Step by step implementing Authentication And Authorization With Identity Framework and use Jwt in web api

In this Post, I implementing authentication and authorization in web api by using Identity and JWT

I will try to teach step by step how to set up implementing authentication and authorization

1. Install below packages using Nuget
2. Explanation about jwt and identity
3. Settings class (jwt,identity)
4. Add Configuration jwt and identity
5. its implementation
6. implementation login and register

JWT: JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

Identity : ASP.NET Core Identity is a membership system which allows you to add login functionality to your application.used to implement forms authentication

Get latest source code here : https://github.com/abolfazlSadeqi/Net7JwtAuthentication

Steps

1.Install below packages using Nuget

Package	Description	Category
Microsoft.AspNetCore.Authentication.JwtBearer	Contains types that enable support for JWT bearer based authentication.	Jwt
System.IdentityModel.Tokens.Jwt	Includes types that provide support for creating, serializing and validating JSON Web Tokens.	Jwt
Microsoft.AspNetCore.Identity.EntityFrameworkCore	Provides types for persisting Identity data with Entity Framework Core.	Identity

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi

2. Setting Identity in DbContext

a) Change Application DbContext inherit for IdentityDbContext

```
public partial class TestContext :
    IdentityDbContext<ApplicationUser, ApplicationRole, long, ApplicationUserClaim,
ApplicationUserRole, ApplicationUserLogin,
        ApplicationRoleClaim, ApplicationUserToken>
```

b) If you want to change the name of the related table Identity , you must change it in this section

```
protected override void OnModelCreating(ModelBuilder builder)
    {
       base.OnModelCreating(builder);

        builder.Entity<ApplicationUser>()
            .ToTable("Users");

        builder.Entity<ApplicationRole>().ToTable("Roles");
        builder.Entity<ApplicationUserRole>().ToTable("UserRoles");
        builder.Entity<ApplicationUserLogin>().ToTable("UserLogins");
        builder.Entity<ApplicationUserClaim>().ToTable("UserClaims");
        builder.Entity<ApplicationRoleClaim>().ToTable("RoleClaims");
        builder.Entity<ApplicationUserToken>().ToTable("UserTokens");
```

c) also,You can change the related table Identity (example user) with inherit for original table can use this table instead of original table in whole project

```
public class ApplicationUser : IdentityUser<long>
{
     public string FirstName { get; set; }
}
```

3.add Configuration Jwt in appsettings.json

```
"Jwt": {
    "Key": "",
    "Issuer": ""”
    "Audience": "",
    "Subject": ""
  }
```

4.Add Configuration base to startup

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

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi

5.add Configuration Identity to startup

a) Add the default identity system configuration for the specified User and Role types.

```
services.AddIdentity< ApplicationUser, ApplicationRole>()
                .AddEntityFrameworkStores<Context>()
                .AddDefaultTokenProviders();
```

b)Add ContextConnection in startup

```
services.AddDbContext<Context>(options => {
  options.UseSqlServer(Configuration.GetConnectionString("ContextConnection"));});
```

6) Add Authentication(Authentication schemes are specified by registering authentication services) to startup(The config is in the following Code)

https://github.com/abolfazlSadeqi/Net7JwtAuthentication/blob/master/Common/Common/UI/Method/HelperAuthentication.cs

7) Add Configure related an authorization header to your AddSwaggerGen (The config is in the following Code)

https://github.com/abolfazlSadeqi/Net7JwtAuthentication/blob/master/Common/Common/UI/Method/HelperSwagger.cs

8)add Configure SecurityToken jwt(The config is in the following Code)

https://github.com/abolfazlSadeqi/Net7JwtAuthentication/blob/master/Common/Common/UI/Jwt/HelperJwt.cs

9) implementation register Action(It uses identity base classes (UserManager<ApplicationUser>))

https://github.com/abolfazlSadeqi/Net7JwtAuthentication/blob/master/UI/API/Controller/Users/UserJwtController.cs

```
    [HttpPost]
    [Route("register")]
    public async Task<IActionResult> Register([FromBody] UserRegistration model)
    {

        if (!ModelState.IsValid)
            return StatusCode(StatusCodes.Status100Continue, "Not Valid");

        var userExists = await _userManager.FindByNameAsync(model.Username);

        if (userExists != null) return StatusCode(StatusCodes.Status500InternalServerError,
"username exists");
```

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi

```
        ApplicationUser user = new()
        {
            Email = model.Email,
            SecurityStamp = Guid.NewGuid().ToString(),
            UserName = model.Username,
            FirstName = model.FirstName,
            LastName = model.LastName,
            Title = model.Title,
            BirthDate = model.BirthDate,
        };

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

        if (!result.Succeeded) return StatusCode(StatusCodes.Status500InternalServerError,
"Failed to create user");


        return Ok("created successfully");
    }
```

10) Implementation login

a) First, check the user, the user is okay(It uses identity base classes (UserManager<ApplicationUser>))

b) If the user is valid, create a jwt token with Claim (Rols,username,userid,…)

```
    [HttpPost]
    [Route("login")]
    public async Task<IActionResult> Login([FromBody] UserLogin model)
    {

        var user = await _userManager.FindByNameAsync(model.Username);

        if (user == null) return Unauthorized();

        var check = await _userManager.CheckPasswordAsync(user, model.Password);

        if (!check) return Unauthorized();

        var userRoles = await _userManager.GetRolesAsync(user);

        var _listClaim = HelperJwt.GetClaim(userRoles, user.UserName, user.Id.ToString());
        var token = HelperJwt.GetToken(_listClaim, _configuration);

        return Ok(new { token = new JwtSecurityTokenHandler().WriteToken(token), expiration =
token.ValidTo });

    }
```

11) for testing with swagger. Click Authorize and enter token

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi

Defined JWT

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

JSON Web Token structure		
Header	**The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.**	**{ "alg": "HS256", "typ": "JWT" }**
Payload	**contains the claims.** **Claims are statements about an entity (typically, the user) and additional data.** **types of claims: registered, public, and private claims.** **Registered claims:** **not mandatory but recommended** **to provide a set of useful, interoperable claims.** **Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.** **Public claims:** **These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.** **Private claims:** **to share information between parties that agree on using them and are neither registered or public claims.**	**{ "sub": "1234567890", "name": "John Doe", "admin": true }**
Signature	**To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.** **The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.**	

Defined *Identity*

ASP.NET Core Identity is a membership system which allows you to add login functionality to your application.used to implement forms authentication

You can configure ASP.NET Core Identity to use a SQL Server database to store user names, passwords, and profile data.

My Linkedin: https://www.linkedin.com/in/abolfazlsadeghi/ My Github: https://github.com/abolfazlSadeqi
My StackoverFlow: https://stackoverflow.com/users/10193401/abolfazl-sadeghi