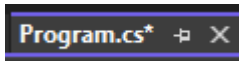
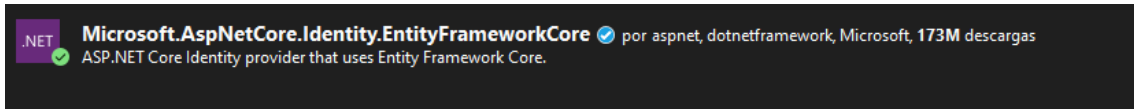


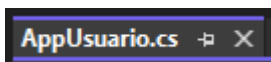
Guía para autenticación y autorización con Identity:

1) Configuración básica para utilizar Identity



```
//Soporte para autenticación con .NET Identity
builder.Services.AddIdentity<AppUsuario, IdentityRole>().AddEntityFrameworkStores<ApplicationDbContext>();
```

Añado personalizaciones a IdentityUser:



```
using Microsoft.AspNetCore.Identity;

namespace APIIdentity.Models
{
    0 referencias
    public class AppUsuario : IdentityUser
    {
        //Aquí añadimos los campos que queramos añadir a los que vienen por defecto en Identity
        0 referencias
        public string Name { get; set; }
    }
}
```



```
using APIIdentity.Modelos;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace APIIdentity.Data
{
    8 referencias
    public class ApplicationDbContext : IdentityDbContext<AppUsuario>
    {
        0 referencias
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {
        }

        0 referencias
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }








        //Aquí pasar todas las entidades (Modelos)
        0 referencias
        public DbSet<Usuario> Usuario { get; set; }
        5 referencias
        public DbSet<AppUsuario> AppUsuario { get; set; }
    }
}
```

Consola del Administrador de paquetes

```
PM> add-migration usuario-identity|
```

```
PM> update-database|
```

Se han creado todas estas tablas:

-  dbo.AspNetRoleClaims
-  dbo.AspNetRoles
-  dbo.AspNetUserClaims
-  dbo.AspNetUserLogins
-  dbo.AspNetUserRoles
-  dbo.AspNetUsers
-  dbo.AspNetUserTokens

2) Empezamos a trabajar con los métodos:

IUsuarioRepositorio.cs

```
using APIIdentity.Modelos.Dtos;
using APIIdentity.Modelos;

namespace APIIdentity.Repositorio.IRepositorio
{
    4 referencias
    public interface IUsuarioRepositorio
    {
        2 referencias
        ICollection<AppUsuario> GetUsuarios();
        2 referencias
        AppUsuario GetUsuario(string usuarioId);
        2 referencias
        bool IsUniqueUser(string usuario);
        2 referencias
        Task<UsuarioLoginRespuestaDto> Login(UsuarioLoginDto usuarioLoginDto);
        2 referencias
        Task<UsuarioDatosDto> Registro(UsuarioRegistroDto usuarioRegistroDto);
    }
}
```

UsuarioMapper.cs*

```
CreateMap<AppUsuario, UsuariosDatosDto>().ReverseMap();
```

UsuarioRepositorio.cs*

Añado propiedades características de Identity y el mapper:

```
namespace APIIdentity.Repositorio
{
    2 referencias
    public class UsuarioRepositorio : IUsuarioRepositorio
    {
        private readonly ApplicationDbContext _bd;
        private string claveSecreta;
        private readonly UserManager<AppUsuario> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly IMapper _mapper;

        0 referencias
        public UsuarioRepositorio(ApplicationDbContext bd, IConfiguration config, UserManager<AppUsuario> userManager, RoleManager<IdentityRole> roleManager, IMapper mapper)
        {
            _bd = bd;
            claveSecreta = config.GetValue<string>("ApiSettings:Secreta");
            _userManager = userManager;
            _roleManager = roleManager;
            _mapper = mapper;
        }
    }
}
```

Me pongo con la función de login, no sin antes ajustar el siguiente Dto:

UsuarioLoginRespuestaDto.cs

```
namespace APIIdentity.Modelos.Dtos
{
    5 referencias
    public class UsuarioLoginRespuestaDto
    {
        3 referencias
        public UsuarioDatosDto Usuario { get; set; }
        0 referencias
        public string Role { get; set; }
        3 referencias
        public string Token { get; set; }
    }
}
```

UsuarioRepositorio.cs

```
2 referencias
public async Task<UsuarioLoginRespuestaDto> Login(UsuarioLoginDto usuarioLoginDto)
{
    var usuario = _bd.AppUsuario.FirstOrDefault(
        u => u.UserName.ToLower() == usuarioLoginDto.NombreUsuario.ToLower());

    bool isValid = await _userManager.CheckPasswordAsync(usuario, usuarioLoginDto.Password);

    //Validamos si el usuario no existe con la combinación de usuario y contraseña correcta
    if (usuario == null || isValid == false)
    {
        return new UsuarioLoginRespuestaDto()
        {
            Token = "",
            Usuario = null
        };
    }

    //Aquí existe el usuario entonces podemos procesar el login
    var roles = await _userManager.GetRolesAsync(usuario);
    var manejadoToken = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(claveSecreta);

    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new Claim[]
        {
            new Claim(ClaimTypes.Name, usuario.UserName.ToString()),
            new Claim(ClaimTypes.Role, roles.FirstOrDefault())
        }),
        Expires = DateTime.UtcNow.AddDays(7),
        SigningCredentials = new(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };

    var token = manejadoToken.CreateToken(tokenDescriptor);

    UsuarioLoginRespuestaDto usuarioLoginRespuestaDto = new UsuarioLoginRespuestaDto()
    {
        Token = manejadoToken.WriteToken(token),
        Usuario = _mapper.Map<UsuarioDatosDto>(usuario),
    };

    return usuarioLoginRespuestaDto;
}
```

Me pongo con la función de registro:

```
2 referencias
public async Task<UsuarioDatosDto> Registro(UsuarioRegistroDto usuarioRegistroDto)
{
    AppUsuario usuario = new AppUsuario()
    {
        UserName = usuarioRegistroDto.NombreUsuario,
        Email = usuarioRegistroDto.NombreUsuario,
        NormalizedEmail = usuarioRegistroDto.NombreUsuario.ToUpper(),
        Nombre = usuarioRegistroDto.Nombre
    };

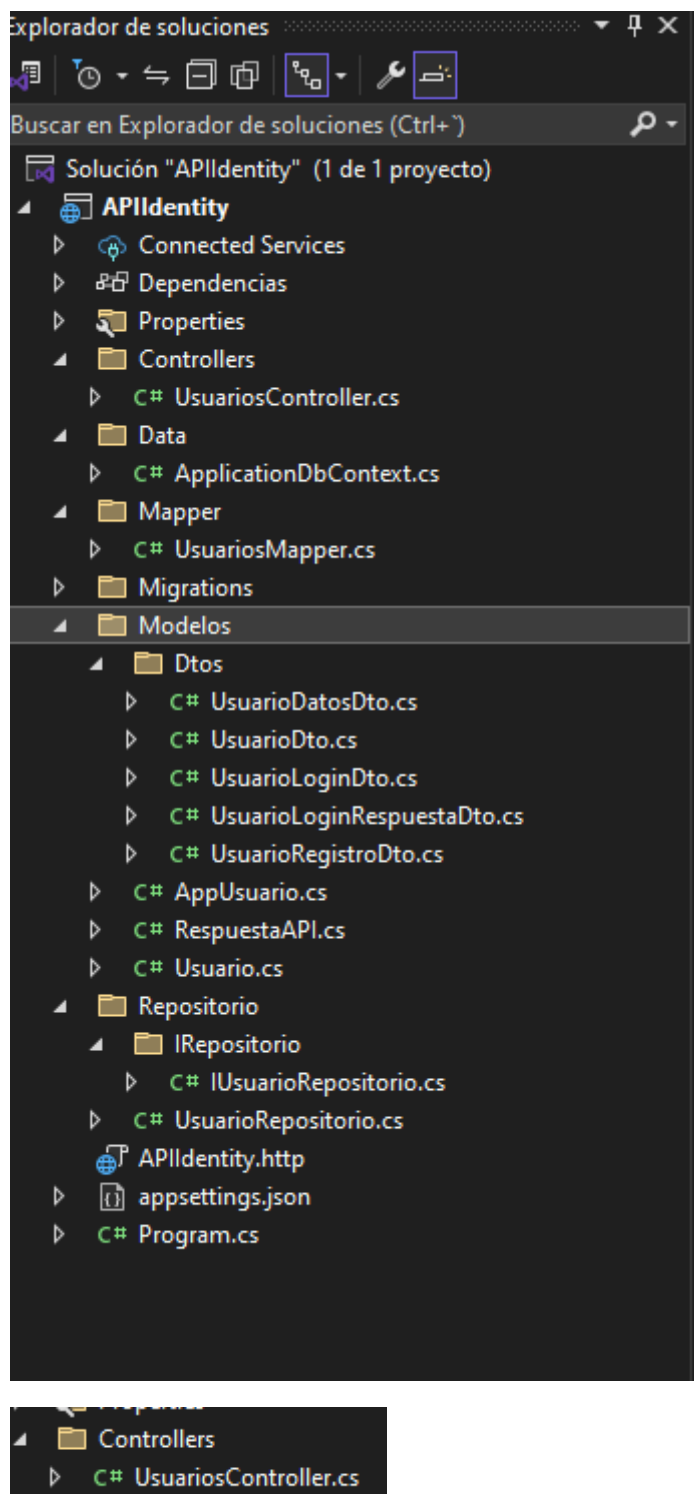
    var result = await _userManager.CreateAsync(usuario, usuarioRegistroDto.Password);
    if (result.Succeeded)
    {
        if (!_roleManager.RoleExistsAsync("Admin").GetAwaiter().GetResult())
        {
            await _roleManager.CreateAsync(new IdentityRole("Admin"));
            await _roleManager.CreateAsync(new IdentityRole("Registrado"));
        }

        await _userManager.AddToRoleAsync(usuario, "Admin");
        var usuarioRetornado = _bd.AppUsuario.FirstOrDefault(u => u.UserName == usuarioRegistroDto.NombreUsuario);

        return _mapper.Map<UsuarioDatosDto>(usuarioRetornado);
    }

    return new UsuarioDatosDto();
}
```

El proyecto queda así:



```

using APIIdentity.Modelos;
using APIIdentity.Modelos.Dtos;
using APIIdentity.Repositorio.IRepositorio;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Net;

namespace APIIdentity.Controllers
{
    [Route("api/usuarios")]
    [ApiController]
    1 referencia
    public class UsuariosController : ControllerBase
    {
        private readonly IUseruarioRepositorio _usRepo;
        protected RespuestaAPI _respuestaApi;
        private readonly IMapper _mapper;

        0 referencias
        public UsuariosController(IUsuarioRepositorio usRepo, IMapper mapper)
        {
            _usRepo = usRepo;
            _mapper = mapper;
            _respuestaApi = new();
        }

        [Authorize(Roles = "Admin")]
        [HttpGet]
        [ResponseCache(CacheProfileName = "PorDefecto30Segundos")]
        [ProducesResponseType(StatusCodes.Status403Forbidden)]
        [ProducesResponseType(StatusCodes.Status200OK)]
        0 referencias
        public IActionResult GetUsuarios()
        {
            var listaUsuarios = _usRepo.GetUsuarios();

            var listaUsuariosDto = new List<UsuarioDto>();

            foreach (var lista in listaUsuarios)
            {
                listaUsuariosDto.Add(_mapper.Map<UsuarioDto>(lista));
            }

            return Ok(listaUsuariosDto);
        }
    }
}

```

```

[Authorize(Roles = "Admin")]
[HttpGet("{usuarioId}", Name = "GetUsuario")]
[ResponseCache(CacheProfileName = "PorDefecto30Segundos")]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 referencias
public IActionResult GetUsuario(string usuarioId)
{
    var itemUsuario = _usRepo.GetUsuario(usuarioId);

    if (itemUsuario == null)
    {
        return NotFound();
    }

    var itemUsuarioDto = _mapper.Map<UsuarioDto>(itemUsuario);

    return Ok(itemUsuarioDto);
}

```

```

[AllowAnonymous]
[HttpPost("registro")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
0 referencias
public async Task<IActionResult> Registro([FromBody] UsuarioRegistroDto usuarioRegistroDto)
{
    bool validarNombreUsuarioUnico = _usRepo.IsUniqueUser(usuarioRegistroDto.NombreUsuario);
    if (!validarNombreUsuarioUnico)
    {
        _respuestaApi.StatusCode = HttpStatusCode.BadRequest;
        _respuestaApi.IsSuccess = false;
        _respuestaApi.ErrorMessages.Add("El nombre de usuario ya existe");
        return BadRequest(_respuestaApi);
    }

    var usuario = await _usRepo.Registro(usuarioRegistroDto);
    if (usuario == null)
    {
        _respuestaApi.StatusCode = HttpStatusCode.BadRequest;
        _respuestaApi.IsSuccess = false;
        _respuestaApi.ErrorMessages.Add("Error en el registro");
        return BadRequest(_respuestaApi);
    }

    _respuestaApi.StatusCode = HttpStatusCode.OK;
    _respuestaApi.IsSuccess = true;
    return Ok(_respuestaApi);
}

```


```

[AllowAnonymous]
[HttpPost("login")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
0 referencias
public async Task<IActionResult> Login([FromBody] UsuarioLoginDto usuarioLoginDto)
{
    var respuestaLogin = await _usRepo.Login(usuarioLoginDto);

    if (respuestaLogin.Usuario == null || string.IsNullOrEmpty(respuestaLogin.Token))
    {
        _respuestaApi.StatusCode = HttpStatusCode.BadRequest;
        _respuestaApi.IsSuccess = false;
        _respuestaApi.ErrorMessages.Add("El nombre de usuario o password son incorrectos");
        return BadRequest(_respuestaApi);
    }

    _respuestaApi.StatusCode = HttpStatusCode.OK;
    _respuestaApi.IsSuccess = true;
    _respuestaApi.Result = respuestaLogin;
    return Ok(_respuestaApi);
}
}

```

 Data
 C# ApplicationDbContext.cs

```

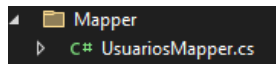
using APIIdentity.Models;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace APIIdentity.Data
{
    0 referencias
    public class ApplicationDbContext : IdentityDbContext<AppUsuario>
    {
        0 referencias
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {
        }

        0 referencias
        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
        }

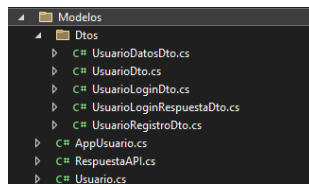
        //Aquí pasar todas las entidades (Modelos)
        0 referencias
        public DbSet<Usuario> Usuario { get; set; }
        5 referencias
        public DbSet<AppUsuario> AppUsuario { get; set; }
    }
}

```



```
using APIIdentity.Modelos.Dtos;
using APIIdentity.Modelos;
using AutoMapper;

namespace APIIdentity.Mapper
{
    2 referencias
    public class UsuariosMapper : Profile
    {
        0 referencias
        public UsuariosMapper() {
            CreateMap<AppUsuario, UsuarioDatosDto>().ReverseMap();
            CreateMap<AppUsuario, UsuarioDto>().ReverseMap();
        }
    }
}
```



```
namespace APIIdentity.Modelos.Dtos
{
    7 referencias
    public class UsuarioDatosDto
    {
        0 referencias
        public string ID { get; set; }
        0 referencias
        public string Username { get; set; }
        0 referencias
        public string Nombre { get; set; }
    }
}
```



```
namespace APIIdentity.Modelos.Dtos
{
    4 referencias
    public class UsuarioDto
    {
        0 referencias
        public string Id { get; set; }
        0 referencias
        public string UserName { get; set; }
        0 referencias
        public string Nombre { get; set; }
        0 referencias
        public string Password { get; set; }
        0 referencias
        public string Role { get; set; }
    }
}
```


UsuarioLoginDto.cs

```
using System.ComponentModel.DataAnnotations;

namespace APIIdentity.Modelos.Dtos
{
    3 referencias
    public class UsuarioLoginDto
    {
        [Required(ErrorMessage = "El usuario es obligatorio")]
        1 referencia
        public string NombreUsuario { get; set; }

        [Required(ErrorMessage = "El password es obligatorio")]
        1 referencia
        public string Password { get; set; }
    }
}
```

UsuarioLoginRespuestaDto.cs

```
namespace APIIdentity.Modelos.Dtos
{
    5 referencias
    public class UsuarioLoginRespuestaDto
    {
        3 referencias
        public UsuarioDatosDto Usuario { get; set; }
        0 referencias
        public string Role { get; set; }
        3 referencias
        public string Token { get; set; }
    }
}
```

UsuarioRegistroDto.cs

```
using System.ComponentModel.DataAnnotations;

namespace APIIdentity.Modelos.Dtos
{
    3 referencias
    public class UsuarioRegistroDto
    {
        [Required(ErrorMessage = "El usuario es obligatorio")]
        5 referencias
        public string NombreUsuario { get; set; }
        [Required(ErrorMessage = "El nombre es obligatorio")]
        1 referencia
        public string Nombre { get; set; }
        [Required(ErrorMessage = "El password es obligatorio")]
        1 referencia
        public string Password { get; set; }
        0 referencias
        public string Role { get; set; }
    }
}
```

AppUsuario.cs ➦ ✕

```
using Microsoft.AspNetCore.Identity;

namespace APIIdentity.Modelos
{
    13 referencias
    public class AppUsuario : IdentityUser
    {
        1 referencia
        public string Nombre { get; set; }
    }
}
```

RespuestaAPI.cs ➦ ✕

```
using System.Net;

namespace APIIdentity.Modelos
{
    3 referencias
    public class RespuestaAPI
    {
        1 referencia
        public RespuestaAPI()
        {
            ErrorMessages = new List<string>();
        }

        5 referencias
        public HttpStatusCode StatusCode { get; set; }
        5 referencias
        public bool IsSuccess { get; set; } = true;
        4 referencias
        public List<string> ErrorMessages { get; set; }
        1 referencia
        public object Result { get; set; }
    }
}
```

Usuario.cs ➦ ✕

```
using System.ComponentModel.DataAnnotations;

namespace APIIdentity.Modelos
{
    1 referencia
    public class Usuario
    {
        [Key]
        0 referencias
        public int Id { get; set; }
        0 referencias
        public string NombreUsuario { get; set; }
        0 referencias
        public string Nombre { get; set; }
        0 referencias
        public string Password { get; set; }
        0 referencias
        public string Role { get; set; }
    }
}
```

```
└─ Repositorio
   └─ IRepositorio
      ├── IUsuarioRepositorio.cs
      └─ UsuarioRepositorio.cs
```

UsuarioRepositorio.cs

```
using APIIdentity.Modelos.Dtos;
using APIIdentity.Modelos;

namespace APIIdentity.Repositorio.IRepositorio
{
    4 referencias
    public interface IUsuarioRepositorio
    {
        2 referencias
        ICollection<AppUsuario> GetUsuarios();
        2 referencias
        AppUsuario GetUsuario(string usuarioId);
        2 referencias
        bool IsUniqueUser(string usuario);
        2 referencias
        Task<UsuarioLoginRespuestaDto> Login(UsuarioLoginDto usuarioLoginDto);
        2 referencias
        Task<UsuarioDatosDto> Registro(UsuarioRegistroDto usuarioRegistroDto);
    }
}
```

UsuarioRepositorio.cs

```
using APIIdentity.Data;
using APIIdentity.Modelos.Dtos;
using APIIdentity.Modelos;
using APIIdentity.Repositorio.IRepositorio;
using AutoMapper;
using Microsoft.AspNetCore.Identity;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace APIIdentity.Repositorio
{
    2 referencias
    public class UsuarioRepositorio : IUsuarioRepositorio
    {
        private readonly ApplicationDbContext _bd;
        private string claveSecreta;
        private readonly UserManager<AppUsuario> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly IMapper _mapper;

        0 referencias
        public UsuarioRepositorio(ApplicationDbContext bd, IConfiguration config, UserManager<AppUsuario> userManager, RoleManager<IdentityRole> roleManager, IMapper mapper)
        {
            _bd = bd;
            claveSecreta = config.GetValue<string>("ApiSettings:Secreta");
            _userManager = userManager;
            _roleManager = roleManager;
            _mapper = mapper;
        }

        2 referencias
        public AppUsuario GetUsuario(string usuarioId)
        {
            return _bd.AppUsuario.FirstOrDefault(c => c.Id == usuarioId);
        }
    }
}
```

```

2 referencias
public ICollection<AppUsuario> GetUsuarios()
{
    return _bd.AppUsuario.OrderBy(c => c.UserName).ToList();
}

2 referencias
public bool IsUniqueUser(string usuario)
{
    var usuarioBd = _bd.AppUsuario.FirstOrDefault(u => u.UserName == usuario);
    if (usuarioBd == null)
    {
        return true;
    }
    return false;
}

2 referencias
public async Task<UsuarioLoginRespuestaDto> Login(UsuarioLoginDto usuarioLoginDto)
{
    var usuario = _bd.AppUsuario.FirstOrDefault(
        u => u.UserName.ToLower() == usuarioLoginDto.NombreUsuario.ToLower());

    bool isValid = await _userManager.CheckPasswordAsync(usuario, usuarioLoginDto.Password);

    //Validamos si el usuario no existe con la combinación de usuario y contraseña correcta
    if (usuario == null || isValid == false)
    {
        return new UsuarioLoginRespuestaDto()
        {
            Token = "",
            Usuario = null
        };
    }

    //Aquí existe el usuario entonces podemos procesar el login
    var roles = await _userManager.GetRolesAsync(usuario);
    var manejadoToken = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(claveSecreta);

```

```

var tokenDescriptor = new SecurityTokenDescriptor
{
    Subject = new ClaimsIdentity(new Claim[]
    {
        new Claim(ClaimTypes.Name, usuario.UserName.ToString()),
        new Claim(ClaimTypes.Role, roles.FirstOrDefault())
    }),
    Expires = DateTime.UtcNow.AddDays(7),
    SigningCredentials = new(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
};

var token = manejadoToken.CreateToken(tokenDescriptor);

UsuarioLoginRespuestaDto usuarioLoginRespuestaDto = new UsuarioLoginRespuestaDto()
{
    Token = manejadoToken.WriteToken(token),
    Usuario = _mapper.Map<UsuarioDatosDto>(usuario),
};

return usuarioLoginRespuestaDto;
}

```

```

2 referencias
public async Task<UsuarioDatosDto> Registro(UsuarioRegistroDto usuarioRegistroDto)
{
    AppUsuario usuario = new AppUsuario()
    {
        UserName = usuarioRegistroDto.NombreUsuario,
        Email = usuarioRegistroDto.NombreUsuario,
        NormalizedEmail = usuarioRegistroDto.NombreUsuario.ToUpper(),
        Nombre = usuarioRegistroDto.Nombre
    };

    var result = await _userManager.CreateAsync(usuario, usuarioRegistroDto.Password);
    if (result.Succeeded)
    {
        if (!_roleManager.RoleExistsAsync("Admin").GetAwaiter().GetResult())
        {
            await _roleManager.CreateAsync(new IdentityRole("Admin"));
            await _roleManager.CreateAsync(new IdentityRole("Registrado"));
        }

        await _userManager.AddToRoleAsync(usuario, "Admin");
        var usuarioRetornado = _bd.AppUsuario.FirstOrDefault(u => u.UserName == usuarioRegistroDto.NombreUsuario);

        return _mapper.Map<UsuarioDatosDto>(usuarioRetornado);
    }

    return new UsuarioDatosDto();
}
}

```

appsettings.json

```

{
  "ApiSettings": {
    "Secreta": "Contratame como desarrollador fullstack en tu empresa y no te arrepentirás ya que garantizo dedicación y resultados."
  },
  "ConnectionStrings": {
    "ConexionSql": "Server=LEASBA-00-053\\EXPRI9;Database=APIIdentity;User ID=sa;Password=Tiburón01;TrustServerCertificate=true;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}

```

Program.cs

```

using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;
using APIIdentity.Mapper;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddDbContext<ApplicationDbContext>(opciones =>
    opciones.UseSqlServer(builder.Configuration.GetConnectionString("ConexionSql")));

//Soporte para autenticación con .NET Identity
builder.Services.AddIdentity<AppUsuario, IdentityRole>().AddEntityFrameworkStores<ApplicationDbContext>();

//Soporte para cache
builder.Services.AddResponseCaching();

//Agregamos los Repositorios
builder.Services.AddScoped<IUsuarioRepositorio, UsuarioRepositorio>();

var key = builder.Configuration.GetValue<string>("ApiSettings:Secreta");

// Agregar AutoMapper a los servicios.
builder.Services.AddAutoMapper(typeof(UsuariosMapper));

//Aquí se configura la Autenticación
builder.Services.AddAuthentication
(
    x =>
    {
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    }
).AddJwtBearer(x =>
{
    x.RequireHttpsMetadata = false;
    x.SaveToken = true;
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(key)),
        ValidateIssuer = false,
        ValidateAudience = false
    };
});

```

```

builder.Services.AddControllers(option =>
{
    //Cache profile. Un cache global y así no tener que ponerlo en todas partes
    option.CacheProfiles.Add("PorDefecto30Segundos", new CacheProfile() { Duration = 30 });
});

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
    options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description =
            "Autenticación JWT usando el esquema Bearer. \r\n\r\n " +
            "Ingresa la palabra 'Bearer' seguido de un [espacio] y después su token en el campo de abajo.\r\n\r\n\r\n " +
            "Ejemplo: \"Bearer tkljkl25jhhk\"",
        Name = "Authorization",
        In = ParameterLocation.Header,
        Scheme = "Bearer"
    });
    options.AddSecurityRequirement(new OpenApiSecurityRequirement()
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                },
                Scheme = "oauth2",
                Name = "Bearer",
                In = ParameterLocation.Header
            },
            new List<string>()
        }
    });
});
});

//Soporte para CORS

```

```

//Soporte para CORS
//Se pueden habilitar: 1-Un dominio, 2-multiples dominios,
//3-cualquier dominio (Tener en cuenta seguridad)
//Usamos de ejemplo el dominio: http://localhost:3223, se debe cambiar por el correcto
//Se usa (*) para todos los dominios
builder.Services.AddCors(p => p.AddPolicy("PoliticaCors", build =>
{
    build.WithOrigins("*").AllowAnyMethod().AllowAnyHeader();
}));

var app = builder.Build();

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

app.UseHttpsRedirection();

//Soporte para CORS
app.UseCors("PoliticaCors");

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();

```