Here's an example of how to implement JWT authentication in an ASP.NET Core application using SQL Server for storing user credentials. This example includes:

1. Setting up the database.
2. Configuring JWT authentication.
3. Creating a simple login API to generate JWT tokens.

**Step 1: Set up the Database**

1. Create a SQL Server database (e.g., AuthDb) and a table Users to store user information.

sql

Copy

CREATE DATABASE AuthDb;

USE AuthDb;

CREATE TABLE Users

(

Id INT PRIMARY KEY IDENTITY,

Username NVARCHAR(50) NOT NULL,

Password NVARCHAR(255) NOT NULL

);

1. Add a user to test the login functionality.

sql

Copy

INSERT INTO Users (Username, Password)

VALUES ('testuser', 'password123'); -- Password should be hashed in production

**Step 2: Create ASP.NET Core Project**

1. Create a new ASP.NET Core Web API project.

bash

Copy

dotnet new webapi -n JwtAuthExample

1. Add the required NuGet packages for JWT authentication.

bash

Copy

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

**Step 3: Configure Services and JWT Authentication**

1. **Create a User model**:

csharp

Copy

public class User

{

public int Id { get; set; }

public string Username { get; set; }

public string Password { get; set; } // Store hashed password in production

}

1. **Create a JwtSettings class** to hold JWT configuration:

csharp

Copy

public class JwtSettings

{

public string SecretKey { get; set; }

public string Issuer { get; set; }

public string Audience { get; set; }

public int ExpirationMinutes { get; set; }

}

1. **Add DbContext** to interact with the SQL Server database:

csharp

Copy

public class AppDbContext : DbContext

{

public DbSet<User> Users { get; set; }

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

}

1. **Add the JWT configuration to appsettings.json:**

json

Copy

{

"JwtSettings": {

"SecretKey": "YourSuperSecretKeyHere1234567890",

"Issuer": "JwtAuthExample",

"Audience": "JwtAuthExampleAudience",

"ExpirationMinutes": 60

},

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=AuthDb;Trusted\_Connection=True;"

}

}

1. **Configure services in Startup.cs (or Program.cs for .NET 6/7):**

csharp

Copy

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

// Configure DbContext

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

// Add JWT configuration

services.Configure<JwtSettings>(Configuration.GetSection("JwtSettings"));

// Configure JWT Authentication

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.RequireHttpsMetadata = false;

options.SaveToken = true;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = Configuration["JwtSettings:Issuer"],

ValidAudience = Configuration["JwtSettings:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["JwtSettings:SecretKey"]))

};

});

// Add other services

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseRouting();

// Enable authentication middleware

app.UseAuthentication();

// Enable authorization middleware

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

**Step 4: Create the Authentication Service**

1. **Create an IAuthService interface and its implementation:**

csharp

Copy

public interface IAuthService

{

Task<string> Authenticate(string username, string password);

}

1. **Implement AuthService:**

csharp

Copy

public class AuthService : IAuthService

{

private readonly AppDbContext \_context;

private readonly JwtSettings \_jwtSettings;

public AuthService(AppDbContext context, IOptions<JwtSettings> jwtSettings)

{

\_context = context;

\_jwtSettings = jwtSettings.Value;

}

public async Task<string> Authenticate(string username, string password)

{

var user = await \_context.Users.SingleOrDefaultAsync(u => u.Username == username);

if (user == null || user.Password != password) // Use hashed passwords in production

{

return null;

}

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_jwtSettings.SecretKey));

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

var token = new JwtSecurityToken(

issuer: \_jwtSettings.Issuer,

audience: \_jwtSettings.Audience,

expires: DateTime.Now.AddMinutes(\_jwtSettings.ExpirationMinutes),

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

**Step 5: Create a Controller for Login**

1. **Create AuthController:**

csharp

Copy

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

[ApiController]

public class AuthController : ControllerBase

{

private readonly IAuthService \_authService;

public AuthController(IAuthService authService)

{

\_authService = authService;

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] LoginRequest request)

{

var token = await \_authService.Authenticate(request.Username, request.Password);

if (string.IsNullOrEmpty(token))

{

return Unauthorized("Invalid credentials");

}

return Ok(new { Token = token });

}

}

public class LoginRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

**Step 6: Add Dependency Injection and Run the Application**

1. **Register the AuthService in Startup.cs (or Program.cs for .NET 6/7)**

csharp

Copy

services.AddScoped<IAuthService, AuthService>();

1. **Run the application:**

Now you can run your ASP.NET Core application. To test:

* Make a POST request to https://localhost:5001/api/auth/login with a JSON body:

json

Copy

{

"username": "testuser",

"password": "password123"

}

You should get a JWT token in the response.

**Step 7: Protect Routes Using JWT**

To protect routes with JWT authentication, add the [Authorize] attribute on controllers or actions:

csharp

Copy

[Authorize]

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

public class SomeProtectedController : ControllerBase

{

[HttpGet]

public IActionResult GetProtectedData()

{

return Ok(new { message = "This is protected data" });

}

}

Now, only requests with a valid JWT token will be able to access this route.

Bottom of Form