**1a)Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.**

**Producer.cs:**

using System;

using Confluent.Kafka;

class Producer

{

public static async Task Start()

{

var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

using var producer = new ProducerBuilder<Null, string>(config).Build();

Console.WriteLine("Enter messages to send (type 'exit' to quit):");

while (true)

{

var message = Console.ReadLine();

if (message == "exit") break;

await producer.ProduceAsync("chat-topic", new Message<Null, string> { Value = message });

}

}

}

**Consumer.cs:**

using System;

using Confluent.Kafka;

class Consumer

{

public static void Start()

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();

consumer.Subscribe("chat-topic");

Console.WriteLine("Waiting for messages... (Ctrl+C to exit)");

try

{

while (true)

{

var cr = consumer.Consume();

Console.WriteLine($"Received: {cr.Value}");

}

}

catch (OperationCanceledException)

{

consumer.Close();

}

}

}

**Program.cs:**

using System;

class Program

{

static async Task Main(string[] args)

{

Console.WriteLine("Type 'p' for producer or 'c' for consumer:");

var choice = Console.ReadLine();

if (choice == "p")

await Producer.Start();

else if (choice == "c")

Consumer.Start();

else

Console.WriteLine("Invalid choice.");

}

}

**Ouput:**

info: Package Reference for package 'Confluent.Kafka' version '2.11.0' added to file 'C:\Users\dell\KafkaChatApp\KafkaChatApp.csproj'.

info: Writing assets file to disk. Path: C:\Users\dell\KafkaChatApp\obj\pro ject.assets.json

log : Restored C:\Users\dell\KafkaChatApp\KafkaChatApp.csproj (in 11.21 sec ).

C:\Users\dell\KafkaChatApp>echo. > Producer.cs

C:\Users\dell\KafkaChatApp>echo. > Consumer.cs

C:\Users\dell\KafkaChatApp>notepad Producer.cs

C:\Users\dell\KafkaChatApp>notepad Consumer.cs

C:\Users\dell\KafkaChatApp>notepad Program.cs

C:\Users\dell\KafkaChatApp>dotnet run

C:\Users\dell\KafkaChatApp\Consumer.cs (26,48): warning CS0618: 'ConsumeResul

t<Ignore, string>.Value' is obsolete: 'Please access the message Value via. M essage. Value.'

C:\Users\dell\KafkaChatApp\Producer.cs(20,91): warning CS8601: Possible null

reference assignment.

Type 'p' for producer or 'c' for consumer:

P

Enter messages to send (type 'exit' to quit):

Hai

Hello

My self Gopesh Sharma

Microsoft Windows [Version 10.0.26100.4652] (c) Microsoft Corporation. All rights reserved.

C:\Users\dell>cd kafkaChatApp

C:\Users\dell\KafkaChatApp>dotnet run

Type 'p' for producer or 'c' for consumer:

C

Waiting for messages... (Ctrl+C to exit)

Received: Hai

Received: Hello

Received: My self Gopesh Sharma

**Create a Chat Application using C# Windows Application using Kafka and consume the message in different client applications**.

**Chatsender:**

using System;

using System.Drawing;

using System.Threading.Tasks;

using System.Windows.Forms;

using Confluent.Kafka;

namespace KafkaWinChatSender

{

public class SenderForm : Form

{

private TextBox txtMessage;

private Button btnSend;

public SenderForm()

{

this.Text = "Kafka Chat - Sender";

this.Size = new Size(400, 150);

txtMessage = new TextBox { Location = new Point(20, 20), Size = new Size(250, 25) };

btnSend = new Button { Text = "Send", Location = new Point(280, 20), Size = new Size(75, 25) };

btnSend.Click += BtnSend\_Click;

Controls.Add(txtMessage);

Controls.Add(btnSend);

}

private async void BtnSend\_Click(object sender, EventArgs e)

{

var message = txtMessage.Text.Trim();

if (!string.IsNullOrEmpty(message))

{

await SendMessage("Sender: " + message);

txtMessage.Clear();

}

}

private async Task SendMessage(string message)

{

var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

using var producer = new ProducerBuilder<Null, string>(config).Build();

await producer.ProduceAsync("chat-topic", new Message<Null, string> { Value = message });

}

[STAThread]

public static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new SenderForm());

}

}

}

**ChatReceiver:**

using System;

using System.Drawing;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Confluent.Kafka;

namespace KafkaWinChatReceiver

{

public class ReceiverForm : Form

{

private ListBox lstMessages;

public ReceiverForm()

{

this.Text = "Kafka Chat - Receiver";

this.Size = new Size(400, 500);

lstMessages = new ListBox

{

Location = new Point(20, 20),

Size = new Size(340, 400)

};

Controls.Add(lstMessages);

StartConsumer();

}

private void StartConsumer()

{

Task.Run(() =>

{

var config = new ConsumerConfig

{

GroupId = "receiver-group",

BootstrapServers = "localhost:9092",

AutoOffsetReset = AutoOffsetReset.Earliest

};

using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();

consumer.Subscribe("chat-topic");

while (true)

{

var result = consumer.Consume(CancellationToken.None);

Invoke((MethodInvoker)(() =>

{

lstMessages.Items.Add(result.Message.Value);

}));

}

});

}

[STAThread]

public static void Main()

{

Application.EnableVisualStyles();

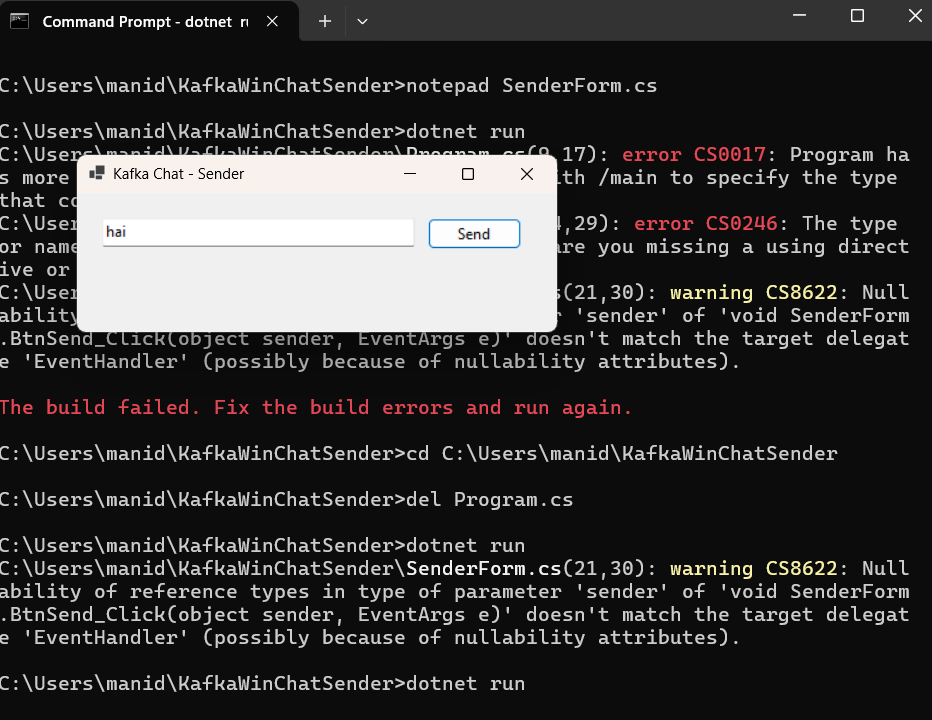
Application.SetCompatibleTextRenderingDefault(false);

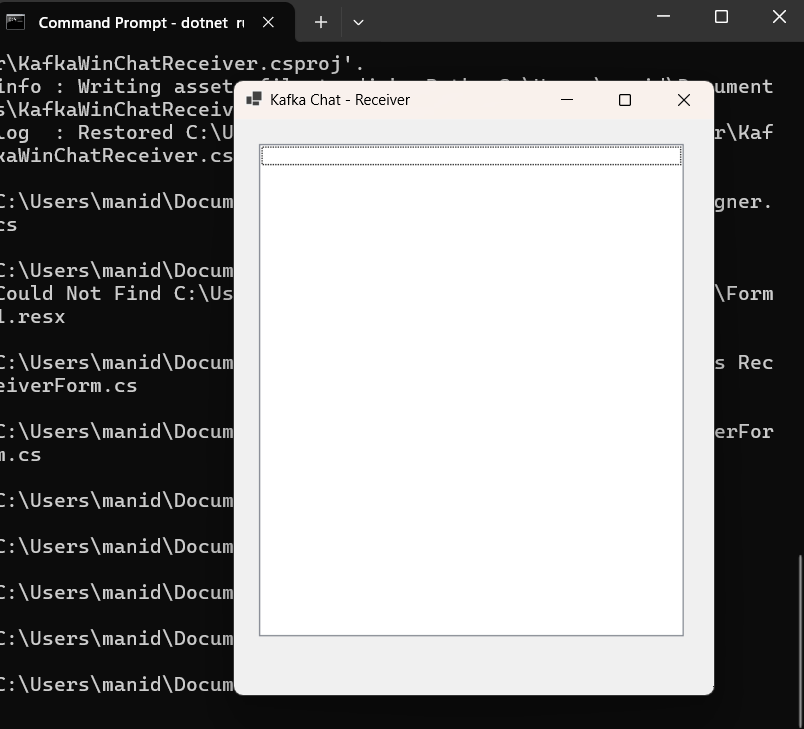
Application.Run(new ReceiverForm());

}

}

}





**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

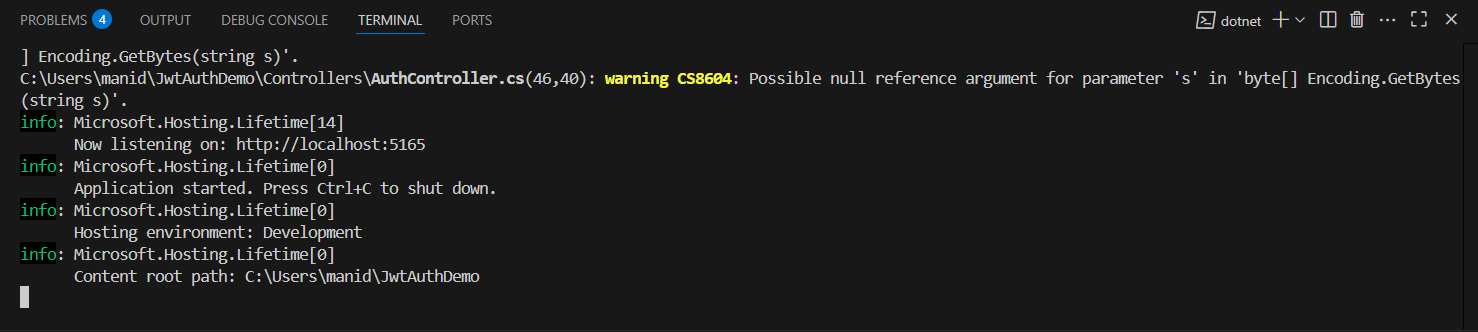
app.UseAuthentication();

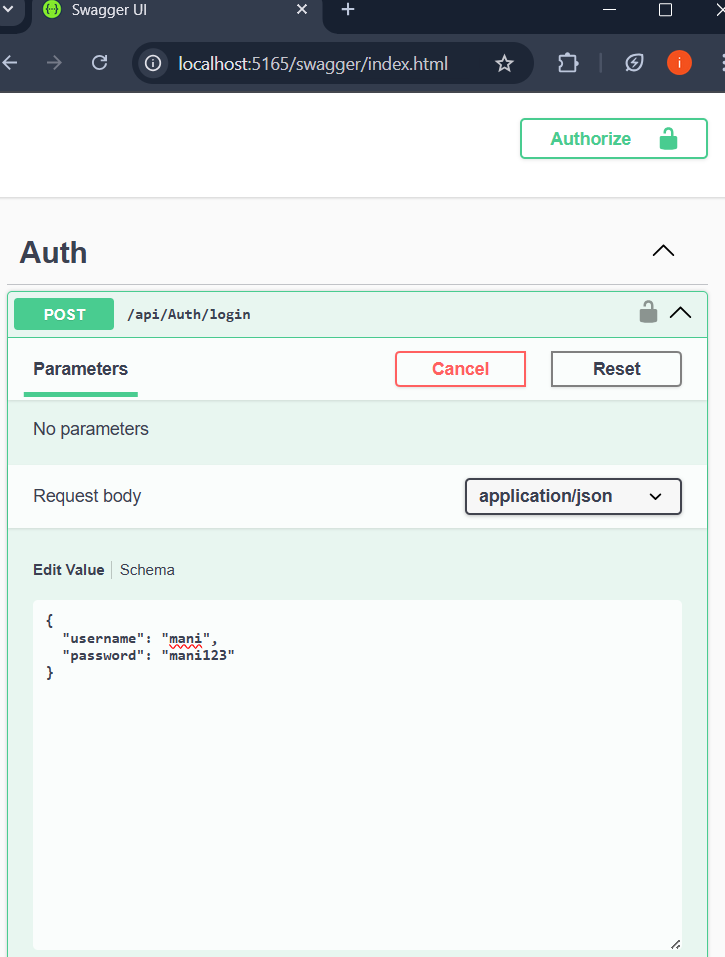
app.UseAuthorization();

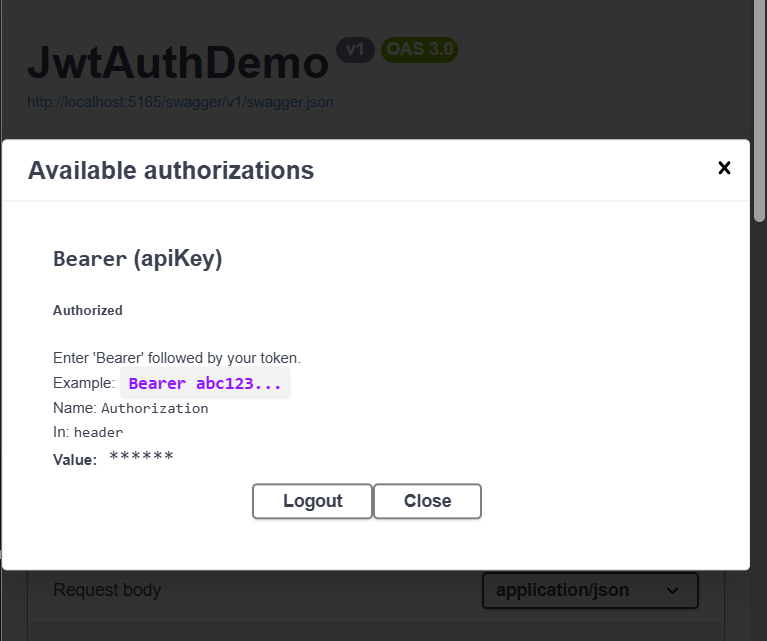
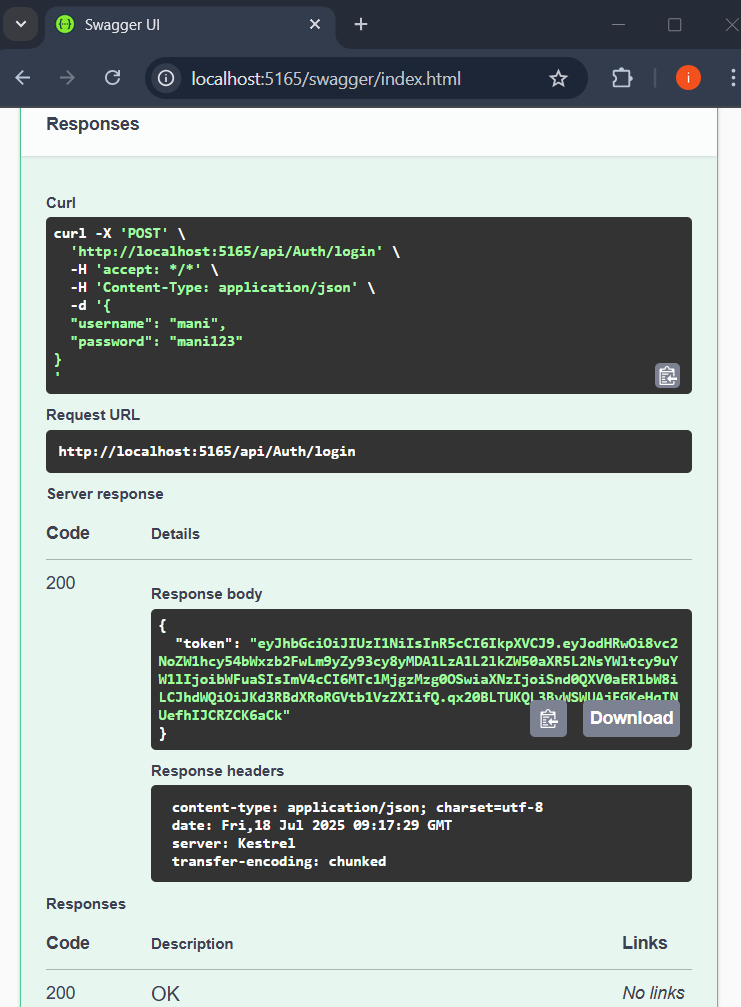
app.MapControllers();

app.Run();

**Ouput:**







**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.**

**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()

        {

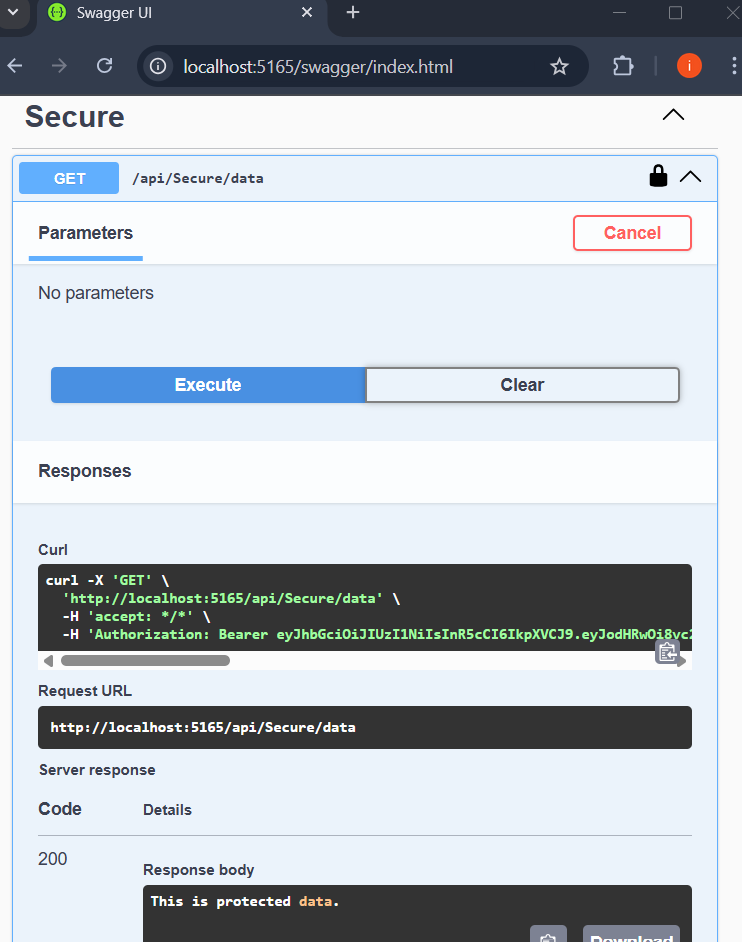
            return Ok(" This is a protected endpoint!");

        }

    }

}

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")]`.**

**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 (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (string.IsNullOrWhiteSpace(model.Username) || string.IsNullOrWhiteSpace(model.Password))

{

return BadRequest("Username and password are required.");

}

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username!);

return Ok(new { Token = token });

}

return Unauthorized();

}

private bool IsValidUser(LoginModel model)

{

// Demo: Only one valid hardcoded user with Admin role

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

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin") // ✅ Add Admin role here

};

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

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

if (Encoding.UTF8.GetBytes(keyString).Length < 32)

throw new InvalidOperationException("JWT key must be at least 256 bits (32 bytes) long.");

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

);

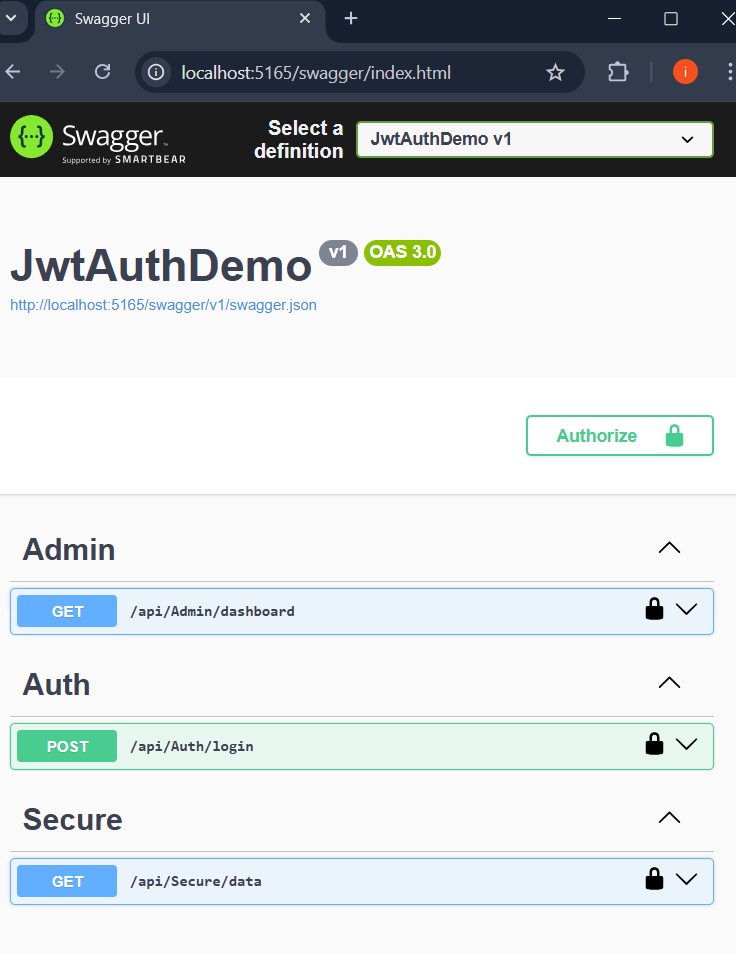
return new JwtSecurityTokenHandler().WriteToken(token);

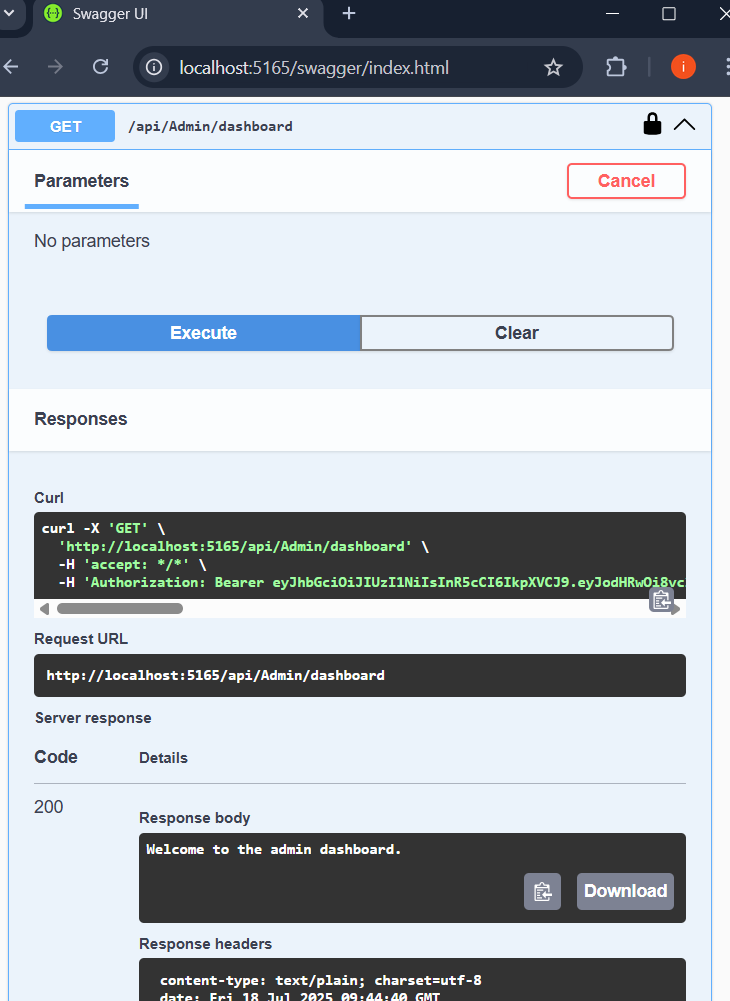
}

}

}

Output:





**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.**

**Program.cs:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.IdentityModel.Tokens.Jwt;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// ✅ JWT Authentication with custom error handling

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"]!))

    };

    // 🔐 Custom error response

    options.Events = new JwtBearerEvents

    {

        OnAuthenticationFailed = async context =>

        {

            context.Response.StatusCode = 401;

            context.Response.ContentType = "application/json";

            var message = context.Exception is SecurityTokenExpiredException

                ? "{\"error\": \"Token has expired.\"}"

                : "{\"error\": \"Invalid token.\"}";

            await context.Response.WriteAsync(message);

        }

    };

});

builder.Services.AddAuthorization();

builder.Services.AddControllers();

// ✅ Swagger with JWT Support

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(options =>

{

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

    options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

    {

        Name = "Authorization",

        Type = SecuritySchemeType.ApiKey,

        Scheme = "Bearer",

        BearerFormat = "JWT",

        In = ParameterLocation.Header,

        Description = "Enter 'Bearer' followed by your token. Example: `Bearer abc123...`"

    });

    options.AddSecurityRequirement(new OpenApiSecurityRequirement

    {

        {

            new OpenApiSecurityScheme

            {

                Reference = new OpenApiReference

                {

                    Type = ReferenceType.SecurityScheme,

                    Id = "Bearer"

                }

            },

            Array.Empty<string>()

        }

    });

});

var app = builder.Build();

// ✅ Always enable Swagger (even in production for now)

app.UseSwagger();

app.UseSwaggerUI();

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

Output:

