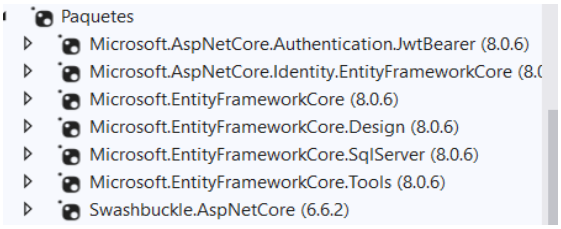
Aplicación WebAPI Auth App ASP .Net Core

Vamos a desarrollar una aplicación de autentificación y autorización de usuarios utilizando **Asp .Net Core**, y una base de datos en **SQL Server**.

Crear un nuevo proyecto en Visual Studio de tipo Web API con el nombre: **AuthAPI**.

Abrir al administrador de paquetes nugget, y en la pestaña de examinar buscar e instalar las siguientes dependencias, en el proyecto.

- Microsoft EntityFramework Core
- Microsoft EntityFramework Core SQLServer
- Microsoft EntityFramework Core Tools
- Microsoft EntityFramework Core Design
- Microsoft AspNetCore Identity EntityFramework Core
- Microsoft AspNetCore AuthenticationJwtBearer
- Swashbuccle.AspNetCore



Actualizar lo que sea necesario.

Agregamos al proyecto una carpeta de nombre Models y agregamos una clase con el nombre: AppUser.cs con el siguiente código:

```
using Microsoft.AspNetCore.Identity;

namespace AuthAPI.Models
{
    public class AppUser : IdentityUser
    {
        public string? FullName { get; set; }
    }
}
```

Agregamos al proyecto una carpeta de nombre Data y agregamos una clase con el nombre: AppDbContext.cs con el siguiente código:

```
using AuthAPI.Models;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace AuthAPI.Data
{
    public class AppDbContext : IdentityDbContext<AppUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

    }
}
```

Modificamos el archivo Program.cs para habilitar Swagger en el proyecto.

```
using Tareas_ASPNet_WebAPI.Middlewares;
using Tareas_ASPNet_WebAPI.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
```

```
//builder.Services.AddOpenApi();

//Añadiendo el generador Swagger a la colección de servicios.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    //app.MapOpenApi();

    //Añadiendo el Middleware de Swagger
    app.UseSwagger();
    app.UseSwaggerUI();
}

//Cada uno de estos es un Middleware
app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

The Swagger UI can be found at: `https://localhost:<port>/swagger`

Agregando la configuración del DbContext para SQLServer.

Abrimos el archivo **Program.cs** y lo modificamos como se muestra a continuación:

```
using Microsoft.EntityFrameworkCore;
using TareasAPI.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

//Obtenemos la cadena de conexión
var connectionString = builder.Configuration.GetConnectionString("cadenaSQL");
//Agregamos la configuración para utilizar SQLServer
builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlServer(connectionString));

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


var app = builder.Build();

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

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Agregando el Conexión String y la configuración de JWToken al archivo appsettings.

Para esto debemos abrir el archivo **appsettings.json** para agregar allí la cadena de conexión correspondiente y la configuración **JWTSetting** como se muestra en el siguiente código:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "cadenaSQL": "Server=DESKTOP-URLB235; Initial Catalog=AuthDB; user id=sa;
password=root;TrustServerCertificate=true; MultipleActiveResultSets=true"
  },
  "JWTSetting": {
    "securityKey":
"xyz2l303kkejoejeke23423sdfsf3r4wef4k044494kfgrerersdfe2r2errfewre4343434erererererererererererr",
    "ValidAudience": "http://localhost:4200",
    "ValidIssuer": "http://localhost:5000",
    "expireInMinutes": 60
  }
```

Agregando toda la configuración para utilizar JWToken y ASP .Net Core Identity al proyecto

```
using AuthAPI.Data;
using AuthAPI.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

//Obtenemos la configuración del JWTSettings de appsettings
var JWTSettings = builder.Configuration.GetSection("JWTSetting");

//Obtenemos la cadena de conexión
var connectionString = builder.Configuration.GetConnectionString("cadenaSQL");
//Agregamos la configuración para utilizar SQLServer
builder.Services.AddDbContext<AppDbContext>(options => options.UseSqlServer(connectionString));

//Agregamos la configuración para ASP -Net Core Identity
builder.Services.AddIdentity<AppUser,
IdentityRole>().AddEntityFrameworkStores<AppDbContext>().AddDefaultTokenProviders();

builder.Services.AddAuthentication(opt =>
{
    opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    opt.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(opt =>
{
    opt.SaveToken = true;
    opt.RequireHttpsMetadata = false;
    opt.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidAudience = JWTSettings["ValidAudience"],
```

```
        ValidIssuer = JWTSettings["ValidIssuer"],
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(JWTSettings.GetSection("securityKey").Value!))
    };
});

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

//Agregando la Definición de Seguridad
builder.Services.AddSwaggerGen(c =>
{
    c.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme
    {
        Description = @"JWT Authorization Example : 'Bearer eyeleieieekeieieie",
        Name = "Authorization",
        In = Microsoft.OpenApi.Models.ParameterLocation.Header,
        Type = Microsoft.OpenApi.Models.SecuritySchemeType.ApiKey,
        Scheme = "Bearer"
    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement()
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
                },
                Scheme = "outh2",
                Name = "Bearer",
                In = ParameterLocation.Header,
            },
            new List<string>()
        }
    });
});

var app = builder.Build();

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

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Ejecutamos y probamos la aplicación y verificar que aparece el botón de Authorize.

Vamos a generar la migración correspondiente para generar las tablas del modelo de Microsoft Identity en la Base de Datos.

Abrimos la consola del administrador de paquetes NuGet en **Herramientas -> Administrador de paquetes NuGet -> Consola del administrador de paquetes**

```
1. Add-Migration InitialCreate
2. Update-Database
```

Esto generará la base de datos y las tablas correspondientes al modelo de Microsoft Itentity en el servidor.

Abrimos el archivo launchSettings.json y modificamos lo siguiente:

```
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": false,
      "applicationUrl": "https://localhost:5000;http://localhost:5139",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Implementando el Controlador de Usuarios

Agregamos un nuevo controlador a la carpeta Controllers con el nombre AccountController.cs con el siguiente código, revisar en el video la implementación de las clases Dto complementarias.

```
using AuthAPI.Dtos;
using AuthAPI.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace AuthAPI.Controllers
{
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    //api/account
    public class AccountController : ControllerBase
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly IConfiguration _configuration;

        public AccountController(UserManager<AppUser> userManager, RoleManager<IdentityRole>
roleManager, IConfiguration configuration)
        {
            _userManager = userManager;
            _roleManager = roleManager;
```

```
    _configuration = configuration;
}

[AllowAnonymous]
// api/account/register
[HttpPost("register")]
public async Task<ActionResult<string>> Register(RegisterDto registerDto)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = new AppUser
    {
        Email = registerDto.Email,
        FullName = registerDto.FullName,
        UserName = registerDto.Email
    };

    var result = await _userManager.CreateAsync(user, registerDto.Password);

    if (!result.Succeeded)
    {
        return BadRequest(result.Errors);
    }

    if (registerDto.Roles is null)
    {
        await _userManager.AddToRoleAsync(user, "User");
    }
    else
    {
        foreach (var role in registerDto.Roles)
        {
            await _userManager.AddToRoleAsync(user, role);
        }
    }

    return Ok(new AuthResponseDto
    {
        IsSuccess = true,
        Message = "Account Created Sucessfully!!!"
    });
}

[AllowAnonymous]
//api/account/login
[HttpPost("login")]
public async Task<ActionResult<AuthResponseDto>> Login(LoginDto loginDto)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = await _userManager.FindByEmailAsync(loginDto.Email);

    if (user == null)
    {
        return Unauthorized(new AuthResponseDto
        {
            IsSuccess = false,
            Message = "User not found with this email"
        });
    }

    var result = await _userManager.CheckPasswordAsync(user, loginDto.Password);
```

```
            if (!result)
            {
                return Unauthorized(new AuthResponseDto
                {
                    IsSuccess = false,
                    Message = "Invalid Password"
                });
            }

            var token = GenerateToken(user);

            return Ok(new AuthResponseDto
            {
                Token = token,
                IsSuccess = true,
                Message = "Login Success"
            });
        }

        private string GenerateToken(AppUser user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();

            var key =
Encoding.ASCII.GetBytes(_configuration.GetSection("JWTSetting").GetSection("securityKey").Value!);

            var roles = _userManager.GetRolesAsync(user).Result;

            List<Claim> claims = [
                new (JwtRegisteredClaimNames.Email, user.Email??""),
        new (JwtRegisteredClaimNames.Name, user.FullName??""),
        new (JwtRegisteredClaimNames.NameId, user.Id??""),
        new (JwtRegisteredClaimNames.Aud,
_configuration.GetSection("JWTSetting").GetSection("ValidAudience").Value!),
        new (JwtRegisteredClaimNames.Iss,
_configuration.GetSection("JWTSetting").GetSection("ValidIssuer").Value!)
            ];

            foreach (var role in roles)
            {
                claims.Add(new Claim(ClaimTypes.Role, role));
            }

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires = DateTime.UtcNow.AddDays(1),
                SigningCredentials = new SigningCredentials(
                    new SymmetricSecurityKey(key),
                    SecurityAlgorithms.HmacSha256
                )
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);

            return tokenHandler.WriteToken(token);
        }

        //api/account/detail
        [HttpGet("detail")]
        public async Task<ActionResult<UserDetailDto>> GetUserDetail()
        {
            var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
            var user = await _userManager.FindByIdAsync(currentUserId!);

            if (user == null)
            {
```

```
                return NotFound(new AuthResponseDto
                {
                    IsSuccess = false,
                    Message = "User not found"
                });
            }

            return Ok(new UserDetailDto
            {
                Id = user.Id,
                Email = user.Email,
                FullName = user.FullName,
                Roles = [.. await _userManager.GetRolesAsync(user)],
                PhoneNumber = user.PhoneNumber,
                PhoneNumberConfirmed = user.PhoneNumberConfirmed,
                AccessFailedCount = user.AccessFailedCount
            });
        }

        //api/account/
        [HttpGet]
        public async Task<ActionResult<IEnumerable<UserDetailDto>>> GetUsers()
        {
            var users = await _userManager.Users.Select(u => new UserDetailDto
            {
                Id = u.Id,
                Email = u.Email,
                FullName = u.FullName,
                Roles = _userManager.GetRolesAsync(u).Result.ToArray()
            }).ToListAsync();

            return Ok(users);
        }
    }
}
```

Implementando el Controlador de Roles

Agregamos un nuevo controlador a la carpeta **Controllers** con el nombre **RolesController.cs** de tipo Controlador de API en Blanco y agregamos los siguientes métodos de API, revisen en el video la implementación de las clases Dto complementarias.

```
using AuthAPI.Dtos;
using AuthAPI.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace AuthAPI.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class RolesController : ControllerBase
    {
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly UserManager<AppUser> _userManager;

        public RolesController(RoleManager<IdentityRole> roleManager, UserManager<AppUser>
userManager)
        {
            _roleManager = roleManager;
            _userManager = userManager;
```

```
        }

        [HttpPost]
        public async Task<IActionResult> CreateRole([FromBody] CreateRoleDto createRoleDto)
        {
            if (string.IsNullOrEmpty(createRoleDto.RoleName))
            {
                return BadRequest("Role name is Required");
            }

            var roleExist = await _roleManager.RoleExistsAsync(createRoleDto.RoleName);

            if (roleExist)
            {
                return BadRequest("Role already exist");
            }

            var roleResult = await _roleManager.CreateAsync(new
IdentityRole(createRoleDto.RoleName));

            if (roleResult.Succeeded)
            {
                return Ok(new { message = "Role Created successfully" });
            }

            return BadRequest("Role creation failed");
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<RoleResponseDto>>> GetRoles()
        {
            //list of users with total user count

            var roles = await _roleManager.Roles.Select(r => new RoleResponseDto
            {
                Id = r.Id,
                Name = r.Name,
                //TotalUsers = _userManager.GetUsersInRoleAsync(r.Name!).Result.Count
            }).ToListAsync();

            foreach (var role in roles)
            {
                role.TotalUsers = _userManager.GetUsersInRoleAsync(role.Name!).Result.Count;
            }

            return Ok(roles);
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteRole(string id)
        {
            //delete role by their id

            var role = await _roleManager.FindByIdAsync(id);

            if (role is null)
            {
                return NotFound("Role not found.");
            }

            var result = await _roleManager.DeleteAsync(role);

            if (result.Succeeded)
            {
```

```
                return Ok(new { message = "Role deleted successfully." });
            }

            return BadRequest("Role deletion failed.");
        }

        [HttpPost("assign")]
        public async Task<IActionResult> AssignRole([FromBody] RoleAssignDto roleAssignDto)
        {
            var user = await _userManager.FindByIdAsync(roleAssignDto.UserId);

            if (user is null)
            {
                return NotFound("User not found");
            }

            var role = await _roleManager.FindByIdAsync(roleAssignDto.RoleId);

            if (role is null)
            {
                return NotFound("Role not found.");
            }

            var result = await _userManager.AddToRoleAsync(user, role.Name!);

            if (result.Succeeded)
            {
                return Ok(new { message = "Role assigned successfully" });
            }

            var error = result.Errors.FirstOrDefault();

            return BadRequest(error!.Description);
        }
    }
}
```