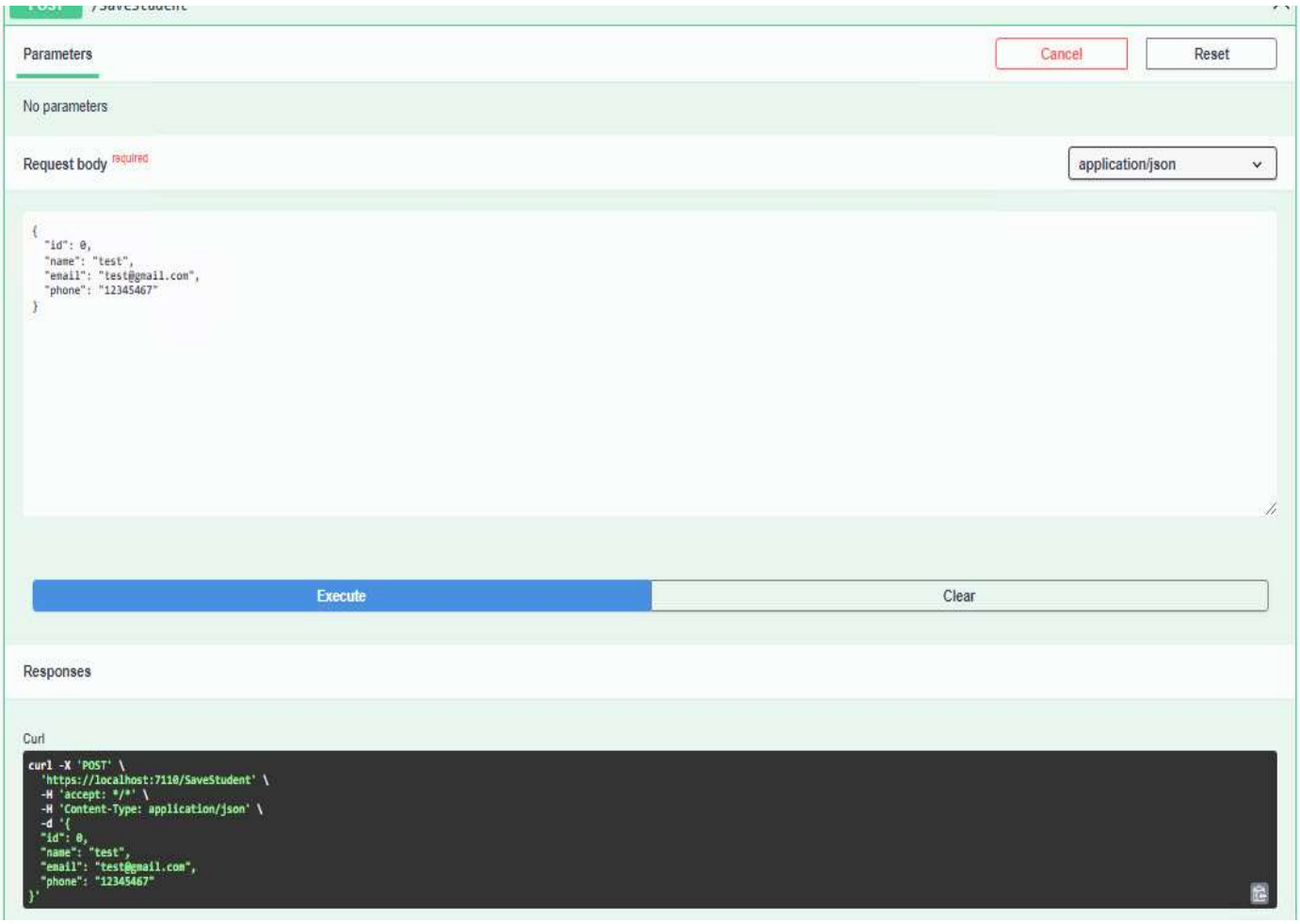
```
using Microsoft.EntityFrameworkCore;
using SampleMinimalAPI.Data;
using SampleMinimalAPI.Model;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetc
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

//Add dbContext
builder.Services.AddDbContext<MyDataContext>(opt => opt.UseInMemoryDataba
var app = builder.Build();
```

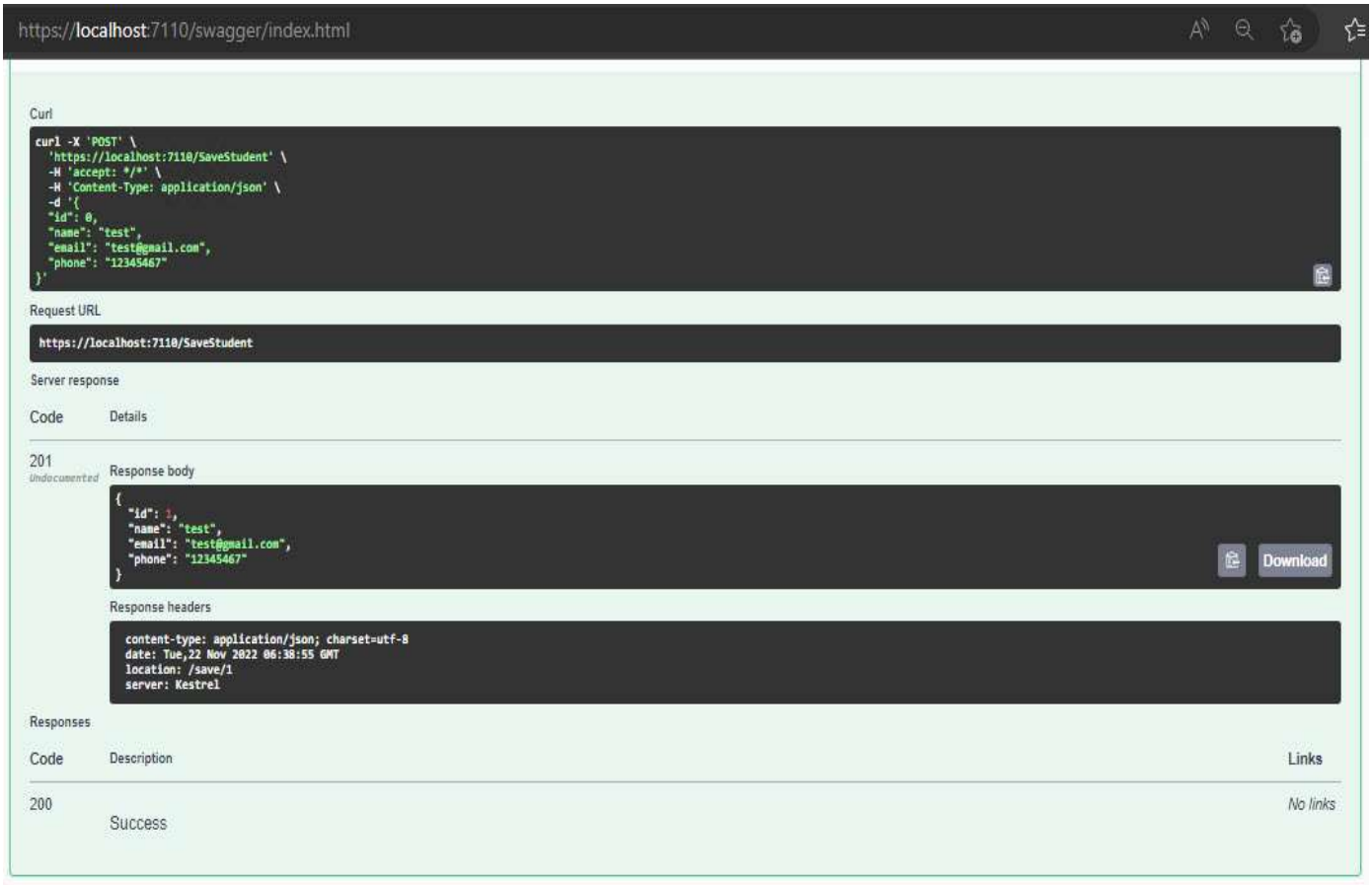
```
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Ho
};

app.MapGet("/weatherforecast", () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        ))
        .ToArray();
    return forecast;
})
.WithName("GetWeatherForecast")
.WithOpenApi();

//Student APIs
app.MapPost("/SaveStudent", async (Student student, MyDataContext db) =>
{
    db.Students.Add(student);
    await db.SaveChangesAsync();

    return Results.Created($"/save/{student.Id}", student);
});

app.MapGet("/GetAllStudent", async (MyDataContext db) =>
    await db.Students.ToListAsync());

app.MapGet("/GetStudentById/{id}", async (int id, MyDataContext db) =>
    await db.Students.FindAsync(id)
        is Student student
            ? Results.Ok(student)
```

```
app.MapPut("/UpdateStudents/{id}", async (int id, Student studentinput, M
{
    var student = await db.Students.FindAsync(id);

    if (student is null) return Results.NotFound();

    student.Name = studentinput.Name;
    student.Phone = studentinput.Phone;

    await db.SaveChangesAsync();

    return Results.NoContent();
});

app.MapDelete("/DeleteStudent/{id}", async (int id, MyDataContext db) =>
{
    if (await db.Students.FindAsync(id) is Student student)
    {
        db.Students.Remove(student);
        await db.SaveChangesAsync();
        return Results.Ok(student);
    }

    return Results.NotFound();
});

app.Run();

internal record WeatherForecast(DateOnly Date, int TemperatureC, string?
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}
```

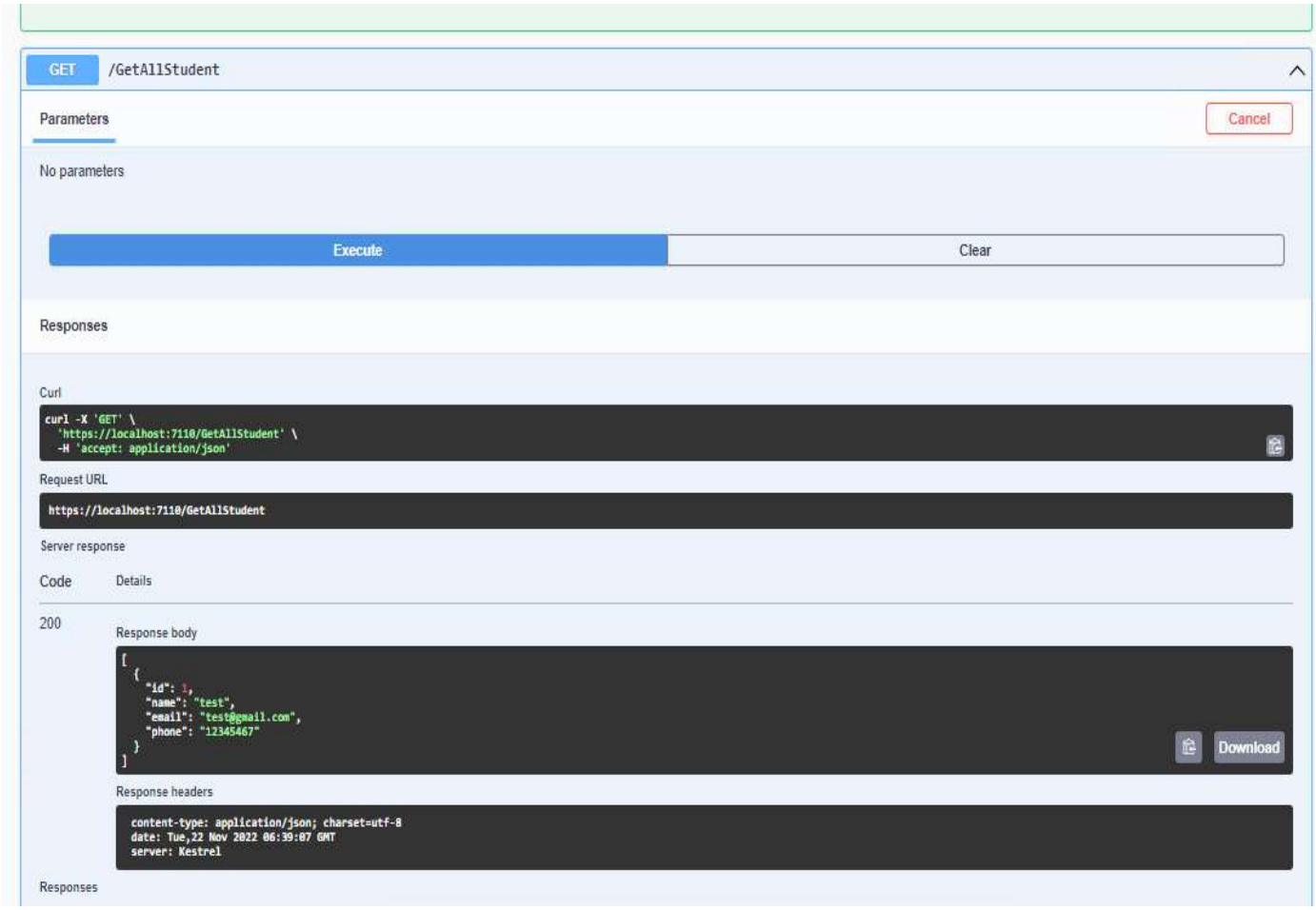
Ahora que nuestras API para estudiantes están listas, podemos ejecutar y probar las API.

A continuación se muestra una imagen de prueba para Save API.





A continuación se muestra una imagen de prueba de la API **GetAll** Student.



De esta manera, podemos probar todas las API de nuestro proyecto de API mínima utilizando una base de datos en memoria.

El código fuente de las API mínimas que se muestran en este artículo se puede encontrar aquí .

Resumen

En pocas palabras, las API mínimas están diseñadas para crear API HTTP con dependencias mínimas. Las API mínimas son adecuadas para microservicios y aplicaciones que incluyen archivos, características y dependencias mínimas con ASP.NET Core. Las API mínimas estaban disponibles desde .NET 6. Por lo tanto, el artículo describe qué es la API mínima y cómo comenzar con la API mínima en .NET 7. Además, el artículo ha demostrado cómo crear una nueva API personalizada con un ejemplo de API para estudiantes. Además, tenemos una idea sobre Entity Framework Core 7 e implementamos el marco de entidad en el proyecto Minimal API, así como también creamos una API mínima para la operación CRUD utilizando el marco de entidad y la base de datos en memoria.

Referencia

rijsat.com

.NETO | NET 7 | Núcleo .NET | ASP.NET | Núcleo ASP.NET | Marco de la entidad | API mínima | API web | Desarrollo web

LIBRO ELECTRÓNICO GRATUITO RECOMENDADO

Desarrollo de API utilizando ASP.NET Core Web API

¡Descargar ahora!

ARTÍCULOS SIMILARES

Operación CRUD utilizando Entity Framework Core y procedimiento almacenado en .NET Core 6 Web API

GraphQL en .NET Core Web API con Entity Framework Core - Cuarta parte

GraphQL en .NET Core Web API con Entity Framework Core - Tercera parte

GraphQL en .NET Core Web API con Entity Framework Core - Primera parte

GraphQL en .NET Core Web API con Entity Framework Core - Segunda parte

Satya Karki TOP 50

Satya Karki es desarrollador de Blockchain, ingeniero de software senior, MCP, MCTS, MCT con más de 10 años de experiencia profesional en tecnologías de Microsoft. Experiencia práctica con ASP.NET, MVC, ASP.NET Core, Azure Cognitive Servi... Leer más

https://rijsat.com https://www.youtube.com/c/RijSat https://www.linkedin.com/in/satya-karki/

13	3m	3	1
0			

Escriba su comentario aquí y presione la tecla Enter (Mínimo 10 caracteres)