Unidad 6 – Capítulo 1 – Laboratorio 1

Creando una Web API con ASP.NET Core

Objetivos

Ejercitar la creación de una Web API utilizando ASP.NET Core como framework para exponer funcionalidades hacia otras aplicaciones siguiendo el estilo REST.

Duración Aproximada

60 minutos

Consignas

Realizar el ABMC / CRUD de una entidad o concepto del dominio y exponer cada una de estas operaciones a través de una Web API utilizando ASP.NET Core. La entidad sobre la que se trabaje deberá persistirse en una base de datos relacional, para lo cual podemos apoyarnos en EF Core.

Utilizar flujos asíncronos (`async` / `await`) para exponer cada una de las operaciones ABMC / CRUD, dado que son los flujos recomendados para interacciones con una base de datos (`I/O-bound`).

En los pasos que se describen a continuación, se adopta como ejemplo la entidad `Alumno` de un hipotético caso de negocio en el contexto de una Universidad, pero el ejemplo es fácilmente trasladable a cualquier otro escenario de negocio.

Pasos sugeridos

1) Utilizando Visual Studio, crear un proyecto de tipo ASP.NET Core Empty.

2) Crear una clase que represente un `Alumno` en el dominio de una Universidad.

    a) Definir algunas propiedades dentro de la clase creada que representen sus datos asociados, estas propiedades podrían ser:
        - `Id` (`int`)
        - `Apellido` (`string`)
        - `Nombre` (`string`)
        - `Legajo` (`int`)
        - `Direccion` (`string`)

3) Agregar el paquete NuGet `Microsoft.EntityFrameworkCore.SqlServer`.

4) Crear una clase que herede de `DbContext` para representar una sesión con la base de datos donde se persistirá la información del `Alumno`, esta clase podría llamarse `UniversidadContext`. Dentro de esta clase:

    a) Definir una propiedad que represente la colección de alumnos, por ejemplo:
        - `Alumnos` (`DbSet<Alumno>`)

    b) Sobrescribir el método `OnConfiguring` de la clase base para configurar el uso de SQL Server como base de datos (adaptar la cadena de conexión según corresponda):

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuil
der)
{
    optionsBuilder.UseSqlServer(@"Server=(localdb)\MSSQLLocalDB;Initial
Catalog=Universidad;Integrated Security=true");
}
```

TIP: si se desea imprimir en la consola el código SQL que el ORM genera al interactuar con la base de datos, se puede configurar en el `optionsBuilder` de esta manera:

```
optionsBuilder.LogTo(Console.WriteLine, LogLevel.Information);
```

c) Establecer en el constructor de la clase que el ORM se asegure de crear la base de datos si la misma no existe:

```
public UniversidadContext()
{
    this.Database.EnsureCreated();
}
```

5) Agregar el paquete NuGet `Swashbuckle.AspNetCore` al proyecto para habilitar OpenAPI / Swagger.

6) En el archivo Program.cs, registrar el contexto con la base de datos y los servicios de OpenAPI / Swagger a continuación de la creación del `builder` de la siguiente manera:

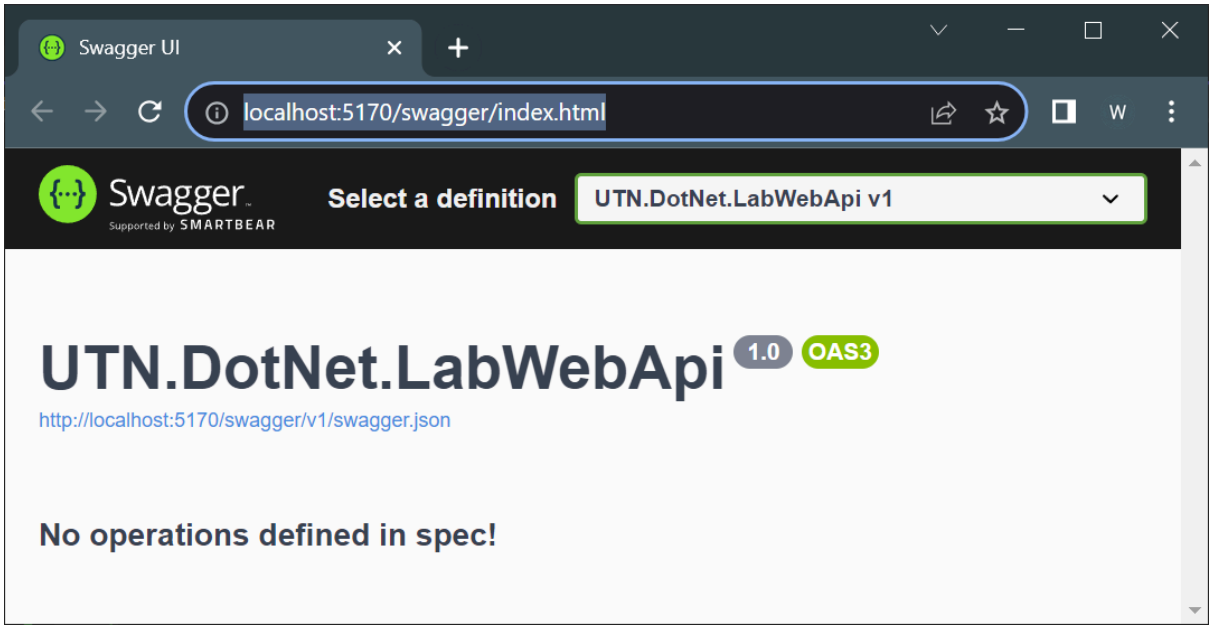
```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<UniversidadContext>(); <-- insertar
builder.Services.AddEndpointsApiExplorer(); <-- insertar
builder.Services.AddSwaggerGen(); <-- insertar

var app = builder.Build();

app.UseSwagger(); <-- insertar
app.UseSwaggerUI(); <-- insertar
```

7) Ejecutar la aplicación para verificar que no hay errores de compilación y que se muestra la UI generada por OpenAPI / Swagger:



8) A continuación, crear las operaciones ABMC / CRUD necesarias para administrar la entidad `Alumno`; para esto, se puede optar por cualquiera de los tres enfoques vistos:

- MVC (tradicional)
- API mínima (nuevo)
- API endpoints (necesario agregar el paquete NuGet `Ardalis.ApiEndpoints`)

En línea con el estilo REST, las operaciones a publicar para completar el ABMC / CRUD son:

Método HTTP	***Comportamiento***	***Código HTTP de respuesta***
`GET`	Devuelve todos los `Alumnos`	`200 OK`
`GET`	Recibe un id y devuelve el `Alumno` asociado o ninguno	`200 OK`, si existe el `Alumno` `404 Not Found`, si no existe
`POST`	Crea un nuevo `Alumno`	`201 Created`
`PUT`	Actualiza un `Alumno` existente o ninguno	`204 No Content`, si existe el `Alumno` `404 Not Found`, si no existe
`DELETE`	Elimina un `Alumno` existente o ninguno	`204 No Content`, si existe el `Alumno` `404 Not Found`, si no existe

A modo de ejemplo, se incluyen dos de las operaciones con el enfoque de API mínima:

```
app.MapGet("/alumnos/{id}", async (int id, UniversidadContext context) =>
    await context.Alumnos.FindAsync(id) is Alumno alumno
        ? Results.Ok(alumno)
        : Results.NotFound());

app.MapPost("/alumnos", async (Alumno alumno, UniversidadContext context) =>
{
    context.Alumnos.Add(alumno);
    await context.SaveChangesAsync();

    return Results.Created($"/alumnos/{alumno.Id}", alumno);
});
```

TIP: recordar que se pueden probar las operaciones que se van creando desde el browser, a través de la UI que automáticamente genera OpenAPI / Swagger.