**Week-05 HandsOn Solution**

**Microservices**

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

* **Code:**

using System;

using JwtMicroservice.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtMicroservice.Controllers

{

    [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 bool IsValidUser(LoginModel model)

        {

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

        }

        private string GenerateJwtToken(string username)

        {

            var claims = new[]

            {

                new Claim(ClaimTypes.Name, username),

                new Claim(ClaimTypes.Role, "Admin")

            };

            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);

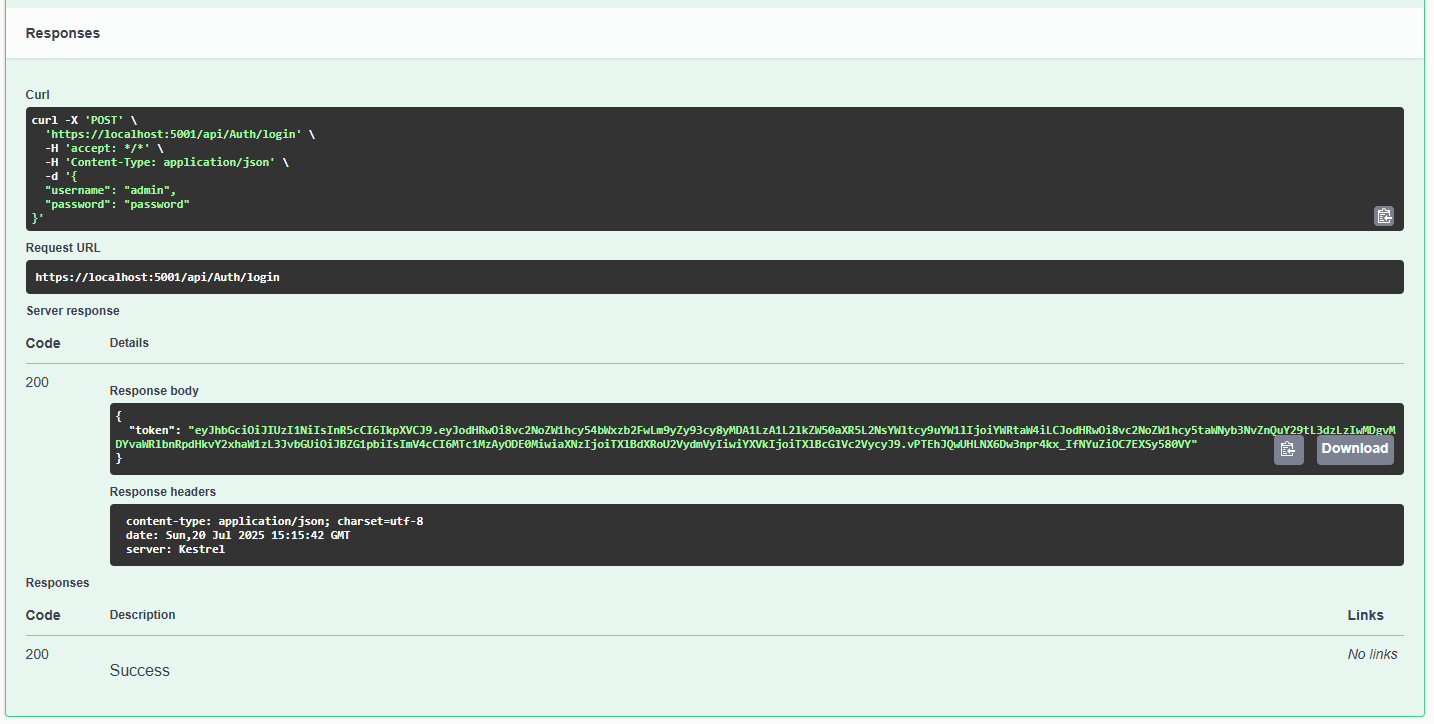
            return new JwtSecurityTokenHandler().WriteToken(token);

        }

    }

}

* **Output:**

****

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

* **Code:**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtMicroservice.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class SecureController : ControllerBase

    {

        [HttpGet("data")]

        [Authorize]

        public IActionResult GetSecureData()

        {

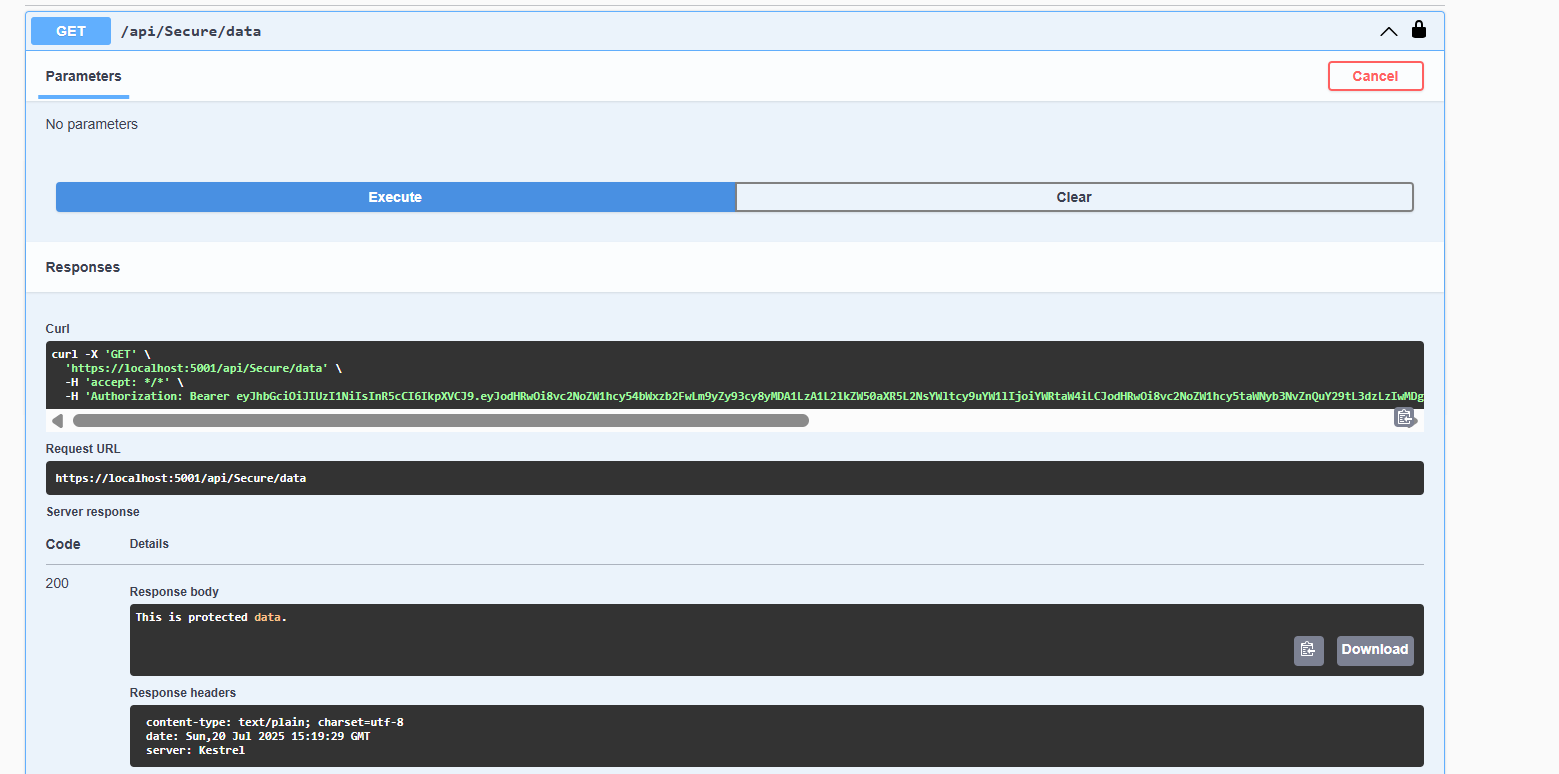
            return Ok("This is protected data.");

        }

    }

}

* **Output:**

****

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

* **Code:**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtMicroservice.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class AdminController : ControllerBase

    {

        [HttpGet("dashboard")]

        [Authorize(Roles = "Admin")]

        public IActionResult GetAdminDashboard()

        {

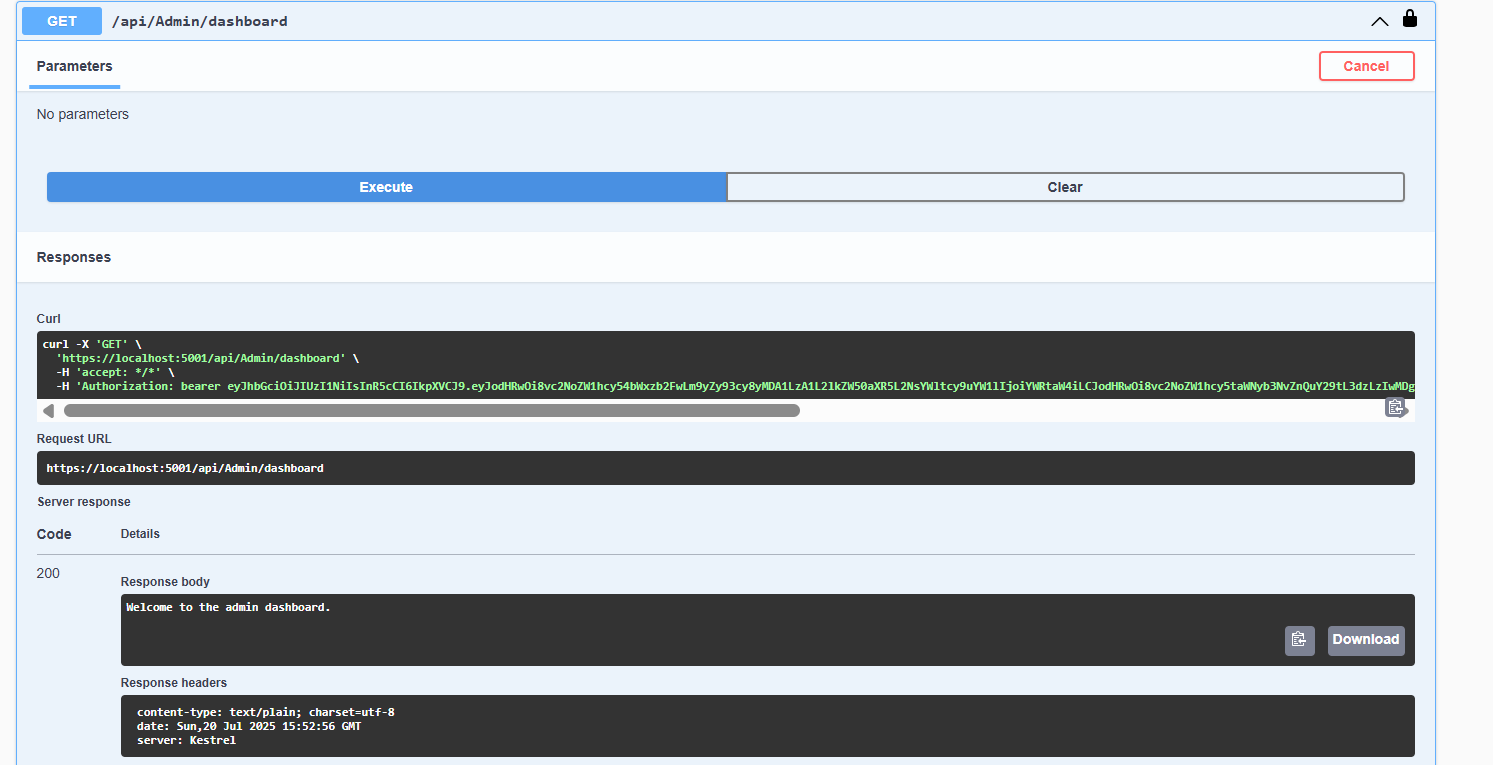
            return Ok("Welcome to the admin dashboard.");

        }

    }

}

* **Output:**

****

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

* **Code:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

using System.Threading.Tasks;

using System;

var builder = Microsoft.AspNetCore.Builder.WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

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"]))

        };

        options.Events = new JwtBearerEvents

        {

            OnAuthenticationFailed = context =>

            {

                if (context.Exception is SecurityTokenExpiredException)

                {

                    context.Response.Headers.Append("Token-Expired", "true");

                }

                return Task.CompletedTask;

            }

        };

});

builder.Services.AddAuthorization();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "JWT API", Version = "v1" });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

    {

        Description = "JWT Authorization header using the Bearer scheme.",

        Name = "Authorization",

        In = ParameterLocation.Header,

        Type = SecuritySchemeType.ApiKey,

        Scheme = "Bearer"

});

    c.AddSecurityRequirement(new OpenApiSecurityRequirement

    {

        {

            new OpenApiSecurityScheme {

                Reference = new OpenApiReference {

                    Type = ReferenceType.SecurityScheme,

                    Id = "Bearer"

                }

            },

            Array.Empty<string>()

        }

    });

});

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();