Asp net core web api

Todo esto existe a partir de .Net Core 6

Viene con un archivo .json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

Habla mucho de que se trabaja con swagger.

Json

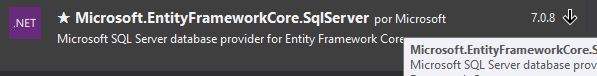
Agrega al Json el connectionString: (es distinto al que he agregado anteriormente.

DESKTOP-MSJO6R8 Podría también ir eso en vez de localhost

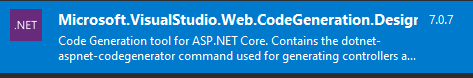
"ConnectionStrings": {" BolsaDeDatos ": "Server=localhost; Database=BolsaDeDatos; Trusted\_Connection=True;"}

Administrar paquetes NuGet

Así gestiona las dependencias. Clickea en el proyecto, no en la solución.

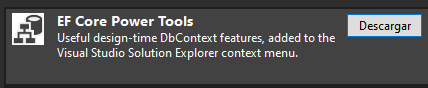






Administrar extensions

Core power tools



Program.cs

Agrega este using:

using Microsoft.EntityFrameworkCore;

const string ConnectionName = "BolsaDeDatos";

var connectionString = builder.Configuration.GetConnectionString(ConnectionName);

Crea carpeta DataAccess y allí una clase UniversityContext. En esa clase, agrega:

using Microsoft.EntityFrameworkCore;

Y la hace heredera de DbContext.

public class UniversityContext:DbContext

Y un constructor así:

public UniversityContext(DbContextOptions<UniversityContext>opciones): base(opciones)

{

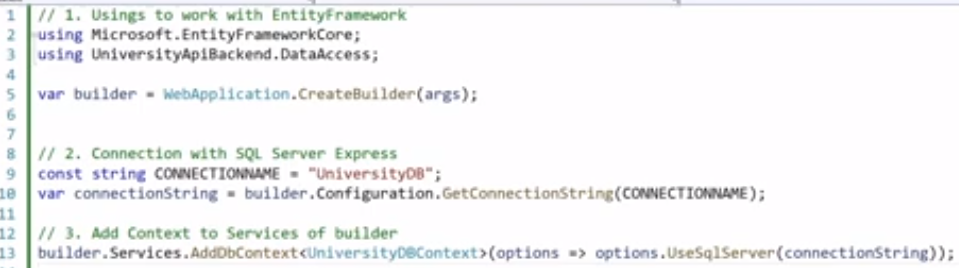
}

Otro using (program):

using API1OpenBootcamp.AccesoDatos;

y otro builder

builder.Services.AddDBContext<UniversityContext>(options => options.UseSqlServer(connectionString));

En baseEntity establece los requisitos que toda tabla quiere que tenga:

public class BaseEntity

{

[Required]

[Key]

public int Id { get; set; }

public string CreatedBy { get; set; }=string.Empty;

public DateTime CreatedAt { get; set; }= DateTime.Now;

public string UpdatedBy { get; set; } = string.Empty;

public DateTime? UpdatedAt { get; set; }

//Con el ? se hace opcional

public string DeletedBy { get; set; } = string.Empty;

public DateTime? DeletedAt { get; set; }

public bool IsDeleted { get; set; }

}

Luego, crea otra clase, user, la cual hereda esta anterior.

public class User:BaseEntity

{

[Required, StringLength(50)]

public string Name { get; set; } = string.Empty;

[Required, StringLength(50)]

public string LastName { get; set; } = string.Empty;

[Required, EmailAddress]

public string Email { get; set; }=string.Empty;

[Required]

public string Password { get; set; }= string.Empty;

}

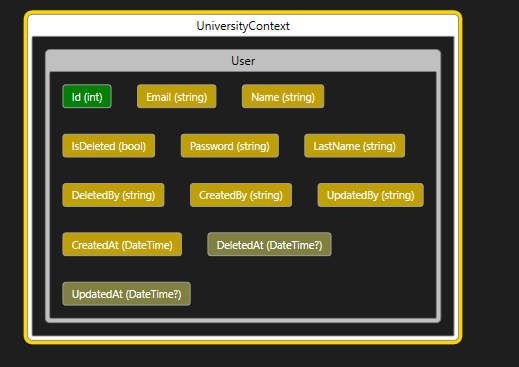
Luego, dice que hay que comprobar si el modelo es correcto=> botón derecho en el proyecto=> EfPowerTools=>Generar DBContext diagram

El resultado no es muy espectacular:

En la clase UniversityContext, agrega:

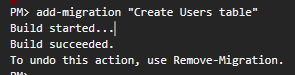
public DbSet<User>? Users { get; set; }

Ahora, el EF Power Tools muestra esto:

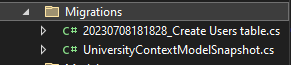


Luego, va a la consola de administrador de paquetes y tipea:

add-migration "Create Users table"



Crea esto:



public partial class CreateUserstable : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Users",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false).Annotation("SqlServer:Identity", "1, 1"),

Name = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),

LastName = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),

Email = table.Column<string>(type: "nvarchar(max)", nullable: false),

Password = table.Column<string>(type: "nvarchar(max)", nullable: false),

CreatedBy = table.Column<string>(type: "nvarchar(max)", nullable: false),

CreatedAt = table.Column<DateTime>(type: "datetime2", nullable: false),

UpdatedBy = table.Column<string>(type: "nvarchar(max)", nullable: false),

UpdatedAt = table.Column<DateTime>(type: "datetime2", nullable: true),

DeletedBy = table.Column<string>(type: "nvarchar(max)", nullable: false),

DeletedAt = table.Column<DateTime>(type: "datetime2", nullable: true),

IsDeleted = table.Column<bool>(type: "bit", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Users", x => x.Id);

});

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "Users");

}

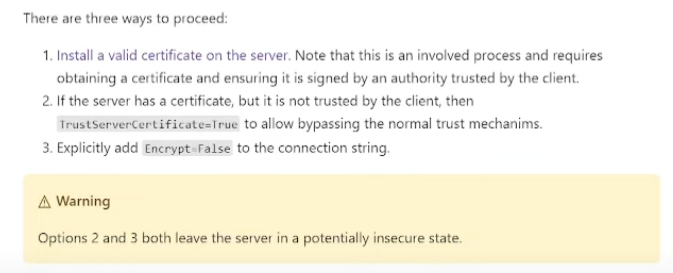
}

Luego, hay que hacer:

update-database

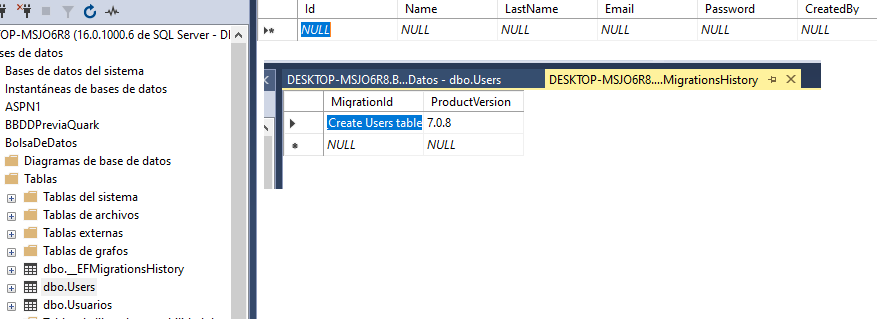
Me tira este error:

A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL Provider, error: 0 - La cadena de certificación fue emitida por una entidad en la que no se confía.)



Agrego esto a la cadena del Json y funca. No sería lo mas seguro:

TrustServerCertificate=True;



Finalmente se crea la BBDD.

Poniendo script-migration muestra lo que se ejecuta:

IF OBJECT\_ID(N'[\_\_EFMigrationsHistory]') IS NULL

BEGIN

CREATE TABLE [\_\_EFMigrationsHistory] (

[MigrationId] nvarchar(150) NOT NULL,

[ProductVersion] nvarchar(32) NOT NULL,

CONSTRAINT [PK\_\_\_EFMigrationsHistory] PRIMARY KEY ([MigrationId])

);

END;

GO

BEGIN TRANSACTION;

GO

CREATE TABLE [Users] (

[Id] int NOT NULL IDENTITY,

[Name] nvarchar(50) NOT NULL,

[LastName] nvarchar(50) NOT NULL,

[Email] nvarchar(max) NOT NULL,

[Password] nvarchar(max) NOT NULL,

[CreatedBy] nvarchar(max) NOT NULL,

[CreatedAt] datetime2 NOT NULL,

[UpdatedBy] nvarchar(max) NOT NULL,

[UpdatedAt] datetime2 NULL,

[DeletedBy] nvarchar(max) NOT NULL,

[DeletedAt] datetime2 NULL,

[IsDeleted] bit NOT NULL,

CONSTRAINT [PK\_Users] PRIMARY KEY ([Id])

);

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20230708181828\_Create Users table', N'7.0.8');

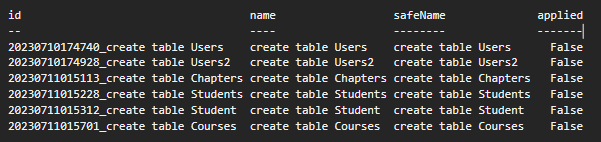
GO

COMMIT;

GO

Remove-migration eliminaría la última migración.

Get-migration devuelve esto:



Clase 3

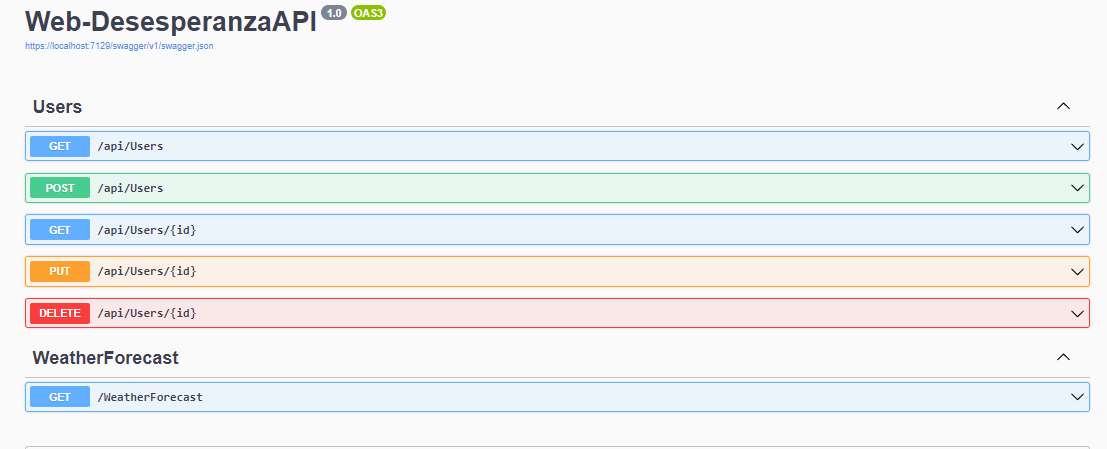
Agregar controlador para web api con entity framework.

Pide “Clase de modelo”. User en este caso.

También pide la clase de contexto. DBContext en este caso.

En cuanto al nombre del controlador, Swagger busca que el nombre al final diga controller, por lo que es imprescindible. Recomienda que a eso se le sume al ppio el nombre de la clase + s al ppio.

Llama a lo que sucede cuando se clickea, “Técnica de scaffolding”, te genera código preestablecido. No hay mas que comprobar que esté todo OK.



Se crea este CRUD

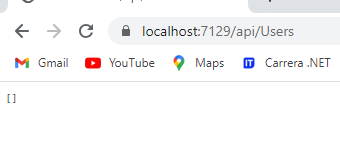
<https://localhost:7129/swagger/index.html>

Ese es el sitio que muestra todo eso.

A la vez, está este otro:

<https://localhost:7129/api/Users>

que devuelve esto, de momento:



Creo que se vería la tabla. Se ejecuta la siguiente función:

// GET: api/Users

[HttpGet]

public async Task<ActionResult<IEnumerable<User>>> GetUsers()

{

if (\_context.Users == null)

{

return NotFound();

}

return await \_context.Users.ToListAsync();

}

Esto también es un GET pero se le pasa un id, que debe ser primary key. Te devuelve un usuario.

// GET: api/Users/5

[HttpGet("{id}")]

public async Task<ActionResult<User>> GetUser(int id)

{

if (\_context.Users == null)

{

return NotFound();

}

var user = await \_context.Users.FindAsync(id);

if (user == null)

{

return NotFound();

}

return user;

}

Método post. Añade a la table, un usuario.

// POST: api/Users

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPost]

public async Task<ActionResult<User>> PostUser(User user)

{

if (\_context.Users == null)

{

return Problem("Entity set 'UniversityContext.Users' is null.");

}

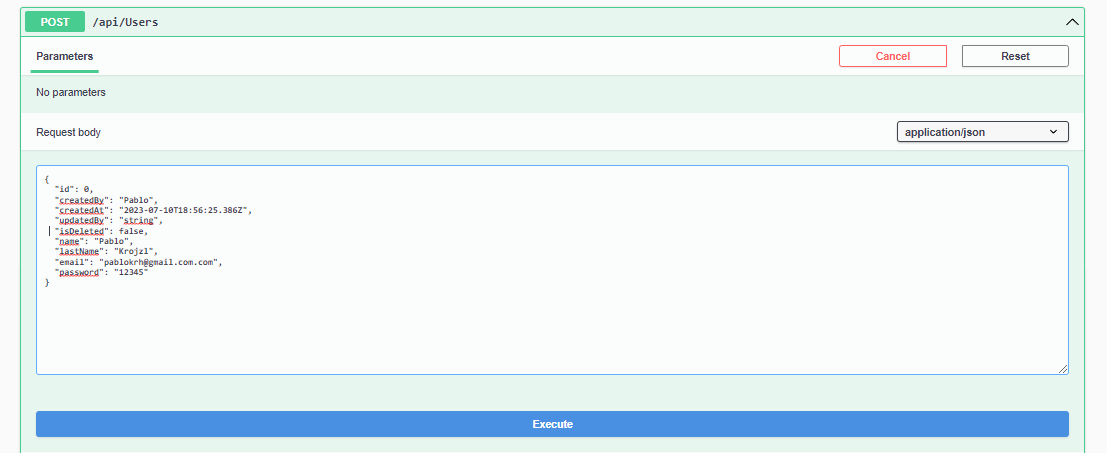
\_context.Users.Add(user);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetUser", new { id = user.Id }, user);

}

Se ve que retorna algo. Llama al método GetUser, para que devuelva este usuario recién creado.



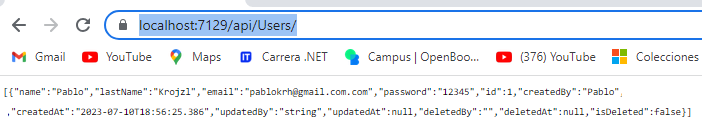
Borré un par de los campos, ya que eran opcionales.

Se puede acceder a la data:

<https://localhost:7129/api/Users/>

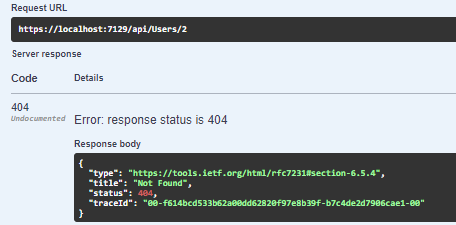
https://localhost:7129/api/Users/1

En el primer caso se vería toda, en la otra, la que coincida con el primer valor.

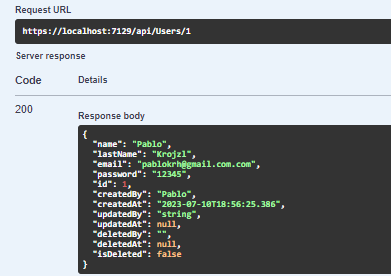


También se puede ver la data a través de Swagger:

En este caso puse un id no existente:



En este no:



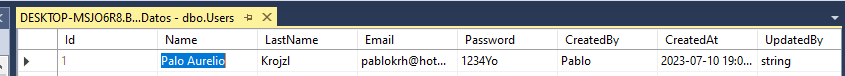
Método Put. Update.

public async Task<IActionResult> PutUser(int id, User user)

Nosotros le pasamos el id, y el User ya lo tiene la app. El retorno es noContent, un msg 404, de que todo ha ido bien.



Ahí se ve el código 204 y el 200. Acá SSMS



Método Delete.

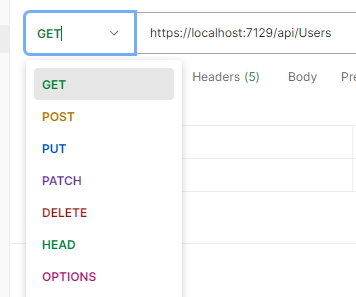
Busca por Id.

Todo esto es con Swagger. Ahora pega volantazo:

Postman

Colección=>Crear Colección

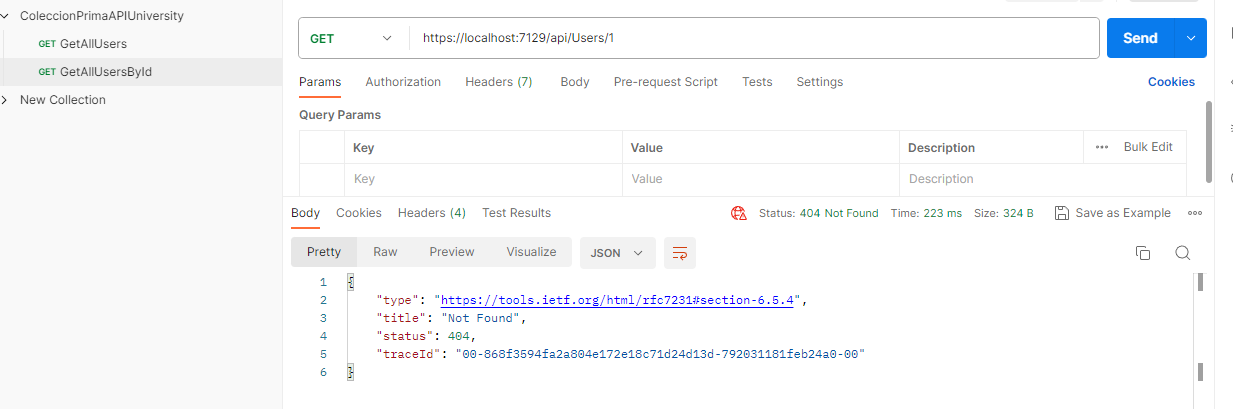
Add a request=> Elegir método:



Le copié la dirección: <https://localhost:7129/api/Users>

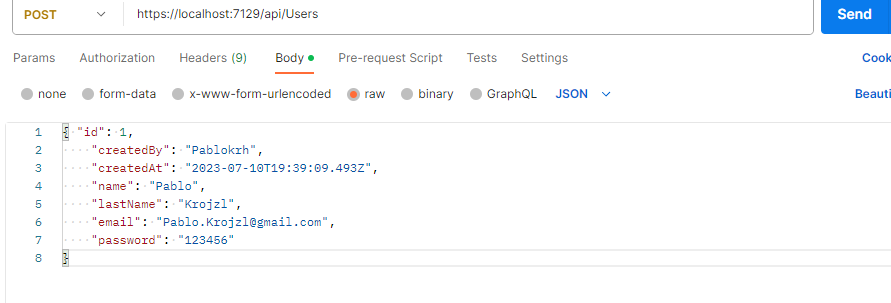
Luego se toca Send

Tuve que descargar Postman desktop si no me tiraba errores.



Ahí puse un segundo método. GetUserById con una id inexistente, tira un 404.

Se hace un método post. En el body se espera un raw, de tipo Json. Luego se escribe el cuerpo de la petición.



Eso da error. El Id es autoincrementable y no hay que especificarlo.

De todos modos, recomienda utilizar Swagger.

--------------------------------FIN-----------------------------------------

Ah re…

Crea clase curso que herede de BaseEntity.

public enum Level

{

Basic,

Advanced,

Medium,

Expert

}

public class Curse:BaseEntity

{

[Required,StringLength(50)]

public string Name { get; set; }=String.Empty;

[Required, StringLength(250)]

public string ShortDescription { get; set; } = String.Empty;

[Required]

public string FullDescription { get; set; } = String.Empty;

public Level Level { get; set; } = Level.Basic;

}

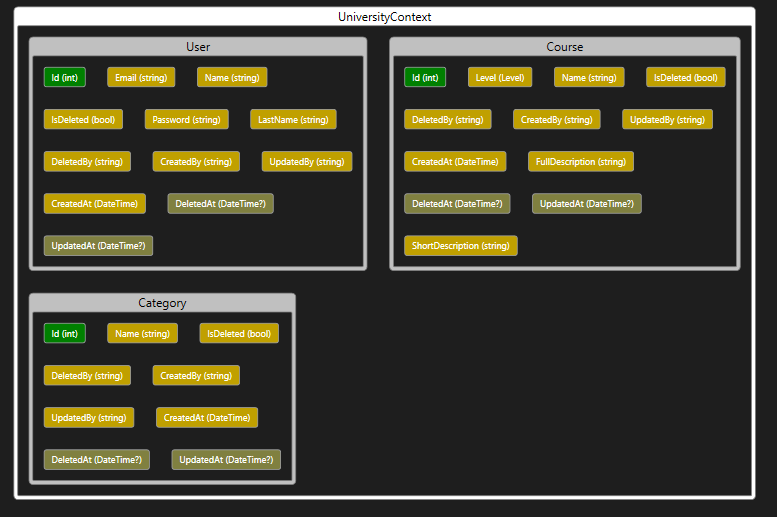
El enum se codea así. Acá:public Level Level { get; set; } = Level.Basic; establece el básico como el valor por default.

También agrega una clase Categorías.

Para que las convierta en tablas, hay que agregarlas a UniversityContext:

public DbSet<Category>? Categories { get; set; }

public DbSet<Course>? Courses { get; set; }

Todo está dentro de University Context y OK, lo que si, faltan las relaciones entre tablas.

Así se hace la relación, dentro de la clase de cursos:

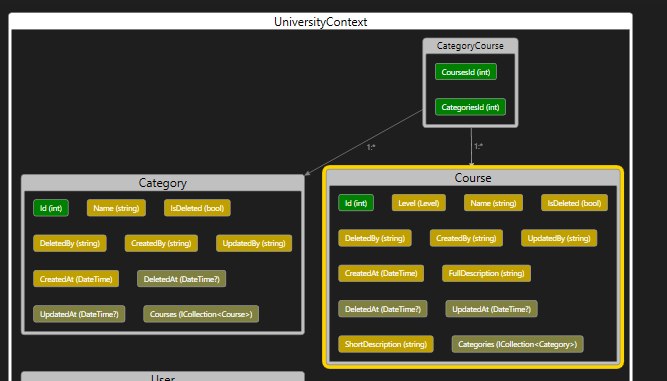
[Required]

public ICollection<Category> Categories { get; set; }=new List<Category>();

Lo mismo, pero al vesre se hace en la clase categorías.

[Required]

public ICollection<Course> Courses { get; set; } = new List<Course>();



Crea también la clase estudiante y la vincula a la clase cursos.

ATENTO!!!

Modifica BaseEntity: Incorpora User en created, updated y deleted by.

[Required]

[Key]

public int Id { get; set; }

public int UserId { get; set; }

public virtual User CreatedBy { get; set; } = new User();

public DateTime CreatedAt { get; set; } = DateTime.Now;

public User UpdatedBy { get; set; } = new User();

public DateTime? UpdatedAt { get; set; }

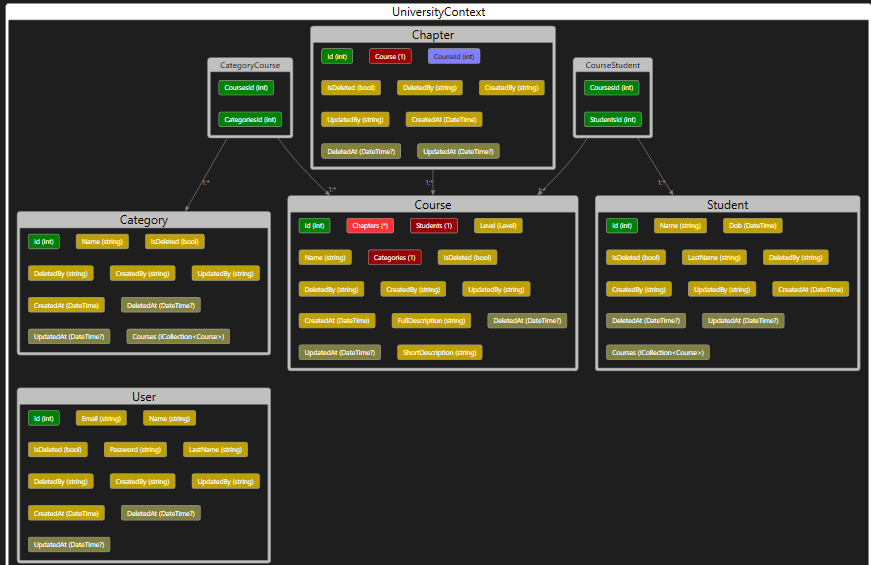
public User DeletedBy { get; set; } = new User();

public DateTime? DeletedAt { get; set; }

public bool IsDeleted { get; set; }

Luego retrocede en chancletas para que funcione y vuelve con todo string.

Agrega temarios (chapter), que vinculará con los cursos.



Agrega luego un controlador de Course.

Clase 3

Linq

Dice que las consultas sql puras no son recomendables, de cara a la seguridad.

Se lo realiza para hacer búsquedas directamente sobre el contexto.

public class Employee

{

public int Id { get; set; }

public string? Name { get; set; }

public int Salary { get; set; }

public string Email { get; set; }

}

public class Enterprise

{

public int Id { get; set; }

public string? Name { get; set; }

public Employee[]? Employees { get; set; }

}

public class Snippets

{

static public void BasicLinq()

{

string[] cars =

{

"VW Golf",

"VW California",

"Audi A3",

"Audi A5",

"Fiat Punto",

"Seat Ibiza",

"Seat León"

};

//Select \* from cars

var carList = from car in cars select car;

//Selecciona todos los autos del array cars.

//Select where

var audiList = from car in cars where cars.Contains("Audi") select car;

}

static public void Numbers()

{

List<int> numbers = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

var processedNumberList = numbers

.Select(num => num \* 3)

.Where(num=>num!=9)

.OrderBy(num=>num);

//Multiplica todo por 3, luego, deja por fuera al número que sea 9 y luego lo ordena ascendentemente.

}

static public void SearchExamples()

{

List<string> textList = new List<string>() { "a", "bx", "c", "e", "cj", "f", "c" };

var first=textList.First();

var cText=textList.First(text=>text.Equals("c"));

//Acá se guarda la primera vez que un texto sea (no que contenga) la c.

var firstElement = textList.First(text=>text.Contains("j"));

var firstElement2 =textList.Contains("j"); //Funca esta?

string? firstOrDefaultText = textList.FirstOrDefault(text => text.Contains("z"));

//Devuelve una Z o una cadena de texto vacía.

var lastOrDefaultText = textList.LastOrDefault(text => text.Contains("z"));

var uniqueElements = textList.SingleOrDefault();

//En caso de haber elementos repetidos en la lista, tomaría uno solo de ellos. Es casi igual que el .Single();

int[] evenNumbers = { 0, 2, 4, 6, 8 };

int[] oddEvenNumbers = { 0,2,6 };

//obtener números que no se repiten en ninguna de las listas:

var myEvenNumbers = evenNumbers.Except(oddEvenNumbers);

}

static public void MultipleSelects()

{

string[] MyOpinion =

{

"Opinión 1, text 1",

"Opinión 2, text 2",

"Opinión 3, text 3",

};

var myOpinionSelection = MyOpinion.SelectMany(opinion => opinion.Split(","));

//Guarda los valores, separados, por la coma, habrían 6 entonces valores, y no tres.

var enterprises = new[]

{

new Enterprise()

{

Id = 1,

Name = "Enterprise One",

Employees = new[]

{

new Employee()

{

Id = 1,

Name = "Carlos Sluzky",

Email = "Charly@gmail.com",

Salary = 3600,

},

new Employee()

{

Id = 2,

Name = "Francisco Varela",

Email = "PanchitoChileno@argentina.com.ar",

Salary = 1600,

},

new Employee()

{

Id = 3,

Name = "Osvaldo Laport",

Email = "TroncazoDeMalo@hotmail.com",

Salary = 2200,

}

}

},

new Enterprise()

{

Id = 2,

Name = "Enterprise Twelve",

Employees = new[]

{

new Employee()

{

Id = 4,

Name = "Marcos Olazábal ",

Email = "Ola@demar.com",

Salary = 658,

},

new Employee()

{

Id = 5,

Name = "Valentino Moroder",

Email = "Tino@Atolonico.com",

Salary = 1000,

},

new Employee()

{

Id = 6,

Name = "Pablo Krojzl",

Email = "pablokrh@hotmail.com",

Salary = 360,

}

}

}

};

//Obtener todos los empleados de todas las empresas:

var employeeList = enterprises.SelectMany(enterprise => enterprise.Employees);

//Ver si una lista está vacía

var hasEnterprises=enterprises.Any();

//Ver si una lista tiene empleados. Si una lista de empresas, no si una empresa tiene empleados.

var hasEmployees=enterprises.Any(enter=>enter.Employees.Any());

//Que tengan empleados de mas de 900 euros

bool hasEmployeeWithSalaryMoreOrEqualThan900 = enterprises.Any(enterprise =>

enterprise.Employees.Any(employee => employee.Salary >= 900));

}

public static void linqCollections()

{

var firstList = new List<string> { "a", "b", "c" };

var secondList = new List<string> { "a", "c", "d" };

//inner join. BUsca los elementos compartidos por las 2 listas.

var commonResult = from element in firstList

join secondElement in secondList

on element equals secondElement

select new { element, secondElement };

var commonResult2 = firstList.Join(

secondList, element => element, secondElement => secondElement,

(element, secondElement) => new { element, secondElement });

//2 formas distintas

//Outer joint -LEFT:

var leftOuterJoin = from element in firstList

join secondElement in secondList

on element equals secondElement

into temporalList

from temporalElement in temporalList.DefaultIfEmpty()

where element != temporalElement

select new { Element = element };

//Hasta el into, busca los elementos comunes en ambas listas, y las guarda en una lista temporal.

//Luego quiere obtener los elementos de la lista 1 que no están en la lista temporal (de los elementos comunes)

//Left significa que es lo de la izquierda, es decir, de la lista 1.

//Forma "Simplificada"

var leftOuterJoin2 = from element in firstList

from secondElement in secondList.Where(s => s == element).DefaultIfEmpty()

select new { Element = element, SecondElement = secondElement };

//Outer joint -Right:

var RightOuterJoin = from secondElement in secondList

join element in firstList

on secondElement equals element

into temporalList

from temporalElement in temporalList.DefaultIfEmpty()

where secondElement != temporalElement

select new { Element = secondElement };

//Unión!! Sin los elementos repetidos.

var unionList = leftOuterJoin.Union(RightOuterJoin);

}

static public void SkipTakeLinq()

{

var myList = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var skipOneAndTwo=myList.Skip(2);

//Devuelve la lista sin los primeros 2 índices.

var skipLastTwo=myList.SkipLast(2);

//Ignora los últimos 2.

var skipWhile = myList.SkipWhile(num => num < 4);

//Devuelve la lista cuyos número no sean menores que 4

var take=myList.Take(2);

//Toma solamente los primeros 2 valores

}

Clase 4

Paginación

Generación de variables locales con LINQ.

ZIP

Repeat y Range

3 formas de crear IEnumerables:

IEnumerable <string> CincoXsPrimeraForma = Enumerable.Repeat("X", 5);

IEnumerable CincoXsSegundaForma = Enumerable.Repeat("X", 5);

var CincoXsTerceraForma = Enumerable.Repeat("X", 5);

ALL

Aggregate

Distinct

Group By

foreach (var item in certifiedQuery)

{

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*{0}\*\*\*\*\*\*\*\*\*\*\*\*", item.Key);

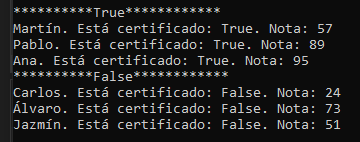
foreach (var student in item)

{

Console.WriteLine(student.Name + ". Está certificado: "+student.Certified+ ". Nota: "+student.Grade);

}

}



var comentariosConContenido = posts.SelectMany(

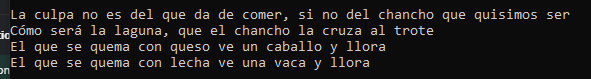
post => post.Comments, (post, comment) => new { PostId = post.Id, commentContent = comment.Content });

foreach (var item in comentariosConContenido)

{

Console.WriteLine(item.commentContent);

}



Agrega al proyecto original una carpeta de servicios. Rápidamente le enchufa una interfaz IStudentService. Luego una clase StudentService, que la implementa.

Hacen falta servicios, para cada uno de los modelos.

Hay que añadir los servicios al program Cs. Justo debajo de los controllers.

builder.Services.AddScoped< IStudentService, StudentService > ();

Agrega un controlador para students, de no haberlo hecho ya.

Al controlador, le agrega el servicio.

private readonly StudentService \_studentService;

public StudentsController(UniversityContext context, StudentService studentService)

{

\_context = context;

\_studentService = studentService;

}

Se le pasa por parámetro y se lo inicializa con el constructor.

Ahí ya podrían utilizarse los métodos codeados en la carpeta de servicios. Los mismos se inyectan.

CORS

En program.cs se habilita el cors, y así habilitar, quienes pueden hacer peticiones a la API, desde dónde, qué apps puedan, y etc.

//CORS

builder.Services.AddCors(options =>

{

options.AddPolicy(name: "CorsPolicy", builder =>

{

builder.AllowAnyOrigin();

builder.AllowAnyMethod();

builder.AllowAnyHeader();

});

});

Eso es lo mas genérico que hay. Hay luego que decirle a la App que haga uso de cors.

JWT

JSON Web Token

Sistema de autenticación. Genera una clave privada. Tienen una validez limitada. Es muy utilizado.

Es una información verificada y confiable.

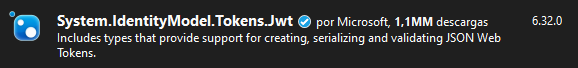
Partes:

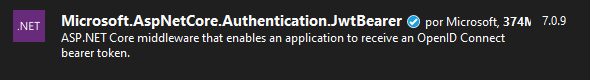
Header (Cabecera)- 2 partes: tipo de token (JWT); algoritmo de cifrado que utiliza (RSA, SHA256, por ejemplo).

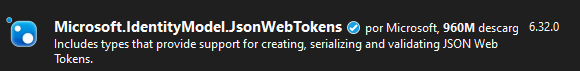
Payload (o Contenido)- Contiene el contenido y los claims (los hay registrados, públicos y privados), además, hay info adicional..

Signature (o Firma)- Generada a partir de la cabecera, el payload y una clave secreta, mas el algoritmo cifrado especificado en la cabecera.

Paquetes NuGet.







Dice que es mas o menos igual en todos los proyectos y altamente reutilizable.

Crea clase pública en Data Models, llamada JWTSettings. Y otra que es UserTokens.

AppSettings.Json. Ajoba del connectionString. Con todo eso hace una configuración genérica en todo el proyecto y evitar tener que ponerlo dentro del propio código.

public class JwtSettings

{

public bool ValidateIssuerSigningKey { get; set; }

public string IssuerSigningKey { get; set; }

public bool ValidateIssuer { get; set; } = true;

public string ValidIssuer { get; set; }

public bool ValidateAudience { get; set; } = true;

public string ValidAudience { get; set; }

public bool RequiredExpirationTime { get; set; }

public bool ValidateLifeTime { get; set; } = true;

}

"JsonWebTokenKeys": {

"ValidateIssuerSigningKey": true,

"IssuerSigningKey": "",

"ValidateIssuer": "true",

"ValidIssuer": "https://localhost:7129",

"ValidateAudience": true,

"ValidAudience": "https://localhost:7129",

"RequiredExpirationTime": true,

"ValidateLifeTime": true

}

Luego realiza unos helpers. Crea una carpeta y ahí una clase estática JWTHelpers.

Utiliza el import system.Security.Claims.

public static IEnumerable<Claim> GetClaims(this UserTokens userAccounts, Guid id)

{

List<Claim> claims = new List<Claim>()

{

new Claim("Id", userAccounts.Id.ToString()),

new Claim(ClaimTypes.Name, userAccounts.Username),

new Claim(ClaimTypes.Email, userAccounts.EmailId),

new Claim(ClaimTypes.NameIdentifier, userAccounts.Id.ToString()),

new Claim(ClaimTypes.Expiration, DateTime.UtcNow.AddDays(1).ToString("MMM ddd dd HH: mm:ss tt")),

//El 1 significa que le suma un día a la hora actual.

//Así, expira en 24 hs. Luego define con forma muy copada cómo se verá la hora

};

//Luego crea nombres distintos para quienes usen la página.

//Está bastante hardcodeado y usa el userName como base para distinguir:

if (userAccounts.Username == "Admin")

{

claims.Add(new Claim(ClaimTypes.Role, "Adiministrator"));

}

else if (userAccounts.Username =="User 1")

{

claims.Add(new Claim(ClaimTypes.Role, "User"));

claims.Add(new Claim("UserOnly", "User 1"));

}

return claims;

}

public static IEnumerable<Claim> GetClaims(this UserTokens userAccounts, out Guid Id)

{

Id = Guid.NewGuid();

return GetClaims(userAccounts, Id);

}

Esta segunda genera un Id y llama a la primera.

using Microsoft.IdentityModel.Tokens; Para el signingCredentials.

public static UserTokens GenTokenKeys (UserTokens model, JwtSettings jwtSettings)

{

try

{

var userToken = new UserTokens();

if (model == null)

{

throw new ArgumentNullException(nameof(model));

}

//Genera clave secreta

var key = System.Text.Encoding.ASCII.GetBytes(jwtSettings.IssuerSigningKey);

Guid Id;

//Expira en un día

DateTime expireTime = DateTime.UtcNow.AddDays(1);

//Validez del token

userToken.Validity = expireTime.TimeOfDay;

//Generar el token

var JwToken = new JwtSecurityToken(

issuer: jwtSettings.ValidIssuer,

audience: jwtSettings.ValidAudience,

claims: GetClaims(model, out Id),

notBefore: new DateTimeOffset(DateTime.Now).DateTime,

expires: new DateTimeOffset(expireTime).DateTime,

signingCredentials: new SigningCredentials(

new SymmetricSecurityKey(key),

SecurityAlgorithms.HmacSha256));

userToken.Token = new JwtSecurityTokenHandler().WriteToken(JwToken);

userToken.Username = model.Username;

userToken.Id = model.Id;

userToken.GuidId = Id;

return userToken;

}

catch (Exception ex)

{

throw new Exception("Error generating the JWT", ex);

}

}

Crea otra clase en model: userLogins

public class UserLogins

{

[Required]

public string UserName { get; set; }

[Required]

public string Password { get; set; }

}

Crea otro controlador para esto, pero en esta ocasión, en blanco. Lo crea como una clase, accountController.

namespace Web\_DesesperanzaAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class AccountController:ControllerBase

{

private readonly JwtSettings \_jwtSettings;

public AccountController(JwtSettings jwtSettings)

{

\_jwtSettings = jwtSettings;

}

private IEnumerable<User> Logins = new List<User>()

{

new User()

{

Id = 1,

Email="pablokrh@gmail.com",

Name="Admin",

Password="Admin"

},

new User()

{

Id = 2,

Email="pepe@gmail.com",

Name="User 1",

Password="Pepe"

}

};

[HttpPost]

public IActionResult GetToken(UserLogins userLogin)

{

try

{

var token = new UserTokens();

var valid=Logins.Any(user=>user.Name.Equals(userLogin.UserName, StringComparison.OrdinalIgnoreCase));

//Esto busca una coincidencia entre el name de la lista logins y de userLogin

if (valid)

{

var user=Logins.FirstOrDefault(user=>user.Name.Equals(userLogin.UserName,StringComparison.OrdinalIgnoreCase));

token = JWTHelpers.GenTokenKeys(new UserTokens()

{

Username = user.Name,

EmailId = user.Email,

Id = user.Id,

GuidId = Guid.NewGuid()

}, \_jwtSettings);

}

else

{

return BadRequest("Wrong Credentials");

}

return Ok(token);

}

catch (Exception ex)

{

throw new Exception("Get token error: " + ex);

}

}

[HttpGet]

[Authorize(AuthenticationSchemes =JwtBearerDefaults.AuthenticationScheme,Roles ="Administrator")]

//A esta ruta solo accede el rol de administrador.

public IActionResult GetUserList()

{

return Ok(Logins);

}

}

}

Clase 5

Retoca el JSON, modificando esto:

"IssuerSigningKey": "52523423-fgh4-5h45-424t-erg-34t6"

Le da una contraseña, dice que puede ser de cualquier tipo.

Agrega una clase dentro del proyecto: AddJWTokenServicesExtensions

static public class AddJWTokenServicesExtensions

{

public static void AddJWTokenServices(this IServiceCollection services, IConfiguration configuration)

{

//Agrega los settings del JWTSettings

var bindJwtSettings = new JwtSettings();

//Entre comillas va el nombre de lo que se puso en el JSON

configuration.Bind("JsonWebTokenKeys", bindJwtSettings);

//Add Singleton

services.AddSingleton(bindJwtSettings);

services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme=JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme=JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(options =>

{

options.RequireHttpsMetadata = false;

options.SaveToken = true;

options.TokenValidationParameters = new TokenValidationParameters()

{

ValidateIssuerSigningKey = bindJwtSettings.ValidateIssuerSigningKey,

IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(bindJwtSettings.IssuerSigningKey)),

//acá se genera la clave secreta

ValidateIssuer = bindJwtSettings.ValidateIssuer,

ValidIssuer = bindJwtSettings.ValidIssuer,

ValidateAudience = bindJwtSettings.ValidateAudience,

ValidAudience = bindJwtSettings.ValidAudience,

RequireExpirationTime = bindJwtSettings.RequiredExpirationTime,

ValidateLifetime = bindJwtSettings.ValidateLifeTime,

ClockSkew = TimeSpan.FromDays(1)

};

EN Program.cs agrega el servicio anterior.

builder.Services.AddJWTokenServices(builder.Configuration);

Luego añade una política de autorización:

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("UserOnlyPolicy", policy => policy.RequireClaim("UserOnly", "User 1"));

});

Y luego esto:

builder.Services.AddSwaggerGen(options =>

{

options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name="Authorization",

Type=SecuritySchemeType.Http,

Scheme="Bearer",

BearerFormat="JWT",

In=ParameterLocation.Header,

Description="JWT autorización Header using Bearer Scheme"

});

options.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference=new OpenApiReference

{

Type=ReferenceType.SecurityScheme,

Id="Bearer"

}

},

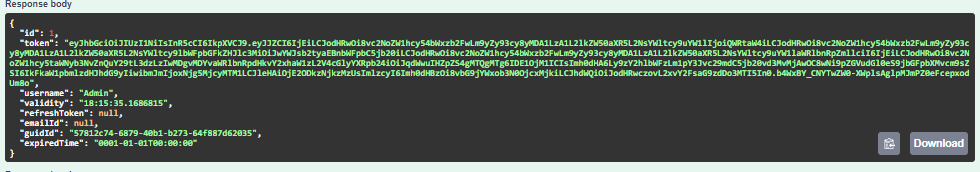
new string[] { }

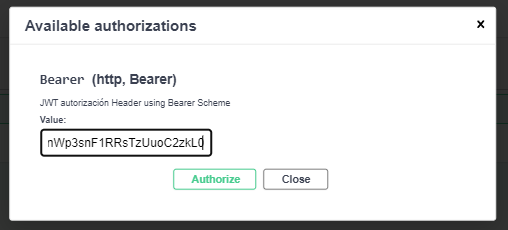
}

});

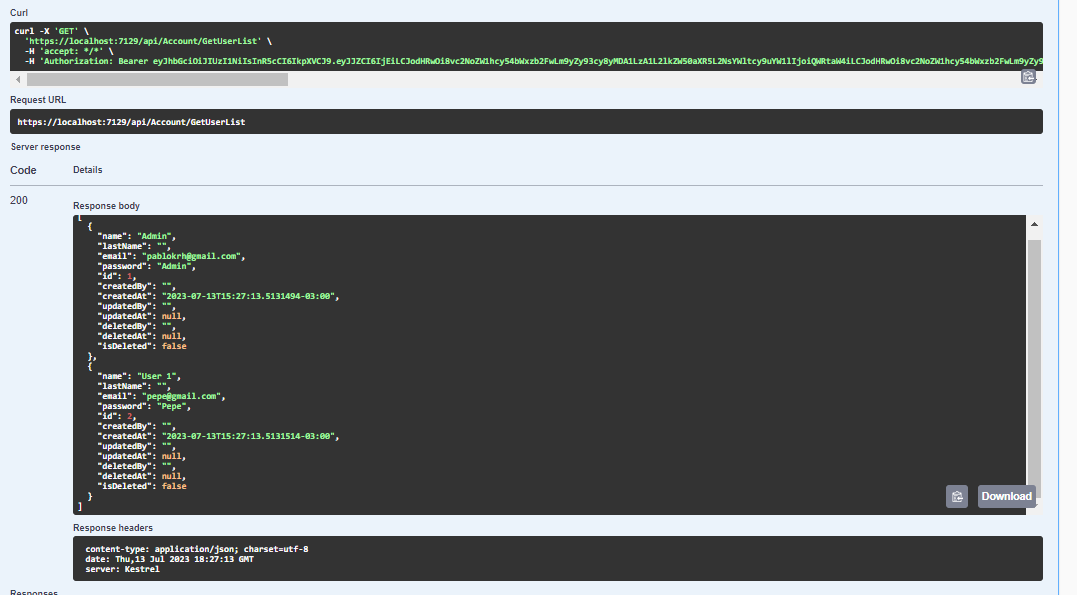
});

Ejecuto la App y me da la posibilidad, en el controlador, escribir username=”Admin” y password=”Admin” y me devuelve este token:





Se agrega el token enorme ahí y te da acceso a la info de la lista.



[HttpGet]

[Authorize(AuthenticationSchemes =JwtBearerDefaults.AuthenticationScheme,Roles ="Administrator")]

//A esta ruta solo accede el rol de administrador.

public IActionResult GetUserList()

{

return Ok(Logins);

}

Ahí se define, en AccountController, la ruta que se desbloquea introduciendo el token de Admin.

Luego toca una línea en el WeatherForeCast para pedir autorización para acceder dentro del swagger a uno de sus métodos:

[HttpGet(Name = "GetWeatherForecast")]

[Authorize(AuthenticationSchemes =JwtBearerDefaults.AuthenticationScheme,Roles = "Administrator, User")]

public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

})

.ToArray();

}

En este caso, tanto administrador como usuarios pueden acceder.

A falta de 7´dice unas cosas de Postman.

Retoca creo que todo, para no hardcodear y usar el context.

public class AccountController:ControllerBase

{

private readonly UniversityContext \_context;

private readonly JwtSettings \_jwtSettings;

public AccountController(JwtSettings jwtSettings, UniversityContext context)

{

\_jwtSettings = jwtSettings;

\_context = context;

}

Igual esto último no lo finaliza (aún).

Clase 7

Búsquedas sobre el contexto usando LINQ.

En AccountController hace justamente eso y reemplaza en función del resultado de la consulta Linq

[HttpPost]

public IActionResult GetToken(UserLogins userLogin)

{

try

{

var token = new UserTokens();

var searchUser = (from user in \_context.Users

where user.Name == userLogin.UserName

&& user.Password == userLogin.Password

select user).FirstOrDefault();

//Sin el FirstOrDefault() no se bien por qué te tira error después. Creo que no te devuelve un

//objeto de tipo User, sino algún tipo de IQueryable, por eso no se puede usar. Así si.

if (searchUser != null)

{

token = JWTHelpers.GenTokenKeys(new UserTokens()

{

Username = searchUser.Name,

EmailId = searchUser.Email,

Id = searchUser.Id,

GuidId = Guid.NewGuid()

}, \_jwtSettings);

}

else

{

return BadRequest("Wrong Credentials");

}

return Ok(token);

}

catch (Exception ex)

{

throw new Exception("Get token error: " + ex);

}

}

Dice que haría falta a cada usuario agregarle un campo para el rol, lo cual lo haría mediante un Enum.

Gestionar versiones de una ApiRestful. Por ejemplo, que devuelva data distinta en función del lenguaje del usuario.

Internacionalización implica globalización y localización.

Hay que localizar al usuario, para devolver info globalizada de manera distinta y adaptarse a los distintos formatos culturales.

Proyecto nuevo. Se suele hacer mediante archivos de traducción. Nombreidioma.resx

En -Fr – De – Es

Program.cs

Añade servicio de localización.

// 1-Localization

builder.Services.AddLocalization(options => options.ResourcesPath = "Resources") ;

//2- Idiomas soportados

var supportedCultures = new[] { "en-US", "es-ES", "fr-FR" }; //Ingles yanqui, español español, franchute de Francia.

var localizationOptinos = new RequestLocalizationOptions()

.SetDefaultCulture("en-US")

.AddSupportedCultures(supportedCultures)

.AddSupportedUICultures(supportedCultures); //Esto es una vista que devuelve (UI).

//3 - Add Localization to app

app.UseRequestLocalization(localizationOptinos)

Crea la carpeta Resources.

Ahí, crea carpeta controllers y otra llamada Entities.

Agrega un controlador en blanco. PostController.

namespace ApiOBInternational.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class PostController : ControllerBase

{

private readonly IStringLocalizer<PostController> \_stringLocalizer;

private readonly IStringLocalizer<SharedResource> \_sharedResourceLocalizer;

public PostController(IStringLocalizer<PostController> stringLocalizer, IStringLocalizer<SharedResource> sharedResourceLocalizer)

{

\_stringLocalizer = stringLocalizer;

\_sharedResourceLocalizer = sharedResourceLocalizer;

}

[HttpGet]

[Route("PostControllerResource")]

public IActionResult GetUsingPostControllerResource()

{

//Encontrar el texto

var article = \_stringLocalizer["Article"];

var postName = \_stringLocalizer.GetString("Welcome").Value ?? ¡Sé bienvenido a mi país, amigo!string.Empty;

// Value ?? string.Empty => En caso de que no esté, está Empty.

return Ok(new

{

PostType = article.Value,

postName=postName,

});

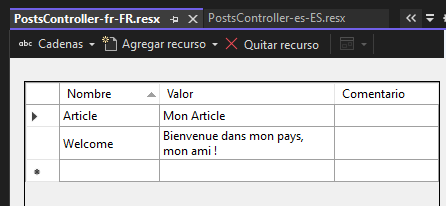
}

}

}

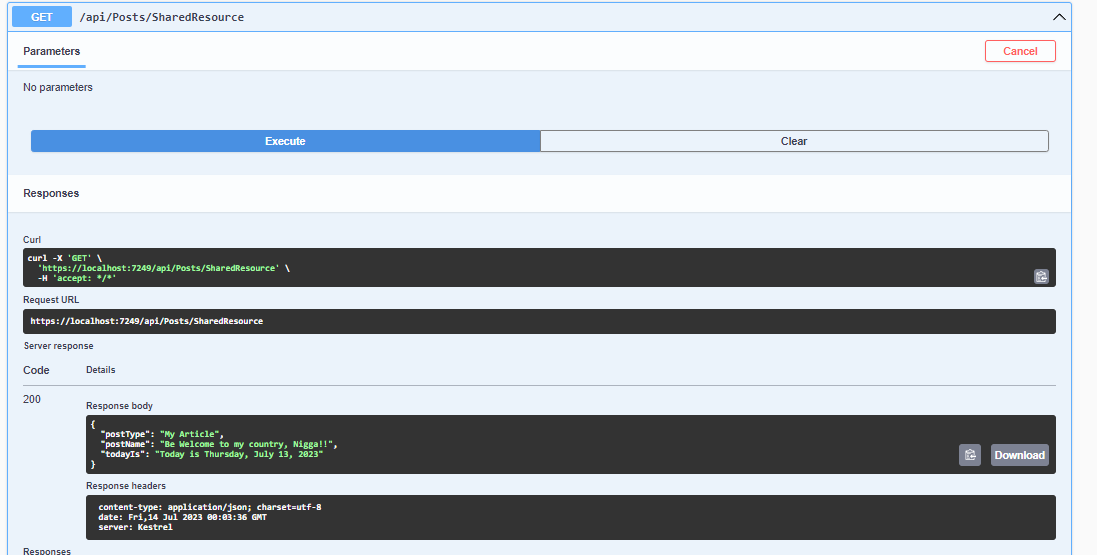
En Resources/Controllers crea un archivo de recursos llamado: PostsController-en-US.resx

Igual que el controlador + especificación del idioma+ extensión .resx

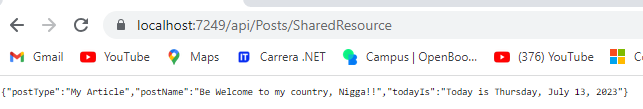


Luego, hace lo mismo con las fechas.

Chrome=>Configuración=>Idiomas=>Añadir otros idiomas.



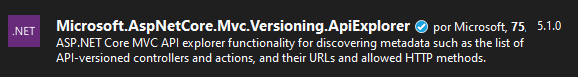
Lo de los idiomas no lo pude cambiar. El muchacho lo hizo con Postman.

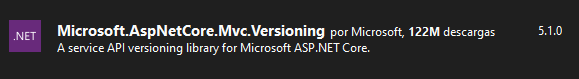


Ahí sin Swagger.

Crea, a su vez una clase SharedResource en carpeta Entities, dentro de controllers. Finalmente la dejó vacía.

Gestión del versionado





Agrega carpetas en la carpeta controller. Una para la versión 1 y otra para la versión 2.

Agrega en raíz una carpeta DTO y allí una clase User. Allí simula el envío de info por parte del usuario. UsersResponseData que hace la respuesta.

Api DummyApi.io permite generar peticiones. Te devuelve usuarios aleatorios a través de JSON.

Base URL

<https://dummyapi.io/data/v1/>

64b4c2bc9c54294957bf51a5

UserData

public class UserData

{

public string? Id { get; set; }

public string? Title { get; set; } //Mr o Mrs

public string? FirstName { get; set; }

public string? LastName { get; set;}

public string? Picture { get; set; }

}

public class UserDataResponse

{

public UserData[]? data { get; set; }

public int ? total { get; set; }

public int? page { get; set; }

public int limit { get; set; }

//Hace falta un controlador para cada uno de estos.

}

Así queda la parte de arriba del controlador:

[ApiVersion("1.0")]

[Route("api/v{version:apiVersion}[controller]")]

[ApiController]

Ambas clases de los controladores se llaman UsersController, pero los archivos se llaman UsersControllerV1 y V2.

UsersControllers V1 y V2

[ApiVersion("1.0")]

[Route("api/v{version:apiVersion}[controller]")]

[ApiController]

public class UsersController : ControllerBase

{

private const string ApiTestUrl = "https://dummyapi.io/data/v1/user?limit=30";

private const string ApiId = "64b4af0295a071ebb0f98631";

//Crea una instancia de http Client. Habrá que enviarlo desde program.cs

private readonly HttpClient \_httpClient;

public UsersController(HttpClient httpClient)

{

\_httpClient = httpClient;

}

[MapToApiVersion("1.0")]

[HttpGet(Name ="GetUserData")]

async Task <IActionResult> GetUsersDataAsync()

{

\_httpClient.DefaultRequestHeaders.Clear();

\_httpClient.DefaultRequestHeaders.Add("app-id", ApiId);

var response = await \_httpClient.GetStreamAsync(ApiTestUrl);

//Acá ya se tendría una respuesta con la lista de usuarios.

var usersData = await JsonSerializer.DeserializeAsync<UserDataResponse>(response);

return Ok(usersData);

}

}

[ApiVersion("2.0")]

[Route("api/v{version:apiVersion}[controller]")]

public class UsersController : ControllerBase

{

private const string ApiTestUrl = "https://dummyapi.io/data/v1/user?limit=30";

private const string ApiId = "64b4af0295a071ebb0f98631";

//Crea una instancia de http Client. Habrá que enviarlo desde program.cs

private readonly HttpClient \_httpClient;

public UsersController(HttpClient httpClient)

{

\_httpClient = httpClient;

}

[MapToApiVersion("2.0")]

[HttpGet(Name = "GetUserData")]

async Task<IActionResult> GetUsersDataAsync()

{

\_httpClient.DefaultRequestHeaders.Clear();

\_httpClient.DefaultRequestHeaders.Add("app-id", ApiId);

var response = await \_httpClient.GetStreamAsync(ApiTestUrl);

//Acá ya se tendría una respuesta con la lista de usuarios.

var usersData = await JsonSerializer.DeserializeAsync<UserDataResponse>(response);

var users = usersData.data;

return Ok(users);

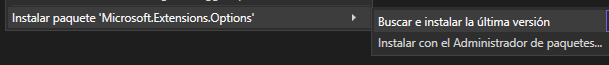
}

}

Son prácticamente idénticas. El único cambio está en el return.

ConfigureSwaggerOptions

Agrega una clase en raíz ConfigureSwaggerOptions. Podría ponerse en carpeta helpers o utils. Esto hace para no empiojar tanta el program.cs



public class ConfigureSwaggerOptions : IConfigureNamedOptions<SwaggerGenOptions>

{

private readonly IApiVersionDescriptionProvider \_provider;

public ConfigureSwaggerOptions(IApiVersionDescriptionProvider provider)

{

\_provider = provider;

}

private OpenApiInfo CreateVersionInfo(ApiVersionDescription description)

{

var info = new OpenApiInfo()

{

Title = "My .Net Api Restful",

Description = "This is my first Api versioning control",

Version = description.ApiVersion.ToString(),

Contact = new OpenApiContact()

{

Email = "pablokrh@gmail.com",

Name = "Pablo Krojzl"

}

};

if (description.IsDeprecated)

{

info.Description += "This Api version has been deprecatted";

}

return info;

}

public void Configure(SwaggerGenOptions options)

{

//Add documentación para cada una de las versiones.

foreach (var description in \_provider.ApiVersionDescriptions) {

options.SwaggerDoc(description.GroupName, CreateVersionInfo(description));

}

}

public void Configure(string? name, SwaggerGenOptions options)

{

Configure(options);

}

}

Program.cs

//Configurar cómo se quiere documentar

builder.Services.AddVersionedApiExplorer(setup =>

{

setup.GroupNameFormat = "'v'VVV";

setup.SubstituteApiVersionInUrl = true;

});

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

//ESTA LINEA DE ABAJO LA DESHABILITO YO: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(); //esta ya viene

//Configurar las opciones

builder.Services.ConfigureOptions<ConfigureSwaggerOptions>();

var app = builder.Build(); //Esta ya viene

//Configurar endpoints para swagger DOcs para cada una de las versiones de la API

var apiVersionDescriptionProvider=app.Services.GetRequiredService<IApiVersionDescriptionProvider>();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(options =>

{

foreach (var description in apiVersionDescriptionProvider.ApiVersionDescriptions)

{

options.SwaggerEndpoint(

$"/swagger/{description.GroupName}/swagger.json",

description.GroupName.ToUpperInvariant();

};

// ESTO ES LO QUE SE OBTIENE: /swagger/v2/swagger.json

});

}

Clase 10: Logs

Es un proceso de registrar eventos que suceden durante la ejecución de un programa.

Tracing, traceo o seguimiento. Sigue el flujo de datos de una app. Se deja constancia por dónde va, para ver dónde ha podido romperse.

Hay varios proveedores de esto:

Proveedores internos:

-Console.Writeline.

-Debbug.

-EventSource

-EventLog

Son opciones integradas

Proveedores externos:

-SeriLog.

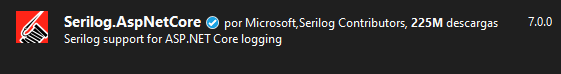
-Log4Net.

-Nlog.

Hay distintos niveles de logueo: info, error, warning, critical, etc.

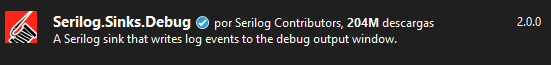
Se puede guardar en BBDD o en archivos de texto plano. Elastic Search lo nombra como la papa, pero no se bien qué es y no se verá aquí.

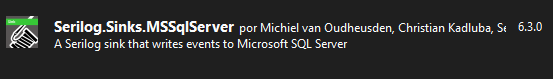
Paquetes NuGet SeriLog:











Program Cs

using Serilog;

builder.Host.UseSerilog((hostbuilderCtx, loggerConf) =>

{

loggerConf

.WriteTo.Console()

.WriteTo.Debug()

.ReadFrom.Configuration(hostbuilderCtx.Configuration);

});

//Decirle a la app que use el serilog (MUCHO MAS ABAJO)

app.UseSerilogRequestLogging();

AppSettings

"Serilog": {

"MinimumLevel": {

"Default": "Verbose",

"Override": {

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"WriteTo": [

{

"Name": "File",

"Args": {

"path": "./logs/log-.txt",

"rollingInterval": "Day"

}

},

{

"Name": "MSSqlServer",

"Args": {

"connectionString": "BolsaDeDatos",

"sinkOptionsSection": {

"tableName": "Logs",

"schemaName": "EventLogging",

"autoCreateSqlTable": true,

"batchPostingLimit": 1000,

"period": "0.00:00:30"

},

"RetrictedToMinimumLevel": "Verbose"

}

}

]

}

WeatherController

Ya viene con este constructor:

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

Se agrega esto:

En este método: public IEnumerable<WeatherForecast> Get()

{

\_logger.LogTrace($"{nameof(WeatherForecastController)} - {(nameof(Get))} - Trace Level Log");

\_logger.LogDebug($"{nameof(WeatherForecastController)} - {(nameof(Get))} - Debug Level Log");

\_logger.LogInformation($"{nameof(WeatherForecastController)} - {(nameof(Get))} - Information Level Log");

\_logger.LogWarning($"{nameof(WeatherForecastController)} - {(nameof(Get))} - Information Level Log");

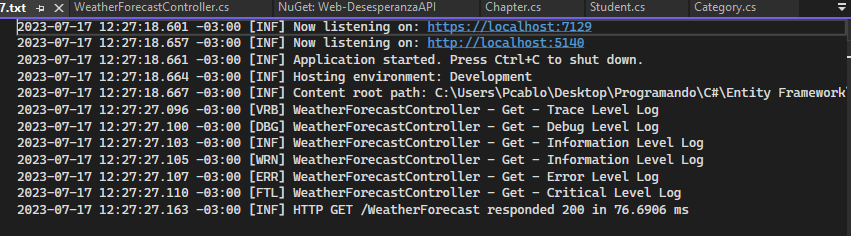
\_logger.LogError($"{nameof(WeatherForecastController)} - {(nameof(Get))} - Error Level Log");

\_logger.LogCritical($"{nameof(WeatherForecastController)} - {(nameof(Get))} - Critical Level Log");

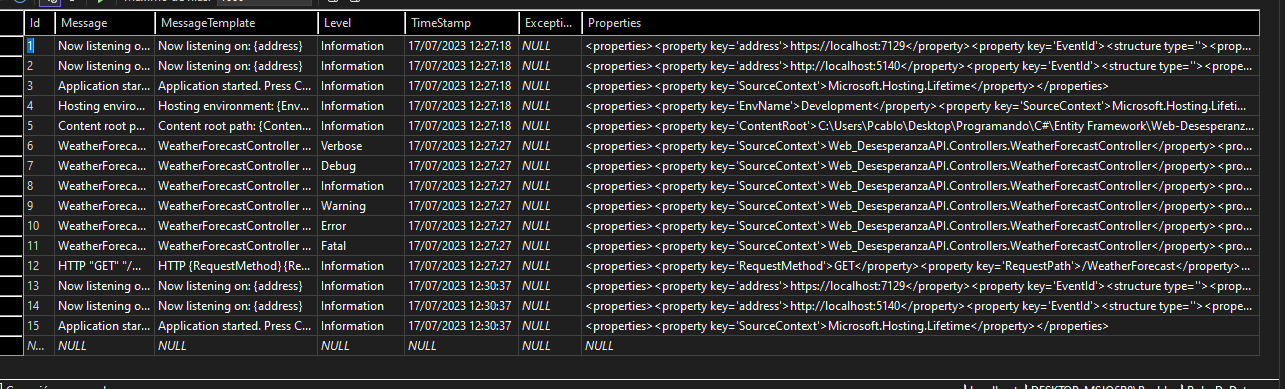
Devolvería primero el nombre del controlador y luego el método, mas la leyenda escrita por nosotros, indicando el nivel del log.

Crea carpeta Logs

Así queda el Txt



Así la BBDD



UniversityContext

private readonly ILoggerFactory \_loggerFactory;

public UniversityContext(DbContextOptions<UniversityContext> opciones, ILoggerFactory loggerFactory) : base(opciones)

{

\_loggerFactory = loggerFactory;

}

Replicamos lo de weathercontroller y sumamos esto:

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var logger = \_loggerFactory.CreateLogger<UniversityContext>();

//optionsBuilder.LogTo(d => logger.Log(LogLevel.Information, d, new[] { DbLoggerCategory.Database.Name }));

//optionsBuilder.EnableSensitiveDataLogging();

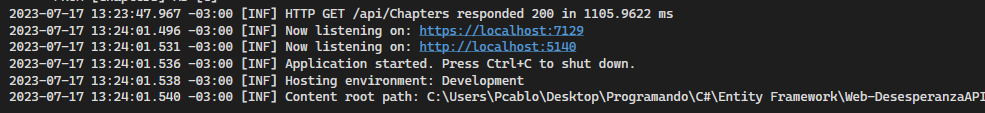
//Con esas 2 lineas ya funciona, pero no hace logs personalizados

optionsBuilder.LogTo(d => logger.Log(LogLevel.Information, d, new[] { DbLoggerCategory.Database.Name }), LogLevel.Information)

.EnableSensitiveDataLogging()

.EnableDetailedErrors();

}



Clase 11: Flujo asíncrono

Es la ejecución de tareas de forma asíncrona. No se ejecuta de forma secuencial sino en paralelo.

Las tareas no asíncronas se ejecutan secuencialmente. Al terminar una, se pasa a la siguiente.

Asincrónicamente, se tarda lo que tarda la tarea mas extensa.

En .Net se usa el modelo TAP (Patrón asincrónico basado en tareas).

App de consola.

Con esto se pueden hacer análisis de tiempos:

using System.Diagnostics;

//Iniciamos un contador de tiempo síncrono

Stopwatch sw = Stopwatch.StartNew();

sw.Start();

/\*

Console.WriteLine("\nBienvenido a la calculadora de Hipotécas Síncrona");

var aniosDeVidaLaboral = CalculadoraHipotecaSync.ObtenerAnosDeVidaLaboral();

Console.WriteLine("\nAños de vida laboral obtenidos: "+ aniosDeVidaLaboral);

var esTipoDeContratoIndefinido = CalculadoraHipotecaSync.EsTipoDeContratoIndefinido();

Console.WriteLine("\nTipo de contrato indefinido: "+ esTipoDeContratoIndefinido);

var sueldoNeto = CalculadoraHipotecaSync.ObtenerSueldoNeto();

Console.WriteLine("\nSueldo neto: $"+ sueldoNeto);

var gastos = CalculadoraHipotecaSync.ObtenerGastosMensuales();

Console.WriteLine("\nGastos mensuales: $"+gastos);

var hipotecaConcedida = CalculadoraHipotecaSync.AnalizarInformacionParaConcederHipoteca(aniosDeVidaLaboral, esTipoDeContratoIndefinido, sueldoNeto, gastos, cantidadSolicitada:50000, aniosAPagar: 25);

//Signigica: si es true, se guarda Aprobada Y SI NO Denegada

var resultado = hipotecaConcedida ? "Aprobada" : "Denegada";

Console.WriteLine($"\nAnálisis finalizado. Su solicitud de hipotéca ha sido: {resultado} ");

sw.Stop();

Console.WriteLine("\nLa operación ha tardado: "+ sw.Elapsed);

\*/

//Reiniciamos el contador de tiempo. Tareas asíncronas:

sw.Restart();

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\nBienvenido a la calculadora de Hipotécas Asíncrona");

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Task <int> aniosDeVidaLaboralTask = CalculadoraHipotecaAsync.ObtenerAnosDeVidaLaboral();

Task <bool> esTipoDeContratoIndefinidoTask = CalculadoraHipotecaAsync.EsTipoDeContratoIndefinido();

Task <int> sueldoNetoTask = CalculadoraHipotecaAsync.ObtenerSueldoNeto();

Task <int> gastosTask = CalculadoraHipotecaAsync.ObtenerGastosMensuales();

//Arma una lista con estas tareas.

var analisisHipotecasTasks = new List<Task>

{

aniosDeVidaLaboralTask,

esTipoDeContratoIndefinidoTask,

sueldoNetoTask,

gastosTask

};

//Se ejecutará mientras haya tareas pendientes

while (analisisHipotecasTasks.Any())

{

Task tareaFinalizada = await Task.WhenAny(analisisHipotecasTasks);

if (tareaFinalizada == aniosDeVidaLaboralTask)

{

Console.WriteLine("\nAños de vida laboral obtenidos: " + aniosDeVidaLaboralTask.Result);

}

else if (tareaFinalizada == esTipoDeContratoIndefinidoTask)

{

Console.WriteLine("\nTipo de contrato indefinido: " + esTipoDeContratoIndefinidoTask.Result);

}

else if (tareaFinalizada == sueldoNetoTask)

{

Console.WriteLine("\nSueldo neto: $" + sueldoNetoTask.Result);

}

else if (tareaFinalizada == gastosTask)

{

Console.WriteLine("\nGastos mensuales: $" + gastosTask.Result);

}

//Esto hace que se vaya printeado en el orden en que van terminando.

//Si se completa alguna se printea eso, y luego la quita de la lista.

//Solo si sale del bucle (es decir, si terimna todas las tareas), pasa a la tarea siguiente.

//Igual sin el bucle la app no se rompe.

analisisHipotecasTasks.Remove(tareaFinalizada);

}

var hipotecaConcedidaAsync = CalculadoraHipotecaAsync.AnalizarInformacionParaConcederHipoteca(aniosDeVidaLaboralTask.Result, esTipoDeContratoIndefinidoTask.Result, sueldoNetoTask.Result, gastosTask.Result, cantidadSolicitada: 50000, aniosAPagar: 25);

var resultadoAsync = hipotecaConcedidaAsync ? "Aprobada" : "Denegada";

Console.WriteLine($"\nAnálisis asincrónico finalizado. Su solicitud de hipoteca ha sido: {resultadoAsync} ");

sw.Stop();

Console.WriteLine($"\nLa operación asíncrona ha durado {sw.Elapsed}");

Console.ReadLine();

Clase asíncrona:

public static class CalculadoraHipotecaAsync

{

public static async Task<int> ObtenerAnosDeVidaLaboral()

{

Console.WriteLine("\nObteniendo años de vida laboral...");

await Task.Delay(5000);

//Simula un proceso que demora 5 segundos.

return new Random().Next(1, 35);

}

public static async Task <bool> EsTipoDeContratoIndefinido()

{

Console.WriteLine("\nVerificando si el tipo de contrato es indefinido...");

await Task.Delay(5000);

return (new Random().Next(1, 10)) % 2 == 0;

}

public static async Task <int> ObtenerSueldoNeto()

{

Console.WriteLine("\nObteniendo sueldo neto...");

await Task.Delay(4000);

return (new Random().Next(800, 6001));

}

public static async Task <int> ObtenerGastosMensuales()

{

Console.WriteLine("\nObteniendo gastos mensuales...");

await Task.Delay(6000);

return (new Random().Next(200, 1000));

}

//Este no es asincrónico

public static bool AnalizarInformacionParaConcederHipoteca(

int anosDeVidaLaboral,

bool tipoDeContratoEsIndefinido,

int sueldoNeto,

int gastosMensuales,

int cantidadSolicitada,

int aniosAPagar

)

{

Console.WriteLine("\nAnalizando información para conceder hipóteca...");

if (anosDeVidaLaboral < 2)

{

return false;

}

var cuota = (cantidadSolicitada / aniosAPagar) / 12;

if (cuota >= sueldoNeto || cuota > (sueldoNeto / 2))

{

return false;

}

var PorcentajeGastoSobreSueldo = ((gastosMensuales \* 100) / sueldoNeto);

if (PorcentajeGastoSobreSueldo > 30)

{

return false;

}

if ((cuota + gastosMensuales) >= sueldoNeto)

{

return false;

}

if (!tipoDeContratoEsIndefinido)

{

if ((cuota + gastosMensuales) > (sueldoNeto / 3))

{

return false;

}

else

{

return true;

}

}

return true;

}

}

Clase sincrónica

public static class CalculadoraHipotecaSync

{

public static int ObtenerAnosDeVidaLaboral()

{

Console.WriteLine("\nObteniendo años de vida laboral...");

Task.Delay(5000).Wait();

//Simula un proceso que demora 5 segundos.

return new Random().Next(1,35);

}

public static bool EsTipoDeContratoIndefinido()

{

Console.WriteLine("\nVerificando si el tipo de contrato es indefinido...");

Task.Delay(5000).Wait();

return (new Random().Next(1,10)) % 2 == 0;

}

public static int ObtenerSueldoNeto()

{

Console.WriteLine("\nObteniendo sueldo neto...");

Task.Delay(4000).Wait();

return (new Random().Next(800, 6001));

}

public static int ObtenerGastosMensuales()

{

Console.WriteLine("\nObteniendo gastos mensuales...");

Task.Delay(6000).Wait();

return (new Random().Next(200, 1000));

}

public static bool AnalizarInformacionParaConcederHipoteca(

int anosDeVidaLaboral,

bool tipoDeContratoEsIndefinido,

int sueldoNeto,

int gastosMensuales,

int cantidadSolicitada,

int aniosAPagar

)

{

Console.WriteLine("\nAnalizando información para conceder hipóteca...");

if (anosDeVidaLaboral < 2)

{

return false;

}

var cuota = (cantidadSolicitada / aniosAPagar) / 12;

if (cuota>=sueldoNeto || cuota> (sueldoNeto / 2))

{

return false;

}

var PorcentajeGastoSobreSueldo = ((gastosMensuales \* 100) / sueldoNeto);

if (PorcentajeGastoSobreSueldo > 30)

{

return false;

}

if ((cuota + gastosMensuales) >= sueldoNeto)

{

return false;

}

if (!tipoDeContratoEsIndefinido)

{

if ((cuota + gastosMensuales) > (sueldoNeto / 3))

{

return false;

}

else

{

return true;

}

}

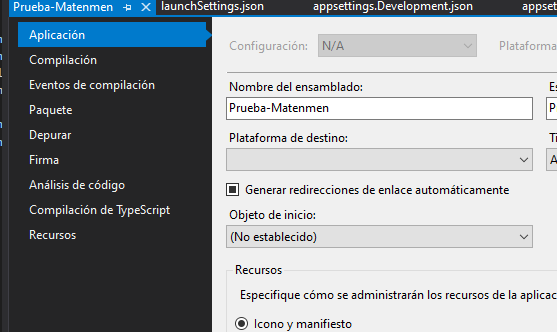
return true;

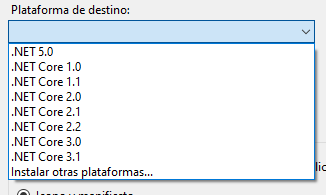
}

}

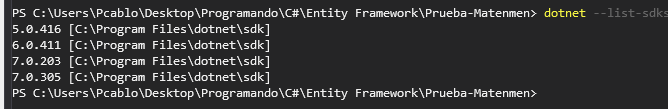
<TargetFramework>net6.0</TargetFramework>

Si pongo la línea de arriba, en Properties no aparece nada.





Nota al pie, ya instalé otras versiones de .Net

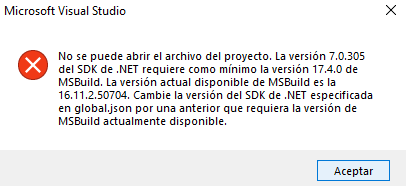


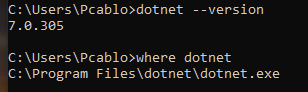
Comandos Dotnet

Dotnet –version

Where dotnet

Dotnet –list-sdks





IActionResult

Devuelve una vista o una redirección.

Random Potatoe!!!

return new Random().Next(1,35);

Acá el return es un booleano. Si se cumple esa condición se devuelve true;

return (new Random().Next(1,10)) % 2 == 0;