**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 JWTbased 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]`

AuthController.cs:

using JwtAuthDemo.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtAuthDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class AuthController : ControllerBase

    {

        private readonly IConfiguration \_config;

        public AuthController(IConfiguration config)

        {

            \_config = config;

        }

        [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)

        {

            // For demo purposes; replace with real user check

            return model.Username == "mani" && model.Password == "mani123";

        }

        private string GenerateJwtToken(string username)

        {

            var claims = new[]

            {

                new Claim(ClaimTypes.Name, username)

            };

            // Use GetValue with null-check to avoid CS8604

            var keyString = \_config.GetValue<string>("Jwt:Key")

                         ?? throw new InvalidOperationException("JWT Key is missing in configuration.");

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

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

            var token = new JwtSecurityToken(

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

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

                claims: claims,

                expires: DateTime.Now.AddMinutes(

                    Convert.ToDouble(\_config["Jwt:DurationInMinutes"] ?? "60")),

                signingCredentials: creds

            );

            return new JwtSecurityTokenHandler().WriteToken(token);

        }

    }

}

SecureController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class SecureController : ControllerBase

    {

        [HttpGet]

        [Authorize]

        public IActionResult Get()

        {

            return Ok(" This is a protected endpoint!");

        }

    }

}

LoginModels.cs:

using System.ComponentModel.DataAnnotations;

namespace JwtAuthDemo.Models

{

    public class LoginModel

    {

        [Required]

        public string Username { get; set; }

        [Required]

        public string Password { get; set; }

    }

}

Appsettings.json:

{

  "Jwt": {

    "Key": " 12345678901234567890123456789012",

    "Issuer": "JwtAuthDemo",

    "Audience": "JwtAuthDemoUser"

  },

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  },

  "AllowedHosts": "\*"

}

Program.cs:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add 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"]))

    };

});

builder.Services.AddAuthorization();

builder.Services.AddControllers();

// Add Swagger with JWT Support

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(options =>

{

    options.SwaggerDoc("v1", new OpenApiInfo { Title = "JwtAuthDemo", Version = "v1" });

    //

JWT Bearer Configuration for Swagger

    options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

    {

        Name = "Authorization",

        Type = SecuritySchemeType.ApiKey,

        Scheme = "Bearer",

        BearerFormat = "JWT",

        In = ParameterLocation.Header,

        Description = "Enter 'Bearer' followed by your token.\n\nExample: `Bearer abc123...`"

    });

    options.AddSecurityRequirement(new OpenApiSecurityRequirement

    {

        {

            new OpenApiSecurityScheme

            {

                Reference = new OpenApiReference

                {

                    Type = ReferenceType.SecurityScheme,

                    Id = "Bearer"

                }

            },

            Array.Empty<string>()

        }

    });

});

var app = builder.Build();

// Swagger middleware (only in dev)

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

// Enable Authentication & Authorization

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

Output:





