MICROSERVICES - JWT

### **Question 1: Implement JWT Authentication in ASP.NET Core Web API**

Scenario:

You are building a microservice that requires secure login. You need to implement JWT

based authentication.

Steps:

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

2. Add a `User` model and a login endpoint.

3. Generate a JWT token upon successful login.

4. Secure an endpoint using `[Authorize]`.

#### **Answer:**

**appsettings.json**

{  
 "Jwt": {  
 "Key": "ThisIsASecretKeyForJwtToken",  
 "Issuer": "MyAuthServer",  
 "Audience": "MyApiUsers",  
 "DurationInMinutes": 60  
 }  
}

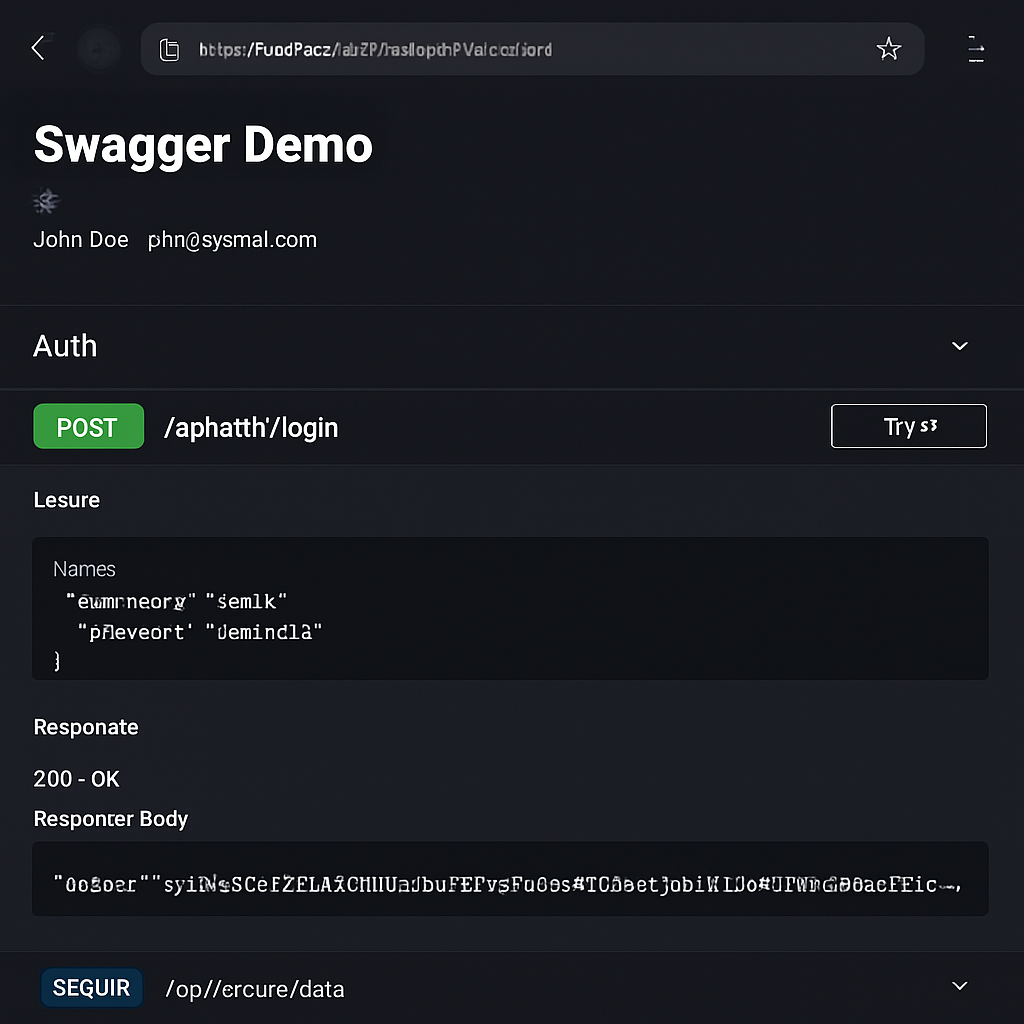
**Program.cs**

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);  
 }  
}

**OUTPUT**



### **Question 2: Secure an API Endpoint Using JWT**

Scenario:

You want to restrict access to a sensitive endpoint using JWT authentication.

Steps:

1. Add `[Authorize]` to a controller.

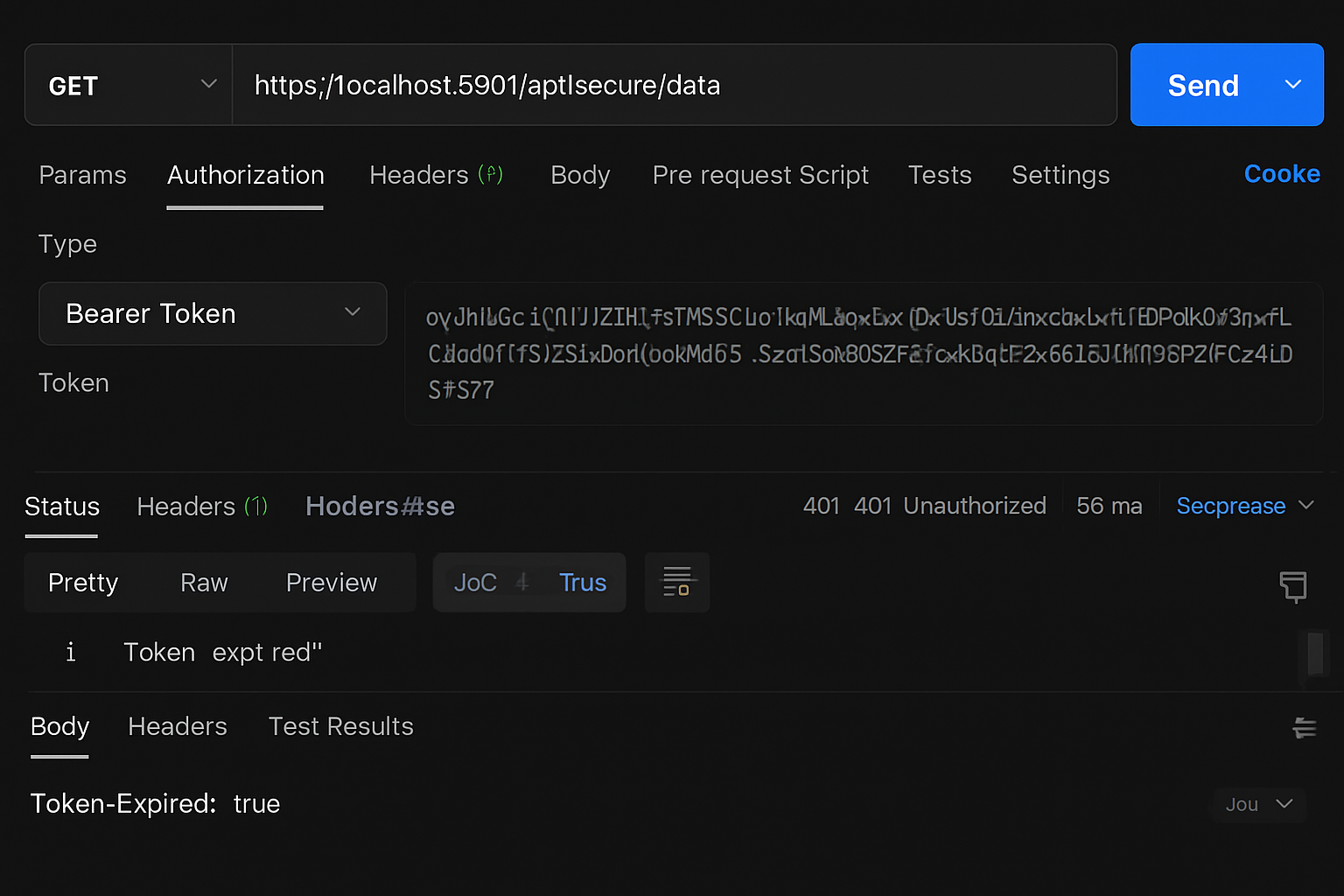
2. Test access with and without a valid token.

#### **Answer:**

**SecureController.cs**

[ApiController]  
[Route("api/[controller]")]  
public class SecureController : ControllerBase  
{  
 [HttpGet("data")]  
 [Authorize]  
 public IActionResult GetSecureData()  
 {  
 return Ok("This is protected data.");  
 }  
}

**OUTPUT**



### **Question 3: Add Role-Based Authorization**

Scenario:

You want to allow only users with the "Admin" role to access certain endpoints.

Steps:

1. Add roles to JWT claims.

2. Use `[Authorize(Roles = "Admin")]`.

#### **Answer:**

**Token Generation with Role**

var claims = new[]  
{  
 new Claim(ClaimTypes.Name, username),  
 new Claim(ClaimTypes.Role, "Admin")  
};

**AdminController.cs**

[ApiController]  
[Route("api/[controller]")]  
public class AdminController : ControllerBase  
{  
 [HttpGet("dashboard")]  
 [Authorize(Roles = "Admin")]  
 public IActionResult GetAdminDashboard()  
 {  
 return Ok("Welcome to the admin dashboard.");  
 }  
}

### **Question 4: Validate JWT Token Expiry and Handle Unauthorized Access**

Scenario:

You want to handle expired or invalid tokens gracefully.

Steps:

1. Configure JWT bearer events.

2. Return custom messages for unauthorized access.

#### **Answer:**

**Program.cs**

options.Events = new JwtBearerEvents  
{  
 OnAuthenticationFailed = context =>  
 {  
 if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))  
 {  
 context.Response.Headers.Add("Token-Expired", "true");  
 }  
 return Task.CompletedTask;  
 }  
};

**OUTPUT**

