# 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));

}

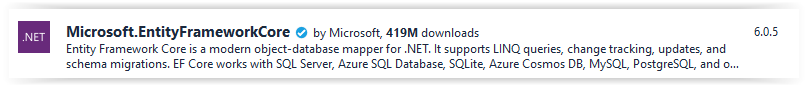
return numeros;

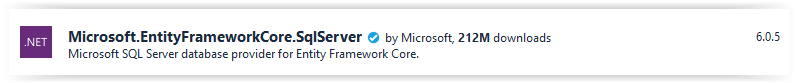
});

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>

A continuación, vamos a realizar una aplicación Mvc Net Core para consumir el Servicio Minimal Api.

Vamos a reutilizar las clases **Serie** y **Personaje**

Creamos una nueva aplicación llamada **MvcCoreMinimalApi**

Agregamos los siguiente Nuget para trabajar

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Incluimos, dentro de **appsettings.json** la URL de acceso a nuestra Api

**APPSETTINGS.JSON**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ApiUrls": {

"ApiMinimalPersonajes": "https://apiminimalpersonajesseries.azurewebsites.net/"

}

}

Creamos una carpeta llamada **Services** y una clase llamada **ServiceMinimalApiPersonajes**

**SERVICEMINIMALAPIPERSONAJES**

public class ServiceMinimalApiPersonajes

{

private MediaTypeWithQualityHeaderValue Header;

private string UrlApi;

public ServiceMinimalApiPersonajes(IConfiguration configuration)

{

this.UrlApi = configuration.GetValue<string>("ApiUrls:ApiMinimalPersonajes");

this.Header = new MediaTypeWithQualityHeaderValue("application/json");

}

//METODO GENERICO PARA RECUPERAR DATOS

private async Task<T> CallGetApiAsync<T>(string request)

{

using (HttpClient client = new HttpClient())

{

client.BaseAddress = new Uri(this.UrlApi);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

HttpResponseMessage response =

await client.GetAsync(request);

if (response.IsSuccessStatusCode)

{

T data = await response.Content.ReadAsAsync<T>();

return data;

}

else

{

return default(T);

}

}

}

private async Task CallPostAsync<T>(string request, T data)

{

using (HttpClient client = new HttpClient())

{

client.BaseAddress = new Uri(this.UrlApi);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

string json =

JsonConvert.SerializeObject(data);

StringContent content =

new StringContent(json, Encoding.UTF8, "application/json");

HttpResponseMessage response =

await client.PostAsync(request, content);

}

}

public async Task<List<Personaje>> GetPersonajesAsync()

{

string request = "/personajes";

List<Personaje> personajes =

await this.CallGetApiAsync<List<Personaje>>(request);

return personajes;

}

public async Task<List<Personaje>> FindPersonajesSerieAsync(int idserie)

{

string request = "/personajes/serie/idserie";

List<Personaje> personajes =

await this.CallGetApiAsync<List<Personaje>>(request);

return personajes;

}

public async Task AddPersonajeAsync(Personaje personaje)

{

string request = "/personajes/post";

await this.CallPostAsync<Personaje>(request, personaje);

}

public async Task<List<Serie>> GetSeriesAsync()

{

string request = "/series";

List<Serie> series =

await this.CallGetApiAsync<List<Serie>>(request);

return series;

}

public async Task AddSerieAsync(Serie serie)

{

string request = "/series/post";

await this.CallPostAsync<Serie>(request, serie);

}

}

Vamos a separar las acciones en dos controladores

Creamos un controlador llamado **PersonajesController**

**PERSONAJESCONTROLLER**

public class PersonajesController : Controller

{

private ServiceMinimalApiPersonajes service;

public PersonajesController(ServiceMinimalApiPersonajes service)

{

this.service = service;

}

public async Task<IActionResult> Index()

{

List<Personaje> personajes =

await this.service.GetPersonajesAsync();

return View(personajes);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(Personaje personaje)

{

await this.service.AddPersonajeAsync(personaje);

return RedirectToAction("Index");

}

}

**SERIESCONTROLLER**

public class SeriesController : Controller

{

private ServiceMinimalApiPersonajes service;

public SeriesController(ServiceMinimalApiPersonajes service)

{

this.service = service;

}

public async Task<IActionResult> Index()

{

List<Serie> series =

await this.service.GetSeriesAsync();

return View(series);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(Serie serie)

{

await this.service.AddSerieAsync(serie);

return RedirectToAction("Index");

}

public async Task<IActionResult> BuscarPersonajeSerie(int idserie)

{

List<Personaje> personajes =

await this.service.FindPersonajesSerieAsync(idserie);

return View(personajes);

}

}

Creamos las vistas de forma dinámica con **Scaffolding**

Inyectamos el servicio dentro de **Program.cs**

**PROGRAM.CS**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**IMPLEMENTAR SEGURIDAD USUARIO DENTRO DE MINIMAL API**

Nosotros estamos acostumbrados a validar mediante Token, lo que se llama **OAuth** y utilizando JWT Bearer.

En un Api normal se utiliza un Controller dedicado a la seguridad y se habilita dicha seguridad y devolvemos un Token en las peticiones.

En Minimal Api esto no es recomendado porque ocupa una gran cantidad de código dentro de Program.cs.

En Minimal Api se utiliza la seguridad Basic mediante un Handler de autentificación básica.

Dicha autentificación se crea en una clase externa y se incluye como autorización dentro del código de Program.cs.

No ocupa mucho espacio dentro de Program.

Vamos a realizar una seguridad de Header que incluya usuario/password.

En este tipo de seguridad, en la petición del Api viene un Header con Authorization y Token encriptado. En dicho Token viene el formato de usuario/password separado por los dos puntos:

**Username:Password**

Volvemos al **Minimal Api**

Creamos una carpeta llamada **Authentication** y una clase llamada **BasicAuthenticationHandler**

**BASICAUTHENTICATIONHANDLER**

public class BasicAuthenticationHandler:

AuthenticationHandler<AuthenticationSchemeOptions>

{

//CONTENDRA UN CONTRUSCTOR QUE RECIBIRA

//UNOS PARAMETROS (INTERNOS) DENTRO DE LA CLASE BASE

public BasicAuthenticationHandler(

IOptionsMonitor<AuthenticationSchemeOptions> options,

ILoggerFactory logger,

UrlEncoder encoder,

ISystemClock clock

) : base(options, logger, encoder, clock)

{

}

protected override Task<AuthenticateResult> HandleAuthenticateAsync()

{

var authHeader = Request.Headers["Authorization"].ToString();

//RECIBE UN PARAMETRO QUE ES AUTHORIZATION CON UN CONTENIDO

//DE basic Y USERNAME:PASSWORD ENCRIPTADO COMO TOKEN

if (authHeader != null &&

authHeader.StartsWith("basic", StringComparison.OrdinalIgnoreCase))

{

//RECUPERAMOS EL TOKEN CIFRADO

var token = authHeader.Substring("Basic ".Length).Trim();

//DECODIFICAMOS EL TOKEN PARA QUEDARNOS CON USERNAME:PASSWORD

var credentialsString =

Encoding.UTF8.GetString(Convert.FromBase64String(token));

var credential = credentialsString.Split(':');

if (credential[0] == "admin" && credential[1] == "admin")

{

//CREAMOS LOS CLAIMS PARA LA VALIDACION DE USUARIOS

var claims = new[]

{

new Claim("name", credential[0]),

new Claim(ClaimTypes.NameIdentifier, credential[0]),

new Claim(ClaimTypes.Role, "Basic")

};

var identity = new ClaimsIdentity(claims, "Basic");

var claimsPrincipal = new ClaimsPrincipal(identity);

return Task.FromResult(AuthenticateResult.Success

(new AuthenticationTicket(claimsPrincipal, Scheme.Name)));

}

//NO TIENE LAS CREDENCIALES CORRECTAS

Response.StatusCode = 401;

Response.Headers.Add("WWW-Authenticate", "Basic realm=\"dotnetthoughts.net\"");

return Task.FromResult(AuthenticateResult.Fail("Invalid Authorization Header"));

}

else

{

Response.StatusCode = 401;

Response.Headers.Add("WWW-Authenticate", "Basic realm=\"dotnetthoughts.net\"");

return Task.FromResult(AuthenticateResult.Fail("Invalid Authorization Header"));

}

}

En **Program.cs** habilitamos la seguridad en las peticiones

**PROGRAM.CS**