**Step 1: Install Required Packages**

You need to install the necessary NuGet packages to handle JWT authentication. These packages will allow you to work with JWTs in the .NET Core app.

bash

Copy code

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

**Step 2: Configure Authentication in Startup.cs or Program.cs**

In .NET Core 6+ or .NET 7, you can add JWT authentication configuration in Program.cs. Otherwise, you would configure it in Startup.cs.

**For .NET Core 6+ or .NET 7:**

Modify the Program.cs file.

csharp

Copy code

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Configure JWT Authentication

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

var app = builder.Build();

// Enable authentication

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Step 3: Define JWT Settings in appsettings.json**

You can store your JWT configuration, like the signing key, issuer, and audience, in appsettings.json.

json

Copy code

{

"Jwt": {

"Key": "YourSuperSecretKey", // Should be stored securely (e.g., in environment variables)

"Issuer": "YourIssuer",

"Audience": "YourAudience"

}

}

**Step 4: Create a JWT Token Generation Method**

Next, create a method that will generate JWT tokens. You can add this in a service class or directly in your controller.

csharp

Copy code

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using Microsoft.IdentityModel.Tokens;

using System.Text;

public class JwtTokenService

{

private readonly IConfiguration \_configuration;

public JwtTokenService(IConfiguration configuration)

{

\_configuration = configuration;

}

public string GenerateToken(UserInfo userInfo)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, userInfo.Username),

new Claim(JwtRegisteredClaimNames.Email, userInfo.EmailAddress),

new Claim("DateOfJoing", userInfo.DateOfJoing.ToString("yyyy-MM-dd")),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var token = new JwtSecurityToken(

issuer: \_configuration["Jwt:Issuer"],

audience: \_configuration["Jwt:Audience"],

claims: claims,

expires: DateTime.Now.AddMinutes(30),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**Step 5: Create a Login API to Generate JWT**

Now, create an API that generates the JWT when the user successfully logs in. You can use the JwtTokenService to generate the token.

csharp

Copy code

[Route("api/[controller]")]

[ApiController]

public class AuthController : ControllerBase

{

private readonly JwtTokenService \_jwtTokenService;

public AuthController(JwtTokenService jwtTokenService)

{

\_jwtTokenService = jwtTokenService;

}

[HttpPost("login")]

public IActionResult Login([FromBody] UserLoginRequest request)

{

// Normally, you would validate the user credentials here.

// If valid, generate JWT token.

if (IsValidUser(request))

{

var userInfo = new UserInfo { Username = request.Username, EmailAddress = "test@example.com", DateOfJoing = DateTime.Now };

var token = \_jwtTokenService.GenerateToken(userInfo);

return Ok(new { Token = token });

}

return Unauthorized();

}

private bool IsValidUser(UserLoginRequest request)

{

// Validate user credentials (for demo, any username/password is valid)

return request.Username == "test" && request.Password == "password";

}

}

**Step 6: Protect Routes with JWT Authentication**

Once the token is generated, you can secure your API endpoints by adding the [Authorize] attribute. This ensures that only requests with valid JWTs can access these routes.

csharp

Copy code

[Authorize]

[Route("api/[controller]")]

[ApiController]

public class UserController : ControllerBase

{

[HttpGet("profile")]

public IActionResult GetProfile()

{

var username = User.Identity.Name; // Retrieve username from JWT token

return Ok(new { Message = $"Welcome {username}!" });

}

}

**Step 7: Testing the Flow**

1. **Login and Get the Token:**
   * Send a POST request to /api/auth/login with valid credentials to receive a JWT token in the response.

Example Request:

json

Copy code

{

"username": "test",

"password": "password"

}

Example Response:

json

Copy code

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

}

1. **Access Protected Route:**
   * Use the token received to access a protected route (e.g., /api/user/profile) by adding it to the Authorization header:

http

Copy code

GET /api/user/profile

Authorization: Bearer <your-token-here>

**Step 8: Token Validation**

When a request is made to a protected endpoint, the JwtBearer middleware validates the token by checking the signature, issuer, audience, and expiration. If any of these checks fail, the request will be rejected with a 401 Unauthorized response.

**Summary of Key Parts:**

1. **Token Creation:** Create JWT tokens using JwtSecurityTokenHandler.
2. **Token Configuration:** Store keys, issuer, and audience in appsettings.json.
3. **Authentication Setup:** Configure JWT Bearer authentication in Program.cs or Startup.cs.
4. **Protected Routes:** Use the [Authorize] attribute to protect routes.
5. **Token Validation:** The middleware validates the JWT token for every request to a protected route.

**1. Registered Claims:**

These are predefined, optional claims that have standard meanings according to the JWT specification (RFC 7519). Examples include:

* **sub** (Subject): Identifies the principal that is the subject of the JWT, often the user ID or username.
* **email**: Email address of the user.
* **jti** (JWT ID): A unique identifier for the JWT, used to avoid token replay.
* **iss** (Issuer): Identifies the issuer of the token.
* **aud** (Audience): Identifies the recipients that the JWT is intended for.
* **exp** (Expiration Time): The time after which the JWT is no longer valid.
* **iat** (Issued At): The time at which the JWT was issued.
* **nbf** (Not Before): The time before which the JWT must not be accepted for processing.

Example from your code:

csharp

Copy code

new Claim(JwtRegisteredClaimNames.Sub, userInfo.Username), // Registered claim for 'sub'

new Claim(JwtRegisteredClaimNames.Email, userInfo.EmailAddress), // Registered claim for 'email'

**2. Public Claims:**

These are custom claims that are not part of the standard but are defined and understood by both the token issuer and the token consumer. They should avoid collisions with registered claims.

Example:

csharp

Copy code

new Claim("DateOfJoing", userInfo.DateOfJoing.ToString("yyyy-MM-dd")) // Custom claim for date of joining

**3. Private Claims:**

These are custom claims agreed upon by the parties using the JWT (issuer and consumer), and they do not have any significance outside the specific use case. Private claims should be used carefully to avoid conflicts with public or registered claims.

Private claims may look like:

csharp

Copy code

new Claim("role", "admin") // Custom private claim indicating user role

In your example:

* JwtRegisteredClaimNames.Sub and JwtRegisteredClaimNames.Email are **registered claims**.
* "DateOfJoing" is a **public claim**.
* JwtRegisteredClaimNames.Jti is another **registered claim**.

Generate Token  
  
 public TokenVM GenerateTokenFromClaims(ICollection<Claim> claims)

{

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_options.JwtKey));

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

var expires = DateTime.Now.AddMinutes(\_options.JwtExpireInMinutes);

IdentityModelEventSource.ShowPII = true;

var token = new JwtSecurityToken(

\_options.JwtIssuer,

\_options.JwtAudience,

claims,

expires: expires,

signingCredentials: creds

);

return new TokenVM()

{

token = new JwtSecurityTokenHandler().WriteToken(token),

tokenexptime = token.ValidTo.Subtract(token.ValidFrom).TotalMilliseconds

};

}

Validate Token  
  
public bool ValidateToken(string token)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_options.JwtKey));

try

{

tokenHandler.ValidateToken(token, new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = key,

ValidateIssuer = false,

ValidateAudience = false,

ClockSkew = TimeSpan.Zero // Token expires exactly at the expiry time, no additional tolerance.

}, out SecurityToken validatedToken);

// Token is valid

return true;

}

catch (SecurityTokenExpiredException)

{

// Token has expired

return false;

}

catch (Exception)

{

// Token is invalid for some other reason

return false;

}

}