

En este resumen ejecutivo de creacion de aplicaciones WebApi se aplicara el modelo MVC de capas en la cual se implementara un Apirest.

#### Requerimientos:

- \* Visual studio code
- \* SDK Dotnet Version 7.0
- \* Conocimientos en C#

## CREACION DEL PROYECTO

Para la creacion del proyecto se utilizará el terminal del sistema operativo ya sea Powershell de windows o la terminal de Linux

#### Consideraciones Iniciales

Inicie el terminal del sistema operativo que se encuentre utilizando o de Preferencia.

Cree una carpeta contenedora de todos los proyectos que va a desarrollar. Puede usar el comando mkdir Nombre de la carpeta.

Cree un repositorio en Git para llevar un control progresivo del trabajo.

En la carpeta que creo clone el repositorio puede utilizar el comando git clone [Repositorio Remoto]. Recuerde que debe tener debidamente configurado el git en la computadora donde se encuentre desarrollando el proyecto.

Ingrese a la carpeta local del repositorio que ha clonado previamente.

#### Creacion del proyecto

\* Cree la solucion del proyecto (Recuerde que la solucion es la que contendra todos los proyectos que son utilizados para el desarrollo del Backend. el comando para la creación de la solucion es dotnet new sln (En nombre de la solucion de asigna teniendo en cuenta el nombre de la carpeta contenedora).

\* Teniendo en cuenta el modelo de 4 capas cree los siguientes proyectos:

- \* Aplicacion (Tipo classlib). dotnet new classlib -o Aplicacion
- \* Dominio (Tipo classlib). dotnet new classlib -o Dominio
- \* Persistencia (Tipo classlib). dotnet new classlib -o Persistencia
- \* Api (Tipo WebApi). dotnet new webapi classlib -o Api[NombreProyecto]

#### Agregar los proyectos a la Solución

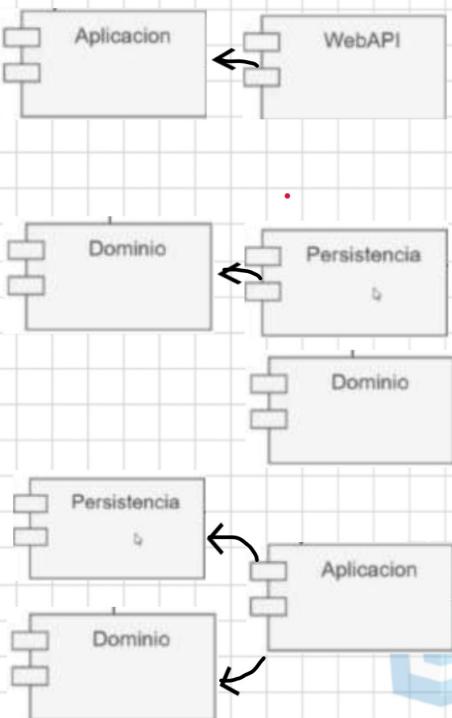
Sintaxis

dotnet sln add [Proyecto]

- dotnet sln add .\Aplicacion
- dotnet sln add .\Dominio
- dotnet sln add .\Persistencia
- dotnet sln add .\Api

Nota: Estos comandos se ejecutan desde la raiz del proyecto.

## Referencias entre proyectos



```
PS D:\NetCore\incidencias-app> cd ..\Aplicacion\  
PS D:\NetCore\incidencias-app\Aplicacion> dotnet add reference ..\Dominio\  
Se ha agregado la referencia "..\Dominio\Dominio.csproj" al proyecto.  
PS D:\NetCore\incidencias-app\Aplicacion> dotnet add reference ..\Persistencia\  
Se ha agregado la referencia "..\Persistencia\Persistencia.csproj" al proyecto.  
PS D:\NetCore\incidencias-app\Aplicacion> cd ..  
PS D:\NetCore\incidencias-app> cd ..\ApiIncidencias\  
PS D:\NetCore\incidencias-app\ApiIncidencias> dotnet add reference ..\Aplicacion\  
Se ha agregado la referencia "..\Aplicacion\Aplicacion.csproj" al proyecto.  
PS D:\NetCore\incidencias-app\ApiIncidencias> cd ..  
PS D:\NetCore\incidencias-app> cd persistencia  
PS D:\NetCore\incidencias-app\persistencia> dotnet add reference ..\Dominio\  
Se ha agregado la referencia "..\Dominio\Dominio.csproj" al proyecto.  
PS D:\NetCore\incidencias-app\persistencia> cd ..  
PS D:\NetCore\incidencias-app> code .
```

## Instalacion de paquetes



Microsoft.AspNetCore.Authentication.JwtBearer  
Microsoft.AspNetCore.OpenApi  
Microsoft.EntityFrameworkCore  
Microsoft.EntityFrameworkCore.Design  
Microsoft.Extensions.DependencyInjection  
System.IdentityModel.Tokens.Jwt



Microsoft.EntityFrameworkCore  
Pomelo.EntityFrameworkCore.MySql



FluentValidation.AspNetCore  
itext7.pdfhtml  
Microsoft.EntityFrameworkCore



Nota. Recuerde que la instalacion de los paquetes la puede realizar usando el Administrador de paquetes Nuget o el terminal dependiendo del sistema operativo anfitrion. Si realiza la instalacion desde el terminal debe asegurarse que se encuentre ubicado en la carpeta de cada proyecto segun grafico anterior.

Comandos de instalacion de paquetes : <https://www.nuget.org/>

## WebApi

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 7.0.10  
dotnet add package Microsoft.EntityFrameworkCore --version 7.0.10  
dotnet add package Microsoft.EntityFrameworkCore.Design --version 7.0.10  
dotnet add package Microsoft.Extensions.DependencyInjection --version 7.0.0  
dotnet add package System.IdentityModel.Tokens.Jwt --version 6.32.3
```

## Persistencia

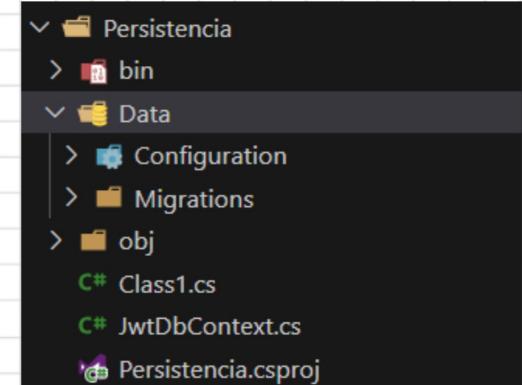
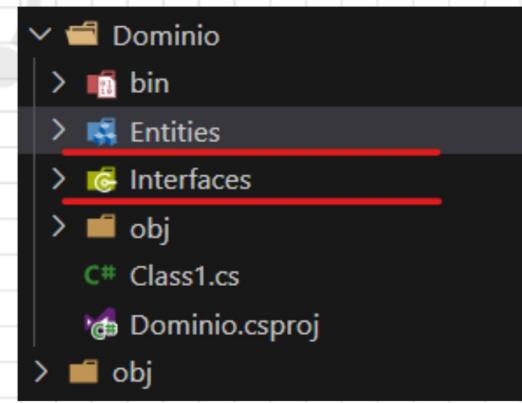
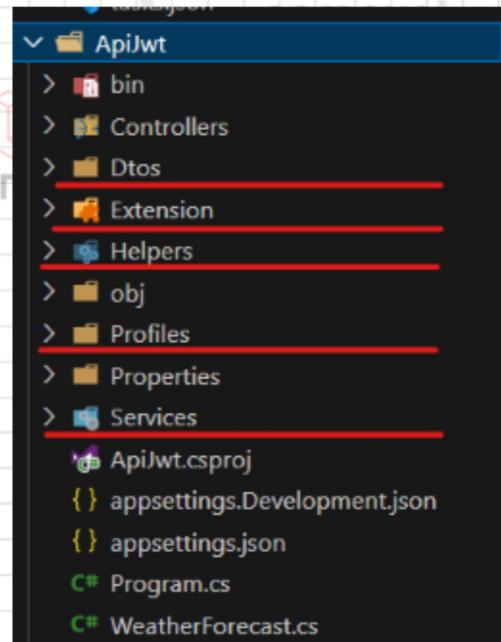
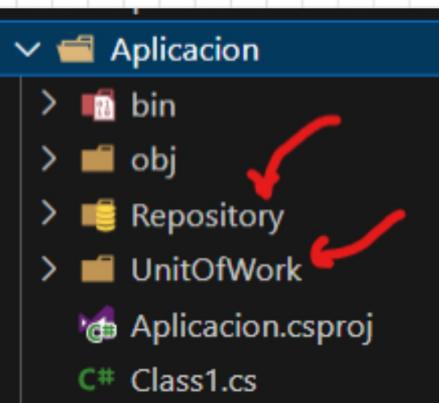
```
dotnet add package Pomelo.EntityFrameworkCore.MySql --version 7.0.0  
dotnet add package Microsoft.EntityFrameworkCore --version 7.0.10
```

## Dominio

```
dotnet add package FluentValidation.AspNetCore --version 11.3.0  
dotnet add package itext7.pdfhtml --version 5.0.1  
dotnet add package Microsoft.EntityFrameworkCore --version 7.0.10
```

**Nota. Recuerde ejecutar cada uno de los comandos desde cada proyecto e instalar la version compatible con el SDK. En el caso del documento SDK Ver 7.0**

## Estructura de carpetas Adicionales



# HOW TO DO JWT

## IMPLEMENTAR JWT EN UN PROYECTO NETCORE

- \* Abra el proyecto donde desea implementar JWT o Cree un nuevo proyecto
- \* Para el caso práctico del How to do crearemos un nuevo proyecto llamado JwtDemo. Recomendación inicie el proyecto en Git <https://github.com/trainingLeader/jwtdemo-app.git>

```
PS D:\NetCore> git clone https://github.com/trainingLeader/jwtdemo-app.git
```

```
PS D:\NetCore\jwtdemo-app> dotnet new sln
```

La plantilla "Archivo de la solución" se creó correctamente.

```
PS D:\NetCore\jwtdemo-app> dotnet new classlib -o Application
```

```
PS D:\NetCore\jwtdemo-app> dotnet new classlib -o Domain
```

```
PS D:\NetCore\jwtdemo-app> dotnet new classlib -o Persistence
```

```
PS D:\NetCore\jwtdemo-app> dotnet new webapi -o ApiJwt
```

- \* Agregue los proyectos creados anteriormente a la solución principal

```
PS D:\NetCore\jwtdemo-app> dotnet sln add .\Application\
```

```
PS D:\NetCore\jwtdemo-app> dotnet sln add .\Domain\
```

```
PS D:\NetCore\jwtdemo-app> dotnet sln add .\Persistence\
```

```
PS D:\NetCore\jwtdemo-app> dotnet sln add .\ApiJwt\
```

- \* Puede comprobar los proyectos que están asociados a la solución principal con el siguiente comando

```
PS D:\NetCore\jwtdemo-app> dotnet sln list
```

Proyectos

-----

Application\Application.csproj

Domain\Domain.csproj

Persistence\Persistence.csproj

ApiJwt\ApiJwt.csproj

```
PS D:\NetCore\jwtdemo-app>
```

- \* Abra el proyecto en Visual Studio Code desde la consola. Recuerde que se debe ubicar en la carpeta principal del proyecto.

```
PS D:\NetCore\jwtdemo-app> code .
```

Referenciar los proyectos. Teniendo en cuenta la estructura de capas use el comando reference para referenciar los proyectos. Tenga en cuenta las indicaciones de la pagina 2 del documento.

```
PS D:\NetCore\jwtdemo-app> cd .\Application  
PS D:\NetCore\jwtdemo-app\Application> dotnet add reference ..\Domain\  
Se ha agregado la referencia "..\Domain\Domain.csproj" al proyecto.  
PS D:\NetCore\jwtdemo-app\Application> dotnet add reference ..\Persistence\  
Se ha agregado la referencia "..\Persistence\Persistence.csproj" al proyecto.  
PS D:\NetCore\jwtdemo-app\Application> cd ..  
PS D:\NetCore\jwtdemo-app> cd .\ApiJwt\  
PS D:\NetCore\jwtdemo-app\ApiJwt> dotnet add reference ..\Application\  
Se ha agregado la referencia "..\Application\Application.csproj" al proyecto.  
PS D:\NetCore\jwtdemo-app\ApiJwt> cd..  
PS D:\NetCore\jwtdemo-app> cd .\Persistence\  
PS D:\NetCore\jwtdemo-app\Persistence> dotnet add reference ..\Domain\  
Se ha agregado la referencia "..\Domain\Domain.csproj" al proyecto.  
PS D:\NetCore\jwtdemo-app\Persistence>
```

Realice la instalacion de cada uno de los paquetes necesarios para construir el proyecto. Ver Pagina 3. Copie cada uno de los comandos teniendo en cuenta la carpeta del proyecto destino.

**Nota. Cuando finalice la instalacion de todos los paquetes se recomienda ejecutar los comandos:  
dotnet restore y dotnet build.**

```
PS D:\NetCore\jwtdemo-app\Domain> dotnet restore  
PS D:\NetCore\jwtdemo-app\Domain> dotnet build  
Versión de MSBuild 17.7.1+971bf70db para .NET  
Determinando los proyectos que se van a restaurar...  
Todos los proyectos están actualizados para la restauración.  
Domain -> D:\NetCore\jwtdemo-app\Domain\bin\Debug\net7.0\Domain.dll
```

Compilación correcta.

0 Advertencia(s)  
0 Errores

Tiempo transcurrido 00:00:04.47

El orden en la implementacion de JWT no es estricto. En el how to do llevaremos un orden recomendado pero no estricto. Recuerde crear la estructura de directorios recomendado en la pagina 3 apartado **Estructura de carpetas**

...

1. Cree una nueva carpeta llamada Services en el proyecto de tipo WebApi
2. Cree la clase BaseEntity en la carpeta Entities que se encuentra en Domain

namespace Domain.Entities;

```
public class BaseEntity
{
    public int Id { get; set; }
}
```

3. Cree la Interface IGenericRepository en la carpeta Interfaces que se encuentra en Domain

```
using System.Linq.Expressions;
using Domain.Entities;
```

namespace Domain.Interfaces;

```
public interface IGenericRepository<T> where T : BaseEntity
{
    Task<T> GetByIdAsync(string id);
    Task<IEnumerable<T>> GetAllAsync();
    IEnumerable<T> Find(Expression<Func<T, bool>> expression);
    void Add(T entity);
    void AddRange(IEnumerable<T> entities);
    void Remove(T entity);
    void RemoveRange(IEnumerable<T> entities);
    void Update(T entity);
}
```

4. Cree el archivo de contexto en Persistence

```
using System.Reflection;
using Microsoft.EntityFrameworkCore;
namespace Persistence;
public class JwtApplicationContext : DbContext
{
    public JwtApplicationContext(DbContextOptions options) : base(options)
    {
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
    }
}
```

5. Cree las entidades de User, Rol y UserRol. Estas entidades se deben crear en la carpeta Entities que se encuentra en la carpeta Domain

```
namespace Domain.Entities;  
  
public class UserRol  
{  
    public int Usuarioid { get; set; }  
    public User Usuario { get; set; }  
    public int RolId { get; set; }  
    public Rol Rol { get; set; }  
}
```

```
namespace Domain.Entities;  
  
public class Rol : BaseEntity  
{  
    public string Nombre { get; set; }  
    public ICollection<User> Users { get; set; } = new HashSet<User>();  
    public ICollection<UserRol> UsersRols { get; set; }  
}
```

```
namespace Domain.Entities;  
public class User : BaseEntity  
{  
    public string Username { get; set; }  
    public string Email { get; set; }  
    public string Password { get; set; }  
    public ICollection<Rol> Rols { get; set; } = new HashSet<Rol>();  
    public ICollection<RefreshToken> RefreshTokens { get; set; } = new HashSet<RefreshToken>();  
    public ICollection<UserRol> UsersRols { get; set; }  
}
```

```
namespace Domain.Entities;  
  
public class RefreshToken : BaseEntity  
{  
    public int UserId { get; set; }  
    public User User { get; set; }  
    public string Token { get; set; }  
    public DateTime Expires { get; set; }  
    public bool IsExpired => DateTime.UtcNow >= Expires;  
    public DateTime Created { get; set; }  
    public DateTime? Revoked { get; set; }  
    public bool IsActive => Revoked == null && !IsExpired;  
}
```

## 6. Genere implementación del repositorio genérico.

```
using System.Linq.Expressions;
using Domain.Entities;
using Domain.Interfaces;
using Microsoft.EntityFrameworkCore;
using Persistence;
namespace Application.Repository;
public class GenericRepository<T> : IGenericRepository<T> where T : BaseEntity
{
    private readonly JwtApplicationContext _context;

    public GenericRepository(JwtApplicationContext context)
    {
        _context = context;
    }
    public virtual void Add(T entity)
    {
        _context.Set<T>().Add(entity);
    }
    public virtual void AddRange(IEnumerable<T> entities)
    {
        _context.Set<T>().AddRange(entities);
    }
    public virtual IEnumerable<T> Find(Expression<Func<T, bool>> expression)
    {
        return _context.Set<T>().Where(expression);
    }
    public virtual async Task<IEnumerable<T>> GetAllAsync()
    {
        return await _context.Set<T>().ToListAsync();
    }
    public virtual async Task<T> GetByIdAsync(int id)
    {
        return await _context.Set<T>().FindAsync(id);
    }
    public virtual async Task<T> GetByIdAsync(string id)
    {
        return await _context.Set<T>().FindAsync(id);
    }
    public virtual void Remove(T entity)
    {
        _context.Set<T>().Remove(entity);
    }
    public virtual void RemoveRange(IEnumerable<T> entities)
    {
        _context.Set<T>().RemoveRange(entities);
    }
    public virtual void Update(T entity)
    {
        _context.Set<T>()
            .Update(entity);
    }
}
```

**7. Agregue los DbSet al Archivo de contexto.**

```
using System.Reflection;
using Domain.Entities;
using Microsoft.EntityFrameworkCore;

namespace Persistence;

public class JwtApplicationContext : DbContext
{
    public JwtApplicationContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Rol> Rols { get; set; }
    public DbSet<User> Users { get; set; }
    public DbSet<UserRol> UsersRols { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
    }
}
```

**8. Agregue el DbContext en el contenedor de dependencias Program.cs.**

```
using Microsoft.EntityFrameworkCore;
using Persistence;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddDbContext<JwtApplicationContext>(options =>
{
    string connectionString = builder.Configuration.GetConnectionString("ConexMysql");
    options.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```



## 9. Genera los archivos de configuración de cada una de las entidades.

```
using Domain.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Persistence.Data.Configuration;

public class UserConfiguration : IEntityTypeConfiguration<User>
{
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.ToTable("user");

        builder.Property(p => p.Id)
            .IsRequired();

        builder.Property(p => p.Username)
            .HasColumnName("username")
            .HasColumnType("varchar")
            .HasMaxLength(50)
            .IsRequired();

        builder.Property(p => p.Password)
            .HasColumnName("password")
            .HasColumnType("varchar")
            .HasMaxLength(255)
            .IsRequired();

        builder.Property(p => p.Email)
            .HasColumnName("email")
            .HasColumnType("varchar")
            .HasMaxLength(100)
            .IsRequired();

        builder
            .HasMany(p => p.Rols)
            .WithMany(r => r.Users)
            .UsingEntity<UserRol>()

            j => j
            .hasOne(pt => pt.Rol)
            .WithMany(t => t.UsersRols)
            .HasForeignKey(ut => ut.RolId);

            j => j
            .hasOne(et => et.Usuario)
            .WithMany(et => et.UsersRols)
            .HasForeignKey(el => el.UsuarioId);

        builder.HasMany(p => p.RefreshTokens)
            .WithOne(p => p.User)
            .HasForeignKey(p => p.UserId);
    }
}
```

```
using Domain.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Persistence.Data.Configuration;

public class RolConfiguration : IEntityTypeConfiguration<Rol>
{
    public void Configure(EntityTypeBuilder<Rol> builder)
    {
        // Aquí puedes configurar las propiedades de la entidad
        // utilizando el objeto 'builder'.
        builder.ToTable("rol");
        builder.Property(p => p.Id)
            .IsRequired();
        builder.Property(p => p.Nombre)
            .HasColumnName("rolName")
            .HasColumnType("varchar")
            .HasMaxLength(50)
            .IsRequired();
    }
}
```

```
using Domain.Entities;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Persistence.Data.Configuration;

public class RefreshTokenConfiguration : IEntityTypeConfiguration<RefreshToken>
{
    public void Configure(EntityTypeBuilder<RefreshToken> builder)
    {
        builder.ToTable("RefreshToken");
    }
}
```

9. Agregue la cadena de conexión en el archivo appsetting y appsetting-Development. Estos son archivos de configuración de variables de entorno. El primero se aplica en producción(despliegue) y el segundo en desarrollo.

ApiJwt > {} appsettings.json > ...

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*",  
  "ConnectionStrings": {  
    "ConexSqlServer": "Data Source=localhost\\sqlexpress;Initial Catalog=db;Integrate Security=True",  
    "ConexMysql": "server=localhost;user=root;password=123456;database=jwtdemo"  
  }  
}
```

ApiJwt > {} appsettings.Development.json > ...

```
{  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "ConnectionStrings": {  
    "ConexSqlServer": "Data Source=localhost\\sqlexpress;Initial Catalog=db;Integrate Security=True",  
    "ConexMysql": "server=localhost;user=root;password=123456;database=jwtdemo"  
  }  
}
```

## 10. Ejecutando la primera migración.

```
dotnet ef migrations add InitialCreateMig --project .\Persistence\ --startup-project .\ApiJwt\ --output-dir ./Data/Migrations
```



Nombre de la migración.  
Cada vez que se realice una migración se debe colocar un nombre diferente.



Proyecto donde se va crear la migración.



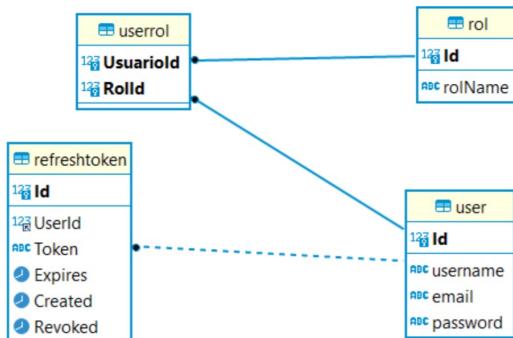
Proyecto que contiene los paquetes que permiten ejecutar la migración



Carpeta donde se van a crear las migraciones

## 11. Aplicando la migración : Recuerde que este comando se debe ejecutar desde la carpeta principal del proyecto

```
PS D:\NetCore\jwtdemo-app> dotnet ef database update --project .\Persistence\ --startup-project .\ApiJwt\
```



HTML

JS

## 12. Genere Interfaces de Rol, User y Unitofwork

```
using Domain.Entities;
```

```
namespace Domain.Interfaces;
```

```
public interface IUserRepository : IGenericRepository<User>
```

```
{
```

```
    Task<User> GetByUsernameAsync(string username);
```

```
    Task<User> GetByRefreshTokenAsync(string username);
```

```
}
```

```
using Domain.Entities;
```

```
namespace Domain.Interfaces;
```

```
public interface IRolRepository : IGenericRepository<Rol> {
```

```
}
```

```
namespace Domain.Interfaces;
```

```
public interface IUnitOfWork
```

```
{
```

```
    IRolRepository Roles { get; }
```

```
    IUserRepository Users { get; }
```

```
    Task<int> SaveAsync();
```

```
}
```

13. Creación de helpers. Los "helpers" (ayudantes o asistentes) son funciones, métodos o clases que se utilizan para realizar tareas comunes o repetitivas en la programación. Estas funciones o clases "ayudan" a simplificar y organizar el código, lo que puede hacer que el desarrollo de software sea más eficiente y menos propenso a errores.

```
namespace ApiJwt.Helpers;

public class Authorization
{
    public enum Roles
    {
        Administrator,
        Manager,
        Employee
    }

    public const Roles rol_default = Roles.Employee;
}
```

```
namespace ApiJwt.Helpers;

public class JWT
{
    public string Key { get; set; }
    public string Issuer { get; set; }
    public string Audience { get; set; }
    public double DurationInMinutes { get; set; }
}
```

14. Genere el archivo de extensión en la carpeta Extension en el proyecto WebApi. Una clase de extensión en .NET Core (y en el lenguaje C# en general) es una clase estática que contiene métodos estáticos diseñados para agregar funcionalidad adicional a clases existentes sin modificar directamente su código fuente. Estos métodos de extensión permiten extender la funcionalidad de tipos de datos sin la necesidad de herencia o modificación de la clase original.

```
using ApiJwt.Services;
using Domain.Entities;
using Domain.Interfaces;
using Microsoft.AspNetCore.Identity;

namespace ApiJwt.Extension;

public static class ApplicationServiceExtensions
{
    public static void ConfigureCors(this IServiceCollection services) =>
        services.AddCors(options =>
    {
        options.AddPolicy("CorsPolicy", builder =>
            builder.AllowAnyOrigin() //WithOrigins("https://domain.com")
                .AllowAnyMethod() //WithMethods("GET", "POST")
                .AllowAnyHeader()); //WithHeaders("accept", "content-type")
    });
    public static void AddAplicacionServices(this IServiceCollection services)
    {
        services.AddScoped<IPasswordHasher<User>, PasswordHasher<User>>();
    }
}
```



15. Cree las clase que implementa las interfaces de rol y user.

```
using Domain.Entities;
using Domain.Interfaces;
using Microsoft.EntityFrameworkCore;
using Persistence;
namespace Application.Repository;
public class UserRepository : GenericRepository<User>, IUserRepository
{
    private readonly JwtApplicationContext _context;

    public UserRepository(JwtApplicationContext context) : base(context)
    {
        _context = context;
    }
    public async Task<User> GetByRefreshTokenAsync(string refreshToken)
    {
        return await _context.Users
            .Include(u => u.Rols)
            .Include(u => u.RefreshTokens)
            .FirstOrDefaultAsync(u => u.RefreshTokens.Any(t => t.Token == refreshToken));
    }
    public async Task<User> GetByUsernameAsync(string username)
    {
        return await _context.Users
            .Include(u => u.Rols)
            .Include(u => u.RefreshTokens)
            .FirstOrDefaultAsync(u => u.Username.ToLower() == username.ToLower());
    }
}
```

```
using Domain.Entities;
using Domain.Interfaces;
using Persistence;

namespace Application.Repository;

public class RolRepository : GenericRepository<Rol>, IRolRepository
{
    private readonly JwtApplicationContext _context;

    public RolRepository(JwtApplicationContext context) : base(context)
    {
        _context = context;
    }
}
```



## 16. Genera Dtos

```
using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations;
namespace ApiJwt.Dtos;
public class AddRoleDto
{
    [Required]
    public string Username { get; set; }
    [Required]
    public string Password { get; set; }
    [Required]
    public string Role { get; set; }
}
```

```
namespace ApiJwt.Dtos;
public class LoginDto
{
    [Required]
    public string Username { get; set; }
    [Required]
    public string Password { get; set; }
}
```

```
using System.Text.Json.Serialization;
```

```
namespace ApiJwt.Dtos;
```

```
public class DataUserDto
{
    public string Message { get; set; }
    public bool IsAuthenticated { get; set; }
    public string UserName { get; set; }
    public string Email { get; set; }
    public List<string> Roles { get; set; }
    public string Token { get; set; }
```

```
[JsonIgnore] // ->this attribute restricts the property to be shown in the result
public string RefreshToken { get; set; }
public DateTime RefreshTokenExpiration { get; set; }
}
```

```
using System.ComponentModel.DataAnnotations;
namespace ApiJwt.Dtos;
public class RegisterDto
{
    [Required]
    public string Name { get; set; }
    [Required]
    public string Lastname { get; set; }
    [Required]
    public string Email { get; set; }
    [Required]
    public string Username { get; set; }
    [Required]
    public string Password { get; set; }
}
```

## 17. Cree la interface de servicio IUserService y UserService

```
using ApiJwt.Dtos;  
  
namespace ApiJwt.Services;  
  
public interface IUserService  
{  
    Task<string> RegisterAsync(RegisterDto model);  
    Task<DataUserDto> GetTokenAsync(LoginDto model);  
    Task<string> AddRoleAsync(AddRoleDto model);  
    Task<DataUserDto> RefreshTokenAsync(string refreshToken);  
}
```

```
using System.IdentityModel.Tokens.Jwt;  
using System.Security.Claims;  
using System.Security.Cryptography;  
using System.Text;  
using ApiJwt.Dtos;  
using ApiJwt.Helpers;  
using Domain.Entities;  
using Domain.Interfaces;  
using Microsoft.AspNetCore.Identity;  
using Microsoft.Extensions.Options;  
using Microsoft.IdentityModel.Tokens;  
namespace ApiJwt.Services;  
public class UserService : IUserService  
{  
    private readonly JWT _jwt;  
    private readonly IUnitOfWork _unitOfWork;  
    private readonly IPasswordHasher<User> _passwordHasher;  
    public UserService(IUnitOfWork unitOfWork, IOptions<JWT> jwt, IPasswordHasher<User> passwordHasher)  
    {  
        _jwt = jwt.Value;  
        _unitOfWork = unitOfWork;  
        _passwordHasher = passwordHasher;  
    }
```

```
public async Task<string> RegisterAsync(RegisterDto registerDto)
{
    var user = new User
    {
        Email = registerDto.Email,
        Username = registerDto.Username
    };
    user.Password = _passwordHasher.HashPassword(user, registerDto.Password); //Encrypt password

    var existingUser = _unitOfWork.Users
        .Find(u => u.Username.ToLower() == registerDto.Username.ToLower())
        .FirstOrDefault();
    if (existingUser == null)
    {
        var rolDefault = _unitOfWork.Roles
            .Find(u => u.Nombre == Authorization.rol_default.ToString())
            .First();
        try
        {
            user.Rols.Add(rolDefault);
            _unitOfWork.Users.Add(user);
            await _unitOfWork.SaveAsync();
            return $"User {registerDto.Username} has been registered successfully";
        }
        catch (Exception ex)
        {
            var message = ex.Message;
            return $"Error: {message}";
        }
    }
    else
    {
        return $"User {registerDto.Username} already registered.";
    }
}
```

```
public async Task<DataUserDto> GetTokenAsync(LoginDto model)
{
    DataUserDto dataUserDto = new DataUserDto();
    var user = await _unitOfWork.Users
        .GetByUsernameAsync(model.Username);

    if (user == null)
    {
        dataUserDto.isAuthenticated = false;
        dataUserDto.Message = $"User does not exist with username {model.Username}.";;
        return dataUserDto;
    }

    var result = _passwordHasher.VerifyHashedPassword(user, user.Password, model.Password);

    if (result == PasswordVerificationResult.Success)
    {
        dataUserDto.isAuthenticated = true;
        JwtSecurityToken jwtSecurityToken = CreateJwtToken(user);
        dataUserDto.Token = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken);
        dataUserDto.Email = user.Email;
        dataUserDto.UserName = user.Username;
        dataUserDto.Roles = user.Rols
            .Select(u => u.Nombre)
            .ToList();

        if (user.RefreshTokens.Any(a => a.IsActive))
        {
            var activeRefreshToken = user.RefreshTokens.Where(a => a.IsActive == true).FirstOrDefault();
            dataUserDto.RefreshToken = activeRefreshToken.Token;
            dataUserDto.RefreshTokenExpiration = activeRefreshToken.Expires;
        }
        else
        {
            var refreshToken = CreateRefreshToken();
            dataUserDto.RefreshToken = refreshToken.Token;
            dataUserDto.RefreshTokenExpiration = refreshToken.Expires;
            user.RefreshTokens.Add(refreshToken);
            _unitOfWork.Users.Update(user);
            await _unitOfWork.SaveChangesAsync();
        }
        return dataUserDto;
    }

    dataUserDto.isAuthenticated = false;
    dataUserDto.Message = $"Credenciales incorrectas para el usuario {user.Username}.";;
    return dataUserDto;
}
```

```
public async Task<string> AddRoleAsync(AddRoleDto model)
{
    var user = await _unitOfWork.Users.GetByUsernameAsync(model.Username);
    if (user == null)
    {
        return $"User {model.Username} does not exists.";
    }
    var result = _passwordHasher.VerifyHashedPassword(user, user.Password, model.Password);
    if (result == PasswordVerificationResult.Success)
    {
        var rolExists = _unitOfWork.Roles.Find(u => u.Nombre.ToLower() == model.Role.ToLower())
            .FirstOrDefault();
        if (rolExists != null)
        {
            var userHasRole = user.Rols.Any(u => u.Id == rolExists.Id);
            if (userHasRole == false)
            {
                user.Rols.Add(rolExists);
                _unitOfWork.Users.Update(user);
                await _unitOfWork.SaveAsync();
            }
            return $"Role {model.Role} added to user {model.Username} successfully.";
        }
        return $"Role {model.Role} was not found.";
    }
    return $"Invalid Credentials";
}
public async Task<DataUserDto> RefreshTokenAsync(string refreshToken)
{
    var dataUserDto = new DataUserDto();
    var usuario = await _unitOfWork.Users
        .GetByRefreshTokenAsync(refreshToken);
    if (usuario == null)
    {
        dataUserDto.isAuthenticated = false;
        dataUserDto.Message = $"Token is not assigned to any user.";
        return dataUserDto;
    }
    var refreshTokenBd = usuario.RefreshTokens.Single(x => x.Token == refreshToken);
    if (!refreshTokenBd.IsActive)
    {
        dataUserDto.isAuthenticated = false;
        dataUserDto.Message = $"Token is not active.";
        return dataUserDto;
    }
    //Revoke the current refresh token and
    refreshTokenBd.Revoked = DateTime.UtcNow;
    //generate a new refresh token and save it in the database
    var newRefreshToken = CreateRefreshToken();
    usuario.RefreshTokens.Add(newRefreshToken);
    _unitOfWork.Users.Update(usuario);
    await _unitOfWork.SaveAsync();
    //Generate a new Json Web Token
    dataUserDto.isAuthenticated = true;
    JwtSecurityToken jwtSecurityToken = CreateJwtToken(usuario);
    dataUserDto.Token = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken);
    dataUserDto.Email = usuario.Email;
    dataUserDto.UserName = usuario.Username;
    dataUserDto.Roles = usuario.Rols.Select(u => u.Nombre).ToList();
    dataUserDto.RefreshToken = newRefreshToken.Token;
    dataUserDto.RefreshTokenExpiration = newRefreshToken.Expires;
    return dataUserDto;
}
```



```
private RefreshToken CreateRefreshToken()
{
    var randomNumber = new byte[32];
    using (var generator = RandomNumberGenerator.Create())
    {
        generator.GetBytes(randomNumber);
        return new RefreshToken
        {
            Token = Convert.ToBase64String(randomNumber),
            Expires = DateTime.UtcNow.AddDays(10),
            Created = DateTime.UtcNow
        };
    }
}

private JwtSecurityToken CreateJwtToken(User usuario)
{
    var roles = usuario.Rols;
    var roleClaims = new List<Claim>();
    foreach (var role in roles)
    {
        roleClaims.Add(new Claim("roles", role.Nombre));
    }
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, usuario.Username),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new Claim(JwtRegisteredClaimNames.Email, usuario.Email),
        new Claim("uid", usuario.Id.ToString())
    };
    .Union(roleClaims);
    var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwt.Key));
    var signingCredentials = new SigningCredentials(symmetricSecurityKey, SecurityAlgorithms.HmacSha256);
    var jwtSecurityToken = new JwtSecurityToken(
        issuer: _jwt.Issuer,
        audience: _jwt.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddMinutes(_jwt.DurationInMinutes),
        signingCredentials: signingCredentials);
    return jwtSecurityToken;
}
```

**18. Modifique el archivo UnitOfWork de acuerdo al siguiente código.**

```
using System.Text;
using ApiJwt.Helpers;
using ApiJwt.Services;
using Application.UnitOfWork;
using Domain.Entities;
using Domain.Interfaces;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;

namespace ApiJwt.Extension;

public static class ApplicationServiceExtensions
{
    public static void ConfigureCors(this IServiceCollection services) =>
        services.AddCors(options =>
    {
        options.AddPolicy("CorsPolicy", builder =>
            builder.AllowAnyOrigin() //WithOrigins("https://domain.com")
                .AllowAnyMethod() //WithMethods("GET", "POST")
                .AllowAnyHeader()); //WithHeaders("accept", "content-type")
    });
    public static void AddAplicacionServices(this IServiceCollection services)
    {
        services.AddScoped<IPasswordHasher<User>, PasswordHasher<User>>();
        services.AddScoped<IUserService, UserService>();
        services.AddScoped<IUnitOfWork, UnitOfWork>();
    }
    public static void AddJwt(this IServiceCollection services, IConfiguration configuration)
    {
        //Configuration from AppSettings
        services.Configure<JWT>(configuration.GetSection("JWT"));

        //Adding Authentication - JWT
        services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
        .AddJwtBearer(o =>
    {
        o.RequireHttpsMetadata = false;
        o.SaveToken = false;
        o.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ClockSkew = TimeSpan.Zero,
            ValidIssuer = configuration["JWT:Issuer"],
            ValidAudience = configuration["JWT:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["JWT:Key"]))
        };
    });
}
}
```

## HOW TO DO LOGS Y EXCEPCIONES

Configurar Serilog : Se instala en el proyecto tipo WebAPI

```
dotnet add package Serilog.AspNetCore --version 7.0.0
```

Agregar la configuración de logs en el contenedor de dependencia.

```
var logger = new LoggerConfiguration()
    .ReadFrom.Configuration(builder.Configuration)
    .Enrich.FromLogContext()
    .CreateLogger();

//builder.Logging.ClearProviders();
builder.Logging.AddSerilog(logger);
```

El código configura y agrega un proveedor de registro Serilog a una aplicación ASP.NET Core.

```
1  using System.Reflection;
2  using System.Text;
3  using ApiIncidencias.Extensions;
4  using ApiIncidencias.Helpers.Errors;
5  using AspNetCoreRateLimit;
6  using Microsoft.AspNetCore.Authentication.JwtBearer;
7  using Microsoft.AspNetCore.Authorization;
8  using Microsoft.AspNetCore.Mvc.Authorization;
9  using Microsoft.EntityFrameworkCore;
10 using Microsoft.IdentityModel.Tokens;
11 using Persistencia;
12 using Serilog;
13
14 var builder = WebApplication.CreateBuilder(args);
15 var logger = new LoggerConfiguration()
16     .ReadFrom.Configuration(builder.Configuration)
17     .Enrich.FromLogContext()
18     .CreateLogger();
19
20 //builder.Logging.ClearProviders();
21 builder.Logging.AddSerilog(logger);
22 // Add services to the container.
23 //builder.Services.AddJwt(builder.Configuration);
24 builder.Services.AddControllers(options =>
25     options.EnableEndpointRouting = false;
```

Configurar parámetros de entorno de Serilog en appsettings

```
"Serilog": {
  "Using": [ "Serilog.Sinks.File" ],
  "MinimumLevel": {
    "Default": "Information"
  },
  "WriteTo": [
    {
      "Name": "File",
      "Args": {
        "path": "../logs/webapi-.log",
        "rollingInterval": "Day",
        "outputTemplate": "[{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} {Level:u3}] {Message:lj}{NewLine}{Exception}"
      }
    }
  ]
}
```

El código corresponde una configuración de Serilog, un marco de registro de eventos para aplicaciones .NET. Esta configuración específica se encuentra en un archivo de configuración típico de ASP.NET Core, como `appsettings.json`, y controla cómo se registran y almacenan los eventos y mensajes en tu aplicación.

"**Serilog**": Esto establece la raíz del objeto de configuración de Serilog.

"**Using**": Esta sección enumera los ensamblados que se deben cargar para los complementos de salida de Serilog que utilizarás. En este caso, se utiliza "Serilog.Sinks.File", que es un complemento que permite escribir registros en archivos.

"**MinimumLevel**

"**WriteTo**

"**Name**

"**Args**

"**path**

"**rollingInterval**

"**outputTemplate**

```
19  "Serilog": {  
20    "Using": [ "Serilog.Sinks.File" ],  
21    "MinimumLevel": {  
22      "Default": "Information"  
23    },  
24    "WriteTo": [  
25      {  
26        "Name": "File",  
27        "Args": {  
28          "path": "../logs/webapi-.log",  
29          "rollingInterval": "Day",  
30          "outputTemplate": "[{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} {Level:u3}] {Message:lj}{NewLine}{Exception}"  
31        }  
32      }  
33    ]  
34  }
```

Aplicar el siguiente código en el contenedor de dependencias.

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var loggerFactory = services.GetRequiredService<ILoggerFactory>();
    try
    {
        var context = services.GetRequiredService<JwtApplicationContext>();
        await context.Database.MigrateAsync();
    }
    catch (Exception ex)
    {
        var _logger = loggerFactory.CreateLogger<Program>();
        _logger.LogError(ex, "Ocurrió un error durante la migración");
    }
}
```

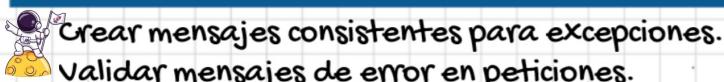
Prueba Inicial : En el controller WeatherForecastController agregue el siguiente código en el constructor de la clase.

```
4  namespace ApiJwt.Controllers;
5
6  {
7      3 references
8      public class WeatherForecastController : ApiBaseController
9      {
10          2 references
11          private static readonly string[] Summaries = new[]
12          {
13              "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering",
14          };
15
16          2 references
17          private readonly ILogger<WeatherForecastController> _logger;
18
19          0 references
20          public WeatherForecastController(ILogger<WeatherForecastController> logger)
21          {
22              _logger = logger;
23              _logger.LogInformation("Cordial saludo desde el pronosticador del clima");
24          }
25      }
26  }
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Now listening on: http://localhost:5258
info: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: D:\NetCore\jwtdemo-app\ApiJwt
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
Failed to determine the https port for redirect.
info: ApiJwt.Controllers.WeatherForecastController[0]
Cordial saludo desde el pronosticador del clima
```

```
logs > webapi-20230913.log
11 [2023-09-13 22:29:06.854 -05:00 INF] Now listening on: http://localhost:5258
12 [2023-09-13 22:29:06.856 -05:00 INF] Application started. Press Ctrl+C to shut down.
13 [2023-09-13 22:29:06.856 -05:00 INF] Hosting environment: Development
14 [2023-09-13 22:29:06.856 -05:00 INF] Content root path: D:\NetCore\jwtdemo-app\ApiJwt
15 [2023-09-13 22:29:37.132 -05:00 INF] Request starting HTTP/1.1 GET http://localhost:5258/api/WeatherForecast
16 [2023-09-13 22:29:37.533 -05:00 WRN] Failed to determine the https port for redirect.
17 [2023-09-13 22:29:38.385 -05:00 INF] Executing endpoint 'ApiJwt.Controllers.WeatherForecastController'
18 [2023-09-13 22:29:38.396 -05:00 INF] Route matched with {action = "Get", controller = "WeatherForecast", id = null}
19 [2023-09-13 22:29:38.397 -05:00 INF] Cordial saludo desde el pronosticador del clima
20 [2023-09-13 22:29:38.404 -05:00 INF] Executing ObjectResult, writing value of type 'ApiJwt.WeatherForecast'
21 [2023-09-13 22:29:38.450 -05:00 INF] Executed action 'ApiJwt.Controllers.WeatherForecastController.Get'
22 [2023-09-13 22:29:38.451 -05:00 INF] Executed endpoint 'ApiJwt.Controllers.WeatherForecastController.Get'
23 [2023-09-13 22:29:38.453 -05:00 INF] Request finished HTTP/1.1 GET http://localhost:5258/api/WeatherForecast?<id>=1
24 [2023-09-13 22:30:32.112 -05:00 INF] Application is shutting down...
```



Crear mensajes consistentes para excepciones.

Validar mensajes de error en peticiones.

## 1. Cree una clase llamada ApiResponse (Helpers)

```
namespace ApiJwt.Helpers;

public class ApiResponse
{
    public int StatusCode { get; set; }
    public string Message { get; set; }

    public ApiResponse(int statusCode, string message = null)
    {
        StatusCode = statusCode;
        Message = message ?? GetDefaultMessage(statusCode);
    }

    public ApiResponse()
    {
    }

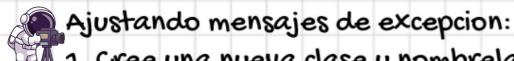
    private string GetDefaultMessage(int statusCode)
    {
        return statusCode switch
        {
            400 => "Has realizado una petición incorrecta.",
            401 => "Usuario no autorizado.",
            404 => "El recurso que has intentado solicitar no existe.",
            405 => "Este método HTTP no está permitido en el servidor.",
            500 => "Error en el servidor. No eres tú, soy yo. Comunícate con el administrador XD.",
            _ => throw new NotImplementedException()
        };
    }
}
```

## 2. Implementando mensajes en los endpoint.

```
0 references
public async Task<IActionResult> Delete(string id){
    var pais = await _unitOfWork.Paises.GetByIdAsync(id);
    if(pais == null){
        ► return NotFound(new ApiResponse(404,"El país solicitado no existe."));
    }
    _unitOfWork.Paises.Remove(pais);
    await _unitOfWork.SaveAsync();
    return NoContent();
}
```

```
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 references
public async Task<ActionResult<PaisDto>> Get(string id)
{
    var pais = await _unitOfWork.Paises.GetByIdAsync(id);
    if (pais == null){
        ► return NotFound(new ApiResponse(404,"El país solicitado no existe."));
    }
    return _mapper.Map<PaisDto>(pais);
}
```

```
[ProducesResponseType(StatusCodes.Status400BadRequest)]
1 reference
public async Task<ActionResult<Pais>> Post(PaisDto paisDto){
    var pais = _mapper.Map<Pais>(paisDto);
    this._unitOfWork.Paises.Add(pais);
    await _unitOfWork.SaveAsync();
    if (pais == null)
    {
        ► return BadRequest(new ApiResponse(400));
    }
    paisDto.Id = pais.Id;
    return CreatedAtAction(nameof(Post),new {id= paisDto.Id}, paisDto);
}
```



## Ajustando mensajes de excepcion:

1. Cree una nueva clase u nombrela ApiException (Helpers)

```
namespace ApiJwt.Helpers;

public class ApiException : ApiResponse
{
    public ApiException(int statusCode, string message = null, string details = null)
        : base(statusCode, message)
    {
        Details = details;
    }

    public string Details { get; set; }
}
```



## Cree la clase Middleware para llevar el control de excepciones de forma global.



```
using System.Net;
using System.Text.Json;

namespace ApiJwt.Helpers;

public class ExceptionMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<ExceptionMiddleware> _logger;
    private readonly IHostEnvironment _env;

    public ExceptionMiddleware(RequestDelegate next,
        ILogger<ExceptionMiddleware> logger, IHostEnvironment env)
    {
        _next = next;
        _logger = logger;
        _env = env;
    }

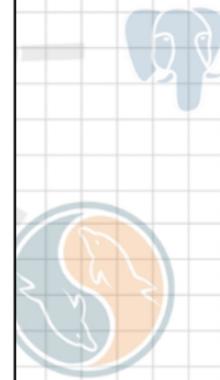
    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (Exception ex)
        {
            var statusCode = (int) HttpStatusCode.InternalServerError;

            _logger.LogError(ex, ex.Message);
            context.Response.ContentType = "application/json";
            context.Response.StatusCode = statusCode;

            var response = _env.IsDevelopment()
                ? new ApiException(statusCode, ex.Message, ex.StackTrace.ToString())
                : new ApiException(statusCode);

            var options = new JsonSerializerOptions
            {
                PropertyNamingPolicy = JsonNamingPolicy.CamelCase
            };
            var json = JsonSerializer.Serialize(response, options);

            await context.Response.WriteAsync(json);
        }
    }
}
```





Agregar la inyección de dependencia del middleware creado.

```

31
32 var app = builder.Build();
33 app.UseMiddleware<ExceptionMiddleware>();
34 // Configure the HTTP request pipeline.
35 if (app.Environment.IsDevelopment())
36 {
37     app.UseSwagger();
38     app.UseSwaggerUI();
39 }
40 using (var scope = app.Services.CreateScope())
41 {

```



Validación de atributos. Para validar atributos se debe incorporar la anotación Required sobre el atributo que se desea validar.

```

Domain > Entities > C# User.cs > ...
1 using System.ComponentModel.DataAnnotations;
2 namespace Domain.Entities;
3
4 public class User : BaseEntity
5 {
6     [Required(ErrorMessage = "El nombre de usuario es obligatorio")]
7     public string Username { get; set; }
8     public string Email { get; set; }
9     public string Password { get; set; }
10    public ICollection<Role> Roles { get; set; } = new HashSet<Role>();
11    public ICollection<RefreshToken> RefreshTokens { get; set; } = new HashSet<RefreshToken>();
12    public ICollection<UserRole> UsersRoles { get; set; }
13 }

```

El "model state" se refiere al estado de un modelo cuando intentas enlazar datos a él, por lo general, cuando recibes datos en un controlador de una solicitud HTTP. El model state rastrea si la unión de datos y la validación del modelo son exitosas o no.

Cuando se envían datos a un método de un controlador, ASP.NET Core intenta enlazar esos datos a los parámetros del método. Si estás utilizando anotaciones de validación en tu modelo, como [Required] o [StringLength(100)], y los datos no cumplen con esas validaciones, se agregará un error al model state.

Para acceder al model state se debe generar un método de extensión e injectarlo al contenedor de dependencias program.cs

```

59     0 references
60     public static void AddValidationErrors(this IServiceCollection services)
61     {
62         services.Configure<ApiBehaviorOptions>(options =>
63         {
64             options.InvalidModelStateResponseFactory = actionContext =>
65             {
66                 var errors = actionContext.ModelState.Where(u => u.Value.Errors.Count > 0)
67                     .SelectMany(u => u.Value.Errors)
68                     .Select(u => u.ErrorMessage).ToArray();
69
70                 var errorResponse = new ApiValidation()
71                 {
72                     Errors = errors
73                 };
74
75                 return new BadRequestObjectResult(errorResponse);
76             };
77         });
78     }

```

Se requiere crear la clase  
ApiValidation. Esta clase se crea  
en helpers



## Datos importantes del código:

```
public static void AddValidationErrors(this IServiceCollection services)
```

Esta es una definición de método de extensión para `IServiceCollection`. Los métodos de extensión son una forma de agregar funcionalidad adicional a una clase existente sin modificarla. En este caso estás agregando un nuevo método llamado

```
services.Configure<ApiBehaviorOptions>(options =>
```

Aquí se está configurando las opciones de comportamiento de la API para la aplicación. `ApiBehaviorOptions` ofrece diversas configuraciones que pueden alterar el comportamiento de cómo el API responde en ciertas situaciones.

```
options.InvalidModelStateResponseFactory = actionContext =>
```

`InvalidModelStateResponseFactory` es una propiedad en `ApiBehaviorOptions` que permite definir una función personalizada para generar una respuesta cuando el `ModelState` es inválido. Esta propiedad espera una función que toma un `ActionContext` y devuelve un `IActionResult`.

```
var errors = actionContext.ModelState.Where(u => u.Value.Errors.Count > 0)
    .SelectMany(u => u.Value.Errors)
    .Select(u => u.ErrorMessage).ToArray();
```

Este fragmento de código está filtrando y seleccionando todos los errores de validación del `ModelState`:

`Where(u => u.Value.Errors.Count > 0)`: Filtra los estados del modelo que tienen errores.  
`SelectMany(u => u.Value.Errors)`: "Aplana" la colección de errores en todos los estados del modelo.  
`Select(u => u.ErrorMessage)`: Selecciona solo el mensaje de error de cada error.



## Inyectar el método de extensión en el contenedor de dependencias.

```
22 builder.Services.ConfigureRateLimiting();
23
24 // Add services to the container.
25 builder.Services.ConfigureCors();
26 builder.Services.AddAplicacionServices();
27 builder.Services.ConfigureApiVersioning();
28 builder.Services.AddJwt(builder.Configuration);
29
30 builder.Services.AddControllers(options =>
31 {
32     options.RespectBrowserAcceptHeader = true;
33     options.ReturnHttpNotAcceptable = true;
34 }).AddXmlSerializerFormatters();
35
36 builder.Services.AddValidationErrors(); ←
37
38 builder.Services.AddDbContext<TiendaContext>(options =>
39
40
```



La inyección de este método se debe realizar después de `AddController`