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 podrá persistirse en una lista o colección estática en memoria.

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) Dentro de la clase `Alumno` declarar un miembro estático de tipo `List<Alumno>` que sea público cuyo nombre podría ser `Lista`:

```
public static readonly List<Alumno> Lista = new();
```

4) Agregar también en la clase `Alumno` un método estático que retorne el próximo `Id` de un nuevo alumno: si la lista está vacía, el próximo `Id` será 1, si la lista no está vacía, el próximo `Id` será el mayor `Id` de `Alumno` existente más 1. La firma del método podría ser:

```
public static int ObtenerProximoId()
```

   NOTA: este método está pensado solamente para poner en práctica los conceptos abordados en la presente unidad; en una aplicación real el `Id` de una entidad típicamente es generado por un motor de base de datos o a través de algún mecanismo que asegure su unicidad en aplicaciones

multi usuario. Este enfoque no es seguro para una aplicación real porque no garantiza esa unicidad.

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

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

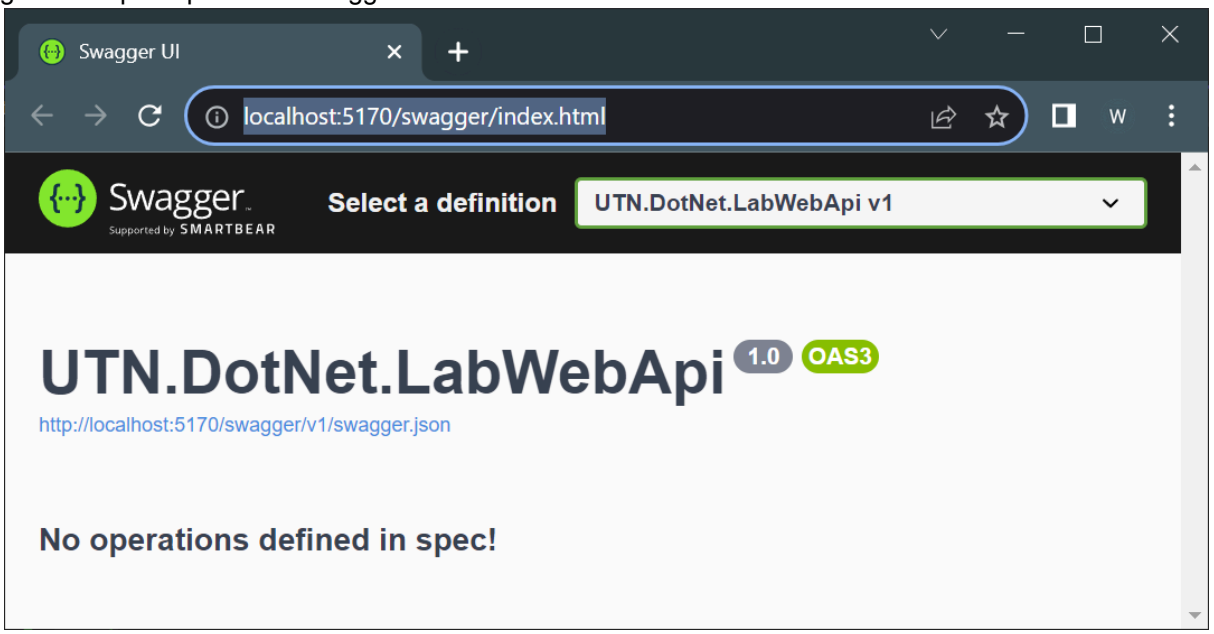
```
var builder = WebApplication.CreateBuilder(args);

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}", (int id) =>
    Alumno.Lista.Find(a => a.Id == id) is Alumno alumno
        ? Results.Ok(alumno)
        : Results.NotFound());

app.MapPost("/alumnos", (Alumno alumno) =>
{
    alumno.Id = Alumno.ObtenerProximoId();
    Alumno.Lista.Add(alumno);
    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.