**MICROSERVICES JWT**

NuGet Package

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

appsettings.json

{

"Jwt": {

"Key": "ThisIsASecretKeyForJwtToken",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}

Program.cs

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", 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"]))

};

});

builder.Services.AddAuthorization();

AuthController.cs

[ApiController]

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

public class AuthController : ControllerBase

{

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized();

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("ThisIsASecretKeyForJwtToken"));

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

var token = new JwtSecurityToken(

issuer: "MyAuthServer",

audience: "MyApiUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(60),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

Example of Secured Endpoint

[Authorize]

[HttpGet("secure-data")]

public IActionResult GetSecureData()

{

return Ok("This is protected data");

}

LoginModel and IsValidUser() Stub

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

private bool IsValidUser(LoginModel model)

{

return model.Username == "admin" && model.Password == "password";

}

Call POST /api/auth/login with:

{

"username": "admin",

"password": "password"

}

ANALYSIS:

The implementation of JWT authentication in the ASP.NET Core Web API effectively demonstrates a foundational, secure login mechanism suitable for microservices. It allows users to authenticate via a /login endpoint, receive a signed JWT token, and access protected endpoints using the [Authorize] attribute. The configuration correctly uses token validation parameters such as issuer, audience, and signing key, ensuring basic security. However, the approach is simplistic and not production-ready as it includes hardcoded secrets, lacks user database validation, omits refresh tokens, and doesn't support token revocation. While it's excellent for learning and prototyping, enhancing it with secure key storage, role-based claims, token persistence, and integration with ASP.NET Identity would be essential for a robust, real-world application.