# Laboratorio Minimal Api

Documentación estará en mi **GitHub**.

Usuario: **SERRAGUTI**

Un Minimal Api es un Api (de toda la vida) que no implementa **Controllers**

Su funcionalidad estará dentro de la (nueva) clase **Program.cs**

Realmente se utiliza para Apis rápidas, es decir, Apis que tengan pocos métodos y que no implementan muchas características.

Nosotros vamos a realizar métodos CRUD y Mapeo de métodos en la Api.

Herramientas para el laboratorio:

* Visual Studio 2022
* SQL Server (Local/Azure)
* Un cliente Api (Postman o Insomnia)

Para el laboratorio de hoy vamos a realizar las siguientes acciones:

1. Crear un Minimal Api con unos métodos inventados y comprobar que funciona
2. Implementar un Minimal Api con BBDD y múltiples operaciones
3. Publicar el Api en Azure y visualizar si funciona
4. Crear una aplicación Mvc Core que consumirá el Api en código C#
5. Implementar seguridad dentro del Api (BasicAuthentication)
6. Validarnos dentro del Mvc y comprobar que funciona la seguridad

Comenzamos creando una aplicación llamada **ApiMinimalPersonajesSeries** de tipo ASP Net Core Web Api.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**PROGRAM.CS**

var builder = WebApplication.CreateBuilder(args);

//INYECTAMOS LOS SERVICIOS TAL Y COMO HACIAMOS EN STARTUP

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

//HABILITAMOS LOS SERVICIOS COMO EN STARTUP

if (app.Environment.IsDevelopment())

{

}

app.UseSwagger();

app.UseSwaggerUI();

app.UseHttpsRedirection();

//UN MINIMAL API NO TIENE CONTROLLERS

//TODOS LOS ENDPOINT DE ACCESO ESTARAN DENTRO

//DE Program.cs MAPEADOS

app.MapGet("/numeros", () =>

{

//AQUI DENTRO DEVOLVEMOS OBJETOS O RESPUESTAS

List<int> numeros = new List<int>();

Random random = new Random();

for (int i = 1; i <= 10; i++) {

numeros.Add(random.Next(1, 800));

}

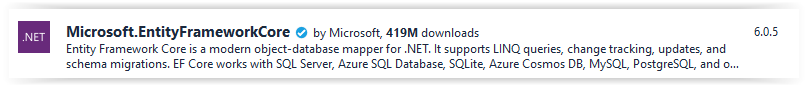
return numeros;

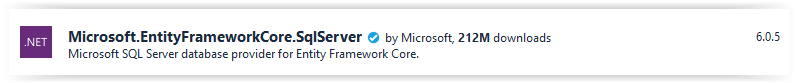
});

app.Run();

Implementar métodos de acceso a Base de datos.

Agregamos los siguientes Nuget





Comenzamos creando una carpeta llamada **Models** y creamos una clase llamada **Personaje** y otra clase llamada **Serie**

**PERSONAJE**

[Table("PERSONAJESLAB")]

public class Personaje

{

[Key]

[Column("IDPERSONAJE")]

public int IdPersonaje { get; set; }

[Column("PERSONAJE")]

public string Nombre { get; set; }

[Column("IDSERIE")]

public int IdSerie { get; set; }

[Column("IMAGEN")]

public string Imagen { get; set; }

[Column("USERNAME")]

public string UserName { get; set; }

[Column("PASSWORD")]

public string Password { get; set; }

}

**SERIE**

[Table("SERIESLAB")]

public class Serie

{

[Key]

[Column("IDSERIE")]

public int IdSerie { get; set; }

[Column("SERIE")]

public string Nombre { get; set; }

[Column("IMAGEN")]

public string Imagen { get; set; }

[Column("PUNTUACION")]

public double Puntuacion { get; set; }

[Column("AÑO")]

public int Anyo { get; set; }

}

El siguiente paso es crear una carpeta llamada **Data** y una clase llamada **SeriesPersonajesContext**

**SERIESPERSONAJESCONTEXT**

public class SeriesPersonajesContext: DbContext

{

public SeriesPersonajesContext(DbContextOptions<SeriesPersonajesContext> options)

: base(options) { }

public DbSet<Personaje> Personajes { get; set; }

public DbSet<Serie> Series { get; set; }

}

Creamos una carpeta llamada **Repositories** y una clase llamada **RepositorySeriesPersonajes**

**REPOSITORYSERIESPERSONAJES**

public class RepositorySeriesPersonajes

{

private SeriesPersonajesContext context;

public RepositorySeriesPersonajes(SeriesPersonajesContext context)

{

this.context = context;

}

public List<Personaje> GetPersonajes()

{

return this.context.Personajes.ToList();

}

public Personaje FindPersonaje(int id)

{

return this.context.Personajes.SingleOrDefault(x => x.IdPersonaje == id);

}

private int GetMaxIdPersonaje()

{

if (this.context.Personajes.Count() == 0)

{

return 1;

}

else

{

return this.context.Personajes.Max(x => x.IdPersonaje) + 1;

}

}

public async Task AddPersonajeAsync(string nombre, string imagen

, int idserie, string username, string password)

{

Personaje personaje = new Personaje()

{

IdPersonaje = this.GetMaxIdPersonaje(),

Nombre = nombre,

Imagen = imagen,

IdSerie = idserie,

UserName = username,

Password = password

};

await this.context.Personajes.AddAsync(personaje);

await this.context.SaveChangesAsync();

}

public List<Personaje> GetPersonajesSerie(int idSerie)

{

return this.context.Personajes.Where(x => x.IdSerie == idSerie).ToList();

}

public List<Serie> GetSeries()

{

return this.context.Series.ToList();

}

private int GetMaxIdSerie()

{

if (this.context.Series.Count() == 0)

{

return 1;

}

else

{

return this.context.Series.Max(x => x.IdSerie) + 1;

}

}

public async Task AddSerieAsync(string nombre, string imagen

, double puntuacion, int anyo)

{

Serie serie = new Serie

{

IdSerie = this.GetMaxIdSerie(),

Nombre = nombre,

Imagen = imagen,

Puntuacion = puntuacion,

Anyo = anyo

};

await this.context.Series.AddAsync(serie);

await this.context.SaveChangesAsync();

}

public async Task UpdateSerieAsync

(int idserie, string nombre, string imagen

, double puntuacion, int anyo)

{

Serie serie = this.context.Series.SingleOrDefault(x => x.IdSerie == idserie);

serie.Nombre = nombre;

serie.Imagen = imagen;

serie.Puntuacion = puntuacion;

serie.Anyo = anyo;

await this.context.SaveChangesAsync();

}

public async Task DeleteSerieAsync

(int idserie)

{

Serie serie = this.context.Series.SingleOrDefault(x => x.IdSerie == idserie);

this.context.Series.Remove(serie);

await this.context.SaveChangesAsync();

}

}

Incluimos nuestra cadena de conexión dentro de **appsettings.json**

**APPSETTINGS.JSON**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"AzureSql": "Data Source=azurecastraininglabs.database.windows.net;Initial Catalog=AZUREDATABASE;Persist Security Info=True;User ID=adminsql;Password=Admin123"

}

}

Implementamos la inyección y las acciones dentro de **Program.cs**

**PROGRAM.CS**

using ApiMinimalPersonajesSeries.Data;

using ApiMinimalPersonajesSeries.Models;

using ApiMinimalPersonajesSeries.Repositories;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

//INYECTAMOS LOS SERVICIOS TAL Y COMO HACIAMOS EN STARTUP

builder.Services.AddTransient<RepositorySeriesPersonajes>();

string connectionString =

builder.Configuration.GetConnectionString("AzureSql");

builder.Services.AddDbContext<SeriesPersonajesContext>

(options => options.UseSqlServer(connectionString));

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

//HABILITAMOS LOS SERVICIOS COMO EN STARTUP

if (app.Environment.IsDevelopment())

{

}

app.UseSwagger();

app.UseSwaggerUI();

app.UseHttpsRedirection();

//METODOS PARA PERSONAJES

app.MapGet("/personajes", (RepositorySeriesPersonajes repo) =>

{

return repo.GetPersonajes();

});

//FIND(id)

app.MapGet("/personajes/find/{id}", (int id,

RepositorySeriesPersonajes repo) =>

{

return repo.FindPersonaje(id);

});

app.MapPost("/personajes/post", async (Personaje personaje,

RepositorySeriesPersonajes repo) =>

{

await repo.AddPersonajeAsync(personaje.Nombre, personaje.Imagen

, personaje.IdSerie, personaje.UserName, personaje.Password);

return Results.Ok();

});

app.MapGet("/personajes/serie/{idserie}", (int idserie,

RepositorySeriesPersonajes repo) =>

{

return repo.GetPersonajesSerie(idserie);

});

//METODOS PARA LAS SERIES

app.MapGet("/series", (RepositorySeriesPersonajes repo) =>

{

return repo.GetSeries();

});

app.MapPost("/series/post", async (Serie serie

, RepositorySeriesPersonajes repo) =>

{

await repo.AddSerieAsync(serie.Nombre, serie.Imagen, serie.Puntuacion

, serie.Anyo);

return Results.Ok();

});

app.MapDelete("/series/delete/{id}", async (int id,

RepositorySeriesPersonajes repo) =>

{

await repo.DeleteSerieAsync(id);

return Results.Ok;

});

app.MapPut("/series/put/{id}", async (int id

, Serie serie

, RepositorySeriesPersonajes repo) =>

{

await repo.UpdateSerieAsync(id, serie.Nombre, serie.Imagen

, serie.Puntuacion, serie.Anyo);

return Results.Ok();

});

app.Run();

Si deseamos habilitar Swagger en Azure, debemos incluir las siguientes líneas

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La URL de publicación es la siguiente:

<https://apiminimalpersonajesseries.azurewebsites.net/index.html>