**Step-by-Step JWT Setup in ASP.NET Core Web API**

This guide explains how to add JWT (JSON Web Token) to your Web API project.

Step 1: Add NuGet Packages

* Microsoft.AspNetCore.Authentication.JwtBearer

Step 2: Add JWT Configuration in appsettings.json

{

"Jwt": {

"Key": "ThisIsAStrongSecretKeyForJwt12345678",

"Issuer": "SimpleAuthApi",

"Audience": "SimpleAuthApiUser"

}

}

**Explanation:**

* Key: A long string used to sign the token securely.
* Issuer: Who created the token (your API name).
* Audience: Who will use the token (like your frontend or user).

Step 3: Configure JWT in Program.cs

Read values from appsettings.json

var key = builder.Configuration["Jwt:Key"];

var issuer = builder.Configuration["Jwt:Issuer"];

var audience=builder.Configuration["Jwt:Audience"];

**Explanation:**

* These lines **read the JWT values** from appsettings.json so they can be used in the code.

1. Add Authentication and JWT Bearer

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme) .AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true, // Check the token issuer

ValidateAudience = true, // Check the token audience

ValidateLifetime = true, // Check if token is expired ValidateIssuerSigningKey = true, // Check if token is valid using secret key

ValidIssuer = issuer,

ValidAudience = audience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key))

};

});

**Explanation (each line):**

* AddAuthentication(...): Tells .NET Core that we want to use JWT for authentication.
* AddJwtBearer(...): Configure JWT validation settings.
* ValidateIssuer: Check if the token is from the right issuer.
* ValidateAudience: Check if the token is meant for this API.
* ValidateLifetime: Check if the token has expired.
* ValidateIssuerSigningKey: Verify token’s signature using your key.
* ValidIssuer, ValidAudience: Must match what's in your appsettings.json.
* IssuerSigningKey: Converts the string key to bytes so it can validate the token.

3. Add Authentication and Authorization middleware app.UseAuthentication(); app.UseAuthorization();

Step 4: Add Swagger JWT Support

Inside

AddSwaggerGen: options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name = "Authorization",

Type = SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT",

In = ParameterLocation.Header,

Description = "Enter: Bearer <token>"

});

options.AddSecurityRequirement(new OpenApiSecurityRequirement

{ {

new OpenApiSecurityScheme {

Reference = new OpenApiReference {

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

**Explanation:**

* This adds a **🔒 button** in Swagger.

Step 5: Generate JWT in your Login Method

1. Create user identityvar claims = new[] { new Claim(ClaimTypes.Name, user.Username)

};

**Explanation:**

* Claim: Basic info stored in the token (like username).
* ClaimTypes.Name: Standard way to store username.

1. Create signing keyvar key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));

Converts the Jwt:Key string to bytes, so it can be used securely.

1. var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

This line says: “Sign my token using this secret key and SHA256 algorithm.”

1. Create tokenvar token = new JwtSecurityToken( issuer: \_configuration["Jwt:Issuer"], audience: \_configuration["Jwt:Audience"], claims: claims, expires: DateTime.Now.AddMinutes(30), signingCredentials: creds

);

**Explanation:**

* This creates the token with:
  + issuer: who created it
  + audience: who can use it
  + claims: user info inside the token
  + expires: valid for 30 minutes
  + signingCredentials: signed with your key

1. Return token stringreturn new JwtSecurityTokenHandler().WriteToken(token);

**Explanation:**

* Converts the token object to a string.
* This string is what the frontend/client uses in the Authorization header.

Step 6: Use [Authorize] to Protect APIs

[Authorize] public IActionResult Profile() { ... }

**Explanation:**

* [Authorize]: Only logged-in users with valid tokens can access.